Oracle® Fusion Middleware

WebCenter Sites Developer's Guide

11*g* Release 1 (11.1.1.8.0)

E29634-03

June 2014

Oracle Fusion Middleware WebCenter Sites Developer's Guide, 11*g* Release 1 (11.1.1.8.0)

E29634-03

Primary Author: Promila Chitkara

Contributing Author: Melinda Rubenau, Sean Cearley, Tatiana Kolubayev

Contributor: Avadhani Chandrashekar, Joseph Granato, Kannan Appachi, Patrice Palau, Ravi Khanuja, Saikat Chaudhuri, Vijayalakshmi Rajan

Contents

Part I Developing Oracle WebCenter Sites

1 Introduction to Oracle WebCenter Sites

2 About Content Management Sites

3 Oracle WebCenter Sites Development Process

4 Programming with Oracle WebCenter Sites

5 Page Design and Caching

6 Intelligent Cache Management with WebCenter Sites

7 Advanced Page Caching Techniques

8 WebCenter Sites Tools and Utilities

9 Sessions and Cookies

10 Error Logging and Debugging

11 Data Design: The Asset Models

12 The WebCenter Sites Database

13 Managing Data in Non-Asset Tables

14 Resultset Caching and Queries

15 Designing Basic Asset Types

16 Designing Flex Asset Types

17 Flex Filters

18 Designing Attribute Editors

19 Configuring Bundled Attribute Editors

20 Importing Assets of Any Type

21 Importing Flex Assets

22 Importing Flex Assets with the BulkLoader Utility

23 Creating Template, CSElement, and SiteEntry Assets

24 Creating Templates and Wrappers

25 Coding Templates for In-Context and Presentation Editing

26 Creating Collection Assets, Query Assets, and Page Assets

27 Best Practices for Creating Future Site Preview Assets and Templates

28 Coding Elements for Templates and CSElements

29 Template Element Examples for Basic Assets

30 Configuring Sites for Multilingual Support

33 The Burlington Financial Sample Site

34 Customizing the WebCenter Sites Admin Interface

35 Customizing Workflow

36 Understanding Web Services

37 Creating and Consuming Web Services

38 Creating Visitor Data Assets

39 Recommendation Assets

40 Coding Engage Pages

41 Memory-Centric Visitor Tracking

42 Asset API Tutorial

49 RealTime Publishing Customization Hooks

50 Coding the Crawler Configuration File

Part II Working with the Developer Tools

51 About Oracle WebCenter Sites: Developer Tools

52 Developer Tools: Installing and Configuring

53 Developer Tools: WebCenter Sites Features in Eclipse

54 Developer Tools: Developing JSPs

55 Developer Tools: Creating Templates for Mobile Websites

56 Developer Tools: Synchronization and Data Exchange

57 Developer Tools: Workspaces

58 Developer Tools: Command-Line Tool

59 Developer Tools: Integrating with Version Control Systems

60 Developer Tools: Development Team Integration Use Case

61 Developer Tools: Using the Command-line Tool to Create Reusable Modules

Part III Customizing the Contributor Interface

66 Contributor Interface: Customizing Global Properties, Toolbar, and Menu Bar

67 Contributor Interface: Customizing Asset Forms

Part IV Configuring Oracle WebCenter Sites: Mobility

68 Configuring Oracle WebCenter Sites: Mobility to Support Mobile Websites

Part V Developing Applications with the Web Experience Management (WEM) Framework

69 About the Web Experience Management (WEM) Framework

70 WEM Framework: Understanding the Framework and Services

71 WEM Framework: The Articles Sample Application

72 WEM Framework: Developing Applications

73 WEM Framework: Developing Custom REST Resources

74 WEM Framework: Single Sign-On for Production Sites

75 WEM Framework: Using REST Resources

76 WEM Framework: Customizable Single Sign-On Facility

77 WEM Framework: Buffering

78 WEM Framework: Registering Applications Manually

Part VI Developing the Community-Gadgets Application

79 About Oracle WebCenter Sites: Community-Gadgets

80 Community-Gadgets: Integrating with Social Networking Services

81 Community-Gadgets: Customizing Its Functionality

84 Community-Gadgets: Guidelines for Maintaining the Application

85 Community-Gadgets: Analyzing Community Widget Tags

86 Community-Gadgets: Enabling SEO Support for Community Widgets

87 Community-Gadgets: Deploying the CSS Tag

Part VII Developing Community Blogs

88 Community Blogs: Data Model

89 Community Blogs: Sample Blog Pages

90 Community Blogs: Customizing Blog Components

Part VIII Developing Gadgets

91 About Developing Gadgets

Preface

This guide contains information about developing Oracle WebCenter Sites to support content contributors and administrators in creating, managing, and delivering highly interactive desktop and mobile websites.

Audience

This guide is written primarily for developers. It is assumed that developers have a clear knowledge of their company's business needs, and a basic understanding of their roles in the development of the online site and its back end. This guide is also useful to administrators, who collaborate with developers by setting up content management sites, site users, workflow processes, publishing methods, and Oracle WebCenter Sites client options.

Developers must know Java, JavaServer Pages (JSP), XML, and HTML. Administrators are not required to have programming experience, although a technical background is assumed.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

These additional documents may also be useful:

- *Oracle WebCenter Sites Certification Matrix*
- *Oracle WebCenter Sites Release Notes*
- *Oracle Fusion Middleware WebCenter Sites Installation Guide*
- *Oracle Fusion Middleware WebCenter Sites User's Guide*
- *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*

- *Oracle Fusion Middleware WebCenter Sites Tag Reference*
- *Oracle Fusion Middleware WebCenter Sites Java API Reference*
- *Oracle Fusion Middleware WebCenter Sites: Installing and Configuring Supporting Software*
- *Oracle Fusion Middleware WebCenter Sites Property Files Reference*
- *Oracle Fusion Middleware WebCenter Sites REST API Resource Reference*
- *Oracle Fusion Middleware WebCenter Sites REST API Bean Reference*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide

This preface introduces the new and changed features of WebCenter Sites 11*g* Release 1 (11.1.1.8.0) and other significant changes that are described in this guide, and provides pointers to additional information. This preface also lists deprecated features.

New and Changed Features in 11*g* Release 1 (11.1.1.8.0)

WebCenter Sites 11*g* Release 1 (11.1.1.8.0) includes the following new and changed features, which are described in this guide:

- **New Features**
  - WebCenter Sites: Mobility. This feature supports creating, previewing, and delivering websites to mobile devices such as phones and tablets. See Chapter 68, "Configuring Oracle WebCenter Sites: Mobility to Support Mobile Websites" in Part IV, "Configuring Oracle WebCenter Sites: Mobility" for more information.

    The ability to create templates for rendering mobile websites has also been added to WebCenter Sites: Developer Tools. For more information, see Chapter 55, "Developer Tools: Creating Templates for Mobile Websites."
  - Proxy asset type framework. This feature enables the integration of external web content into websites delivered by WebCenter Sites. See Chapter 48, "Proxy Assets: Integrating Third-Party Content Sources" for more information.
  - Vanity URLs. This feature enables users to create user-friendly URLs to web pages. Developers use `render:gettemplateurl` to create vanity URL links. See Section 45.4, "Generating Vanity URL Links in a Web Page" in Section 45, "WebCenter Sites URL Assemblers" for more information. Additional information about Vanity URLs is available in the *Oracle Fusion Middleware WebCenter Sites User's Guide* and *Oracle Fusion Middleware WebCenter Sites Administrator's Guide.*
  - Integration with Oracle Real-Time Decisions. This functionality helps site visitors make decisions by dynamically recommending their best options. See the topic Chapter , "Creating Templates for Recommendations Using Oracle Real-Time Decisions" in Chapter 40, "Coding Engage Pages" for more information.
- **Changed Features**
  - WebCenter Sites: Developer Tools now supports WebCenter Sites: Mobility, as described under "New Features," in this preface. Chapter 52, "Developer Tools: Installing and Configuring" contains instructions for installing WebCenter Sites: Developer Tools in this release.

  - Customizing the WebCenter Sites Contributor interface. Context menu configuration is now defined by the `config.contextMenus` property. See Section 66.4, "Customizing Context Menus" for more information.
  - Packaging of WebCenter Sites Community and Gadgets applications. These applications remain Web Experience Management (WEM) Framework applications that have their own interfaces, but they are now packaged together in a web application called *WebCenter Sites: Community-Gadgets*. Part VI, "Developing the Community-Gadgets Application" has been updated to reflect the change.

- **Document Consolidation**

  This guide consolidates developer's documentation that was once available in separate books. Topics that have been added to this guide are the following:

  - WebCenter Sites: Site Capture crawler code. See Chapter 50, "Coding the Crawler Configuration File" for more information. This chapter contains information about the `BaseConfigurator` class, implementing its methods and interfaces to control a crawler's site capture process, and about sample code that is available in the Site Capture installation for the FirstSiteII crawler.
  - Part II, "Working with the Developer Tools." This part describes the toolkit that is used to integrate Oracle WebCenter Sites with the Eclipse Integrated Development Environment (IDE). The Developer Tools kit enables developers to work in a distributed environment using the Eclipse IDE and version control systems (VCS).
  - Part III, "Customizing the Contributor Interface." This part describes the process of customizing the following Contributor interface features: dashboard, search views, global properties, toolbar, menu bars, context menus, and asset forms.
  - Part IV, "Configuring Oracle WebCenter Sites: Mobility"

    Note: This part is new. It describes a new feature, WebCenter Sites: Mobility, and provides information for configuring its framework to support the development of mobile websites.

  - Part V, "Developing Applications with the Web Experience Management (WEM) Framework." This part contains an overview of the WEM Framework, describes the process of developing applications and custom Representational State Transfer (REST) resources, and provides information about implementing and customizing Single Sign-On (SSO).
  - Part VI, "Developing the Community-Gadgets Application."This part describes the Community-Gadgets application and the process of integrating the application with Facebook, Twitter, Google, and Janrain. This part also provides information about customizing Community-Gadgets, translating its functionality into various languages, and maintaining the Community-Gadgets website presence.
  - Part VII, "Developing Community Blogs." This part contains procedures for customizing the blog data model. It also covers template code and provides guidelines for developing blog functionality on different content management sites.
  - Part VIII, "Developing Gadgets." This part introduces template developers to the process of creating gadgets for the Oracle WebCenter Sites: Gadgets

application. Sample gadgets, which are included with the Gadgets application, are used throughout to illustrate the development task.

   – Part IX, "Developing a Java Connector and Plugin for CIP" shows developers how to extend the Oracle WebCenter Sites: Content Integration Platform to publish from systems of their own choice to Oracle WebCenter Sites.

Deprecation Notice

The following features are deprecated in WebCenter Sites 11*g* Release 1 (11.1.1.8.0):

- SOAP-based web services. This feature is replaced by REST services.
- Mirror publishing. This feature is replaced by RealTime publishing.
- Static publishing
- Page debugging
- WebCenter Sites Desktop
- WebCenter Sites DocLink

Part I

Developing Oracle WebCenter Sites

This part describes the Oracle WebCenter Sites developer's environment. It begins with an overview of Oracle WebCenter Sites and the development process you will follow to create your content management (CM) framework. The rest of this part explains your main task of building the framework for the online site.

Building the framework for the online site requires developing the data model as well as designing and writing code to deliver content with the look and feel that best represents your organization's business. It also involves implementing page caching, security, session management techniques and implementing optional functionality; such as web services and features that are provided by add-on products.

This part provides information on developing WebCenter Sites and contains the following chapters:

- Chapter 21, "Importing Flex Assets"
- Chapter 22, "Importing Flex Assets with the BulkLoader Utility"
- Chapter 23, "Creating Template, CSElement, and SiteEntry Assets"
- Chapter 24, "Creating Templates and Wrappers"
- Chapter 25, "Coding Templates for In-Context and Presentation Editing"
- Chapter 26, "Creating Collection Assets, Query Assets, and Page Assets"
- Chapter 27, "Best Practices for Creating Future Site Preview Assets and Templates"
- Chapter 28, "Coding Elements for Templates and CSElements"
- Chapter 29, "Template Element Examples for Basic Assets"
- Chapter 30, "Configuring Sites for Multilingual Support"
- Chapter 31, "User Management on the Delivery System"
- Chapter 32, "The HelloAssetWorld Sample Site"
- Chapter 33, "The Burlington Financial Sample Site"
- Chapter 34, "Customizing the WebCenter Sites Admin Interface"
- Chapter 35, "Customizing Workflow"
- Chapter 36, "Understanding Web Services"
- Chapter 37, "Creating and Consuming Web Services"
- Chapter 38, "Creating Visitor Data Assets"
- Chapter 39, "Recommendation Assets"
- Chapter 40, "Coding Engage Pages"
- Chapter 41, "Memory-Centric Visitor Tracking"
- Chapter 42, "Asset API Tutorial"
- Chapter 43, "Public Site Search"
- Chapter 44, "Creating a Hierarchical Flex Family"
- Chapter 45, "WebCenter Sites URL Assemblers"
- Chapter 46, "White Space and Compression"
- Chapter 47, "Asset and Publish Events in WebCenter Sites"
- Chapter 48, "Proxy Assets: Integrating Third-Party Content Sources"
- Chapter 49, "RealTime Publishing Customization Hooks"
- Chapter 50, "Coding the Crawler Configuration File"

1

Introduction to Oracle WebCenter Sites

Oracle WebCenter Sites is a high-performance, large-scale content management and delivery system. You and your development team use Oracle WebCenter Sites to create and manage large and complex websites, sites that run businesses, and other types of sites, all of which are generically referred to as "online sites" or "websites."

The following sections provide an overview of WebCenter Sites:

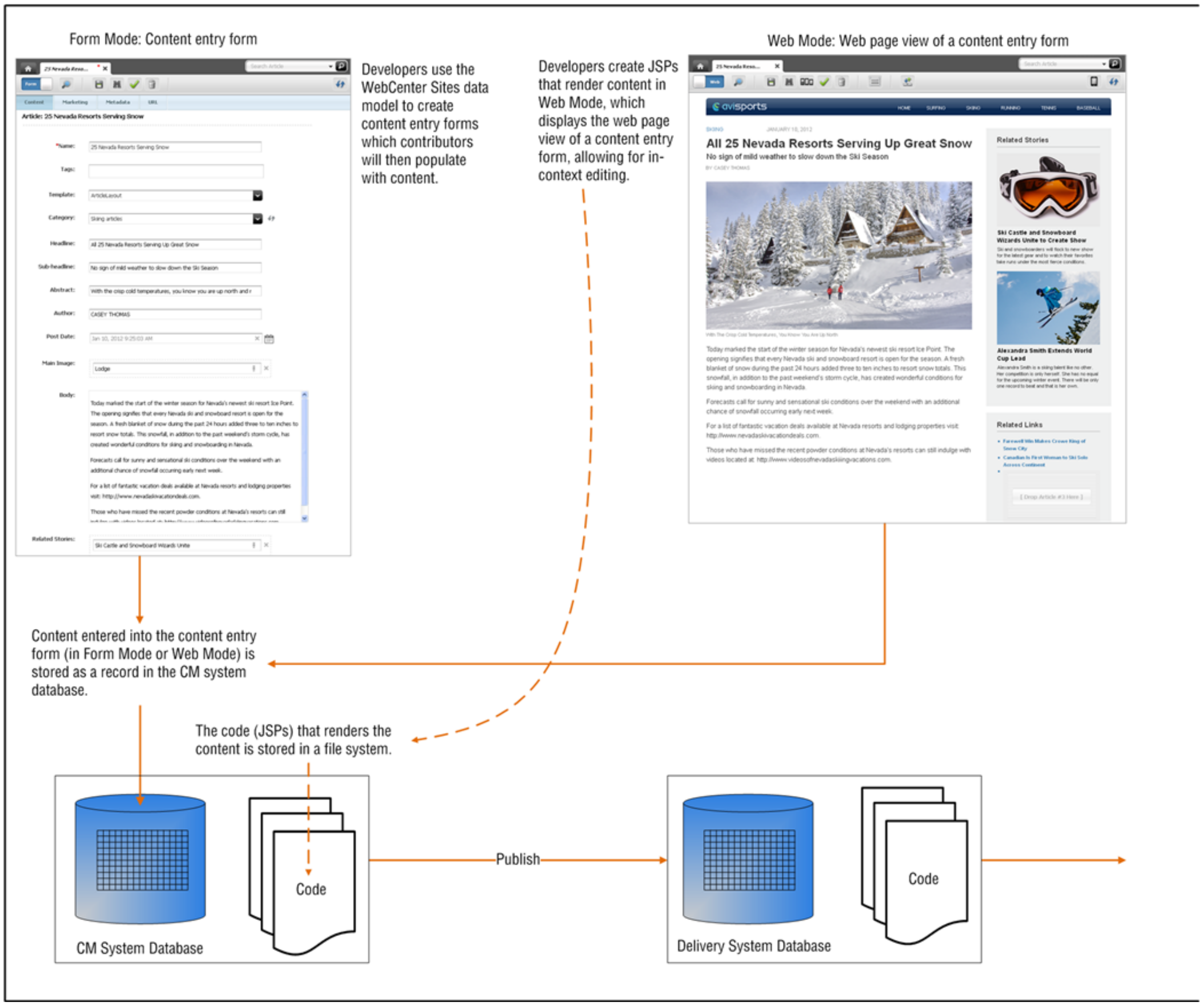
- Section 1.1, "From the Content Entry Form to the Website"
- Section 1.2, "Developing a Dynamic Website"
- Section 1.3, "WebCenter Sites Data Design"
- Section 1.4, "Presentation Logic"
- Section 1.5, "WebCenter Sites Systems"
- Section 1.6, "Approval and Publishing"
- Section 1.7, "Performance: Caching"
- Section 1.8, "WebCenter Sites Interfaces"
- Section 1.9, "Sample Sites"

> **Note:** The following features are deprecated in WebCenter Sites 11*g* Release 1 (11.1.1.8.0): Sites Desktop, Sites DocLink, Static publishing.

1.1 From the Content Entry Form to the Website

As a WebCenter Sites developer, your job is to create the content entry forms which contributors will then use to create the content for your website. In addition, you will also develop the JSPs that render content entry forms in Web Mode as well as render published content on the website. When content is ready for public delivery, it can be published to the website using either dynamic or static publishing.

Figure 1–1 Content Entry Forms



Form Mode: Content entry form
Web Mode: Web page view of a content entry form
All 25 Nevada Resorts Serving Up Great Snow
No sign of mild weather to slow down the Ski Season
Related Stories
Related Links
Code
CM System Database
Publish
Code
Delivery System Database

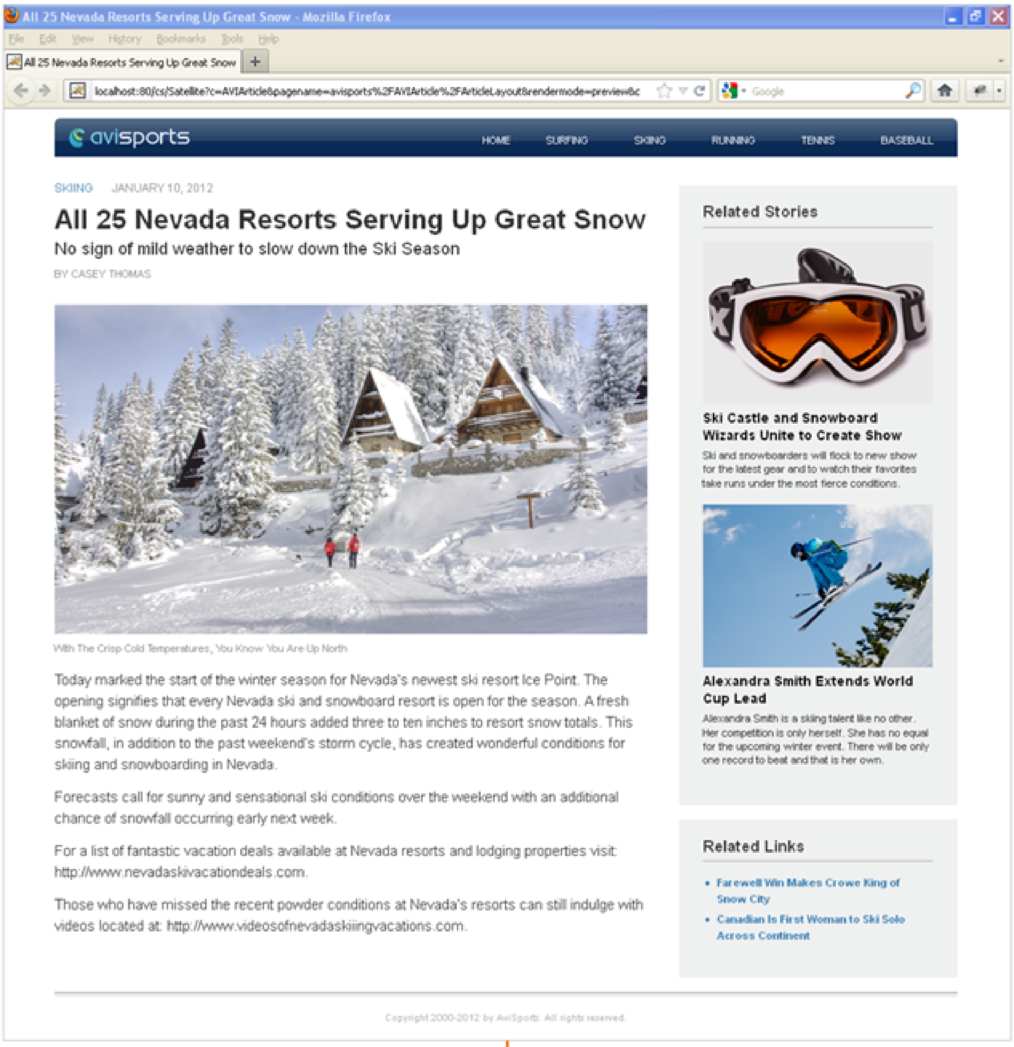
Developers use the WebCenter Sites data model to create content entry forms which contributors will then populate with content.

Developers create JSPs that render content in Web Mode, which displays the web page view of a content entry form, allowing for in-context editing.

Content entered into the content entry form (in Form Mode or Web Mode) is stored as a record in the CM system database.

The code (JSPs) that renders the content is stored in a file system.

Figure 1–2 Website (online site)


All 25 Nevada Resorts Serving Up Great Snow - Mozilla Firefox
File Edit View History Bookmarks Tools Help
All 25 Nevada Resorts Serving Up Great Snow
localhost:80/cs/Satellite?c=AVIArticle&pagename=avisports%2FAVIArticle%2FArticleLayout&rendermode=preview&c
Google
avisports
HOME
SURFING
SKIING
RUNNING
TENNIS
BASEBALL

SKIING JANUARY 10, 2012

All 25 Nevada Resorts Serving Up Great Snow

No sign of mild weather to slow down the Ski Season

BY CASEY THOMAS

With The Crisp Cold Temperatures, You Know You Are Up North

Today marked the start of the winter season for Nevada's newest ski resort Ice Point. The opening signifies that every Nevada ski and snowboard resort is open for the season. A fresh blanket of snow during the past 24 hours added three to ten inches to resort snow totals. This snowfall, in addition to the past weekend's storm cycle, has created wonderful conditions for skiing and snowboarding in Nevada.

Forecasts call for sunny and sensational ski conditions over the weekend with an additional chance of snowfall occurring early next week.

For a list of fantastic vacation deals available at Nevada resorts and lodging properties visit: http://www.nevadaskivacationdeals.com.

Those who have missed the recent powder conditions at Nevada's resorts can still indulge with videos located at: http://www.videosofnevadaskiingvacations.com.

Related Stories

Ski Castle and Snowboard Wizards Unite to Create Show

Ski and snowboarders will flock to new show for the latest gear and to watch their favorites take runs under the most fierce conditions.

Alexandra Smith Extends World Cup Lead

Alexandra Smith is a skiing talent like no other. Her competition is only herself. She has no equal for the upcoming winter event. There will be only one record to beat and that is her own.

Related Links

- Farewell Win Makes Crowe King of Snow City
- Canadian Is First Woman to Ski Solo Across Continent

Copyright 2000-2012 by AviSports. All rights reserved.

Formatted content is displayed on the website by JSPs.

1.2 Developing a Dynamic Website

WebCenter Sites supports both static and dynamic pages. Dynamic page generation is its main function, and is the subject of this section of the guide.

A dynamic WebCenter Sites page differs from a typical HTML page, as shown in the following table:

Table 1–1 Static Page vs. Dynamic Page

Static Page (HTML Page)	Dynamic Page (WebCenter Sites Page)
Single disk file, served via a web server.	Composed and created upon request.
One-to-one association between the HTML page and the page the visitor sees in the web browser.	The web page that the visitor sees can be composed of multiple components called *pagelets*, created from within WebCenter Sites.

Table 1–1 (Cont.) Static Page vs. Dynamic Page

Static Page (HTML Page)	Dynamic Page (WebCenter Sites Page)
No separation of presentation and content. As a result, it is difficult to modify presentation and content independently of each other.	Separation of presentation and content.As a result, presentation and content can be modified and maintained independently of each other.

1.3 WebCenter Sites Data Design

As a developer, you use the data models provided by WebCenter Sites to create the content entry forms (shown in Figure 1–1) that contributors will use to create content for the website. Each field in a content entry form maps to a corresponding column in a database table (or multiple tables).

WebCenter Sites supports the following data models:

- **WebCenter Sites basic asset model**, supports a flat data structure, which means that basic assets cannot inherit each other's properties (called attributes in this guide). Content is entered by WebCenter Sites users and is stored as objects called **assets** in the WebCenter Sites database. Each type of asset is contained in one primary storage table in the database, such that basic assets of one type can be associated with basic assets of another type.
- **WebCenter Sites flex asset model**, is a comprehensive data model in which each asset type uses several storage tables such that hierarchical data structures can be created, and child assets inherit attribute values from their parent assets. The flex asset model also supports flat data structures, within its own framework. (Note that the flex asset model functions independently of the basic asset model; tables created within the two models do not intersect.)

Whether you choose the flex asset model or the basic asset model depends on the complexity of the data you plan to serve to your visitors. The flex asset model has historically been used for creating large online catalogs of products. However, it can be used in less complex situations, and is especially desirable when the intent is to eventually convert flat data structures to hierarchical structures. The conversion process does not require you to re-create the data.

1.4 Presentation Logic

As a developer you use APIs and JSP tags to code templates and elements which will be used to render content on the website. This section provides an overview of the programming components you will use in the process of coding.

This section contains the following topics:

- Section 1.4.1, "Element Files"
- Section 1.4.2, "APIs and JSP Tags"
- Section 1.4.3, "Sessions and Cookies"
- Section 1.4.4, "WebCenter Sites Utilities"
- Section 1.4.5, "Website Navigation"

1.4.1 Element Files

In very simplistic terms, the main function of WebCenter Sites is to separate format from content. By separating the two, WebCenter Sites enables you to reuse the same bits of formatting code for many pieces of content. For example, if you want to change the format of articles, you rewrite the code in one place, rather than having to rewrite code for every article in your system.

Your formatting code is stored in files called **elements**. The code extracts the content from the database and formats the content. Because content is formatted only when a page is requested, you have the opportunity to design pages that will be constructed on-the-fly, according to the identity of the visitor requesting them.

Element files are stored in the `ElementCatalog` table in the WebCenter Sites database. The names of your pages are stored in the `SiteCatalog` table. That is, the `SiteCatalog` table stores the entries for all the legal page names for your website.

Each row in the `SiteCatalog` table is a page entry. Each page entry points to an element in the `ElementCatalog` table. The element being pointed to by a page entry is called the **root element** of the page entry.

WebCenter Sites renders your content into an online page by executing `SiteCatalog` page entries. Here is how it works:

1. A visitor enters a URL to your website in a browser.
2. The web server that processes the HTTP request maps that URL to a WebCenter Sites URL. For example, this is a WebCenter Sites URL:

   ```
   http://www.BurlingtonFinancial.com/servlet/ContentServer?pagename=BurlingtonFin
   ancial/Home
   ```

   The text at the end of a WebCenter Sites URL is called the pagename. In this example, the pagename is `BurlingtonFinancial/Home`.
3. WebCenter Sites looks up the page name in the `SiteCatalog` table, determines its root element, locates that element in the `ElementCatalog` table, and then invokes that element.
4. The element is executed. If there are calls to other elements from within the root element, those elements are executed in turn.
5. The results (images, articles, and so on, including any HTML tags) are rendered into HTML code and returned to the visitor's browser.

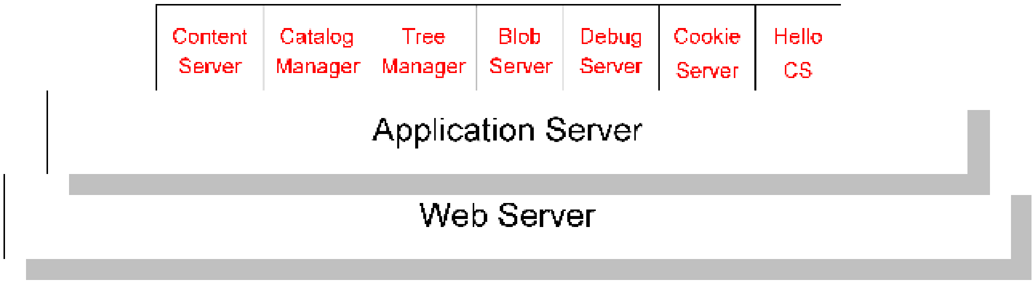
The result is a page that is dynamically rendered on demand.

1.4.2 APIs and JSP Tags

WebCenter Sites includes several tag families that you use to code your elements. The tag families enable you to identify, extract, and then display assets on your website. WebCenter Sites also provides Java methods and utilities that you can use for designing your website, for developing your own content management applications, and for customizing the WebCenter Sites modules/products.

> **Note:** For information about coding pages that display assets that use the basic data model, see Chapter 28, "Coding Elements for Templates and CSElements." For information about all of the custom tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference.*

The WebCenter Sites operating system consists of several servlets that run on top of an application server. Each servlet is invoked when necessary to perform a discrete set of tasks. Each servlet has a corresponding Java API with Java methods and JSP tags that you use to invoke the functions that you need to use. The main WebCenter Sites servlets are shown in the following figure:



The main WebCenter Sites servlets are as follows:

- **ContentServer**: Generates and serves pages dynamically. This servlet provides disk caching, session management, event management, searching, and personalization services.
- **CatalogManager**: Provides most of the database management for the WebCenter Sites database, including revision tracking, security, resultset caching, and publishing services.
- **TreeManager**: Manages the tree tables, which store hierarchical information about other tables in the WebCenter Sites database.
- **BlobServer**: Locates and serves binary large objects (blobs). Blobs are not processed in any way. They are served as is, as they are stored.
- **DebugServer**: Provides tools that help you debug your XML code.
- **CookieServer**: Serves cookies for WebCenter Sites pages, whether those pages are delivered by the ContentServer servlet or by the Satellite Server application.
- **HelloCS**: Displays version information about the WebCenter Sites software installed on your system.

In general, you do not need to know which servlet performs which service or task. You simply invoke the appropriate Java method or XML or JSP tag and let the WebCenter Sites core application determine which servlet to call. The exception to this rule is when you write code that references a servlet URL; that is, when you include a link to a blob or to another page on a WebCenter Sites page. Because the ContentServer servlet and the BlobServer servlet reside at different URLs, you must include the URL of the appropriate servlet in your `<A HREF>` tags.

For information about the coding links to blobs and pages, see Chapter 4, "Programming with Oracle WebCenter Sites" and Chapter 28, "Coding Elements for Templates and CSElements."

1.4.3 Sessions and Cookies

WebCenter Sites automatically creates a session for a visitor when he or she visits your website for the first time. You can store information about that visitor in session variables by using the tags and methods in the WebCenter Sites core. Subsequent elements can then access those variables and respond conditionally to them.

Session variables, however, are volatile. They last only as long as the session lasts, that is, until one of the following events occurs:

- The visitor closes his or her browser.

- The session times out after a period of inactivity. You control session timeouts through the `cs.timeout` property in the `futuretense.ini` file.
- The application server is restarted (except in a cluster).
- The session is disabled in some other way.

To store information on a more permanent basis, you would use cookies. You can code your elements to write cookies that store information about your visitors to their browsers. Then, you can use the stored information to customize pages and display the appropriate version of a page to the appropriate visitor when he or she returns to your website.

For more information about sessions and cookies, see Chapter 9, "Sessions and Cookies."

1.4.4 WebCenter Sites Utilities

WebCenter Sites provides **utilities**, which are the developer's tools for managing the WebCenter Sites database and various code. The utilities are GUI-based and must be manually installed (unless otherwise noted). They are:

- Developer Tools, which integrates WebCenter Sites with the Eclipse Integrated Development Environment (IDE). The Developer Tools kit enables WebCenter Sites developers to work in a distributed environment using tools such as Eclipse and version control system (VCS) integration.
- Sites Explorer, for viewing and editing tables in the WebCenter Sites database.
- CatalogMover, for exporting and importing database tables.
- XMLPost, for incrementally importing data into the WebCenter Sites database.
- BulkLoader, for quickly importing large amounts of data into the WebCenter Sites database.
- Property Editor, for viewing and organizing property files (system configuration files).

1.4.5 Website Navigation

The website navigation for the content management site you are currently logged in to is represented by the **Site Plan** tab in the WebCenter Sites Admin interface.

For information about content management sites and the **Site Plan** tab, see Chapter 2, "About Content Management Sites." For additional information, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

1.5 WebCenter Sites Systems

When you are working with WebCenter Sites for your content management needs, you and the others on your team work with up to four different systems:

- **Development** system, where developers and designers plan and create the website.

  All of the WebCenter Sites products that you have purchased are installed on this system.

- **Management** system, where content providers such as writers, editors, graphic artists, and marketers are assigned to content management sites to develop the content that is delivered to visitors of the website. Revision tracking and workflow features track changes to assets (content), monitoring them until they are approved to be published to the delivery system.

Content management sites represent the real website. For example, you could create separate content management sites for separate sections of your website because the teams who provide content for each section work completely separately from each other and only members of that team should have access to that section (content management site). Or, you could create a content management site that represents an entire website, as does the avisports sample site. For more information about content management sites, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

- **Delivery** system, where the content you are making available or the products that you are selling are served to your visitors or customers.

  If you are delivering your content dynamically, all of the WebCenter Sites products that you purchased are installed on this system. If you are delivering your content statically. That is, if you are serving static HTML pages, your delivery system is a web server only, and you do not need to install any of the WebCenter Sites products on that system.

- **Testing** system, where you or your QA engineers test the performance of both the management system and the delivery system. If a dedicated testing system is not available, testing can be done on the development system.

As a developer, you spend the majority of your time working on the development system. When the asset types that you develop and the site that you have designed are ready, you migrate (publish) your work from the development system to the management system. As assets are created, modified, and approved by the content providers on the management system, they are published from the management system to the delivery system.

1.6 Approval and Publishing

As mentioned in the previous section, when you finish developing the website, you publish your work (templates, elements, asset types, the site plan, and so on) from the development system to the management system. Publishing your work makes it available on the management system. Contributors can then use the asset types and your site design to create content for the website. When contributors are finished creating the site content, that content (along with the supporting asset types, templates, elements, site plan, and so on) can be approved and published to the website.

When assets are ready to be published, someone first marks them as approved. Then, when the publishing process is ready to start, it invokes the approval system which compiles a list of all the approved assets and examines all the dependencies for those assets. If an asset is approved but an asset that it is linked to is not approved, the approved asset is not published until the linked asset is also approved.

The WebCenter Sites publishing and approval systems track and verify all the asset dependencies in order to maintain the integrity of the content on your delivery system. The publishing and approval systems ensure that the assets which you have determined to be ready for publishing are the only assets that get published.

When you publish content and elements, WebCenter Sites copies the content and elements from one system (for example, your management system) to another system (for example, the delivery system). WebCenter Sites delivers two publishing methods that are built from the WebCenter Sites publishing APIs. These publishing methods interact with the WebCenter Sites approval system, an underlying system that determines which assets have been approved.

The WebCenter Sites publishing methods are:

- **RealTime** is the dynamic publishing method. It is built with the WebCenter Sites RealTime API to copy approved assets from the WebCenter Sites database on one system to the WebCenter Sites database on another system.
- **Export to Disk** is the static publishing method. It renders your approved assets into static HTML files, using the template elements assigned to them to format them. An administrator or automated process then copies those files to your delivery system using FTP or another file transfer method.

For more information about configuring publishing, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

For information about coding elements so that they log dependencies appropriately and how WebCenter Sites calculates approval dependencies, see Chapter 28, "Coding Elements for Templates and CSElements."

For information about how you approve assets, see the *Oracle Fusion Middleware WebCenter Sites User's Guide* and the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

1.7 Performance: Caching

As a developer, you implement various caching frameworks that are supported by WebCenter Sites to optimize the performance of your company's website. WebCenter Sites also supports the use of Satellite Server caching, which provides a second level of caching for your WebCenter Sites system and can also be used as a remote cache for your web pages. By default, WebCenter Sites and Satellite Server use inCache as their page caching framework.

This section contains the following topics:

- Section 1.7.1, "Page Caching"
- Section 1.7.2, "Resultset Caching"
- Section 1.7.3, "Asset Caching"
- Section 1.7.4, "Satellite Server"

1.7.1 Page Caching

Page caching is implemented at the template level and is used to cache pages on the WebCenter Sites system. Page caching plays a significant role in system performance. If an element is not changed and it will generate the same page each time it is invoked, why make WebCenter Sites process the element each time it is called? If the generated page is cached, it can be served much faster than it can if it must first be generated.

WebCenter Sites alone (independently of Satellite Server) can separately cache each page or pagelet that is identified by a page entry in the `SiteCatalog` table. You can mark the expiration date of any pagelet in the cache by specifying a value for that page entry in that table.

Page caching is made especially effective by the addition of Satellite Server. Installing a Satellite Server application amounts to installing page caches on the servers that host Satellite Server, thereby extending the WebCenter Sites page cache.

- For information about page caching, see Chapter 5, "Page Design and Caching."

- For information about inCache page caching, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.
- For information about Satellite Server, see Section 1.7.4, "Satellite Server."

1.7.2 Resultset Caching

Resultset caching is another feature that can greatly enhance system performance. When the WebCenter Sites database is queried by any mechanism, the WebCenter Sites application can cache the resultset that it returns. The WebCenter Sites application keeps track of every table in the database; whenever a table is modified, it flushes all the resultsets that were cached for that table.

For more information about resultset caching, see Chapter 14, "Resultset Caching and Queries."

1.7.3 Asset Caching

Asset caching is a memory-based system that is built on the inCache framework to optimize the performance of WebCenter Sites by taking up load that would otherwise affect the database. In WebCenter Sites, programmatic usage of assets consists of loading and rendering their attributes. Given that assets are loaded by templates, which are stored in the WebCenter Sites database, `AssetCache` is used only on WebCenter Sites nodes. Asset caching includes the `AssetCache` container component which functions by caching assets and interacting with existing inCache components.

For more information about the inCache framework and asset caching, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

1.7.4 Satellite Server

Satellite Server is a caching application. It supplements WebCenter Sites caching functionality by providing additional page caches. The tandem use of the WebCenter Sites and Satellite Server caches results in automatic double-buffered caching.

By default, co-resident Satellite Server is installed on the same computer where WebCenter Sites is installed. You can further improve your system's performance by installing Satellite Server remotely so it can cache pages and pagelets closer to their intended audience. Remote Satellite Server hosts are fast, inexpensive caches of WebCenter Sites pages. They reduce the load on the WebCenter Sites host, dramatically increase the speed of page delivery to your site visitors, and provide a simple and inexpensive way to scale your WebCenter Sites system.

1.7.4.1 Handling HTTP Requests

When the load balancer routes an HTTP request for a page to Satellite Server, Satellite Server either serves the page if the page is in its cache, or if the page is not cached, it forwards the HTTP request to WebCenter Sites. The basic chain of events is the following:

1. Satellite Server checks its cache.
2. What happens next depends on whether the page is in the Satellite Server cache:

Table 1–2 Page is in Satellite Server Cache vs. Not in Satellite Server Cache

Page is in Satellite Server Cache	Page is Not in Satellite Server Cache
Satellite Server serves the page to the visitor's browser. In this case, Satellite Server does not have to forward the request to the WebCenter Sites database, thus reducing the load on the WebCenter Sites database.	■ Satellite Server forwards the request to WebCenter Sites. ■ If WebCenter Sites has the page in its cache, it returns the cached page to Satellite Server. If the page is not in the WebCenter Sites cache, WebCenter Sites renders the page, caches a copy, and sends the page to Satellite Server. ■ Satellite Server then caches the page and serves it to the visitor's browser. When requested again, the page will be served from the Satellite Server cache, which reduces the load on the WebCenter Sites database.

Each Satellite Server application is independent of every other Satellite Server application. An individual Satellite Server application has the following characteristics:

- It maintains its own cache.
- It cannot request pages or pagelets from another Satellite Server application. It can request pages or pagelets from only the WebCenter Sites core.

1.7.4.2 Satellite Server Servlets

Satellite Server is made up of several servlets: one that caches and serves pages, and two that manage the cache:

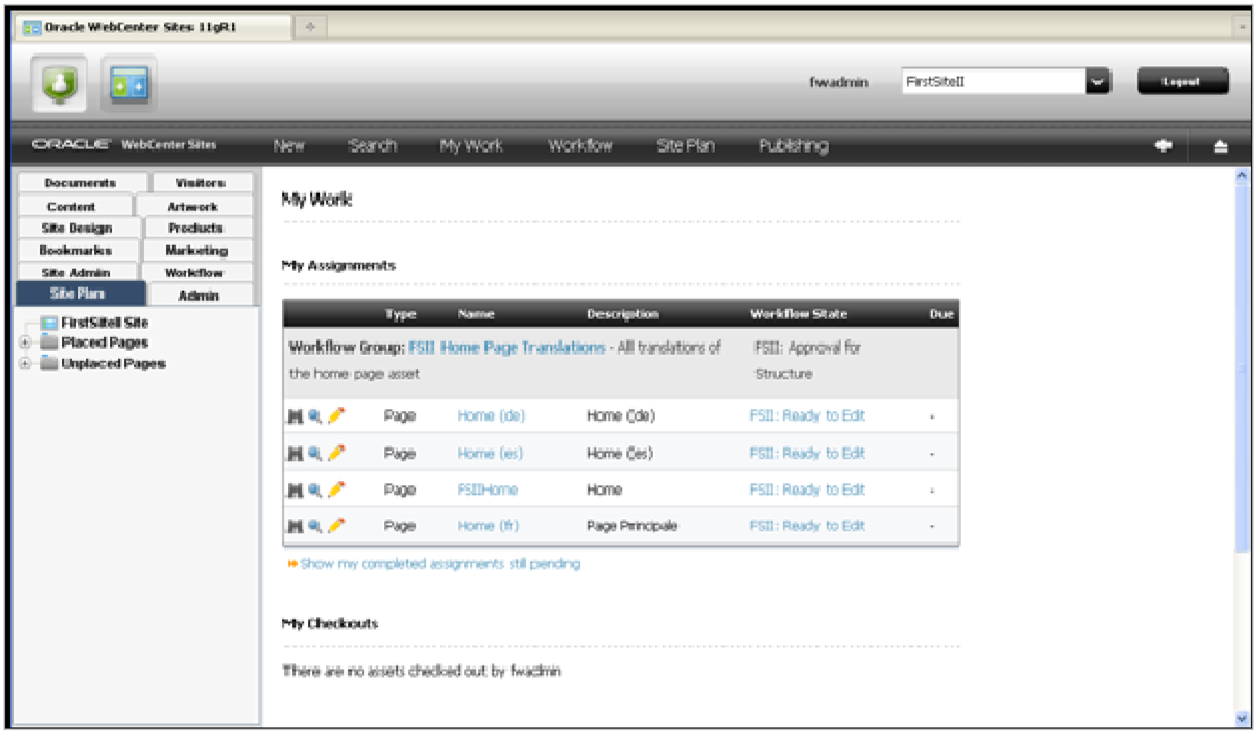
- **Satellite**: Caches pages at the pagelet level. The Satellite XML or JSP tags in your elements indicate which pagelets should be cached, and they control various Satellite Server settings.
- **Inventory**: Enables you to examine the Satellite Server cache so you can obtain the information you need to manually flush individual pages or pagelets from the cache when necessary.
- **FlushServer**: Handles all types of cache-flushing. FlushServer can either flush the entire cache, or can flush individual items from the cache.

For information about coding pages with the Satellite Server tags and page caching in general, see Chapter 5, "Page Design and Caching."

1.8 WebCenter Sites Interfaces

This section provides an overview of the WebCenter Sites interfaces. As a developer, you will work mainly with the WebCenter Sites Admin interface.

- **Admin Interface** is designed for developers and administrators. This interface allows administrators to manage and configure WebCenter Sites.



My Work

My Assignments

Type	Name	Description	Workflow State	Due
Workflow Group: FSII Home Page Translations - All translations of the home page asset			FSII: Approval for Structure	
Page	Home (de)	Home (de)	FSII: Ready to Edit	-
Page	Home (es)	Home (es)	FSII: Ready to Edit	-
Page	FSIIHome	Home	FSII: Ready to Edit	-
Page	Home (fr)	Page Principale	FSII: Ready to Edit	-

Show my completed assignments still pending

My Checkouts

There are no assets checked out by fwadmin

The tree panel on the left contains all the content management elements that developers and administrators need to work with. The workspace area on the right is where all the tasks and operations are performed.

The Admin interface supports code-based operations, and enables you to graphically complete the creation of basic asset types. For example, to create a basic asset type, you would:

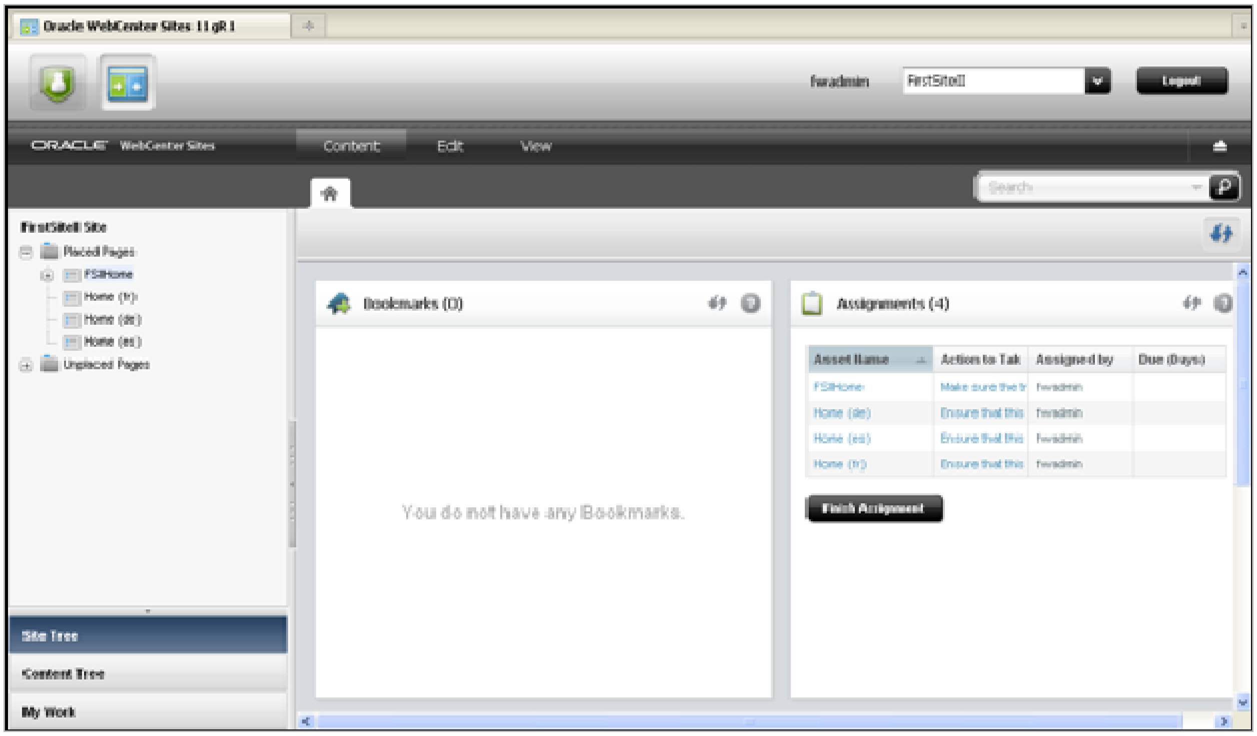
a. Write an XML file (called *asset descriptor* files) to define the basic asset type.

b. Upload the file to WebCenter Sites.

c. Use the WebCenter Sites interface to invoke the AssetMaker utility. One of the functions of the interface (AssetMaker) is to read the asset descriptor file and, from it, create a storage table for the asset type. Other functions in the interface allow you to configure the asset type (for example, name its authorized users).

The same interface is used by administrators to create content management sites, manage system users, control their permissions to content, establish workflow processes, and configure WebCenter Sites features (such as Sites Desktop).

- **Contributor Interface** is designed specifically for content providers and business users. The Contributor interface provides ease of use and quick access to most WebCenter Sites content management functions, such as previewing, creating, editing, deleting, and approving assets.

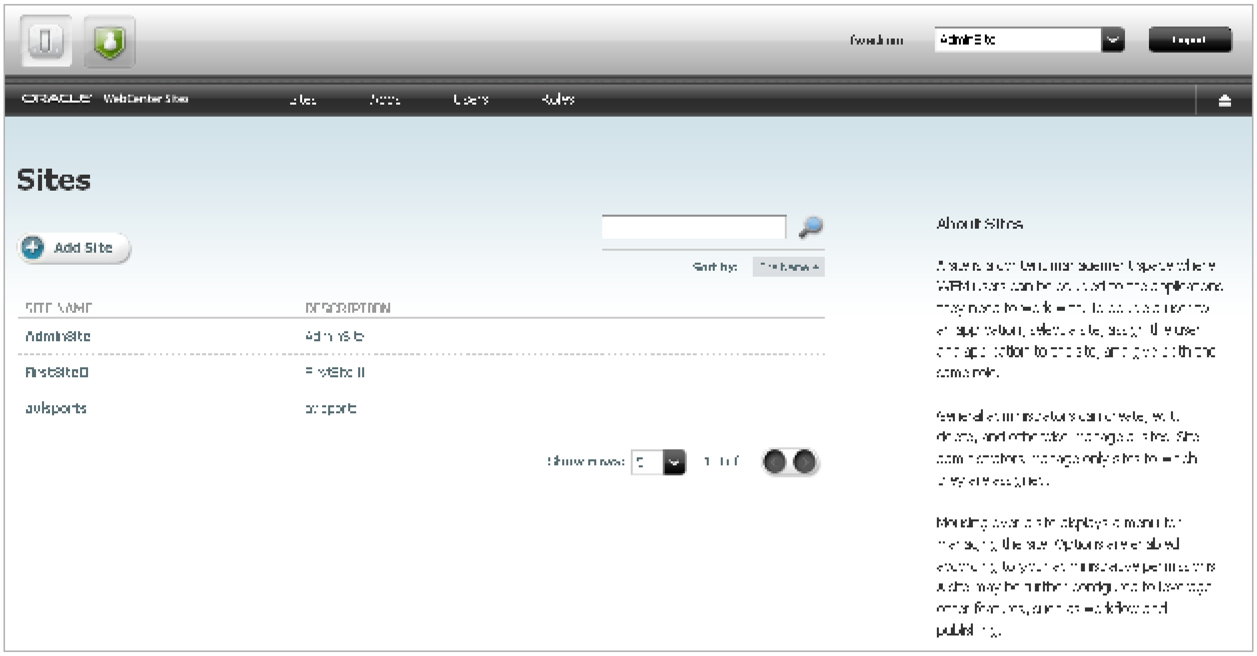


Asset Name	Action to Tak	Assigned by	Due (Days)
FSIIHome	Make sure the tr	fwadmin	
Home (de)	Ensure that this	fwadmin	
Home (es)	Ensure that this	fwadmin	
Home (fr)	Ensure that this	fwadmin	

When you work with assets in the Contributor interface, you may see fields enabled with the following WYSIWYG editors:

- **CKEditor** is an open source WYSIWYG text editor from CKSource which requires no client-side installation. Developers can use CKEditor to create basic assets whose text-entry fields use CKEditor as the input mechanism for the field. Developers can also create attribute editors for flex attributes that use CKEditor as the input medium.
- **Clarkii Online Image Editor (Clarkii OIE)** is a popular third-party image editor from InDis Baltic. Developers can enable Clarkii OIE to allow users to edit images directly in the Form Mode of the WebCenter Sites Contributor interface eliminating the need for an external image editor.

- **WEM Admin Interface** is designed specifically for administrators to manage the assignment of applications and users to sites via roles.



Sites

SITE NAME	DESCRIPTION
AdminSite	AdminSite
FirstSiteII	FirstSite II
avisports	avisports

About Sites

[illegible]

For more information about the WEM Admin interface, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide* and Part V, "Developing Applications with the Web Experience Management (WEM) Framework".

WebCenter Sites also supports clients (optional interfaces) such as MS Word and applications that offer functionality similar to Windows Explorer and Windows Desktop. The clients are as follows:

- **WebCenter Sites Desktop** offers content authors the familiar Microsoft Word interface as an alternative to the WebCenter Sites interfaces. Authors create their content directly in Word documents. However, note that Sites Desktop requires the content in Word documents to be structured, because the content will be parsed to database tables.

  For example, when using Sites Desktop to author content, the user opens a Word document, enters content, and structures the content by tagging it with the same field names as defined in the equivalent content-entry form that WebCenter Sites provides. The tagging utility is embedded in the Word interface, and the selection of fields is determined by the WebCenter Sites administrator. When the Word document is saved, the content in its fields is parsed to fields in the appropriate database table(s). The Word document remains available for editing.

  Note: Sites Desktop must be manually installed on client computers.

- **Sites DocLink** supports unstructured content in the flex asset family.

  Sites DocLink provides a drag-and-drop interface for uploading and downloading unstructured types of content (documents, graphics, and other single binary files) that are managed as flex assets. Sites DocLink also presents the hierarchical structure of any flex asset family in the WebCenter Sites database as folders and files in the Windows Explorer application.

  Note: Sites DocLink must be manually installed on client computers.

For more information about configuring Sites Desktop and Sites DocLink for specific users, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*

1.9 Sample Sites

WebCenter Sites provides the sites described in this section. Avisports and FirstSiteII code can be reused and modified to suit your organization's business needs. This section of the guide uses both of these sample sites to illustrate best coding practices.

This section contains the following topics:

- Section 1.9.1, "The Avisports Sample Site"
- Section 1.9.2, "The FirstSiteII Sample Site"
- Section 1.9.3, "Only Available in the WebCenter Sites Demo JSK"

1.9.1 The Avisports Sample Site

Avisports is a sports-centric sample site containing sports articles illustrated with images. Avisports is used to illustrate features in the WebCenter Sites Contributor interface such as creating and editing assets in Form Mode and Web Mode. In addition, Avisports provides developers with sample templates that are coded to render an asset's Create and/or Edit view in Web Mode.

1.9.2 The FirstSiteII Sample Site

FirstSiteII is an electronics retail site that includes articles, images, a product catalog with several entries, documents, and sample user accounts. Utilizing our best practices in WebCenter Sites site development, FirstSiteII provides an excellent starting point for developers.

1.9.3 Only Available in the WebCenter Sites Demo JSK

The following sample sites are only available in the WebCenter Sites Demo JSK, which is located on the Oracle Technology Network on the WebCenter Sites download page.

The Burlington Financial Sample Site

WebCenter Sites provides a fully functional sample site named Burlington Financial. The site is used in this section of the guide as the source of examples that illustrate the basic asset model, WebCenter Sites functionality, and coding practices. You can examine the examples both in this book and online in the context of the actual site.

The Burlington Financial site contains sample asset types, elements, `SiteCatalog` entries, a workflow process, and so on. For information about the Burlington Financial sample site, see Chapter 33, "The Burlington Financial Sample Site."

The HelloAssetWorld Sample Site

In addition to the Burlington Financial sample site, WebCenter Sites delivers a sample site called *HelloAssetWorld*. The site provides a simple introduction to creating a WebCenter Sites website.

The templates that compose HelloAssetWorld are described in Chapter 32, "The HelloAssetWorld Sample Site." Further information about the site's configuration and users is available in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

2 About Content Management Sites

In the Oracle WebCenter Sites environment, a content management site is an object that you use as an aid in the WebCenter Sites interface for designing the online site and for managing access to assets. The Burlington Financial sample site is a content management site. So is the GE Lighting sample site.

You first create content management sites on the development system. When the sites are tested and approved for use, you must then duplicate the sites (with exactly the same names) on the management and delivery systems.

This chapter contains the following sections:

2.1 Content Management Sites

A *content management (CM) site* is an object that you use as a design aid or organizational construct for the online site that you are delivering from your WebCenter Sites delivery system. A content management site represents your online site.

> **Note:** In this guide, *content management site* is also called CM site, or simply site.

When you log in to WebCenter Sites running any of the WebCenter Sites modules and products, you are logging in to a content management site. If you have access to more than one site, the first decision that you make after logging in is which site to work on. From that point on, all of the tasks that you complete are completed in the context of that site (until you switch sites).

Content management sites are used by three types of users in different ways:

- Developers use content management sites to design the online site:
  - Create the data model (the source of content-entry and editorial forms that business users will use in order to provide content for the online site)
  - Enable the data model for selected content management sites

  - Code templates that extract content from the database, format the content, and deliver the content to the online site. You use the **Site Plan** tab to create a design framework for the online site. Each content management site has a separate site plan, which is stored in the `SitePlanTree` table.
  - Enable the templates for the content management sites
  - Implement page caching
  - Make use of session data
  - Write code that gathers information about site visitors
  - Establish security

    > **Note:** When you create data (for example, asset types and templates), the data is entered into a system-wide pool, regardless of the content management site that you have chosen to log in to.
    >
    > Enabling the data for the site that you are logged in to (and for any other site) links the data to that site and makes it accessible to users of that site.

- WebCenter Sites administrators use content management sites to control users' access to content:
  - WebCenter Sites administrators can restrict users from accessing certain assets and asset types on the WebCenter Sites system.

    Asset types must be enabled for a content management site. Therefore, if an administrator decides not to enable an asset type for a content management site, then users who log in to that content management site do not have access to assets of that type. WebCenter Sites administrators can also restrict users from accessing a content management site.
  - WebCenter Sites administrators can share individual assets among content management sites (as long as the sharing sites have the asset type enabled and have the same users in common).

    WebCenter Sites administrators can also restrict access to specific assets by not sharing them. Even if the asset type is enabled across sites, an asset created in one site is not available in another unless it has been shared to the other site.
- Once content management sites are developed, authorized content providers use the sites to create electronic assets, manage the assets, and deploy the assets to their audiences. The content providers are linked to content-entry forms as well as other authoring tools; they are also given certain editorial permissions; and they are given access to publishing and delivery systems for serving their content as part of the online site to browsers.

Content management sites represent real, online sites. However, they can represent those online sites in any number of ways, depending on what makes sense for your situation. For example:

- One content management site can represent one complete online (public) site.
- Several content management sites can represent separate sections of one large online site. For example, with a catalog, perhaps people who do the data entry for household goods never do data entry for yard goods so there are separate sites that represent those areas. And, in a publication example, perhaps sports writers have a separate site that represents the sports news section and the financial writers have a separate site that represents the financial news section.

- Several content management sites can represent the same online site but exist to restrict users' access to asset types, by role. For example, site one has the article and image asset types enabled and only content providers have access to this site; site two has all asset types including templates enabled and only a small group of developers have access to the site.

2.2 Online Sites

An online site is the set of pages that an organization displays to its target audience of customers, clients, and casual visitors. The online site can be an website or a portal. It can be accessible to the general public or it can be a password-protected site. It can also be a completely exclusive site, such as a corporate intranet or departmental network, operating strictly within the private domain.

Regardless of its nature, an online site originates from either a single CM site, or many CM sites, depending on which model you choose. Throughout our product guides, we use the term online site generically to refer to websites and portals, both of which are supported in this release.

2.3 Developers and the Content Management Site

Because you must log in to a content management site when you use your WebCenter Sites modules and products, all asset development is done in the context of a content management site. As you develop asset types, design your online site pages, and code Template assets, consider the following:

- When you create a Template asset, WebCenter Sites creates entries for it in both the `SiteCatalog` table and the `ElementCatalog` table. The name that it assigns to the page entry in the `SiteCatalog` table includes the name of the site that you were logged in to when you created the Template asset.
- If you share a Template asset with more than one content management site, WebCenter Sites creates a page entry in the `SiteCatalog` for each site that it is shared with. The names of the additional page entries for a shared template include the name of the site that the template was shared with.

Therefore, you must use the same content management site names on your development system that you will use on the management and delivery systems in order for your online site to function properly.

Because content management sites cover both design issues and access issues, you must work with your system administrators when determining how to use content management sites and how many sites you need for your system.

After you determine how many content management sites you need for both design and access control reasons on your management system, you or your system administrators can create the appropriate content management sites and enable the appropriate asset types for those sites on all of your systems. Then, on your development and management systems, you or your system administrators configure which content providers and other users (such as you) have access to which sites.

To configure content management sites, you use the **Site** option on the **Admin** tab.

2.4 Sites and the Site Plan

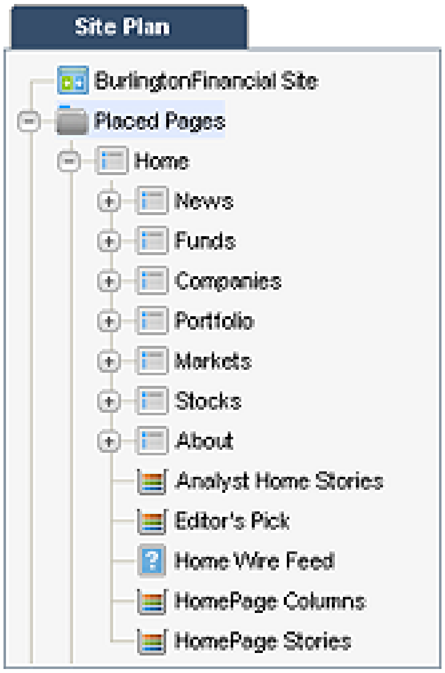
Page assets are site design assets that store references to other assets, organizing your assets according to the design that you and other developers are implementing. During the design phase of your online site, you create page assets, associate other assets with them, and then position the page assets in the tree on the **Site Plan** tab, located in the tree on the left side of the WebCenter Sites window.

When page assets are positioned in the tree on the **Site Plan** tab, information about each page asset's position in that tree is written to the `SitePlanTree` table. If the page assets that are positioned on this tab represent the same hierarchy that your templates and elements are coded to create on your published pages, you can use the WebCenter Sites `SITEPLAN` tag family to build navigational features. See Section 29.6, "Example 6: Displaying Site Plan Information" for more information.

The **Site Plan** tab displays a graphical representation of the layout of your online site, the content management site that you are currently logged in to, as a tree. It starts by querying the `SitePlanTree` table to determine which page assets have been placed. It then displays the page assets at the appropriate level on the tree, with the assets that have been associated with those page assets at subsequent hierarchical levels.

2.4.1 Example: the Burlington Financial Sample Site

The tree on the **Site Plan** tab shows assets, not rendered site pages. In other words, it does not represent all the possible online pages that could be delivered by the actual online site. For example, this is the section of the **Site Plan** tree that shows the Home page asset of the Burlington Financial sample site:



To better understand the connection between your online site and the **Site Plan** tab, display the rendered Burlington Financial News page asset in your browser:

1. Log in to the Burlington Financial sample site.
2. Select the **Site Plan** tab.
3. Select the **News** page asset from the tree, click the right mouse button, and select **Preview**.

Compare the News web page that is rendered in your browser to this section of the **Site Plan** tab and note the following:

- The News page asset represents an actual page that would be rendered if a visitor selected the News link from the online Burlington Financial home page. This is because the template assigned to the News page is coded to display the page asset as a web page.
- The collection assets displayed in the tree under the News page asset do not represent actual rendered pages. The template for the News page is coded to display the headlines of the articles contained in the collections that are associated with the News page as links in the online News page.
- The article assets contained in the NewsTop collection represent actual online pages. This is because the template that displays the article when you click the link to them is coded to display the article in a separate web page.

Because it is the code in the template that determines how an asset is displayed in your online site, there can be many online pages that are not represented as page assets in the **Site Plan** tree.

In order to select the correct assets for your page assets, you must know what the template elements for your assets are coded to do. This is why creating and placing page assets is your responsibility and it is a task that you complete as you code your template elements.

Page assets serve as gateway or index pages that offer access to other assets that represent content in addition to representing actual online pages.

2.5 Sites and the Database

When you create a site, WebCenter Sites writes information about it to the following database tables:

- The `Publication` table, which holds the names, descriptions, and pubids (IDs) of all the sites (publications) created for your system.
- The `PublicationTree` table, which stores information about which asset types have been enabled for which sites.
- The `SitePlanTree` table, which stores information about the hierarchical structure of a site and its page assets. There is a top-level node for each site created for your system. This table lists sites and page assets.

  As a developer, you can code your elements to extract and display information from the `SitePlanTree` table (for example, to create links to the major sections of your online site).

Note: Early versions of WebCenter Sites used the term publication rather than the term site and several of the database tables in the WebCenter Sites database still refer to sites as publications.

3 Oracle WebCenter Sites Development Process

When you are developing an online site that is to be delivered from a WebCenter Sites content management system, you are actually designing two sites:

- The online site that is delivered from your delivery system to visitors' browsers
- The content management site(s) that your content providers will use to input the data that they will publish to the delivery system.

In other words, you are responsible for the user experience of two sets of end users:

- The site visitors who use your delivery system
- The content providers who use the management system

When creating these two closely connected yet separate sites, the development team performs a series of planning, development, and testing steps. This chapter describes the development process in one possible sequence of events and in very general terms. Your own work flow will vary based on your work environment and business needs.

This chapter contains the following sections:

3.1 Step 1: Set Up the Team

The first step is to assemble the development team, which include the following kinds of people:

- Site designers
- XML and JSP developers
- Java application developers
- Database administrators

- System network administrators
- Marketers and advertising staff
- Product managers (if you are developing a commerce site)
- Content providers

You need people such as DBAs, system administrators, and content providers on your development team in addition to the people (like you) who do the actual coding for several reasons:

- Using a WebCenter Sites system requires you to design a data model in addition to creating a page design, which means that you need early input from the DBAs who will be supporting the databases on each system.
- Using a WebCenter Sites system means moving code and data around on multiple separate systems, several of which are probably clustered, which means you need early input from system and network administrators.
- Implementing a WebCenter Sites system will not be optimum unless the work habits of your content providers are accurately reflected in the design of the management system, which means you need early input from those who will use the management system.

3.2 Step 2: Create Functional and Design Specifications

An online site delivered from a Oracle WebCenter Sites content management system is a holistic construct in which everything interacts, intersects, and works with everything else. Therefore, the second step is to create a functional specification and a design specification (to design your online site on paper).

You should complete some version of this step before you begin coding anything (although you might do some proof-of-concept coding while working on the design specification).

3.2.1 Functional Requirements

Before you can begin a design specification, product management and marketing must provide the functional requirements for the online site.

3.2.2 Page Design

After you obtain the functional requirements from your marketing folks, a good place to start is to map out all the types of pages that you want to present on the online site. For example, home page, section page, columnist page, search page, article page, and so on. If you are designing a commerce site, you need other kinds of pages: registration page, product category pages, product description page, article page, FAQ page, invoice page, and so on.

Determine the graphical, navigational, and functional features for each page and the site overall: navigation bars, buy buttons and shopping carts, tell me more buttons, search functions, logo placement, animated graphics, and so on.

If you are using Engage, decide where the merchandising messages (recommendations) are to be placed on the pages and on which pages they'll be placed. For example, perhaps each product category page has a New Products section in the upper-right corner of the page.

Map out the entire structure of the site and create mock-ups of it.

3.2.3 Caching Strategy

One of the major elements in your design is caching: page caching and resultset caching. No online site can reach performance goals without your planning, testing, and implementing a caching strategy.

While designing the pages that you want to present on your online site, you must consider how and when page caching can and should be implemented for each piece on each page.

While designing your queries, you must map out all the tables in the database and determine how the resultset caching settings should be set for each table.

3.2.4 Security Strategy (Access Control)

Will you require your visitors to identify themselves before they are allowed to access any part of your online site? You must determine what kinds of access control you want to enforce early in the design process so that you design your pages correctly.

For example, if you plan to check your visitors' identities before allowing them access to a page, this affects how you would cache the components of that page. For example, you could design a container page, which is never cached, that verifies the identity of the visitor and then assembles the page from cached pagelets only if the verification is successful.

3.2.5 Separate Format from Content (Elements from Assets)

Following the basic proposition of separating content from format, take a look at each piece of each proposed page in your site and determine whether that piece should be represented as data or as logic.

A good design is one in which data is designed to be represented as an asset and is not embedded into element code. Examine every component of design or content, and then determine what your assets are. You make that determination by deciding which category a component belongs to: data or logic/code.

Simply speaking, do not code something into an element (embed it in logic) if it is really data. If it is data, is should be in a separate asset.

Here's another way to look at it:

- Assets that represent content are the responsibility of content providers.
- Logic, anything coded into any element, is the responsibility of the developers.

3.2.5.1 Determine the Asset Types (Content)

Documents, articles, products, and images are easily identified as assets. However, design components such as headers and footers could also be assets:

- When the content in a header or footer is embedded in the code of an element, you or another developer has to change the text in it when anything in it changes (a phone number, a logo, and so on).
- When the content in a header or footer is in an asset, the code in your elements must be able to obtain the identity of the asset; its content becomes the responsibility of a content provider.

Other page components that can be assets include the following kinds of things:

- Online polls
- Animation and other media

- Quote of the day
- Company or stock profiles
- Knowledgebase questions and answers

From your point of view, if the content for a component is represented in an asset, someone else is responsible for that content. You are only responsible for when and where it appears on your online sites and what it looks like when it appears there.

3.2.5.2 Decide How to Handle Images and Other Blobs

You have two general options when deciding how to manage the images and other blobs that you want to use in your online site:

- Treat them as assets: Store them in the WebCenter Sites database and have the BlobServer servlet serve them.
- Treat them as static files: Put them in a file structure on your web server and let the web server serve them.

Either method is a valid option. If you keep your image files on the web server, you can create links to them with the WebCenter Sites tags, and there may be performance benefits when you allow your web server to deliver your images. However, if you keep your images and blobs separate from the WebCenter Sites database:

- You must implement a separate file management process. The publishing methods that move image assets from your management system to your delivery system cannot move content that is not in the WebCenter Sites database. You must manage this process on your own.
- None of the native WebCenter Sites security mechanisms will apply. That is, you cannot use ACLs to limit access to blobs that are not managed by WebCenter Sites.

3.2.5.3 Map Out the Functional Design and Format (Elements)

You also need to analyze all of the functionality that you plan to incorporate into your online site. If you are designing a commerce site, parts of it will no doubt behave more like an application.

Outline what code or logic is required for your visitor registration pages, visitor data collection pages, shopping carts, personalization, and so on.

Remember that your WebCenter Sites system provides you with coding options: Java, XML, and JSP. As you look at each of the functions you want to provide, determine which is the best coding solution for that function.

3.2.6 Data Design

Once you know which pieces of your site should be represented as assets, you can map out what your asset types should be. Each new asset type will use one or more database tables (depending on whether it is a basic or flex asset type).

3.2.6.1 Asset Types

No matter which asset model you are using, basic or flex, consider the following when you design your asset types:

- Asset type design affects both of the user groups that you are designing for (visitors to the online site and the content providers who must enter the data).

- Which types of assets need to be linked or related to other assets of other types in order to successfully implement your page design? Be sure to implement these relationships in the asset type.
- Content providers appreciate efficiency. Be sure that your asset types store only the data that you really plan to use so that content providers do not waste time maintaining data that no one uses.

3.2.6.2 Auxiliary Tables That Support Your Asset Types

The data design that you want to implement for your system extends beyond the database tables that hold your assets. Depending on the kinds of information that you want to provide, you might need to create auxiliary tables that support your asset types.

For example, the Burlington Financial sample site has asset types with a **Mimetype** field. The **Mimetype** field is a drop-down field and a user must select a value from the drop-down list. These values are pulled from a lookup table named `MimeType`. Depending on your needs, you might need to create similar tables for your system.

Your DBAs should be involved in your discussions about the asset types and auxiliary tables that you plan to create so they can understand from the start the kind of database tuning issues that might arise on the management and delivery systems.

3.2.6.3 Visitor Data

If you are using Engage, you also need to determine what kinds of visitor data you plan to gather. These data types are represented by the Engage visitor data assets that you use to create segments for personalizing your site based on the identity of the visitor. (For example, demographics, purchase history, or clickstream information.)

After your WebCenter Sites system goes live and you start collecting visitor data, the tables that store that data grow very quickly. This is another area that you need to consult your DBAs about.

3.3 Step 3: Set Management System Requirements

Before you can begin coding, you must know how the management system will be organized. These decisions affect your design because your design depends on the content management site.

A content management site is an object that you use as an organizational construct for an actual online site and as an access control tool. When you create Template assets, WebCenter Sites creates an entry in the `SiteCatalog` table for it. The naming convention that WebCenter Sites uses for the page entries for templates includes the name of the content management site that you are creating the template for. This means that you must be consistent with site names throughout your entire content management system (development system, management system, and delivery system) and you must know the names of the sites that you are using before you begin coding.

Although your primary concern is the name of each site, the system administrators and business managers must also determine the following:

- How many users and ACLs (access control lists) do you need? (Remember that you may need to create ACLs to assign to the visitors of the online site, as well.)
- How many site roles you do you need?
- Which asset types need a workflow process?
- Which asset types should use revision tracking?

- Who should have access to which asset types on which sites?

Use both this book and the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide* to help you make these decisions.

3.4 Step 4: Implement the Data Design

After you have created your design specification and you understand the organization of the management system, you can implement the data design.

On the development system, you complete the following kinds of tasks:

- Create content management sites with the same names as those that will be used on the management system.
- Design and create your asset types.
- Add any lookup tables or other auxiliary tables for the asset types.
- Create sample assets of each type.

This step and the next step (Section 3.5, "Step 5: Build the Online Site") are iterative and will most likely overlap a great deal. While you need to create asset types so that you can create assets before you create templates for them, it is likely that you will uncover areas that need refinement in your data design only after you have coded a template and tested the code.

Refer to Chapter 11, "Data Design: The Asset Models" when you implement the data design of your online site.

3.5 Step 5: Build the Online Site

After you have sample assets of even one type created on the development system, you can begin coding templates and building the online site. (Actually, you can begin coding elements that do not display assets any time after you have created your design specification.)

In this step, you complete the following kinds of tasks:

- Create the page, query, and collection assets that implement the functionality of your online site.
- If you are using Engage, create the visitor data assets, sample segments, recommendations, and sample promotions.
- Create Template assets (and code template elements) for all of your asset types.
- If content contributors are using the Web Mode feature of the Contributor interface, code the templates using the `insite` family of tags.
- Code the CSElements that implement underlying functionality (that do not display assets).
- If you are developing a commerce site, code pages that implement the shopping cart.
- If you are using Engage, code pages that collect visitor data.
- Test everything. Most likely you will perform both usability and market testing for your online site.

Refer to Chapter 4, "Programming with Oracle WebCenter Sites" as you build your online site.

3.6 Step 6: Set Up the Management System

After you have the online site working on your development system, you move it to the management system.

The developers complete the following kinds of tasks:

- Create the sites.
- Re-create the asset types.
- Mirror the asset type tables and auxiliary table from the development system to the management system.
- Mirror publish the site design assets and the data structure assets created on the development system to the management system.

The system administrators then complete the following kinds of tasks:

- Create users, ACLs, and roles. Assign users their roles for each content management site.
- Configure Sites Desktop users, if you are using that feature.
- Create workflow processes.
- Create StartMenu shortcuts.
- Enable revision tracking.

Refer to the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide* for information about setting up the management system.

3.6.1 Import Content as Assets

It is likely that you already have content in some non-asset format that you want to use. To import this content into the WebCenter Sites database as assets, use the XMLPost utility.

3.6.2 Import Catalog Data and Flex Asset Data

If you are using the flex asset model and you have a large amount of pre-existing data that you want to use, you can import it with the BulkLoader utility. For systematic updates, however, you use the XMLPost utility.

3.6.3 Instruct the Editorial Team About Site Design

Before the editorial team can successfully maintain the online site, they need to understand your design. For example, how frequently are collections supposed to be rebuilt?

If you are using the basic asset model, content providers need to know the following:

- Which categories and sources they should assign to their assets in order for their assets to be located by the appropriate queries and collections.
- Which templates they should assign to which assets.
- Which association fields must be filled out in order for the links on the site pages to function correctly.

It is a good idea to program as much of this information as possible into the Start Menu shortcuts that you and the system administrators create for each asset type.

If you are using the flex asset model, content providers need to know the following:

- The general hierarchy or taxonomy in place for the flex assets.
- Some information about what information a flex asset inherits.
- Which templates they should assign to which assets.

3.7 Step 7: Set Up the Delivery System

When you set up the delivery system, you complete several of the same steps that you complete for the management system. For example:

- Re-create the sites.
- Re-create the asset types (but without their Start Menu shortcuts).
- Mirror the asset type tables and auxiliary table from the development system to the management system.

And then you publish all of the assets on the management system to the delivery system.

Also, because this system is not a management system, you complete the following steps as well:

- Implement your security strategy.
- On the web server, map the URL of your site (`www.example.com`) to the WebCenter Sites URL of your home page.

For information about setting up the delivery system, see the section on publishing in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

3.8 Step 8: Publish to the Delivery System

When the content on the management system is ready, you publish it to the delivery system. After intensive testing (both performance and load) you open your site to the public.

4 Programming with Oracle WebCenter Sites

In addition to managing your content, Oracle WebCenter Sites handles many useful tasks for you, including storing web pages and pieces of web pages, called *pagelets*, in WebCenter Sites caches, and maintaining those caches so that visitors to your website never see an outdated page. In order for WebCenter Sites to do this, you must code with WebCenter Sites tags and Java methods.

A WebCenter Sites page is composed of various element assets, blocks of code that can retrieve the content of your pages from the WebCenter Sites database, or that perform other tasks, such as deleting outdated items from the database, and *Template assets*, which are generally used to format the content of your web pages. Elements and templates can be written in a number of scripting and markup languages, including HTML, XML, JSP, CSS, and JavaScript. Note, however, that WebCenter Sites only evaluates XML and JSP.

This chapter gives you a brief overview of programming with WebCenter Sites. It contains the following sections:

Note: The following feature is deprecated in WebCenter Sites 11*g* Release 1 (11.1.1.8.0): SOAP-based web services. This feature is replaced by REST services.

4.1 Choosing a Coding Language

Choose your coding or markup language based on what the element or template that you are creating does. For example, you typically use HTML and XML for page layout and JSP and Java for logic. Elements that display content that may change, such as a newspaper article, should usually be written in XML or JSP. This is because such elements use logic to retrieve their content from the WebCenter Sites database, and thus are managed using WebCenter Sites XML or JSP tags.

WebCenter Sites also has a Java API, which you will use in conjunction with WebCenter Sites JSP tags if you choose JSP as your coding language.

The following table lists the situations to which each language is best suited:

Table 4–1 Coding Language Usage

Code	When to Use
`XML`	The element contains mostly text, with few loops and conditionals.
`JSP`	■ The element requires conditional operators, or relational operators other than = or !=. ■ The element uses many loops. Loops perform better in JSP than in XML. ■ The element contains calls to Java code.

Note that elements written in XML or JSP can call any type of element, but you cannot mix XML and JSP in the same element. For example, an element written in either XML or JSP can call another element written in HTML, XML, or JSP. However, an element written in HTML cannot call an element written in XML or JSP.

4.2 The Oracle WebCenter Sites Context

When you code for a Oracle WebCenter Sites project, you code within the Oracle WebCenter Sites context. The WebCenter Sites context provides access to the Java servlets that compose WebCenter Sites, and to the WebCenter Sites Java objects whose methods and tags allow you access to WebCenter Sites functionality.

You code in the WebCenter Sites context no matter what language you code your project in; WebCenter Sites XML and JSP tags provide an easy-to use interface to WebCenter Sites's Java objects, so that even web designers with little or no Java experience can create WebCenter Sites web pages.

4.2.1 The ICS Object

When you are coding for Oracle WebCenter Sites, you often access the methods and tags of the Interface to WebCenter Sites (ICS) object. The ICS object encapsulates some of WebCenter Sites's core functionality, allowing you to access servlets that control the WebCenter Sites tree (the TreeManager servlet) and the input of data into the database (the CatalogManager servlet).

You also use ICS methods and tags to perform tasks such as creating and displaying variables and using `if/then` statements to perform tasks based on specified conditions. For a complete list of the ICS object's methods and tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*

4.2.2 The FTCS tag

Each WebCenter Sites element or template begins and ends with the `ftcs` tag. This tag creates the WebCenter Sites context, alerting WebCenter Sites that code contained within the opening and closing `ftcs` tags will contain WebCenter Sites tags and access ICS methods.

If you use the WebCenter Sites user interface or the Oracle WebCenter Sites Explorer tool to create elements and templates, the opening and closing `ftcs` tags are automatically added after the standard directives. You must code within the opening and closing ftcs tags; WebCenter Sites is unaware of any code which falls outside of these tags.

If you create element and template code using some other method, you must add the opening ftcs tag after your directives, and use the closing ftcs tag as the last line of your code.

4.3 WebCenter Sites JSP

JSP programmers have a set of standard tools at their disposal, including directives, actions, and JSP objects. If you are programming in JSP within WebCenter Sites, you have access to many of these features. Sometimes, however, you must substitute a WebCenter Sites tag for a JSP directive or action, or access a WebCenter Sites object rather than one of JSP's implicit objects.

The following sections detail the differences between standard JSP and WebCenter Sites JSP, and how standard JSP functionality maps to WebCenter Sites tags and methods:

- Section 4.3.1, "WebCenter Sites Standard Beginning"
- Section 4.3.2, "JSP Implicit Objects"
- Section 4.3.3, "Syntax"
- Section 4.3.4, "Actions"
- Section 4.3.5, "Declarations"
- Section 4.3.6, "Scriptlets and Expressions"
- Section 4.3.7, "JSP Directives"
- Section 4.3.8, "Oracle WebCenter Sites Tag Libraries"

4.3.1 WebCenter Sites Standard Beginning

If you use either the WebCenter Sites user interface or Oracle WebCenter Sites Explorer to create your Template assets, CSElement assets, and non-asset elements, WebCenter Sites automatically seeds the element or template with a standard beginning.

The standard beginning for a JSP element in Oracle WebCenter Sites Explorer follows:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
<%//
// elementName
//
// INPUT
//
// OUTPUT
//%>
<%@ page import="COM.FutureTense.Interfaces.FTValList" %>
<%@ page import="COM.FutureTense.Interfaces.ICS" %>
<%@ page import="COM.FutureTense.Interfaces.IList" %>
<%@ page import="COM.FutureTense.Interfaces.Utilities" %>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<cs:ftcs>

<!-- user code here -->
```

```
</cs:ftcs>
```

If you use the WebCenter Sites user interface to create Template and CSElement assets, you will also see a standard beginning similar to the preceding code sample. The standard beginning for these assets imports additional tag libraries for use with basic assets and includes tags that log dependencies between the Template and CSElement assets and the content that they render.

If you use a tool other than Oracle WebCenter Sites Explorer or the WebCenter Sites user interface to create your elements and templates, you must copy the standard beginning into your code verbatim.

The following sections explain the standard beginning for Oracle WebCenter Sites Explorer.

4.3.1.1 Taglib Directives

The following taglib directives import the base tag libraries that you will use with WebCenter Sites. If you use the WebCenter Sites user interface to create template and CSElement assets, you will see additional taglib directives in your seed code.

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
```

The first directive imports the `ftcs1_0` tags, which create the FTCS context. These tags are used in each template or element that you create, and indicate that the code enclosed by them will be controlled by WebCenter Sites.

The second directive imports the `ics` tags, which provide access to WebCenter Sites's core functionality.

The third directive imports the satellite tags, which are for use with Satellite Server.

For more information about these tag libraries, see Section 4.3.8, "Oracle WebCenter Sites Tag Libraries."

For information about commonly used tags that are found in these tag libraries, see Section 4.5, "WebCenter Sites Tags."

To add taglib directives to these defaults, modify and save the `OpenMarket/Xcelerate/AssetType/Template/ModelJsp.xml` file.

4.3.1.2 Page Directives

The following page directives import the base Java interfaces that you will use with WebCenter Sites:

```
<%@ page import="COM.FutureTense.Interfaces.FTValList" %>
<%@ page import="COM.FutureTense.Interfaces.ICS" %>
<%@ page import="COM.FutureTense.Interfaces.IList" %>
<%@ page import="COM.FutureTense.Interfaces.Utilities" %>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
```

The first page directive imports the FTValList interface, which creates a list of name/value pairs that you use to pass arguments to WebCenter Sites subsystems like the CatalogManager and TreeManager.

The second page directive imports the `ICS` interface, which provides access to the core WebCenter Sites functionality.

The third page directive imports the IList interface, which contains the methods to access the rows in a WebCenter Sites query or list object. It also contains the methods that a third party must implement when attempting to construct and register a list object for use within an WebCenter Sites XML page.

The fourth page directive imports the `Utilities` interface, which provides a simple interface for some common tasks such as formatting dates, reading and writing files, and sending email.

The fifth page directive imports the ftErrors class, which contains error codes.

The sixth page directive imports the `ftMessage` class, which contains error messages used by WebCenter Sites.

To add page directives to the standard directives for JSP elements, modify and save the `OpenMarket/Xcelerate/AssetType/Template/ModelJsp.xml` file.

4.3.1.3 The cs:ftcs Tag

Each WebCenter Sites JSP template or element must have the `cs:ftcs` tag as its first and last tags. This tag creates the WebCenter Sites context, alerting WebCenter Sites that code contained within the opening and closing `cs:ftcs` tags will contain WebCenter Sites tags.

You must code within the opening and closing `cs:ftcs` tags; WebCenter Sites is unaware of any code which falls outside of these tags.

4.3.2 JSP Implicit Objects

JSP provides several implicit objects that are available for developers to use. In the WebCenter Sites context, however, you are often dealing with WebCenter Sites's objects, and should use WebCenter Sites JSP tags and Java methods to access these objects, instead of using JSP's implicit objects.

The following table maps JSP's implicit objects and some of their commonly used methods to the WebCenter Sites tag or method that you should use to replace them.

Table 4–2 JSP Implicit Object Mapping

Object	Method	WebCenter Sites Tag or Method
`request`	`getParameter`	`ics:getvar tag`
`request`	`getParameterNames`	`ICS.GetVars() method`
`request`	`getCookie`	`ics:getCookie tag`
`response`	`addCookie`	`satellite:cookie tag`
`session`	`getAttribute`	`ics:getssvar tag`
`session`	`setAttribute`	`ics:setssvar tag`

`out`	`println`	`ics:getvar tag or`
		`render:stream tag`

4.3.3 Syntax

Oracle WebCenter Sites uses standard JSP syntax. When you are nesting tags, for example, using a JSP expression as the value of a JSP tag's parameter, remember to use single quotes to contain the expression, as in the following example:

```
name='<%=ics.GetVar("myVariable")%>'
```

4.3.4 Actions

Standard JSP allows developers to use several different actions. The following table describes what actions should be replaced with WebCenter Sites tags and which can be used as usual:

Table 4–3 JSP Actions vs. WebCenter Sites Tags

Action	WebCenter Sites
`<jsp:forward>`	Use the `render:satellitepage` or `render:callelement` tags instead.
`<jsp:getproperty>`	Use this for custom Java Beans. If you want to find the value of one of the WebCenter Sites properties, use the `<ics:getproperty>` tag.
`<jsp:include>`	Use the `render:satellitepage` or `render:callelement` tags instead.
`<jsp:setProperty>`	Use this to set properties in custom Java Beans. Use the WebCenter Sites Property Editor to set WebCenter Sites properties.
`<jsp:useBean>`	Use this for custom Java Beans.

4.3.5 Declarations

In standard JSP, you usually declare variables within a JSP declaration. In WebCenter Sites, you use the `ics:setvar` tag to declare variables that are available in the WebCenter Sites context.

For more information about WebCenter Sites variables, see Section 4.6, "Variables."

4.3.6 Scriptlets and Expressions

You can use scriptlets and expressions without any variation from normal JSP usage.

When you use an expression as the value of the parameter for a WebCenter Sites JSP tag, however, be sure that you nest quotation marks correctly, as described in Section 4.3.3, "Syntax."

4.3.7 JSP Directives

When you are coding JSP in a Oracle WebCenter Sites context, there are some caveats for using directives, which are outlined in the following table:

Table 4–4 JSP Directives vs. WebCenter Sites Tags

Directive	WebCenter Sites
`IncludeDirective`	Use the `render:satellitepage` or `render:callelement` tags to include other files in your JSP pages.

Table 4–4 (Cont.) JSP Directives vs. WebCenter Sites Tags

Directive	WebCenter Sites
`Page Directive`	If you use the WebCenter Sites user interface or the Oracle WebCenter Sites Explorer tool to create elements or templates, your element or template is automatically seeded with standard page directives. In addition to the standard directives, you must add one other page directive to set the `contentType` for each WebCenter Sites element or template that you create. Set your page's content type to text/html and the character set to UTF-8 by providing the following page directive as the first line of every WebCenter Sites JSP file: `<%@ page contentType="text/html; charset=UTF-8" %>`
`Taglib Directive`	WebCenter Sites automatically seeds your templates and elements with commonly used taglib directives. You can add additional WebCenter Sites taglib directives to an element or Template asset as needed; a list of the WebCenter Sites tag libraries follows this table.

4.3.8 Oracle WebCenter Sites Tag Libraries

WebCenter Sites has a series of JSP tag libraries that correspond to functions in WebCenter Sites's APIs.

The following table lists the WebCenter Sites tag libraries and describes their functions. Use this table as a reference when deciding which tag libraries to import into your JSPs.

4.3.8.1 Tag Libraries for Both Basic and Flex Assets

Table 4–5 Tag Libraries for Both Basic and Flex Assets

Tag Library	Description
`acl.tld`	Tags for creating and manipulating Access Control Lists.
`date.tld`	Tags that convert dates with year, month, day, and optional hour, minute, and am/pm fields into epoch format long integers representing milliseconds since Jan 1, 1970, 0:00 GMT. Date tags also convert long integers into dates.
`dir.tld`	Directory Services tags.
`ftcs1_0.tld`	Tags that create the FTCS context. These tags are used in each template or element that you create, and indicate that the code enclosed by them will be controlled by WebCenter Sites.
`ics.tld`	Tags which provide access to core WebCenter Sites functionality, including access to the CatalogManager and TreeManager commands, and basic coding constructs like if/then statements.
`insite.tld`	Tags for Web Mode.
`localestring.tld`	Tags for localizing text strings.
`name.tld`	Tags that access the name of the user who is currently logged in to WebCenter Sites and manipulate usernames in directory services.
`object.tld`	Tags for manipulating WebCenter Sites objects.
`property.tld`	Tags for retrieving values from WebCenter Sites property files.

Table 4–5 (Cont.) Tag Libraries for Both Basic and Flex Assets

Tag Library	Description
`render.tld`	Tags that render basic assets.
`tags for working with satellite server`	Many of these tags have RENDER equivalents (as defined in `render.tld`) that are preferred for building sites with WebCenter Sites.
`soap.tld`	WebCenter Sites SOAP tags.
`time.tld`	Tags that get and set the timing for determining the performance of elements.
`user.tld`	Tags to log users in and out of WebCenter Sites.
`webservices.tld`	Web services tags that allow you to consume certain types of public websites as part of a WebCenter Sites page.

4.3.8.2 Tag Libraries for Basic Assets

Table 4–6 Tag Libraries for Basic Assets

Tag Library	Description
`asset.tld`	Tags that retrieve and manipulate basic assets.
`siteplan.tld`	Tags that allow access to the site plan tree. You use these tags to create navigation for a site that uses basic assets.

4.3.8.3 Tag Libraries for Flex Assets

Table 4–7 Tag Libraries for Flex Assets

Tag Library	Description
`assetset.tld`	Tags for creating assetsets with flex assets.
`blobservice.tld`	Tags for retrieving and manipulating blobs that are attributes of flex assets.
`calculator.tld`	Tags that provide basic calculator and boolean functions.
`cart.tld`	Tags that allow you to add, delete, and otherwise manipulate items in a shopping cart object.
`cartset.tld`	Tags that allow you store, retrieve, delete, and list shopping cart objects for a registered buyer.
`commercecontext.tld`	Tags that access the objects in the visitor context.
`currency.tld`	Tags that convert floating point values and currency strings, and perform formatting and rounding operations on currency strings.
`decimal.tld`	Tags that format floating point values as decimal objects in different locales.
`hash.tld`	Tags that allow you to cast an IList as a hash table and search it by key.
`listobject.tld`	Tags that construct WebCenter Sites resultset lists, which are used throughout your elements as arguments for other tags.
`locale1.tld`	Tags that generate a locale object, which is used to describe the desired locale for various other tags in the system.
`misc.tld`	Miscellaneous tags, including a tag that returns the names of all the columns in an input list

Table 4–7 (Cont.) Tag Libraries for Flex Assets

Tag Library	Description
`searchstate.tld`	Tags for creating searchstates to constrain groups of flex assets (assetsets).
`session.tld`	A tag that flushes all stored objects for a given session.
`string.tld`	Tags that perform string manipulations.
`textformat.tld`	Tags that format text.
`vdm.tld`	Visitor Data Management tags, which enable you to record and retrieve information about website visitors from WebCenter Sites, or from other databases.

For complete descriptions of the WebCenter Sites tags used for template development, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

4.4 WebCenter Sites XML

This section explains the basics of WebCenter Sites XML. WebCenter Sites XML uses standard XML syntax and is defined by the `futuretense_cs.dtd`. As with WebCenter Sites JSP tags, WebCenter Sites XML tags provide access to WebCenter Sites servlets and objects.

The following sections describe things to be aware of when coding with WebCenter Sites XML.

4.4.1 WebCenter Sites Standard Beginning

If you use the WebCenter Sites user interface or the Oracle WebCenter Sites Explorer tool to create your templates and elements, WebCenter Sites automatically seeds the element with the following standard beginning:

```
<?xml version="1.0" ?>
<!DOCTYPE ftcs SYSTEM "futuretense_cs.dtd">
<ftcs version="1.2">
</ftcs>
```

If you use some other tool to create your elements and templates, you must copy this code into them verbatim.

The following sections explain this standard beginning.

4.4.1.1 XML Version and Encoding

The first line in any WebCenter Sites XML template or element must set the XML version, as follows:

```
<?xml version="1.0"?>
```

Note that in order for your element to run, `<?xml version="1.0"?>` must be the first line in the element, with no spaces before the text. The line must also have a hard return at the end, placing it on its own line.

If you need to set the encoding for this template or element, you can do this as follows:

```
<?xml version="1.0" encoding="utf-8"?>
```

4.4.1.2 The DTD File

WebCenter Sites XML is defined by the `futuretense_cs.dtd` file. You must import this file into each WebCenter Sites element or template that you code by entering the following line immediately after the XML version statement:

```
<!DOCTYPE ftcs SYSTEM "futuretense_cs.dtd">
```

4.4.1.3 The FTCS Tag

Each WebCenter Sites XML template or element must have the `ftcs` tag as its first and last tags. This tag creates the WebCenter Sites context, alerting WebCenter Sites that code contained within the opening and closing `FTCS` tags will contain WebCenter Sites tags.

You must code within the opening and closing `ftcs` tags; WebCenter Sites is unaware of any code which falls outside of these tags.

4.4.2 XML Entities and Reserved Characters

Because symbols such as < and > are reserved characters in XML, you must not place them in your content. For example, the following code confuses the XML parser because the less-than sign (<) appears inside some text:

```
<P>4 < 7</P>
```

You must use character entities in place of reserved characters. Character entities begin with `&#` and end with a semicolon. Between the `&#` and the semicolon, you specify the decimal Latin-1 (a superset of ASCII) value of the character. For example, the decimal Latin-1 value of the < character is 60, so the correct way to code the preceding line in XML is:

```
<P>4 &#60; 7</P>
```

See Section 4.8, "Values for Special Characters" for a list of these character entities.

4.4.3 XML Parsing Errors

The XML parser that processes WebCenter Sites tags ensures that the tags are syntactically correct. This simplifies tracking down hard-to-find problems related to tagging syntax errors. A misspelled tag name is not reported as an error. This is because the XML parser doesn't require all tag names to exist in the DTD.

When a page request is made to a WebCenter Sites system and an XML syntax error is detected, the results streamed back will contain useful information to help you locate the problem. The results include a general error description, followed by the line/column location of the error. For example, the following error reports a bad parameter name:

```
Illegal attribute name NAM Illegal attribute name NAM
Location: null(6,11)
Context:
```

And the next error reports an incorrect tag nesting:

```
Close tag IF does not match start tag THEN Close tag IF does not match start tag
THEN
Location: null(13,3)
Context:
```

The XML parser also detects run-time errors. These are errors where the XML tags are syntactically correct, however, some error in the structure is detected during processing. For example, the following error reports an invalid use of ARGUMENT:

```
Failed to run template:c:\FutureTense\elements\dan.xml Runtime error Argument
invalid [Argument 5]
Containing tag: FTCS
```

4.5 WebCenter Sites Tags

WebCenter Sites has an extensive set of tags in both JSP and XML that allow you to access the various functions of WebCenter Sites and its product family. You use these tags in conjunction with HTML, Java, JavaScript, and custom tags that you create, to code your website.

This section provides and overview of the tags that you are most likely to use in your Template assets and elements. For complete information on all of the WebCenter Sites tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*

The tags discussed here are arranged by usage, as follows:

- Section 4.5.1, "Tags That Create the WebCenter Sites Context"
- Section 4.5.2, "Tags That Handle Variables"
- Section 4.5.3, "Tags That Call Pages and Elements."
- Section 4.5.4, "Tags That Create URLs."
- Section 4.5.5, "Tags That Control Caching."
- Section 4.5.6, "Tags That Set Cookies."
- Section 4.5.7, "Programming Construct Tags."
- Section 4.5.8, "Tags That Manage Compositional and Approval Dependencies."
- Section 4.5.9, "Tags That Retrieve Information About Basic Assets."
- Section 4.5.10, "Tags That Create Assetsets (Flex Assets)."
- Section 4.5.11, "Tags That Create Searchstates (Flex Assets)."

4.5.1 Tags That Create the WebCenter Sites Context

The following table describes tags that create the WebCenter Sites context in which you code. You use these tags in every template or element that you write.

FTCS (XML)	ftcs1_0:ftcs (JSP)
`<FTCS>`	`<ftcs1_0:ftcs>`
`</FTCS>`	`</ftcs1_0:ftcs>`

The FTCS tag creates the WebCenter Sites context. The opening FTCS tag should be the first tag in your code, and the closing FTCS tag should be the last tag in your code. WebCenter Sites is unaware of anything that falls outside of the opening and closing FTCS tags. Consequently, content outside the tags will not be cached, and the tags will not operate correctly.

4.5.2 Tags That Handle Variables

The following tags handle variables in WebCenter Sites.

CSVAR (XML)	ics:getvar (JSP)

`<CSVAR NAME="variableName"/>`	`<ics:getvar`
	`  name="variableName"/>`

CSVAR displays the value of a variable, session variable, built-in, or counter.

SETVAR (XML)	ics:setvar (JSP)

`<SETVAR`
`  NAME="variableName"`
`  VALUE="variableValue"/>`
	`<ics:setvar`
	`  name="variableName"`
	`  value="variableValue"/>`

SETVAR sets the value of a regular, WebCenter Sites variable.The value of the variable exists for the duration of the page evaluation unless it is explicitly deleted using REMOVEVAR.

SETSSVAR (XML)	ics:setvar (JSP)

`<SETSSVAR`
`  NAME="variableName"`
`  VALUE="variableValue"/>`
	`<ics:setssvar`
	`  name="variableName"`
	`  value="variableValue"/>`

SETSSVAR sets a session variable.

REPLACEALL (XML)	ics:resolvevariables (JSP)

`<REPLACEALL`
`  NAME="variableName"`
`  VALUE="variableValue"/>`
	`<ics:resolvevariables`
	`  name="variableName"`
	`  [output="variable name"]`
	`  [delimited="true│false"]/>`

REPLACEALL and ics:resolvevariables resolve multiple WebCenter Sites variables. In other words, when you want to use WebCenter Sites variables in HTML tags, you use these tags to resolve the variables.

For more information about variables in WebCenter Sites, see Section 4.6, "Variables."

4.5.3 Tags That Call Pages and Elements

Use the following tags to call elements or templates.

Note: CACHECONTROL, used below, has been deprecated.

RENDER.SATELLITEPAGE (XML)	render:satellitepage (JSP)
<RENDER.SATELLITEPAGE	<render:satellitepage
PAGENAME="nameOfPageEntry"	pagename="nameOfPageEntry"
[CACHECONTROL="expiration_date_and_ time"]	[cachecontrol="expiration_date_and_ time"]>
[ARGS_var1="value1"]/>	<[render:argument name="variable1" value="value1"]/>
	</render:satellitepage>

RENDER.SATELLITEPAGE requests a WebCenter Sites pagelet and caches that pagelet in both WebCenter Sites and Satellite Server, if the pagelet is not already in cache. If you wish to call a page or pagelet without caching it individually, use the RENDER.CALLELEMENT tag. The RENDER.SATELLITEPAGE tag has a stacked scope, so the only variables available to the page are ones that you explicitly pass in.

RENDER.CALLELEMENT (XML)	render:callelement (JSP)
<RENDER.CALLELEMENT	<ics:callelement element="element name">
ELEMENTNAME="nameOfElement"	<ics:argument name="argument name" value="arg value"/>
[ARGS_var1="value"]/>	
	</ics:callelement>

RENDER.CALLELEMENT is similar to the RENDER.SATELLITEPAGE tag in that both tags call other WebCenter Sites code, either in an element or in a page. However, code called by RENDER.CALLELEMENT does not get cached as an individual page or pagelet on Satellite Server.

Use RENDER.CALLELEMENT to process the content of an element that you wrote for the WebCenter Sites Content Applications and you want the scope of that element to be stacked. The element must exist in the ElementCatalog.

4.5.4 Tags That Create URLs

RENDER.GETPAGEURL (XML)	render:getpageurl (JSP)
<RENDER.GETPAGEURL	<render:getpageurl
OUTSTR="myURL"	outstr="myURL"
PAGENAME="SiteCatalogPageEntry"	pagename="SiteCatalogPageEntry"
cid="IDofAsset"	cid="IDofAsset"
[p="IDofParentPage"]	[p="IDofParentPage"]
[c="AssetType"]	[c="AssetType"]
[ADDSESSION="true"]	[addsession="true"]
[DYNAMIC="true"]	[dynamic="true"]
[PACKEDARGS="stringFromPACKARGStag"]	[packedargs="stringFromPACKARGStag"]>
[ARGS_xxx="y"]/>	<[render:argument name="xxx" value="yyy"]/>
	</render:getpageurl>

This tag creates a URL for an asset, processing the arguments passed to it into a URL-encoded string and returning it as the variable specified by the OUTSTR parameter.

If `rendermode` is set to `export`, the tag creates a file name for a static HTML file (unless you specify that you want a dynamic URL). If `rendermode` is set to `live`, the tag creates a dynamic URL.

RENDER.SATELLITEBLOB (XML)	render:satelliteblob (JSP)
`<RENDER.SATELLITEBLOB`	`<render:satelliteblob`
`SERVICE="HTMLtagName"`	`service="HTMLtagName"`
`BLOBTABLE="blobTable"`	`blobtable="blobTable"`
`BLOBKEY="primaryKeyName"`	`blobkey="primaryKeyName"`
`BLOBWHERE="primaryKeyValue"`	`blobwhere="primaryKeyValue"`
`BLOBCOL="columnName"`	`blobcol="columnName"`
`BLOBHEADERNAMEN="headername"`	`blobheadernameN="headername"`
`BLOBHEADERVALUEN="mimetype"`	`blobheadervalueN="mimetype"`
`[ARGS_format1="5"]`	`[cachecontrol="expirationDateAndTime"]>`
`[CACHECONTROL="expirationDateAndTime"]/>`	`<[render:argument name="format1" value="5"]/>`
	`</render:satelliteblob>`

This tag creates an HTML tag with a BlobServer URL for assets that are blobs. For example, imagefile assets from the Burlington Financial sample site are blobs stored in the WebCenter Sites database which means they must be served by the BlobServer servlet. This tag creates an HTML tag that instructs a browser how to find and format the specified blob.

4.5.5 Tags That Control Caching

The following tag lets you control whether or not the output of the current template or element gets cached.

ics.disablecache (XML)	ics:disablecache (JSP)
`<ics.disablecache/>`	`<ics:disablecache/>`

Use `ics.disable` cache in conjunction with `if/then` statements that check for error conditions; if an error is present, the resulting rendered page will not be cached. For complete information and code samples for the `ics.disablecache` tag, see Section 28.8.2, "Ensuring that Incorrect Pages Are Not Cached."

4.5.6 Tags That Set Cookies

The following tag sets cookies in WebCenter Sites.

satellite.cookie (XML)	satellite:cookie (JSP)
`<satellite.cookie`	`<satellite:cookie>`
`name="cookie_name"`	`<satellite:parameter name='name' value='cookie_name'/>`
`value="cookie_value"`	`<satellite:parameter name='value' value='cookie_value'/>`
`timeout="timeout"`	`<satellite:parameter name='timeout' value='cookie_timeout'/>`
`secure="true|false"`	`<satellite:parameter name='secure' value='true|false'/>`
`url="URL"`	`<satellite:parameter name='url' value='url'>`
`[domain="domain"]/>`	`</satellite:cookie>`

satellite.cookie sets a cookie on the user's browser. This tag is the only way to set cookies in either XML or JSP.

4.5.7 Programming Construct Tags

The following tags allow you to use basic programming constructs.

IF/THEN/ELSE (XML)	ics:if/ics:then/ics:else (JSP)
`<IF COND="LOGICAL_EXPRESSION">`	`<ics:if condition="logical expression">`
`<THEN>`	`<ics:then>`
`tags and/or text`	`tags and/or text`
`</THEN>`	`</ics:then>`
`<ELSE>`	`<ics:else>`
`tags and/or text`	`tags and/or text`
`</ELSE>`	`</ics:else>`
`</IF>`	`</ics:if>`

IF, THEN, ELSE determine conditions. You typically use these tags to determine the value of a variable.

LOOP (XML)	ics:listloop (JSP)
`<LOOP [FROM="START"]`	`<ics:listloop`
`[COUNT="LOOP_TIMES"]`	`listname="some list"`
`[LIST="LIST_NAME"]`	`[maxrows="number of loops"]`
`[UNTIL="END"]>`	`[startrow="start row"]`
`...`	`[endrow="end row"]/>`
`</LOOP>`	

LOOP and ics:listloop iterate through items in a list. Remember that excess code within these tags affects the performance of the template. Whenever possible, keep statements that do not need to be repeated outside the LOOP tags.

4.5.8 Tags That Manage Compositional and Approval Dependencies

For complete information about compositional and approval dependencies, see Section 28.1, "About Dependencies."

RENDER.LOGDEP (XML)	render:logdep (JSP)
`<RENDER.LOGDEP ASSET="asset name"`	`<render:logdep asset="asset name"`
`CID="asset id"`	`cid="asset id"`
`C="asset type"/>`	`c="asset type"/>`

Use the RENDER.LOGDEP tag if your template uses tags that obtain an asset's data without loading the asset, such as ASSET.CHILDREN.

RENDER.UNKNOWNDEPS (XML)	render.unknowndeps (JSP)
`<RENDER.UNKNOWNDEPS/>`	`<render:unknowndeps/>`

Use the RENDER.UNKNOWNDEPS tag if a page has a query or some other indeterminate connection to its dependent assets. This tag causes the page or pagelet to be regenerated at every publish because the dependencies cannot be determined. This means that you should use this tag sparingly.

RENDER.FILTER (XML)	render:filter (JSP)
`<RENDER.FILTER LIST="list name"`	`<render:filter list="list name"`
`LISTVARNAME="output list name"`	`listvarname="output list name"`
`LISTDICOL="assetID column"`	`listidcol="assetID column"`
`[LISTTYPECOL="assettype column"]`	`[listtypecol="assettype column"]`
`[TYPE="asset type"]`	`[type="asset type"]`
`[ID="asset id"]`	`[id="asset id"]`
`[VARNAME="output variable"/>`	`[varname="output variable"/>`

Use the RENDER.FILTER tag to check for unapproved assets and prevent them from being included in the exported page. This tag filters either a single asset or list of assets by comparing each asset ID against the assetid column in the ApprovedAssets database table. During export rendering, it filters what can be published based on approval status. During live rendering, RENDER.FILTER does nothing. Use this tag whenever you have a database query for a list of assets in your template.

4.5.9 Tags That Retrieve Information About Basic Assets

ASSET.LOAD (XML)	asset:load (JSP)
`<ASSET.LOAD`	`<asset:load`
`NAME="assetName"`	`name="assetName"`
`TYPE="assetType"`	`type="assetType"`
`OBJECTID="object.id"`	`objectid="object.id"`
`[FIELD="fieldName"]`	`[field="fieldName"]`
`[VALUE="fieldValue"]`	`[value="fieldValue"]`
`[DEPTYPE="EXACT, EXISTS,`	`[deptype="exact,exists,or greater"]/>`
`or GREATER"]/>`	

This tag queries the database for a specific asset and then loads the asset's data into memory as an object. The object is then available to your elements until either the session is flushed or the name that is assigned to the object is overwritten.

The scope of the object names that you assign to loaded assets is **global.** Be sure to use unique object names so that your elements do not overwrite objects by mistake. A convenient naming convention is to include the element name in the asset name. For an example of creating unique asset object names by using this convention, see Section 29.1, "Example 1: Basic Modular Design."

`ASSET.LOAD` automatically logs a dependency between the template or element that uses the tag and the asset data that the tag retrieves.

ASSET.SCATTER (XML)	asset:scatter (JSP)
`<ASSET.SCATTER`	`<asset:scatter`
`NAME="assetName"`	`name="assetName"`
`PREFIX="variablePrefix"/>`	`prefix="variablePrefix"/>`

This tag retrieves values from all of the fields of an asset object that has already been retrieved (loaded) with the `ASSET.LOAD` tag and turns those values into WebCenter Sites variables. For example, if you want to display the headline, byline, description, and so on of an article online, you can use this tag to retrieve all of those values with one call.

ASSET.GET (XML)	asset:get (JSP)
`<ASSET.GET`	`<asset:get`
`NAME="assetName"`	`name="assetName"`
`FIELD="fieldName"`	`field="fieldName"`
`[OUTPUT="outputVariable"]/>`	`[output="outputVariable"]/>`

This tag retrieves the value from one specified field of an asset object that has already been retrieved (loaded) with the `ASSET.LOAD` tag and turns that value into a WebCenter Sites variable. For example, if you need only the headline of an article to use in a link to that article, you can use this tag to retrieve that one value.

ASSET.CHILDREN (XML)	asset:children (JSP)
`<ASSET.CHILDREN`	`<asset:children`
`NAME="assetName"`	`name="assetName"`
`LIST= "listName"`	`list="listName"`
`[CODE= "NameOfAssociation"]`	`[code="NameOfAssociation"]`
`[OBJECTTYPE= "typeOfObject"]`	`[objectype="typeOfObject"]`
`[OBJECTID="objectID"]`	`[objectid="objectID"]`
`[ORDER="nrank"]/>`	`[order="nrank"]/>`

This tag queries the AssetRelationTree table and then builds a list of assets that are children of the asset that you specified. You use this tag to retrieve assets in a collection, to retrieve the image assets associated with article assets, and so on.

Use the `RENDER.LOGDEP` tag in conjunction with `ASSET.CHILDREN` to log a dependency between the element or template in which it appears and the content that `ASSET.CHILDREN` retrieves.

4.5.9.1 Performance Notes About the Asset Tags

- `ASSET.LOAD` and `ASSET.CHILDREN` are database queries, so you should use them only when necessary, because queries to the database take time. For example, you might want to include error checking code after an `ASSET.LOAD` tag and before its subsequent `ASSET.CHILDREN` tag that determines whether an asset was returned by the `ASSET.LOAD`. If there is no asset, there is no reason to invoke the `ASSET.CHILDREN` tag.
- An `ASSET.SCATTER` call takes much longer than a single `ASSET.GET` call.

4.5.10 Tags That Create Assetsets (Flex Assets)

Assetset tags specify a set of one or more flex assets that you want to retrieve from the database.

You can retrieve the following information from an assetset:

- The values for one attribute for each of the flex assets in the assetset
- The values for multiple attributes for each of the flex assets in the assetset
- A list of the flex assets in the assetset
- A count of the flex assets in the assetset
- A list of unique attribute values for an attribute for all flex assets in the assetset
- A count of unique attribute values for an attribute for all flex assets in the assetset

The following tables describe the assetset tags that you will use most frequently.

<table>
<tr><th>ASSETSET.SETASSET (XML)</th><th>assetset:setasset (JSP)</th></tr>
<tr><td>&lt;ASSETSET.SETASSET
  NAME="assetsetname"
  TYPE="assettype"
  ID="assetid"
  [LOCALE="localeobject"]
          [DEPTYPE="exact|exists|none"]
/&gt;</td><td>&lt;assetset:setasset name="assetsetname"
type="assettype" id="assetid"
[locale="localeobject"]
[deptype="exact|exists|none"]/&gt;</td></tr>
</table>

ASSETSET.SETASSET builds an asset set from a single asset that you specify and defines a compositional dependency between the template or element that it appears in and the content that it retrieves.

<table>
<tr><th>ASSETSET.SETSEARCHEDASSETS (XML)</th><th>assetset:setsearchedassets (JSP)</th></tr>
<tr><td>&lt;ASSETSET.SETSEARCHEDASSETS</td><td>&lt;assetset:setsearchedassets</td></tr>
<tr><td> NAME="assetsetname"</td><td>name="assetsetname"</td></tr>
<tr><td>[ASSETTYPES="assettype"]
[CONSTRAINT="searchstateobject"]
[LOCALE="localeobject"]
[SITE="siteidentifier"]
[DEPTYPE="exact|exists|none"]/&gt;</td><td> [assettypes="assettype"]
  [constraint="searchstateobject"]
  [locale="localeobject"]
  [site="siteidentifier"]
  [deptype="exact|exists|none"]/&gt;</td></tr>
</table>

ASSETSET.SETSEARCHEDASSETS creates an assetset object which represents all assets of specific types narrowed by specified search criteria (represented by the searchstate object that you name in the constraint parameter).

This tag also defines a compositional dependency between the template or element in which it appears and the each asset in the set.

<table>
<tr><th>ASSETSET.GETMULTIPLEVALUES (XML)</th><th>assetset:getmultiplevalues (JSP)</th></tr>
<tr><td>&lt;ASSETSET.GETMULTIPLEVALUES</td><td>&lt;assetset:getmultiplevalues</td></tr>
<tr><td> NAME="assetsetname"</td><td> name="assetsetname"</td></tr>
<tr><td> LIST="listname"</td><td> list="listname"</td></tr>
<tr><td> [BYASSET="true|false"]</td><td> [byasset="true|false"]</td></tr>
<tr><td> PREFIX="prefix"/&gt;</td><td> prefix="prefix"/&gt;</td></tr>
</table>

ASSETSET.GETMULTIPLEVALUES scatters attribute values from several attributes (and potentially more than one asset) into several specified lists.

It is recommended that you use ASSETSET.GETMULTIPLEVALUES when the goal is to display a fixed-format table of assets, or to obtain many attributes of a single asset (such as for a product detail page).

ASSETSET.GETMULTIPLEVALUES has the following limitations:

- Only non-foreign attributes can be scattered.
- Text-type attributes cannot be scattered.

ASSETSET.GETATTRIBUTEVALUES (XML)	assetset:getattributevalues (JSP)
`<ASSETSET.GETATTRIBUTEVALUES`	`<assetset:getattributevalues`
`  NAME="assetsetname"`	`  name="assetsetname"`
`  ATTRIBUTE="attribname"`	`  attribute="attribname"`
`  [TYPENAME="assettypename"]`	`  [typename="assettypename"]`
`  LISTVARNAME="varname"`	`  listvarname="varname"`
`[ORDERING="ascending|descending"]/>`	`[ordering="ascending|descending"]/>`

ASSETSET.GETATTRIBUTEVALUES gets the list of values for a specified attribute of the assets represented by an assetset.

ASSETSET.GETASSETLIST (XML)	assetset:getassetlist (JSP)
`<ASSETSET.GETASSETLIST`	`<assetset:getassetlist`
`  NAME="assetsetname"`	`  name="assetsetname"`
`  [LIST="attriblist"]`	`  [list="attriblist"]`
`  [MAXCOUNT="rowcount"]`	`  [maxcount="rowcount"]`
`  [METHOD="random|highest"]`	`  [method="random|highest"]`
`  LISTVARNAME="varname/>`	`  listvarname="varname"/>`

ASSETSET.GETASSETLIST retrieves an ordered list of assets, given optional sort criteria. The resulting list has two columns, assetid and assettype, that are sorted by the criteria that you specify.

4.5.11 Tags That Create Searchstates (Flex Assets)

Searchstate tags assemble criteria that filter the assets that you retrieve using the assetset tags.

You build a searchstate by adding or removing constraints to narrow or broaden the list of flex assets that are described by the searchstate.

The following tables describe the searchstate tags that you will use most frequently.

SEARCHSTATE.CREATE (XML)	searchstate:create (JSP)
`<SEARCHSTATE.CREATE`	`<searchstate:create`
`  NAME="ssname"`	`  name="ssname"`
`  [OP="and|or"]/>`	`  [op="and|or"]/>`

SEARCHSTATE.CREATE builds an empty searchstate object. You must begin constructing a searchstate with this tag.

SEARCHSTATE.ADDSTANDARDCONSTRAINT (XML)	searchstate:addstandardconstraint (JSP)
`<SEARCHSTATE.ADDSTANDARDCONSTRAINT`	`<searchstate:addstandardconstraint`
`NAME="ssname"`	`name="ssname"`
`[BUCKET="bucketname"]`	`[bucket="bucketname"]`
`[TYPENAME="assettype"]`	`[typename="assettype"]`
`ATTRIBUTE="attribname"`	`attribute="attribname"`
`[LIST="listname"]`	`[list="listname"]`
`[IMMEDIATEONLY="true|false"]`	`[immediateonly="true|false"]`
`[CASEINSENSITIVE="true|false"]/>`	`[caseinsensitive="true|false"]/>`

SEARCHSTATE.ADDSTANDARDCONSTRAINT adds an attribute name/value constraint into a new or existing searchstate object.

You can constrain the attribute by a list of values that you specify in the `list` parameter.

SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT (XML)	searchstate:addsimplestandardconstraint (JSP)
`<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT`	`<searchstate:addsimplestandardconstraint`
`NAME="ssname"`	`name="ssname"`
`[BUCKET="bucketname"]`	`[bucket="bucketname"]`
`[TYPENAME="assettype"]`	`[typename="assettype"]`
`ATTRIBUTE="attribname"`	`attribute="attribname"`
`VALUE="value"`	`value="value"`
`[IMMEDIATEONLY="true|false"]/>`	`[immediateonly="value"]/>`

SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT adds an attribute name/single value constraint to an existing searchstate.

This tag is the simple version of SEARCHSTATE.ADDSTANDARDCONSTRAINT. The object referred to by NAME is updated to reflect the new constraint. If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

SEARCHSTATE.ADDRANGECONSTRAINT (XML)	searchstate:addrangeconstraint (JSP)
`<SEARCHSTATE.ADDRANGECONSTRAINT`	`<searchstate:addrangeconstraint`
`NAME="ssname"`	`name="ssname"`
`[BUCKET="bucketname"]`	`[bucket="bucketname"]`
`[TYPENAME="assettype"]`	`[typename="assettype"]`
`ATTRIBUTE="attribname"`	`attribute="attribname"`
`LOWER="lowrange"`	`lower="lowrange"`
`UPPER="uprange"`	`upper="uprange"`
`[CASEINSENSITIVE="true|false"]/>`	`[caseinsensitive="true|false"]`
	`/>`

SEARCHSTATE.ADDRANGECONSTRAINT adds a range constraint for a specific attribute name.

SEARCHSTATE.ADDRICHTEXTCONSTRAINT (XML)	searchstate:addrichtextconstraint (JSP)
<SEARCHSTATE.ADDRICHTEXTCONSTRAINT	<searchstate:addrangeconstraint
NAME="ssname"	name="ssname"
[BUCKET="bucketname"]	[bucket="bucketname"]
[TYPENAME="assettype"]	[typename="assettype"]
ATTRIBUTE="attribname"	attribute="attribname"
VALUE="criteria"	lower="lowrange"
[PARSER="parsername"]	upper="uprange"
CONFIDENCE="minlevel"	[caseinsensitive="true|false"]
[MAXCOUNT="number"] />	/>

SEARCHSTATE.ADDRICHTEXTCONSTRAINT adds an attribute name and rich-text expression to the list of rich-text constraints in the searchstate.

SEARCHSTATE.TOSTRING (XML)	searchstate:tostring (JSP)
<SEARCHSTATE.TOSTRING	<searchstate:tostring
NAME="objname"	name="objname"
VARNAME="varname"/>	varname="varname"/>

SEARCHSTATE.TOSTRING converts a searchstate object into its string representation that is suitable for various uses, such as saving in a session variable or packing into a URL.

SEARCHSTATE.FROMSTRING (XML)	searchstate:fromstring (JSP)
<SEARCHSTATE.FROMSTRING	<searchstate:fromstring
NAME="objname"	name="objname"
VALUE="stringval"/>	value="stringval"/>

SEARCHSTATE.FROMSTRING provides the ability for a searchstate object to be initialized from its string representation. You must create an empty searchstate using the SEARCHSTATE.CREATE tag before you can use this tag.

4.6 Variables

WebCenter Sites supports the following kinds of variables:

- Regular variables, which last for the duration of the current template or element, unless you explicitly remove them. Regular variables have a global scope.
- Session variables, which last for the duration of the current session.

WebCenter Sites provides several standard variables whose names are reserved. You can retrieve the values of these variables, but you cannot use their names for other variables that you create.

This section describes the following topics:

- Section 4.6.1, "Reserved Variables"
- Section 4.6.2, "Setting Regular Variables"
- Section 4.6.3, "Setting Session Variables"
- Section 4.6.4, "Working With Variables"
- Section 4.6.5, "Variables and Precedence"
- Section 4.6.6, "Best Practices with Variables"

4.6.1 Reserved Variables

The following table defines the standard WebCenter Sites variables. Unless otherwise noted, these are regular variables:

Table 4–8 Reserved Variables

Variable	Definition
`tablename`	A variable that is set to a tablename before the `execsql` tags can be run.
`pagename`	The name of the WebCenter Sites page being invoked.
`ftcmd`	A variable used in calls to CatalogManager.
`username`	A session variable that contains the name of the user who is currently logged in to the current session.
`password`	A session variable that contains the password of the user who is currently logged in to the current session.
`authusername`	A variable that you can set to the username of a user who you want to log in to WebCenter Sites. This can be sent to WebCenter Sites via a URL.
`authpassword`	A variable that you can set to the password of a user who you want to log in to WebCenter Sites. This can be sent to WebCenter Sites via a URL.
`currentACL`	A session variable that contains the ACLs that the current user belongs to.
`errno`	Error numbers reported by WebCenter Sites tags.
`context`	Reserved for future use in the `render:calltemplate` tag. For more information, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.
`site`	The full name of the site, as stored in the `name` column of the `Publication` table. The `site` variable is set as a `resarg` in all of the Template and Site Entry assets. The site owns the Template and SiteEntry assets that you create within the site.
`sitepfx`	The site prefix (and short name of the site), as stored in the `cs_prefix` column of the `Publication` table.
`ft_ss`	An internal variable that is automatically set by WebCenter Sites to support communication with Satellite Server. When `ft_ss` is set to `true`, WebCenter Sites infers that a request is from Satellite Server.
`c`	The asset type that a template formats. WebCenter Sites sets this variable by default when you save the Template asset.
`cid`	The ID of the asset being rendered or formatted by a template.

Table 4–8 (Cont.) Reserved Variables

Variable	Definition
ct	The value of a child template, if there is one. For a thorough explanation of child templates, see Section 29.3, "Example 3: Using the ct Variable."
p	The ID of an asset's parent page, if there is one.
rendermode	Specifies whether a page entry is to be delivered live, exported, or previewed. By default, rendermode is live. When you use Export to Disk, or the Preview function, WebCenter Sites automatically overrides the value of this variable with export or preview. This value is used internally and must not be modified.
seid	The ID of a SiteEntry asset.
tid	The ID of a Template asset.
eid	The ID of a CSElement asset, eid is available to the CSElement's root element.

4.6.2 Setting Regular Variables

Most of the variables that you will use while coding WebCenter Sites templates and elements are **regular variables**. Regular variables last for the duration of the current template or element, unless they are explicitly deleted using WebCenter Sites tags.

4.6.2.1 Setting Variables with SETVAR

Inside a WebCenter Sites element, you can call the SETVAR XML or JSP tags to create a variable and establish its initial value. For example, the following SETVAR XML tag creates a variable named dog and sets its value to fido:

```
<SETVAR NAME="dog" VALUE="fido"/>
```

If the variable already exists, SETVAR resets its value to the new value. For example, the following command resets the value of dog to mocha:

```
<SETVAR NAME="dog" VALUE="mocha"/>
```

4.6.2.2 Setting Variables via a URL

WebCenter Sites creates a page when a browser goes to a URL managed by a WebCenter Sites application. Each page is associated with a particular URL. Imagine, for example, a page associated with a URL having the following format:

```
http://host:port/servlet/ContentServer?pagename=Experiment/Hello
```

At the end of every URL, you can set one or more variables. For example, the following URL creates three variables in the Hello page:

```
http://host:port/servlet/ContentServer?pagename=Experiment/Hello&dog=fido&cat=fifi
```

The preceding URL creates the following variables available to Hello:

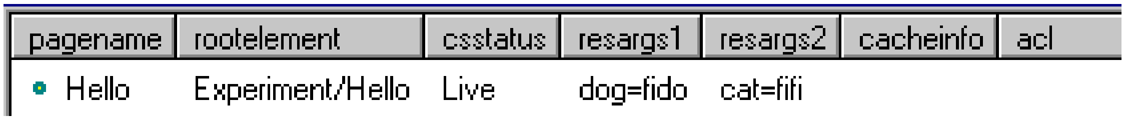
- A variable named pagename whose value is Experiment/Hello
- A variable named dog whose initial value is fido.
- A variable named cat whose initial value is fifi.

4.6.2.3 Setting Default Variables for Elements and Templates with Oracle WebCenter Sites Explorer

You can use Oracle WebCenter Sites Explorer to create default variables in a page by placing the variables in either of the following fields:

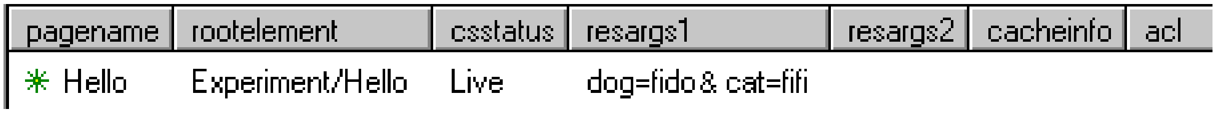
- `resargs1` or `resargs2` fields of the `SiteCatalog` database table
- `resdetails1` or `resdetails2` fields of the `ElementCatalog` database table

For example, you can use Oracle WebCenter Sites Explorer to access the `SiteCatalog` table, and then create variables `dog` and `cat` by placing name/value pairs in the `resargs1` and `resargs2` fields:



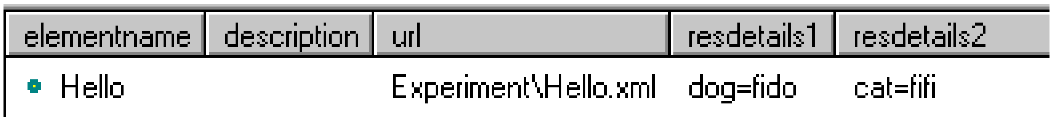
pagename	rootelement	csstatus	resargs1	resargs2	cacheinfo	acl
● Hello	Experiment/Hello	Live	dog=fido	cat=fifi		

Note that we placed one name/value pair in `resargs1` and another in `resargs2`. Alternatively, we could have put both name/value pairs in `resargs1`, as shown in the following diagram:



pagename	rootelement	csstatus	resargs1	resargs2	cacheinfo	acl
✳ Hello	Experiment/Hello	Live	dog=fido& cat=fifi			

You can also set the values of `dog` and `cat` in the `ElementCatalog` table by putting name/value pairs in the `resdetails1` and `resdetails2` fields:



elementname	description	url	resdetails1	resdetails2
● Hello		Experiment\Hello.xml	dog=fido	cat=fifi

Variables set through the URL or through `POST` and `GET` operations take precedence over variables set using the `SiteCatalog` or `ElementCatalog` tables. For example, if a URL sets variable `dog` to `rex` and the `SiteCatalog` sets `dog` to `fido`, then the resulting value of `dog` will be `rex`.

4.6.2.4 Setting Variables Using HTML Forms

In CGI programming, a buyer fills out a form. Then, the browser encodes the buyer's responses as name/value pairs, which get passed to the CGI script.

Although WebCenter Sites does not use traditional CGI programming, a WebCenter Sites element can still display a form. As in traditional programming, the browser encodes the buyer's responses as name/value pairs. However, instead of passing these name/value pairs to a CGI program, the pairs get passed to a different WebCenter Sites page. The receiving WebCenter Sites page can access the name/value pairs as it would access any WebCenter Sites variable.

Cookie names and values are also instantiated as variables. For more information about cookies, see Chapter 9, "Sessions and Cookies"

4.6.3 Setting Session Variables

HTTP is a stateless protocol. To overcome this limitation, WebCenter Sites can maintain state between requests and, thus, keep track of sessions.

A browser connection to the WebCenter Sites system establishes a session. Thereafter, the session is uniquely identified to the system. WebCenter Sites can deliver pages whose content and behavior are based on this unique identity.

When a client first enters your site, a unique session is established. WebCenter Sites associates a default user identity with a new session and maintains that information in session variables. **Session variables** contain values that are available for the duration of the session. They are saved as part of the user's session and are used to retain the value of a variable across page requests.

In a clustered configuration, the session state is maintained across all cluster members. Session variables should be used carefully, since there is a resource cost that is proportionate to the number and size of session variables used.

Session state is lost under these conditions:

- The client exits.
- The session has timed out. WebCenter Sites can optionally terminate a session if no requests have been made for some period of time.
- The application server has been restarted.

Server resources associated with the session are de-allocated when the following occurs:

- The session has been explicitly terminated by the client via a WebCenter Sites tag.
- The session has timed out.
- The application server has been restarted.

Use the `SETSSVAR` XML and JSP tags to create a session variable. If the session variable already exists, `SETSSVAR` resets the variable's value. For example, the following `SETSSVAR XML` tag sets the session variable `profile` to the value `10154`:

```
<SETSSVAR NAME="profile" VALUE="10154"/>
```

4.6.4 Working With Variables

The following sections describe how to work with WebCenter Sites variables.

4.6.4.1 Retrieving a Variable's Value

The syntax you use to read the value of a variable depends on the kind of variable:

Table 4–9 Variables and Syntax

Type of Variable	Syntax	Example
String Variable	`Variables.variable_name`	`Variables.dog`
Counter Variable	`Counters.variable_name`	`Counters.position`
Session Variables	`SessionVariables.variable_name`	`SessionVariables.username`
Property	`CS.Property.property_name`	`cs.use.short.jsp.names`

WebCenter Sites XML provides quite a few methods for accessing list variables.

4.6.4.2 Displaying a Variable's Value

Use the `CSVAR` XML tag to display the value of any kind of variable, including properties and session variables. Use the `ics:getvar` JSP tag to view the value of a regular WebCenter Sites variable; or the `ics:getssvar` JSP tag to display the value of a session variable. For example, if the following code appears in an XML element:

```
<SETVAR NAME="mood" VALUE="happy"/>
<p>My dog is <CSVAR NAME="Variables.mood"/>.</p>
```

then the resulting page displays the following text:

```
My dog is happy.
```

You can also include literal values as part of the `NAME` argument to the `CSVAR` XML tag; for example, the following code will also generate "`My dog is happy.`", but evaluates more slowly:

```
<SETVAR NAME="mood" VALUE="happy"/>
<p><CSVAR NAME="My dog is Variables.mood"/>.</p>
```

4.6.4.3 Assigning One Variable Value to Another Variable

You can assign the value of one variable to another variable. You accomplish this task differently if you are coding with XML than if you are coding with JSP.

JSP

If you are coding with JSP, you cannot use the `ics:getvar` tag to evaluate the variable value because you cannot nest one JSP tag within another JSP tag. To circumvent this limitation, use the `ics.GetVar` Java method to substitute variable values, as shown in the following sample code:

```
<ics:setvar name="myVar" value="Fred"/>
<ics:setvar name="yourVar" value='<%=ics.GetVar("myVar")%>'/>
<ics:getvar name="yourVar"/>
```

> **Note:** You must enclose the expression that evaluates the variable value ('`<%=icsGetVar("myVar")%>'` in the example) in single quotes. Otherwise your JSP element will throw an exception.

XML

The following lines of XML assign the value `carambola` to a variable named `your_favorite`:

```
<SETVAR NAME="my_favorite" VALUE="carambola"/>
<SETVAR NAME="your_favorite" VALUE="Variables.my_favorite"/>
```

Taking this one step further, you can concatenate two variable values and assign the result to a third variable. For example, the following sets the variable `car` to the value `red rabbit`.

```
<SETVAR NAME="color" VALUE="red"/>
<SETVAR NAME="model" VALUE="rabbit"/>
<SETVAR NAME="car" VALUE="Variables.color Variables.model"/>
```

4.6.4.4 Using Variables in HTML Tags

You can use XML and JSP variables within traditional HTML tags, although you code differently to accomplish this in XML and JSP.

JSP

If you are coding with JSP, you use the `ics:getvar` tag or the `ics.GetVar` Java method to evaluate the variable value.

You can also use the `ics.resolvevariables` tag to resolve variables that are contained within a string. For example, the following code displays the phrase, "The date is," along with the value of the `CS.Date` variable:

```
<ics.resolvevariables name="The date is $(CS.Date)." delimited="true"/>
```

The `delimited` parameter indicates that you have used the delimiters `$(` and `)` to explicitly mark the variable or variables that you want to resolve. If you want to use variables to specify a list name and a column in that list, for example, you use the following syntax:

```
<ics.resolvevariables name="$(Variables.listname).$(Variables.columnname)"
delimited="true"/>
```

If the `delimited` parameter is set to `false`, no delimiters are used to set off variables.

XML

You can use XML variables inside HTML tags if you use the appropriate attributes. For example, the following code does not contain the appropriate attributes and, therefore, does not set the background color to red:

```
<SETVAR NAME="color" VALUE="red"/>
<TABLE bgcolor="Variables.color">
...
```

To use XML variable values within an HTML tag, you must use the `REPLACEALL` attribute within that HTML tag. The `REPLACEALL` attribute tells the system to substitute the current value of this XML variable within this HTML tag. Therefore, the correct way to code the preceding lines is as follows:

```
<SETVAR NAME="color" VALUE="red"/>
<TABLE bgcolor="Variables.color" REPLACEALL="Variables.color">
...
```

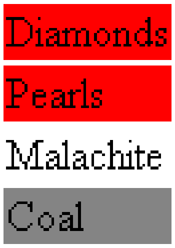
You can combine multiple variable values within one `REPLACEALL` attribute. For example, the following HTML `TABLE` tag uses two XML variables:

```
<SETVAR NAME="color" VALUE="red"/>
<SETVAR NAME="myborder" VALUE="3"/>
<TABLE bgcolor="Variables.color" border="Variables.myborder"
     REPLACEALL="Variables.color,Variables.myborder">
...
```

The `<REPLACEALL>` tag is an alternative to the `REPLACEALL` attribute. The `<REPLACEALL>` tag performs substitutions within its domain; for example:

```
<SETVAR NAME="highlight" VALUE="red"/>
<SETVAR NAME="diminish"  VALUE="gray"/>
<REPLACEALL LIST="Variables.highlight,Variables.diminish">
<TABLE>
  <TR BGCOLOR="Variables.highlight"><TD>Diamonds</TD></TR>
  <TR BGCOLOR="Variables.highlight"><TD>Pearls</TD></TR>
  <TR><TD>Malachite</TD></TR>
  <TR BGCOLOR="Variables.diminish"><TD>Coal</TD></TR>
</TABLE>
</REPLACEALL>
```

The output of this section is:



Diamonds
Pearls
Malachite
Coal

The REPLACEALL tag performs a string search and replace, and is, therefore, potentially very slow. Use the REPLACEALL attribute where possible. If you must use the REPLACEALL tag, keep the amount of code you enclose with it as small as possible.

4.6.4.5 Evaluating Variables with IF/THEN/ELSE

WebCenter Sites XML and JSP provides the IF/THEN/ELSE construct available in most computer languages. However, the only conditional operation for variables is to compare two values for equality or inequality. You can't, for example, compare two values to see if one is greater than another. (You can write Java code to do that, however.)

For example, the following code branches depending on the value of a variable named greeting.

```
<IF COND="Variables.greeting=Hello">
<THEN>
     <p>Welcome.</p>
</THEN>
<ELSE>
     <p>So long.</p>
</ELSE>
</IF>
```

If greeting is set to Hello, then WebCenter Sites generates the HTML:

```
<p>Welcome.</p>
```

If greeting is set to anything other than Hello, WebCenter Sites generates:

```
<p>So long.</p>
```

4.6.5 Variables and Precedence

Variables set through a URL or through HTTP GET and POST operations take precedence over variables set with the resargs and resdetails columns in the SiteCatalog and ElementCatalog tables.

4.6.6 Best Practices with Variables

Because all variables are global and the syntax for accessing variables from items in lists and from other sources is the same, good coding practices help you to avoid errors. For example:

- Because it is easy to reuse base names in your elements, use prefixes in front of variables to define them uniquely. The recommended syntax to use is: `Variables.assettype:fieldname`.

  For example, `Variables.Article:description`.

  The `ASSET.SCATTER` tag makes it easy for you to use this syntax through its `PREFIX` attribute. For more information about this tag, see Chapter 28, "Coding Elements for Templates and CSElements."

- If you are going to use the `RESOLVEVARIABLES` tags to resolve your variables, set the `DELIMITED` parameter to `true` and use the delimiters `$(` and `)` to explicitly indicate the variables you want to resolve.

- Use debugging to catch naming conflicts. Use the Property Editor to set the `com.fatwire.logging.cs` property (in the `commons-logging.properties` file or in `log4j.properties`, depending on which logging framework you are using). When this property is enabled, WebCenter Sites writes a record of all the variables that are created to the WebCenter Sites log file.

For a list of the error values that WebCenter Sites tags can write to the `errno` variable, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

4.7 Other WebCenter Sites Storage Constructs

In addition to regular and session variables, WebCenter Sites supports a number of storage constructs. The following sections describe these constructs and how to use them:

- Section 4.7.1, "Built-ins"
- Section 4.7.2, "Lists"
- Section 4.7.3, "Counters"

4.7.1 Built-ins

WebCenter Sites provides several built-ins, which return values such as the current date.

The general syntax of a built-in is:

```
CS.builtin
```

For example, `UniqueID` is a built-in that generates a unique ID. The following syntax generates or references this built-in variable:

```
CS.UniqueID
```

For a list of built-ins in WebCenter Sites, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

4.7.2 Lists

A list consists of a table of values organized in rows and columns. Use the `SETROW` or `GOTOROW` tags to identity the proper row.

The following entities create lists:

- The `SELECTTO`, `EXECSQL`, `CATALOGDEF`, `STRINGLIST` and `CALLSQL` tags
- CatalogManager commands

- TreeManager commands
- Custom tags

Use the following syntax to refer to a current row's column value:

```
listname.colname
```

For example, if a list named `cars` had a column named `color`, the value of the current row would be referenced as:

```
cars.color
```

4.7.2.1 Looping Through Lists

Use the `LOOP` XML tag or the `ics:listloop` JSP tag to iterate through a list. For each row in the list, WebCenter Sites executes the instructions between the loop tags.

For example, consider a table named `MyCars` containing the following rows:

Table 4–10 Example Table with Rows

id	Model	Color	Year
224	Ford Focus	blue	2001
358	VW Rabbit	red	1998
359	Toyota Corolla	yellow	2000
372	Alpha Romeo Spider	red	1982
401	Porsche 911	red	1984
423	Dodge Voyager	tan	1991

The following XML searches `MyCars` for red cars. The `SELECTTO` XML and JSP tags write this information into a list variable named `carlist`.

```
<SETVAR NAME="color" VALUE="red"/>
<SELECTTO FROM="MyCars" WHERE="color" WHAT="*" LIST="carlist"/>
Red cars: <BR/>
<OL>
<LOOP LIST="carlist">
    <LI><CSVAR NAME="carlist.model"/> </LI>
</LOOP>
</OL>
```

The preceding XML generates the following HTML:

```
Red cars: <BR/>
<OL>
 <LI> VW Rabbit </LI>
 <LI> Alpha Romeo Spider </LI>
 <LI> Porsche 911 </LI>
</OL>
```

4.7.3 Counters

A counter is an XML variable whose value is an integer. Three tags control counters:

Table 4–11 Counters

Tag	What It Does
SETCOUNTER	Initializes a counter variable
INCCOUNTER	Changes the counter's value by a specified amount
REMOVECOUNTER	Destroys the counter variable

To create a counter, you call `SETCOUNTER`. To change its value, call `INCCOUNTER`. For example, consider the following code:

```
<SETCOUNTER NAME="c" VALUE="10"/>
<INCCOUNTER NAME="c" VALUE="3"/>
<p>Current value is <CSVAR NAME="Counters.c"/></p>
```

The output of this code is:

```
Current value is 13
```

Notice that you reference counter variables using the syntax:

```
Counters.name
```

4.8 Values for Special Characters

If you need to use special (non-alphanumeric) characters in your XML or JSP, you will need to use their hexadecimal character representation. For example, the following line specifies a space as part of a variable value:

```
    <SETVAR NAME="foo" VALUE="foo%20bar"/>
```

The following are hexadecimal values for special characters that are commonly used in WebCenter Sites:

Table 4–12 Values for Special Characters

Hexadecimal Value	Character
%22	doublequote (")
%20	one space
%3c	less than sign (<)
%3e	greater than sign (>)
%26	ampersand (&)
%09	tab (\t)
%0a	newline (\n)
%0d	carriage return (\r)
%25	percent (%)

5 Page Design and Caching

This chapter describes how WebCenter Sites caching works. Caching your web pages can improve your site's performance. Whether your site is static or dynamic, you need to design your site so that part or all of a given page is cached.

This chapter contains the following sections:

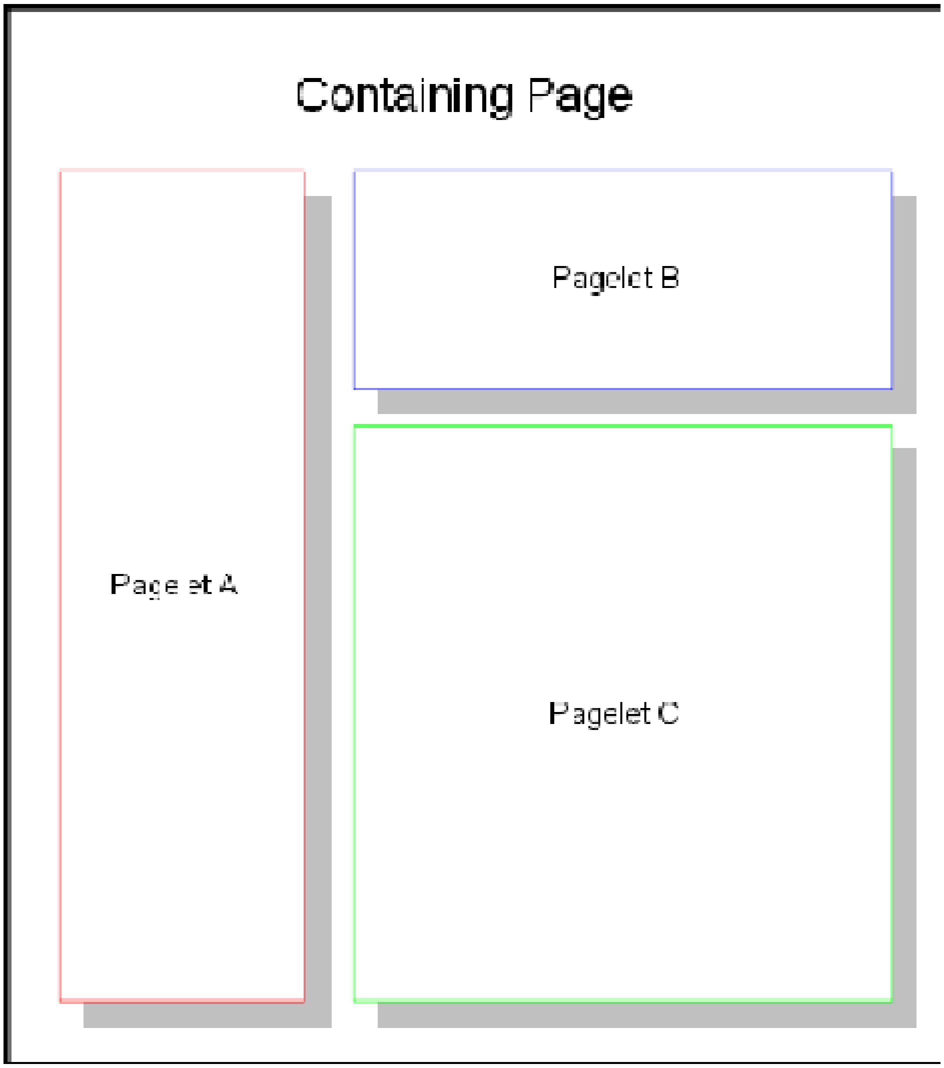
5.1 Modular Page Design

It is recommended that you design your web pages using a modular page design strategy, where a web page that a website visitor sees is composed of multiple elements. Modular page design has several benefits:

- It improves system performance by allowing you to develop an efficient caching strategy
- It lets you code common design elements, like navigation bars, one time and use them on multiple web pages

The following diagram shows a simple modular page:

Figure 5–1 A modular page



Each rectangle represents a pagelet (the generated output of one or more elements). These pagelets are called by a containing page. The containing page lays out how the pagelets appear on the finished page and contains any code that must be evaluated each time the page is viewed (custom ACL checking code, for example). This strategy lets you code an element once and use it in many places in your website.

5.2 Caching

WebCenter Sites lets you cache entire web pages and/or the components that make up those web pages. An efficient page caching strategy improves system performance by reducing load.

Two members of the WebCenter Sites product family implement page caching:

- WebCenter Sites, which caches pages on the WebCenter Sites system
- Satellite Server, which provides a second level of caching for your WebCenter Sites system, and can also be used as a remote cache for your web pages

WebCenter Sites utilizes both the WebCenter Sites and Satellite Server caches to create an efficient caching strategy.

5.2.1 WebCenter Sites Caching

Pagelets generated by requests to the ContentServer servlet can be cached on disk. If a page is accessed frequently and its content depends on a small number of parameters, then it is a good candidate for disk caching.

To disk-cache a pagelet, you use one of the following tags:

Table 5–1 Caching Tags

JSP Tag	XML Tag
`satellite:page`	`SATELLITE.PAGE`
`render:satellitepage`	`RENDER.SATELLITEPAGE`

If the pagelet that you want to cache is not already in the disk cache, ContentServer adds it to the cache and then serves the pagelet. If the specified pagelet is already in the disk cache, ContentServer simply serves it.

The expiration of disk-cached pagelets is time-based and governed by properties in the `futuretense.ini` file, in conjunction with the values set in the `cscacheinfo` and `sscacheinfo` columns of the `SiteCatalog` table (`cscacheinfo` corresponds to the WebCenter Sites servlet; `sscacheinfo` corresponds to the Satellite Server servlet). When a cached page is requested, its expiration time (cache timeout) is stored in the `etime` column of the `SystemPageCache` table. The expiration time for cached pages requested through the WebCenter Sites servlet is calculated from the `cscachinfo` value, and the expiration time for cached pages requested through the Satellite Server servlet is calculated from the `sscacheinfo` value. Items in cache are bound by the same security rules as uncached pages; WebCenter Sites ACLs apply to cached pagelets just as they do to elements.

5.2.2 BlobServer and Caching

The term **blob** is an acronym for **b**inary **l**arge **ob**ject. Although a blob is usually an image file, a blob can be any binary object, including a Microsoft Word file or a spreadsheet. Most websites serve a number of blobs.

To serve blobs, WebCenter Sites offers a special servlet called BlobServer. The BlobServer gathers a blob from a table and performs all relevant security checks.

You access BlobServer with the BlobServer tags:

- `satellite:blob`
- `render:satelliteblob`

Both of these tags cache blobs in the WebCenter Sites and Satellite Server caches. For more information about the BlobServer tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

Deleting Blobs from the WebCenter Sites Memory Cache

To delete a specific blob from the WebCenter Sites cache, you must do the following in the BlobServer URL:

- Rename the `blobtable` parameter to `flushblobtable`
- Authenticate as a user with `SiteGod` privileges by passing credentials through the `authusername` and `password` parameters

For example:

```
http://hostname:port/servlet/BlobServer?blobcol=urlpicture&blobheader=image%2Fgif&
blobkey=id&flushblobtable=NewPortalImage&blobwhere=22&authusername=username&authpa
ssword=password
```

To delete **all** blobs, rename the **blobtable** parameter to **flushblobtables** (notice the "s") and set its value to `true`.

5.2.3 Satellite Server Caching

Satellite Server, automatically installed with WebCenter Sites, provides an additional layer of caching. To improve your WebCenter Sites system's performance, you can add remote Satellite Server systems, putting your content closer to its intended audience.

Satellite Server caches pages, pagelets, and blobs to disk or to memory. You can use the `Inventory` servlet to view the contents of the memory and disk caches in varying degrees of detail. Note that items cached on Satellite Server are not protected by WebCenter Sites APIs. You can overcome this limitation by using the caching strategy outlined in Section 5.4.1.1, "Pagelet Caching Strategies."

Satellite Server caches small items to memory and large items to disk. You control the definitions of small and large through the `file_size` property. For more information on setting Satellite Server properties, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

On a busy site, each Satellite Server system's cache fills up quickly with the most popular pages. When the cache is full, Satellite Server deletes old pages to make room for new ones. Satellite Server uses a Least Recently Used algorithm (LRU) to determine which items should be removed from the cache. In other words, when a new page needs to be cached, Satellite Server removes the page that hasn't been accessed for the longest time. For example, given two cached pages, one that hasn't been accessed in 36 hours and the other that hasn't been accessed in 2 hours, Satellite Server removes the page that hasn't been accessed in 36 hours.

5.2.3.1 Cache Expiration

Page and pagelet expiration on Satellite Server is specified in the `sscacheinfo` column of the `SiteCatalog` table. Each time a page or pagelet is invoked through Satellite Server, Satellite Server processes the `sscacheinfo` field's value and determines when the page or pagelet should expire. Consult Section 7.4, "CacheInfo String Syntax" in Chapter 7, "Advanced Page Caching Techniques"for information about the `sscacheinfo` field.

> **Note:** Deprecation notice: It is possible to override the `sscachinfo` expiration information for pagelets by specifying the `cachecontrol` attribute in the `satellite.page` and `render.satellitepage` tags. However, this practice is deprecated because it can lead to non-deterministic behavior; some pagelets may be accessed through the default method (without the `cachecontrol` attribute) while others may be accessed with an override. The first method invoked will set the expiration for Satellite Server, and the second one will have no effect on the expiration.

Blobs cached on Satellite Server expire using the following algorithm:

- You can use Satellite Server tags to override the default expiration time on a blob-by-blob basis.

- If there is no Satellite tag to override the default expiration, Satellite Server gets the expiration time from the value of the `satellite.blob.cachecontrol.default` property. This property is described in Section 5.3.4.1, "WebCenter Sites Page Caching Properties."
- If no value is set for the `satellite.blob.cachecontrol.default` property, Satellite Server gets the expiration time from the value of the `expiration` property, described in Section 5.3.4.2, "Satellite Server Properties."

5.2.3.2 Caching with the Satellite Servlet

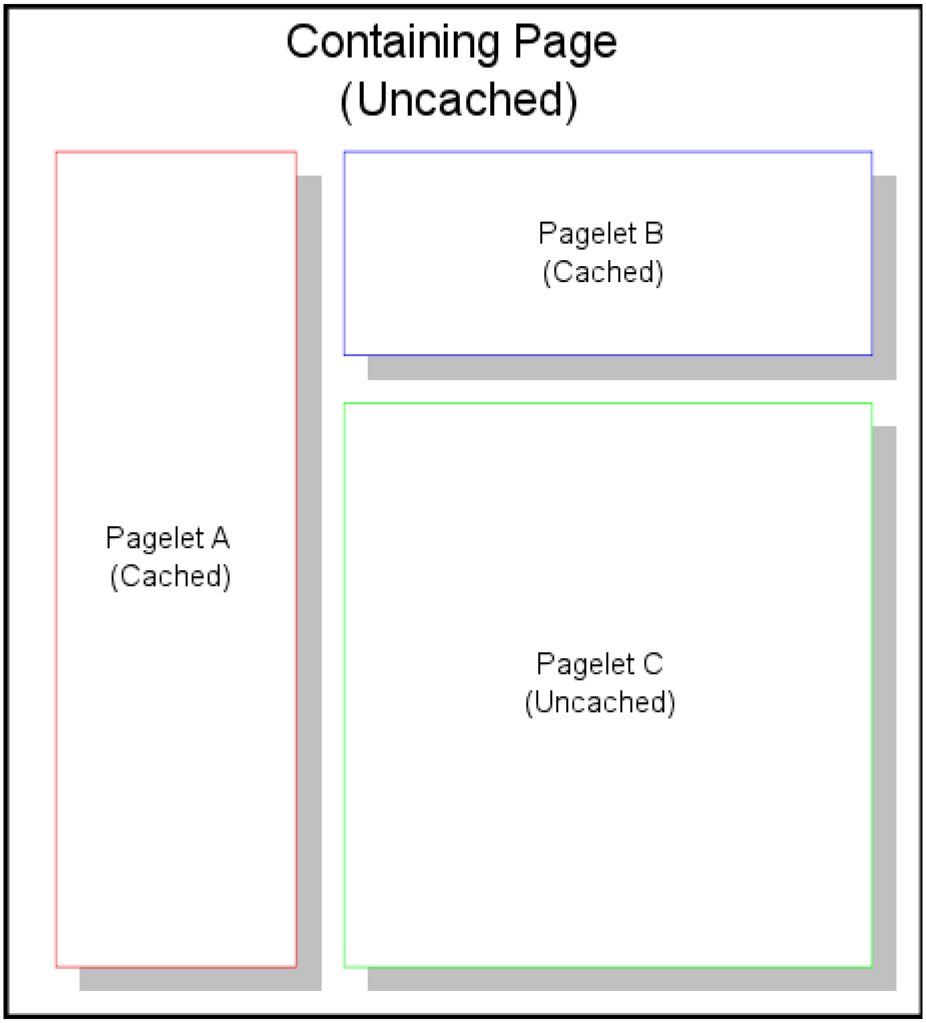
The following sections describe how the Satellite servlet caches web pages and how you can implement Satellite Server caching on your site. Use caching with the Satellite servlet in tandem with modular page design to create a fast, efficient website.

How the Satellite Servlet Caches Pages

The Satellite servlet allows caching at the pagelet level. To implement caching with the Satellite servlet, you use Satellite Server XML or JSP tags in your WebCenter Sites pages, and you access pages using special Satellite URLs.

For example, suppose that you used the Satellite servlet to implement pagelet-level caching on a web page named `myPage`. As shown in Figure 5–2, this page is composed of a containing page and three pagelets; A, B, and C. The containing page and pagelets A and B are already cached on a Satellite Server system, but pagelet C is not cached.

Figure 5–2 Web Page Named myPage



When a user requests `myPage`:

1. Satellite Server examines the URL. If it is a Satellite URL, the Satellite servlet gets the cached copy of the containing page. The servlet then looks for pointers to pagelets which are not currently in its cache, and requests those pagelets from WebCenter Sites. So, in our example, the Satellite servlet gets the containing page, and gets pagelets A and B from its cache.
2. The Satellite servlet requests Pagelet C from WebCenter Sites.
3. WebCenter Sites parses the appropriate XML to create Pagelet C and sends it to the Satellite servlet.
4. The Satellite servlet assembles Pagelets A, B, and C into the page, and sends the assembled page to the requester. The servlet also caches Pagelet C.

Implementing Caching with the Satellite Servlet

To implement pagelet-level caching with the Satellite servlet, you add Satellite tags to your WebCenter Sites templates. You do not develop any XML, JSP, or Java code on Satellite Server systems. In fact, Satellite Server does not know how to parse XML.

The Satellite tags in your elements are interpreted by the Java code you installed as part of Satellite Server. If this code is being called with a Satellite URL, it generates the information that the Satellite servlet uses to cache and construct the pagelets. If you do not call an element containing Satellite tags with a Satellite URL, the resulting page functions as if the Satellite tags were WebCenter Sites tags.

Satellite URLs look like the following example:

```
http://host_name:port/servlet/Satellite?pagename=page
```

where `host_name` and `port` are the hostname and port number of your Satellite Server computer, and `page` is the name of the page you are requesting. A Satellite URL can also include name/value pairs you want to pass to the called page.

Caching a Pagelet

The following sample code uses the `render:satellitepage` tag to call a pagelet. If the pagelet is not already in Satellite Server's cache, the Satellite servlet loads and caches the page. If the pagelet encounters an error during the processing and cannot be evaluated, it is not cached.

The `render:satellitepage` tag (and the `satellite:page` tag and their xml equivalents) identifies a cached pagelet by the pagename and name/value pairs passed to it. If the parameters or the name/value pairs differ from one invocation to another, a different pagelet will be cached, even if the content generated is the same. It is important to use name/value pairs to pass arguments to a pagelet through these tags.

Values passed through the ICS object pool, ICS List pool, page attribute context, and session (including session variables) may not be available to all called pagelets, because nested pagelets may not always be called at the same time as the parent. Furthermore, pagelets that rely on session or context data are rarely cacheable anyway, so attempting to cache them can result in non-deterministic behavior.

All parameters passed to a nested pagelet through `render:satellitepage` (and the `satellite:page` tag, and their xml equivalents) must be specified in the SiteCatalog as page criteria. This is WebCenter Sites's way of determining which parameters are relevant when building a pagelet for caching. Parameters other than those listed in the SiteCatalog are not permitted (an error indicating this will be written to the log).

```
<cs:ftcs>
<html>
  <body>
    <render:satellitepage pagename="My/Sample/Page" />
  </body>
</html>
</cs:ftcs>
```

Caching a Blob

Using Satellite tags to load and cache a blob is similar to the way you use Satellite tags to load and cache a pagelet. The following sample code adds to the previous example by calling a blob as well as a pagelet.

Line 7 uses the `ics:selectto` tag to perform a simple SQL query that retrieves a blob from the database. Results are returned in the form of an `IList` named `imagelist`.

Line 12 uses the `satellite:blob` tag to load the blob that was retrieved from the database. As with the `satellite.page` tag, if the blob is not in Satellite's cache, Satellite will load and cache the blob. The `cachecontrol` parameter is set so that the blob will expire at a given time; in this case, every 30 minutes.

```
1. <html>
```

```
<body>
<!-- NOTE: This will fail if list has no content (== null) -->

<ics:setvar name="category" value="logo"/>
<ics:setvar name="errno" VALUE="0"/>
<ics:selectto from="SmokeImage" list="imagelist" where="category" limit="1"/>

<ics:then>
<!-- Test a blob -->

<render:satelliteblob service="img src"blobtable="SmokeImage" blobkey="id"
blobwhere="imagelist.id" blobcol="urlpicture"
blobheader="image/gif"cachecontrol="*:30:0 */*/*"alt="imagelist.alttext"
border="0" />
</ics:then>

<render:satellitepage
pagename="QA/Satellite/Functional/xml/"pagelet1"cachecontrol="never"/>
</body>
</html>
```

Never-Expiring Blobs

If there are binary files (or blobs) on your site that seldom change or never change, such as company logos, and you are using the Satellite servlet to cache at the pagelet level, you can improve performance by using an alternative method to serve these blobs.

To serve never-expiring blobs

1. Copy the never-expiring images to all your Satellite Server hosts. Place them under the doc root for your web server.
2. Access the images through `<img src="pathname">` HTML tags rather than through `satellite:blob` Satellite tags.

For example, consider a never-expiring corporate logo file named `CorporateLogo.gif`. To use the alternative method of serving blobs, you would first copy the file to the web server's doc root on all your Satellite Server hosts. Then, instead of serving this logo through a `satellite.blob` tag, your element could simply use a tag like the following:

```
<img src="CorporateLogo.gif">
```

> **Note:** Be careful when using this mechanism for serving never-expiring images. For example, Satellite Server cannot warn you that one of the Satellite Server hosts does not contain the same image file as the other hosts.

```
http://myloadbalancer:1234/servlet/ContentServer?pagename=myPage
```

The expiration of the page is controlled by the `expiration` property. For more information on the `expiration` property, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

5.3 Viewing the Contents of the Satellite Server Cache

The Inventory servlet lets you view the various items stored in the cache. You invoke the Inventory servlet by using the following URL:

```
http://host:port/servlet/Inventory?username=username&password=passwordword&detail=
value
```

where:

Table 5–2 Inventory Servlet Parameters

Parameter	Description
`host:port (required)`	The host name and port number of the Satellite Server host whose cache you want to view.
`username (required)`	The user name that you enter to log you in to the Satellite Server host.
`password (required)`	The password that you enter to log you in to the Satellite Server host.
`detail (optional)`	The type of information you wish the Inventory servlet to display. Valid values are: ■ **`names`**: Displays the header information, plus the page names of the pages in the cache. ■ **`keys`**: Displays the header information, plus the page names and keys of the items in the cache. If you do not supply the `detail` parameter, or if you set its value to be anything other than `name` or `keys`, the header information displays.

The header contains the following information:

Table 5–3 Information Types

Information type	Description
Remote host	The host that this Satellite Server system forwards requests to.
Maximum cache objects	The maximum number of items allowed in the cache.
Current size	The number of items currently in the cache.
Cache check interval	How often the cache is checked for expired items, in minutes.
Default cache expiration	The value of the `expiration` property.
Minimum file size (in bytes)	Items larger than this value are stored in files. Items smaller than this value are stored in RAM.

5.3.1 CacheManager

The WebCenter Sites CacheManager object maintains both the WebCenter Sites and Satellite Server caches. CacheManager can do the following:

- Log pagelets in the cache tracking tables.

- Keep a record of the content (assets) that pages and pagelets contain by recording **cache dependency items** in cache-tracking tables. Cache dependency items are items that, when changed, invalidate the cached pages and pagelets that contain them. A cache dependency item is logged as a dependency for the current page and all of that page's parent pages.
- Remove pages and pagelets containing invalid items from the WebCenter Sites and Satellite Server caches.
- Rebuild the WebCenter Sites and Satellite Server caches with updated pages and pagelets after the invalid pages have been removed.

5.3.2 The SiteCatalog Table

WebCenter Sites's `SiteCatalog` table lists the pages and pagelets generated by WebCenter Sites. An element must have an entry in the `SiteCatalog` table to be cached on WebCenter Sites and Satellite Server.

The fields in the `SiteCatalog` table set the default behavior of a WebCenter Sites page, including default caching behavior. For more information on the `SiteCatalog` table and its fields, see Section 23.4, "Creating Template Assets" and Section 23.6, "Creating SiteEntry Assets."

5.3.3 The Cache Key

Items stored in the WebCenter Sites and Satellite Server caches are given a name called a **cache key**. The cache key uniquely identifies each item in the cache. CacheManager locates items in the cache using the cache key. WebCenter Sites and Satellite Server generate cache keys automatically, based on the values in the `pagename`, `resargs`, and `pagecriteria` fields of the `SiteCatalog` table, and other internal data.

5.3.3.1 pagecriteria and the Cache Key

You include variables used by the page in the cache key by specifying them in a comma-separated list in the `pagecriteria` field of the `SiteCatalog` table. For example, suppose that you have a page called `myPage` which uses the values `red` and `blue`. To include `red` and `blue` in the `myPage` cache key, enter:

`favoritecolor,second_favoritecolor` in the `pagecriteria` column and `favoritecolor=red&second_favoritecolor=blue in the resargsl column.`

WebCenter Sites and Satellite Server use the `pagecriteria` and parameters that are passed to cached pages to help generate the cache keys. If the parameters differ from one invocation to another, a different page will be cached even if the content being generated is the same. For example:

```
http://mysatellite:1234/servlet/ContentServer?pagename=myPage&favoritecolor=red
```

calls a different page than:

```
http://mysatellite:1234/servlet/ContentServer?pagename=myPage&second_
favoritecolor=blue
```

whether or not the content being generated is the same. Values passed in by the URL override values set in `pagecriteria`. For example, you have the `myPage` `pagecriteria` set to `red,blue`:

- If the URL passes in a value of `green,` then `green,blue` (not `red,blue`) will go into `myPage's` cache key.

- If the URL passes in values of green,violet, then green,violet (not red,blue) will go into myPage's cache key.
- If the URL passes in values of green,violet,yellow, an error results.

If a page does not have pagecriteria set, the values in the resargs fields go into the cache key. As with pagecriteria, values passed in by a URL override values specified in the resargs fields.

5.3.4 Caching Properties

The default cache settings for WebCenter Sites and Satellite Server are contained in the futuretense.ini file. Additional Satellite Server properties are contained in satellite.properties. All properties can be modified by use of the Property Editor.

This section summarizes the WebCenter Sites and Satellite Server caching properties. For detailed information about the properties and the Property Editor, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

5.3.4.1 WebCenter Sites Page Caching Properties

The following properties in futuretense.ini control disk caching on WebCenter Sites:

- cs.pgCacheTimeout, which specifies the default timeout for pages in the WebCenter Sites cache.
- cs.freezeCache, which controls whether the cache pruning thread should run to remove expired entries from the cache.
- cs.nocache, which disables the entire WebCenter Sites page cache.
- cc.SystemPageCacheTimeout, which specifies the number of minutes a cached page is held in memory.
- cs.alwaysUseDisk, which specifies the default behavior for page entries in the SiteCatalog that have no cache override property specified.
- cc.SystemPageCacheCSz, which specifies the maximum number of pages that can be cached in memory.

5.3.4.2 Satellite Server Properties

Satellite Server has two sets of properties (given in detail in the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*):

- One set of properties is in the futuretense.ini file on your WebCenter Sites system and includes the following:
  - satellite.page.cachecontrol.default, a deprecated property that specifies a default value for the cachecontrol parameter for the satellite.page, and RENDER.SATELLITEPAGE tags and their JSP equivalents.
  - satellite.blob.cachecontrol.default, which specifies a default value for the cachecontrol parameter for the satellite.blob, and RENDER.SATELLITEBLOB tags and their JSP equivalents.
- The other set of properties is in the satellite.properties file on each Satellite Server host and comprises the following:
  - cache_folder: Specifies the directory into which Satellite Server will cache pagelets to disk.

– `file_size`: Separates disk-cached pagelets and blobs from memory-cached pagelets and blobs according to the size that you specify.

– `expiration`: Sets the default value for the length of time blobs stay in Satellite Server's cache.

– `cache_check_interval`: Controls the frequency of the cache cleaner thread, and therefore when expired objects are pruned from cache.

– `cache_max`: Specifies the maximum number of objects (pagelets and blobs) that can be cached (memory cache and disk cache combined) at a time.

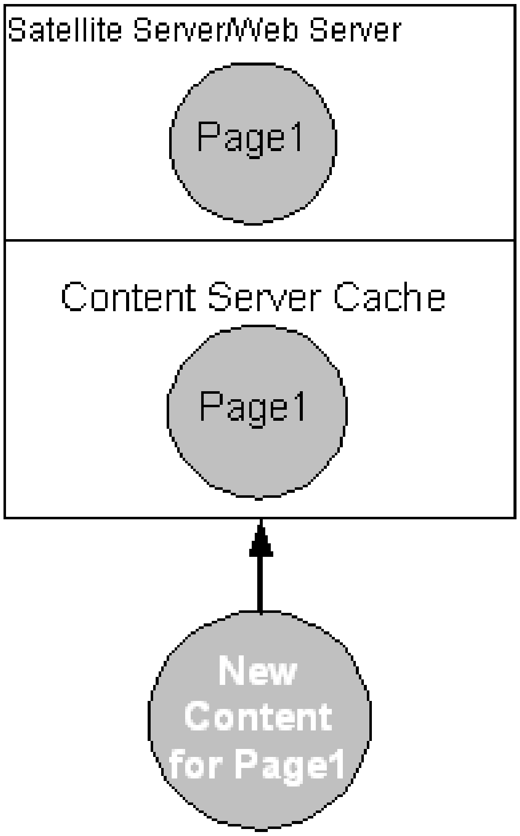
5.4 Double-Buffered Caching

WebCenter Sites and Engage implement a double-buffered caching strategy, which uses the WebCenter Sites and Satellite Server caches in tandem on your live website. This double-buffered caching strategy ensures that pages are always kept in cache, either on WebCenter Sites or Satellite Server.

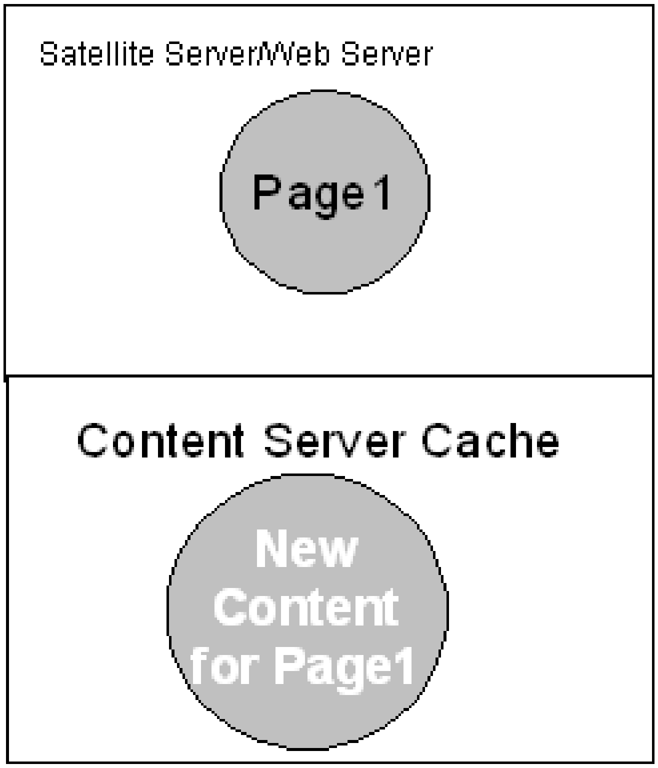
You can implement a similar caching strategy if you are running the WebCenter Sites core and Satellite Server without any of the other CS modules or products, by using the CacheManager Java API. For more information about the CacheManager Java API, see the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.

Both the WebCenter Sites core and Satellite Server caches are maintained by WebCenter Sites's CacheManager object. CacheManager tracks when content changes by logging elements and the assets that those elements call in cache tracking tables.

When assets are updated and published, WebCenter Sites and Satellite Server caches are automatically flushed and updated in the following order:

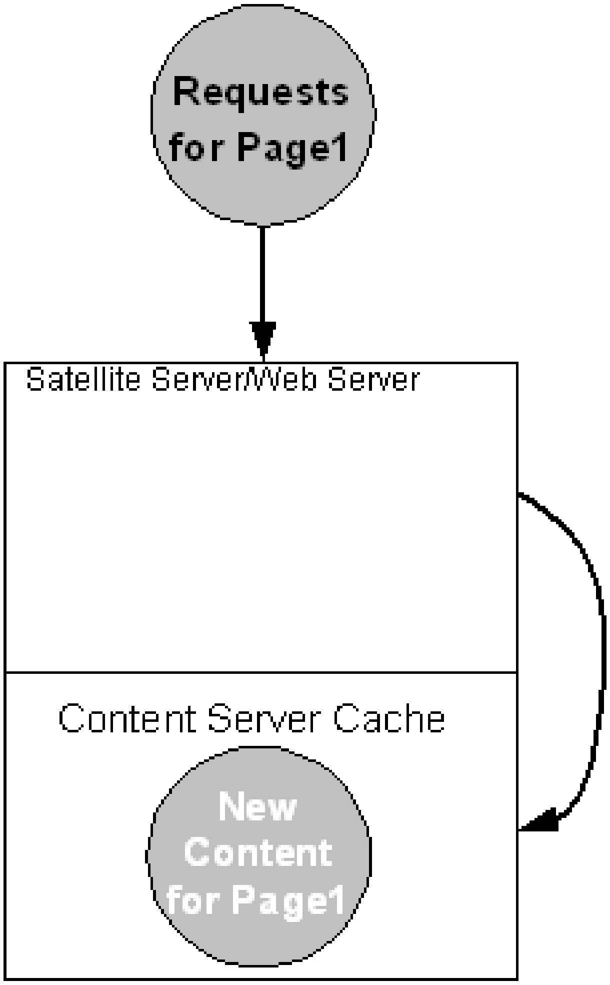


- Content providers publish updated assets to the delivery system. CacheManager checks the cache tracking tables to see which cached items are affected by the updated assets.

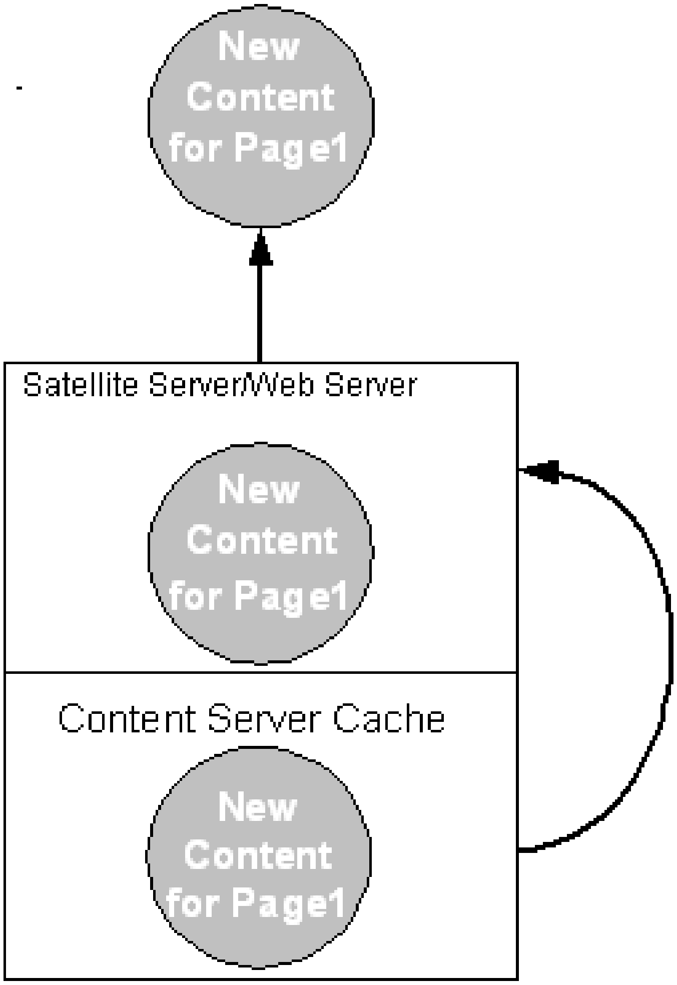


- CacheManager flushes the outdated `Page1` from the WebCenter Sites cache, then reloads the WebCenter Sites cache with the updated `Page1`.

  Any requests for `Page1` will be served the old version of `Page1` from the Satellite Server cache. This protects the WebCenter Sites computer from undue load as it deletes and rebuilds its cache.



- CacheManager flushes the outdated items from the Satellite Server cache. As visitors come to the website and request `Page1`, the Satellite Server searches to see if `Page1` is in its cache. Because `Page1` is not in the Satellite Server cache, the request is passed on to WebCenter Sites.



- The Satellite Server system's cache is filled with an updated version of `Page1`, taken from the WebCenter Sites cache. The updated page is served to the requestors. If `Page1` were requested again, the page would be served from the Satellite Server cache.

5.4.1 Implementing Double-Buffered Caching

The first step in implementing double-buffered caching on your website is to design modular pages, as described in Section 5.1, "Modular Page Design." Once you have developed a modular page design, you implement a double-buffered caching strategy in three steps:

- Develop a pagelet caching strategy
- Set how individual pages and pagelets are cached by using the `pagecriteria` field of the `SiteCatalog` table
- Code your elements with Satellite tags

5.4.1.1 Pagelet Caching Strategies

With a modular page design, caching occurs at the pagelet level; the containing page is never cached, so that any cached pagelets are always protected by ACLs. You choose which pagelets get cached based on how frequently they are updated.

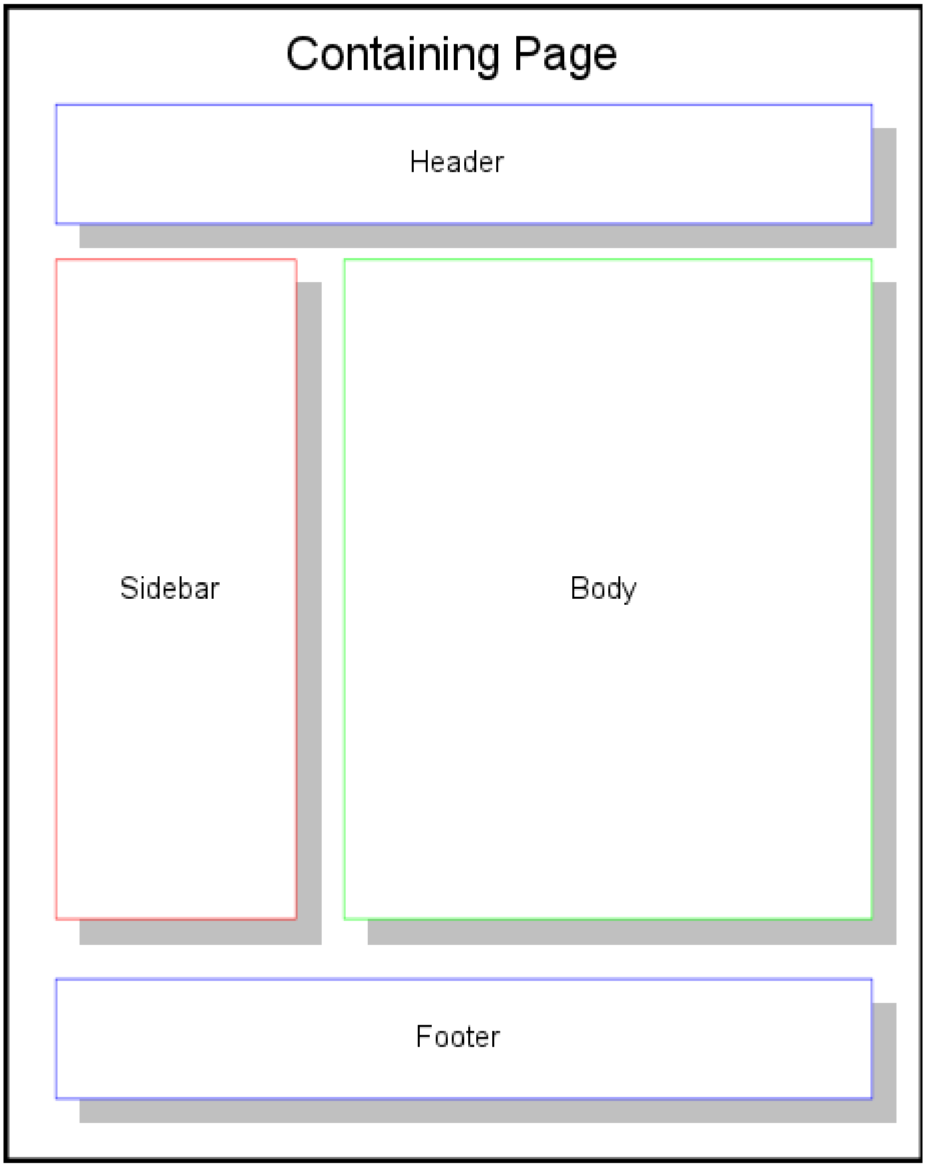
The following table summarizes the guidelines for caching pagelets:

Table 5–4 Guidelines for Caching Pagelets

Cache a Pagelet	Don't Cache a Pagelet
▪ If the content seldom changes. ▪ If the pagelet does not contain logic that requires evaluation to work.	▪ If the content changes frequently. ▪ If the content must be "real time." ▪ If the pagelet contains code that checks for ACLs, or other logic that requires evaluation to work.

The following diagram is an example of a modular page:

Figure 5–3 Modular Page



The containing page should never be cached; this lets you put logic, which requires evaluation by WebCenter Sites, into your pages, while still gaining the performance benefits of caching. It also allows your page to be protected by WebCenter Sites ACLs.

The header and footer pagelets in this example should be disk cached. They rarely gets updated, and should be designed accordingly. The header and footer may be static HTML written into your template, or disk-cached content from WebCenter Sites.

The sidebar is also a good candidate for disk caching. It has a small number of variations, and its content is determined by a small number of parameters.

Determining how to cache the body pagelet is more complex. The contents of the body pagelet probably depend on where the website visitor is in the site. There are three possible types of content for the body pagelet:

- The results of a search that the website visitor runs
- The results of a frequently run query
- An article

Your caching strategy should be as follows:

- If the content of the body pagelet is the result of a search based on parameters that the website visitor enters, you do not want to cache it. Such pages change for each visitor, and there is little benefit to caching them.
- If the content is the product of a standard query that visitors often use, you should use resultset caching. Caching frequently run queries in the memory cache improves performance. For more information on resultset caching, see Chapter 14, "Resultset Caching and Queries."
- If the content of the body pagelet is the text of an article, you should cache the pagelet to disk.

5.4.2 Setting cscacheinfo

The values in the cscacheinfo field of the `SiteCatalog` table allow you to control how pages get cached on WebCenter Sites on a page-by-page basis.

You can change these properties for each page and pagelet in your website. For example, if you want a containing page element to be uncached on WebCenter Sites, set the values in `cscacheinfo` to `false`.

For more information on the cscacheinfo field, see Section 23.4, "Creating Template Assets."

5.4.3 Coding for Caching

To implement double-buffered caching, you code your elements with Satellite Server tags. If you are running WebCenter Sites and Satellite Server only, use the Satellite tags documented in the Satellite Server sections of the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

Automatic cache maintenance is dependent upon logging your assets in the cache tracking tables. If you use the `ASSET.LOAD` tag to load an asset, that asset is automatically logged in the cache tracking tables. For those sections where `ASSET.LOAD` is not used, use the `RENDER.LOGDEP` tag to log content in the cache tracking tables.

Note: Cache dependencies are logged only if a page or pagelet is cached on WebCenter Sites. If a page is uncached on WebCenter Sites but cached on Satellite Server, that page will not be automatically flushed from the cache when its content is updated.

5.4.4 Caching and Security

Cached pagelets require special security considerations as you design your site and develop your caching strategy. The following sections outline security considerations for pages cached in the WebCenter Sites and Satellite Server caches.

5.4.4.1 WebCenter Sites Security

Pagelets that are disk cached on WebCenter Sites are bound by WebCenter Sites's ACLs, allowing you to use those ACLs to prevent unauthorized access to a page.

Note, however, that although WebCenter Sites checks the ACL of a containing page, it does not check the ACLs of the pagelets that the containing page calls. For example, suppose that your site uses three ACLs: `Open`, `Secret`, and `TopSecret`. Your containing page can be viewed by members of the `Open` ACL, but it calls pagelets that should be viewed only by members of the `Secret` and `TopSecret` ACLs. Because WebCenter Sites only checks a visitor's ACL of the containing page, visitors with the `Open` ACL can view content meant for members of the `Secret` and `TopSecret` ACLs.

To ensure that all the relevant ACLs are checked

1. Include the ACL for the page that you want to protect in that page's cache criteria, as shown in the following sample code:

```
<render.satellitepage pagename="innerwrapper" userAcl="SessionVariables.member"
c="Article" cid="123">
```

2. In the pagelet, insert code to check the ACLs, as shown in the following sample:

```
<asset.load name="art" type="Variables.c" OBJECTID="Variables.cid"/>
<ASSET.GET NAME="art" FIELD="myACL"/> <!-- note you need a column in your db to
support this -->
<IF COND="Variables.userACL=Variables.myACL">
<THEN>
<render.satellitepage pagename="protected_art_tmpl1" c="Variables.c"
cid="Variables.cid"/>
</THEN>
<ELSE>
<render.satellitepage pagename="accessDenied"/>
</ELSE>
</IF>
```

5.4.4.2 Satellite Server Security

Pagelets that will be cached on Satellite Server are bound only by WebCenter Sites ACLs under the following circumstances:

- If they are retrieved from the WebCenter Sites cache
- If they must be generated by WebCenter Sites to fulfill the page request

If a pagelet is served from the Satellite Server cache, it is no longer protected by WebCenter Sites ACLs.

To ensure that the content of your Satellite Server pages is secure, never cache your containing page and be sure that you put an ACL checking mechanism in the uncached container.

If your elements are coded with Satellite tags but you do not yet have Satellite Server installed, the page design considerations outlined in Section 5.4.4.1, "WebCenter Sites Security" apply to you. Once Satellite Server is installed, however, WebCenter Sites checks the ACLs of uncached pagelets called from a containing page. The ACLs of pagelets cached on Satellite Server are not checked.

6

Intelligent Cache Management with WebCenter Sites

Whenever a site is built, it is critical that the rendering engine cache be properly configured so that all of the components work in concert. This chapter describes the rendering engine cache and its components. It also describes the cache configuration properties that enable CacheManager to clear all caches (ContentServer, BlobServer, and Satellite Server caches) of any object which becomes obsolete because of changes in its underlying content.

This chapter contains the following sections:

- Section 6.1, "WebCenter Sites's Rendering Engine Cache"
- Section 6.2, "CacheManager"
- Section 6.3, "Enabling CacheManager"

6.1 WebCenter Sites's Rendering Engine Cache

WebCenter Sites's rendering engine cache is a two-tier cache. Tier 1 consists of ContentServer and BlobServer; Tier 2 consists of Satellite Server. Each component is independently configurable with fine controls that tune cache size, cache timeout, and dependency management behavior.

Whenever a site is built, it is critical that the rendering engine cache be properly configured so that all of the components work in concert. If the components are configured correctly, WebCenter Sites can perform extremely well, effectively preventing users from viewing uncached content nearly all of the time. However, if these components are mis-configured, WebCenter Sites's behavior can be non-intuitive and unpredictable. Inadequate caching can hamper performance, and improper co-ordination of the cache inventory can result in stale content being rendered indefinitely. To address this, WebCenter Sites includes a module called CacheManager, which can actively manage the cache on behalf of the whole system.

6.2 CacheManager

When a site uses CacheManager, it can record the existence of a *compositional dependency* against an object that is to be cached by the rendering engine. For example, if a pagelet renders an asset, then the asset is a compositional dependency on that page. If the asset changes, the page is no longer valid and must be flushed from cache.

Utilizing CacheManager to flush the cache requires ceding full control over the lifecycle of rendering engine cache objects to CacheManager by specifying that the objects never expire from the cache. When CacheManager determines that they are obsolete because of changes in the underlying content (i.e., in one of the compositional dependencies recorded against each object), it removes those objects from the cache.

> **Note:** The reason for specifying an infinite expiration time is to ensure that CacheManager keeps a record of all objects that are cached, as well as what dependencies are tracked against them. This record is stored on WebCenter Sites, and it is linked to the existence of the cached object on the first tier. This record enables CacheManager to infer the existence of objects in the second tier cache and therefore flush the objects from the second tier cache.
>
> If, however, an object were to expire from the cache, its record would be removed, leaving CacheManager without the information it requires to properly flush the object from the second tier cache.

Enabling CacheManager's features is almost completely automatic:

- By default, the cache is configured so that objects never expire.
- Compositional dependencies are recorded against the Blob and Page cache on the lower tier. Tags such as `<asset:load>` and `<render:sateliteblob>` provide *automatic* compositional dependency recording (see the *Oracle Fusion Middleware WebCenter Sites Tag Reference* for a complete list), whereas the two tags `<portal:logdep>`, and `<render:logdep>` provide *explicit* compositional dependency recording.
- Whenever assets are modified or published, WebCenter Sites automatically invokes CacheManager to purge the old content from the cache and, in the case of publishing, instructs CacheManager to pre-cache the new content in the background prior to flushing the second tier cache.

Site visitors enjoy the best possible cache performance by never having to view uncached content. For more information about recording compositional dependencies, consult the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

6.3 Enabling CacheManager

This section describes the Tier 1 and Tier 2 cache configuration properties and how they must be set in order to enable CacheManager.

This section contains the following topics:

- Section 6.3.1, "Tier 1 Cache Configuration Properties"
- Section 6.3.2, "Tier 2 Cache Configuration Properties"

6.3.1 Tier 1 Cache Configuration Properties

The tables in this section describe properties that regulate the WebCenter Sites page cache and BlobServer blob cache:

- Table 6–1, " WebCenter Sites Page Cache Properties"
- Table 6–2, " BlobServer Cache Properties"

More detailed descriptions are given in the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

Table 6–1 WebCenter Sites Page Cache Properties

Property	Description
`cs.pgCacheTimeout`	This property specifies how long a page should reside in the WebCenter Sites page cache, and it affects CacheManager as follows: By default, this property is set to `0`, which means that the pages should reside in this cache forever, and that removal of these pages must be done explicitly. **For CacheManager to operate properly, this property must be set to 0**. Otherwise, pages will expire, making it impossible for CacheManager to remove the corresponding pages from the Tier 2 cache, and users will view stale data. Setting this property to a positive integer causes pages to expire after the number of minutes specified by the integer.
`cs.IItemList`	This property specifies the class implementing the `IItemList` interface, and it affects CacheManager as follows: The `IItemList` interface is used to record compositional dependencies in the page cache. If this property is set to a legal class, then dependency items will be recorded against a page id in the `SystemItemCache` table, and this is what enables CacheManager. An illegal value results in CacheManager having no effect.

`cc.SystemPageCacheCSz`
`cc.SystemPageCacheTimeout`
`cs.alwaysusedisk`
`cs.freezeCache`
`cs.nocache`
`cs.requiresessioncookies`
	These properties are used to configure page caching, but have no effect on CacheManager. For more information about the properties, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

Table 6–2 BlobServer Cache Properties

Property	Description
`bs.bCacheTimeout`	This property specifies how many seconds a blob should remain cached by BlobServer, and it affects CacheManager as follows: When compositional dependencies are recorded against a blob in the `SystemItemCache` table, they are configured such that they will be removed from the table after the blob expires from the cache. This prevents excessive growth of the `SystemItemCache` table. However, removing the entry from the table disables CacheManager from removing the corresponding blobs from the Tier 2 cache, and users will view stale data.

Table 6–2 (Cont.) BlobServer Cache Properties

Property	Description
`bs.bCacheSize`	This property specifies how many blobs will be stored in the BlobServer cache, and has no effect on CacheManager. Unlike the `bs.bCacheTimeout` property, when a blob is evicted from the blob memory cache due to the cache being full, the corresponding row is not removed from the `SystemItemCache` table. Consequently, this property has no effect on CacheManager.
`cs.recordBlobInventory`	This property specifies whether compositional dependencies should be recorded against blobs. This property must be set to `true` (the default) for CacheManager to operate on blobs.
`bs.security`	This property controls the security feature of BlobServer, and it affects CacheManager as follows: When BlobServer security is enabled, caching is disabled. Consequently, BlobServer security is incompatible with CacheManager's Intelligent Cache Management features. By default, this level of security is disabled. For more information about BlobServer security, see Chapter 5, "Page Design and Caching" and Chapter 7, "Advanced Page Caching Techniques."
`cs.manage.expired.blob.inventory`	This property controls the removal of blob dependency records from the `SystemItemCache` table after a blob expires from the blob cache. Its effect on CacheManager depends on the value of `bs.bCacheTimeout`. ▪ If **`bs.bCacheTimeout`** is set to 0 or less, this property has no effect. ▪ If **`bs.bCacheTimeout`** is set to a positive integer, setting this property to `true` ensures that CacheManager still operates correctly, but at the cost of growth in the `SystemItemCache` table. The default value is `false`.

6.3.2 Tier 2 Cache Configuration Properties

Tier 2 cache configuration properties deal with the Satellite Server cache, both page and blob.

None of the Tier 2 properties affect the correct operation of CacheManager. They do, however, serve as important diagnostic aids if CacheManager happens to be operating incorrectly. The timeout and configuration values of the Tier 2 cache properties are important in troubleshooting unpredictable behavior.

Typically, unpredictable behavior results when objects are cached on the Tier 2 cache but not on the Tier 1 cache, and so they are not actively flushed when the dependent asset is saved or published. Consult the *Oracle Fusion Middleware WebCenter Sites Property Files Reference* for configuration details.

Unpredictable behavior can also result if no compositional dependency is recorded against an object that is cached. This scenario precludes all active management of that object in the caches. Consult the *Oracle Fusion Middleware WebCenter Sites Tag Reference* for details about which tags automatically record compositional dependencies, and which tags must be used in conjunction with explicit recording using one of the `:logdep` tags.

Note: Do not record excessive compositional dependencies on your pages or blobs. Doing so will cause unnecessary flushing of the cache, which under certain circumstances, can result in severe performance problems during publishing. Be especially judicious when recording unknown compositional dependencies. Consult Chapter 28, "Coding Elements for Templates and CSElements" for more information about compositional dependencies.

7 Advanced Page Caching Techniques

Caching improves WebCenter Sites's performance in serving pages. Caching rendered content eliminates the need to render the content each time it is requested. This reduces the hardware requirements for the WebCenter Sites system, reduces the number of times clients make requests for uncached content, and ultimately improves response time.

The caching system has multiple layers, which allows regeneration of cached objects to be done on one cache level, while the client is being served cached content from another cache level. WebCenter Sites comprises the inner level of cache, and Satellite Server comprises the outer layer of cache.

This chapter describes how rendered object caching works in the WebCenter Sites platform. It describes how pages and blobs are cached, where they are cached, and how they are retrieved from cache on both WebCenter Sites and Satellite Server systems.

This chapter contains the following sections:

7.1 Configuring the WebCenter Sites Cache

Both WebCenter Sites and Satellite Server cache pages, pagelets, and blobs. WebCenter Sites provides three different rendering engine caches: CS page cache, BlobServer cache, and SS cache. All the caches are controllable. They can be configured and emptied as follows:

- Maximum cache size can be configured. For information and instructions, see Section 7.1.1, "Configuring Maximum Cache Size" WebCenter Sites cache, BlobServer cache, and Satellite Server cache.
- Objects can be stored in the cache with expiration information, such that when the object has expired from cache it is removed. For information and instructions, see Section 7.1.2, "Setting Expiration Time for an Individual Entry" WebCenter Sites cache and BlobServer cache.
- Objects can be explicitly removed from the cache either manually, or automatically by using CacheManager. For information and instructions, see Section 7.1.3, "Explicitly Removing Entries from Cache" WebCenter Sites cache, BlobServer cache, and Satellite Server cache.

There are two levels of caching for the WebCenter Sites page cache:

- In the database.
- In memory. Memory cache is a transparent subset of the database cache; however, it is independently configurable.

This section describes the main configuration settings for each cache and contains the following topics:

- Section 7.1.1, "Configuring Maximum Cache Size"
- Section 7.1.2, "Setting Expiration Time for an Individual Entry"
- Section 7.1.3, "Explicitly Removing Entries from Cache"

7.1.1 Configuring Maximum Cache Size

- WebCenter Sites's database cache does not have a maximum size configuration option.
- The memory subset of the page cache lets you specify the maximum number of entries present by using the `cs.SystemPageCacheCSz` property in the `futuretense.ini` file. Setting a positive integer specifies the maximum number of entries that will be allowed to exist in the cache (the cache uses a Least Recently Used (LRU) algorithm for identifying the entry to be pruned when the maximum size has been reached).

  > **Note:** Setting a value of `0` will cause all entries to be added and then promptly removed. However, this should be avoided.
  >
  > The `cs.SystemPageCacheCSz` property must not be set to `-1`.

  The maximum size does not have anything to do with the aggregate number of bytes stored in cache.

7.1.2 Setting Expiration Time for an Individual Entry

The lifetime of an individual entry in the WebCenter Sites page cache is determined by the `cscacheinfo` setting for each entry. The CacheInfo object derives values, if not explicitly set in the cscacheinfo field, from the configuration file. For CacheInfo syntax, see Section 7.4, "CacheInfo String Syntax."

7.1.3 Explicitly Removing Entries from Cache

WebCenter Sites provides two ways of removing entries from cache: manually and automatically, using CacheManager.

Manual Removal

You can manually remove an entry from the page cache, you can use the CacheServer servlet. The CacheServer provides two options:

- Flushing the entire cache.
- Forcing a flush of all pages at the moment they expire. In order to invoke the CacheServer *flush all* functionality, you must be logged in as a user with *destroy* privileges on the `SiteCatalog` table, and specify the parameter `all=true` when invoking the CacheServer servlet. If you do not specify a parameter, then all

expired entries (those whose expiry date is in the past) will be cleared from the cache immediately. Entries that have not yet expired will not be cleared.

> **Note:** In no case will an expired entry be served from the cache, even if it is still in the database table. WebCenter Sites checks the expiry date of any page it retrieves from cache before serving the page. If WebCenter Sites attempts to serve a page that has expired, the page will be removed from the cache immediately and a new page will be generated.

Automatic Removal

CacheManager is a module that ties in very closely with the internals of the WebCenter Sites page and blob cache mechanisms. It is a tool that lets you manage the contents of all of the rendering caches based on the items loaded on a page, on the expiration of the pages, or on parameters passed into pages.

CacheManager itself could be the subject of an entirely independent document. However, an overview of its functionality is described herein. For detailed information about its methods and required arguments, consult the COM.FutureTense.Cache.CacheManager Javadoc.

1. A `CacheManager` is instantiated using one of two constructors. One constructor sets `CacheManager` with all of the currently registered Satellite Servers. The other constructor lets you specify which Satellite Servers this instance of `CacheManager` will actually manage.
2. Next, the `CacheManager` needs to be populated with pages and blobs. This is done by using one of the following methods:

   ```
   setByCachedDate(ICS ics, boolean before, String timestamp)
   setByItemDate(ICS ics, boolean before, String timestamp)
   setPagesByArg(ICS ics, String paramName, String paramValue)
   setPagesByID(ICS ics, String[] ids)
   ```

   The contents of the Page, Blob and Satellite caches are closely tied together. It is always the case, except as a result of a configuration error, that any object cached on Satellite Server will be present in the WebCenter Sites cache. This means that WebCenter Sites has a record of all entries in all rendering engine caches. **CacheManager** uses this record in order to be able to manage the contents of each of the caches, without having to directly interrogate each cache for the information explicitly.

   ```
   setByCachedDate(ICS ics, boolean before, String timestamp)
   ```

   This method lets you populate CacheManager based on the date an entry was last added to the cache. You can choose whether you want to populate it with all of the entries modified either before or after the date specified.

   ```
   setByItemDate(ICS ics, boolean before, String timestamp)
   ```

   This method lets you populate CacheManager based on the date an item on an entry was last modified. As with `setByCachedDate(ICS, boolean, String)`, you can choose whether you want all entries whose items were modified before or after the date specified.

   ```
   setPagesByArg(ICS ics, String paramName, String paramValue)
   ```

This method lets you populate CacheManager based on name-value pairs present in the cache key (including pagename).

```
setPagesByID(ICS ics, String[] ids)
```

This method lets you populate CacheManager based on the exact item IDs of the items stored on the pages or blobs in the cache.

Once fully populated, CacheManager is able to manage the contents of the caches. This is done using one of the four main service methods:

- `flushCSEngine(ICS ics, int mode)`

  This method flushes all of the pages and blobs currently populated in the CacheManager from the WebCenter Sites page and blob caches.

- `flushSSEngines(ICS ics)`

  This method flushes all of the pages and blobs currently populated in the CacheManager from the Satellite Server cache. This is done by sending an HTTP request to the FlushServer servlet with the appropriate `<page>` and `<blob>` tags embedded in it. Satellite Server interprets these tags and converts them into a cache key, then flushes the corresponding pages from cache.

- `refreshCSEngine(ICS ics, int mode)`

  This method sends a request (using `ICS.ReadPage` or `ICS.BlobServer`) that regenerates the object and automatically re-populates the cache

- `refreshSSEngines(ICS ics)`

  This method sends a request via HTTP to Satellite Server to read the pages. The returned bytes are ignored, but the result is that the Satellite Server cache is re-populated.

Using these methods it is possible to take advantage of double-buffered caching, a tool that can enable extremely high performance dynamic sites. For more information about double-buffered caching, see Chapter 5, "Page Design and Caching."

7.2 Configuring the Blob Server Cache

The BlobServer cache is an *all or nothing* cache, entries are either globally cached or globally not cached.

BlobServer caching is disabled if security is enabled. Thus, if `bs.security=true`, caching is disabled.

This section contains the following topics:

- Section 7.2.1, "Configuring Maximum Cache Size"
- Section 7.2.2, "Setting Expiration Time for an Individual Entry"
- Section 7.2.3, "Explicitly Removing Entries from Cache"

7.2.1 Configuring Maximum Cache Size

The property `bs.bCacheSize` in `futuretense.ini` specifies the number of entries the blob cache will contain. If the size is set to a negative number, the blob cache will be allowed to grow indefinitely.

7.2.2 Setting Expiration Time for an Individual Entry

Blob Server does not support individual entry expiration for cached entries. All cached objects will reside in cache for the timeout determined by the `bs.bCacheTimeout` property in `futuretense.ini`. A negative timeout indicates that entries should not time out. A positive integer specifies the number of minutes an object will reside in cache.

7.2.3 Explicitly Removing Entries from Cache

BlobServer supports the flushing of both individual entries and all entries from the cache.

Manual Removal

To manually remove an entry from cache, simply rename the `blobtable` parameter to `flushblobtable`. This will remove the entry corresponding to the rest of the parameters from the cache.

To manually remove all entries from the cache, there are two options. One is to invoke the BlobServer servlet with the parameter `flushblobtables` (notice the "`s`"). The other is to invoke the CacheServer servlet as described above. However, this will flush all pages and all blobs from the cache.

Automatic Removal

Because blob dependency items are recorded when blob links are generated, it is possible to invoke CacheManager to manage blobs as well as pages. (In fact, CacheManager always manages blobs and pages together). Refer to the sections about CacheManager and WebCenter Sites for details about using CacheManager.

> **Note:** Developers should check the asset status before serving BLOBs. This avoids issues such as the BlobServer serving a BLOB after an asset has been voided (this behavior occurs with basic assets only).

7.3 Configuring the Satellite Server Cache

The generic Satellite Server cache configuration is done in the `satellite.properties` file; typically, however, cache configuration is overridden on an object-by-object basis.

This section describes configuring the maximum cache size and explicitly removing entries from cache (manual removal and automatic removal)

Configuring Maximum Cache Size

The maximum number of entries that can be stored in the cache at once is configurable using the `cache_max` property in the `satellite.properties` file. If the property is set to a negative integer, the cache will not be limited by size. Any positive integer will specify the maximum number of entries that can be stored in the cache.

Explicitly Removing Entries from Cache

Individual entries can be removed from the Satellite Server cache either manually or using CacheManager, as explained in this section.

- **Manual Removal**: Satellite Server includes a servlet called FlushServer. By submitting a `GET` request to this servlet (specifying the username, password and reset parameters), it is possible to flush all of the contents of the Satellite Server cache. It is not possible to flush individual entries using `GET`.

- **Automatic Removal**: As described above, it is possible to flush the Satellite Server cache using CacheManager. As described, CacheManager is only able to flush entries on Satellite Server if a corresponding object is cached on WebCenter Sites. This is the case because of the way WebCenter Sites tracks the contents of the Satellite Server cache.

  As described above, it is possible to flush the Satellite Server cache by using CacheManager, as long as a corresponding object is cached on WebCenter Sites. The corresponding object is required because of the way WebCenter Sites tracks the contents of the Satellite Server cache.

  The relevant CacheManager methods for dealing with the Satellite Server cache are `flushSSEngines()` and `refreshSSEngines()`. For information about the methods, see Section 7.1.3, "Explicitly Removing Entries from Cache."

7.4 CacheInfo String Syntax

The `cscacheinfo` and `sscacheinfo` fields of the SiteCatalog are populated with a CacheInfo string. This section describes the format of the string. It is a two-part, comma-separated string. The first part indicates whether the page will be cached. The second part describes the expiration.

Sample values:

```
false
true
true,*
true,~4
true,@1987-06-05 04:32:10
true,#00:00:00 */*/*
*
(blank)
```

CacheInfo String: First Part

The first part in CacheInfo must be one of the following values:

```
false
true
(blank)*
```

- If the value is `false`, then the page will not be cached.
- If the value is `true`, then the page will be cached according to the information provided in the second element.
- If the value is blank, then WebCenter Sites will consult the `futuretense.ini` property `cs.alwaysusedisk`. If this property is set to `yes`, then a blank value will be interpreted as having the same behavior as `true`. If the value is set to `no` (the default value), then a blank value will be interpreted as having the same behavior as `false`.
- If the value is *, then it will be treated as blank.

CacheInfo String: Second Part

The second part in CacheInfo describes when a page that is to be cached should be removed from cache. If the first element is `false` (or is interpreted as `false`), then the second element is ignored.

There are three ways of specifying the expiration of a page:

```
page timeout (in minutes)
instant in time expiration
cron-like TimePattern expiration
```

Legal values include:

```
~<number of minutes>
@<date in JDBC format>
#<COM.FutureTense.Util.TimePattern format>
*
(blank)
```

Page Timeout

If the second element starts with the tilde symbol (~), then the value following the tilde symbol (~) must be an integer. The value of this integer is the number of minutes a page will remain in cache after it was first created. A negative value or "0" indicates that the page will never expire (it will remain in cache forever).

Absolute Moment in Time

If the second element starts with the at symbol (`@`), then the value following the at symbol (`@`) must be a date expressed in the JDBC date string format, namely, `YYYY-MM-DD HH:MM:SS`. Once that date has passed, cached pages will be flushed from cache and the page will no longer be cached.

TimePattern

The TimePattern format is supported for describing page cache expiration. If the second element starts with the hashtag (`#`), then the value following the hashtag (`#`) must be a valid TimePattern string as defined by the public class `COM.FutureTense.Util.TimePattern`.

In general, the TimePattern syntax corresponds to the format used in most UNIX cron tables. It lets you specify expiration at a specific time or times every day, month, week, day of week, and year.

It is expected that the TimePattern format will become the most widely used format for page expiration.

Wildcard

If the second element is *, then the page will assume a timeout expiration behavior, as described in Timeout above. The timeout value will be read from the `futuretense.ini` file's `cs.pgCacheTimeout` property.

Blank

If the second element is blank, then it assumes the same behavior of *.

8

WebCenter Sites Tools and Utilities

WebCenter Sites includes several tools and utilities that you use together with the WebCenter Sites browser-based interface for developing and maintaining your websites. This chapter provides brief descriptions of these utilities, and tells you how to start them.

This chapter contains the following sections:

- Section 8.1, "Oracle WebCenter Sites Explorer"
- Section 8.2, "Connecting to a WebCenter Sites Database"
- Section 8.3, "CatalogMover"
- Section 8.4, "Property Editor"
- Section 8.5, "XMLPost"

8.1 Oracle WebCenter Sites Explorer

The Oracle WebCenter Sites Explorer tool is a Microsoft Windows application for viewing and editing tables and rows in the WebCenter Sites database, and for creating and editing executable elements (or files) written in XML or JSP. You use Oracle WebCenter Sites Explorer to do the following:

- Add entries to tables
- Edit rows within tables
- Track revisions to rows of tables
- Create and drop WebCenter Sites tables
- Organize tables and folders into projects
- Preview `SiteCatalog` records as pages in a browser
- Export and import records as integrated `.cse` type files
- Export and import tables and projects in `.zip` files

Oracle WebCenter Sites Explorer is installed along with WebCenter Sites.

8.2 Connecting to a WebCenter Sites Database

You can use Sites Explorer on any remote Microsoft Windows computer simply by copying the Oracle WebCenter Sites Explorer directory on a computer where WebCenter Sites is installed (`tools/ContentServerExplorer`) to a directory on the remote computer. You then start the Oracle WebCenter Sites Explorer executable file

(`ContentServerExplorer.exe`) and log in to WebCenter Sites by supplying a user name, password, hostname, port, and protocol information.

To connect to a system that is running WebCenter Sites

1. Start Oracle WebCenter Sites Explorer.
2. Choose **File** then **Open WebCenter Sites** to display the Login dialog box.
3. Enter the following values:

   Name: Your WebCenter Sites user name.

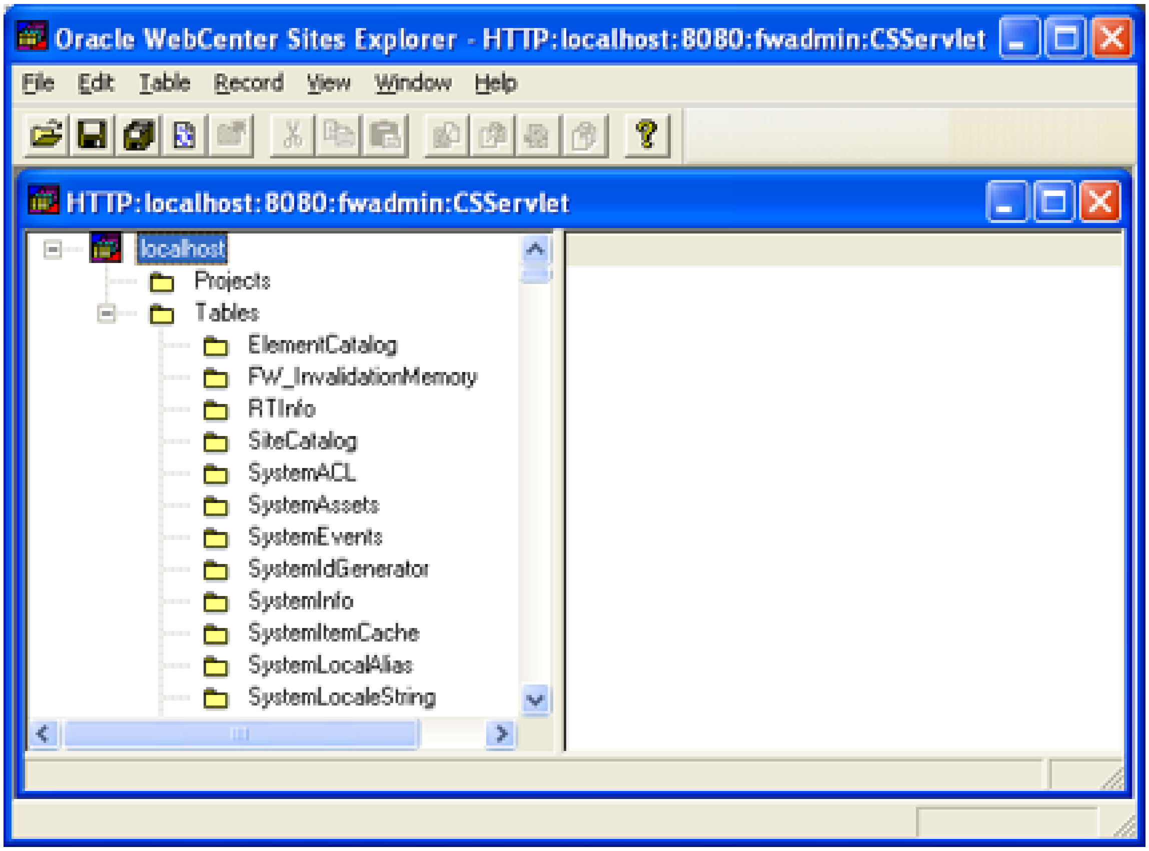
   Password: Your WebCenter Sites password. (Depending on your site security, it may not be necessary to enter a name and password.)

   Host name: The hostname or IP address. You cannot leave this field blank.

   Port: The port number (the default is 80).

   Protocol: Typically, this is HTTP. You may select HTTPS if the web server is running SSL.

   Application server URL path: The type of application server for your site.
4. Click **OK** to log in. The Oracle WebCenter Sites Explorer utility appears:



You may want to create a shortcut on your Windows desktop to Oracle WebCenter Sites Explorer. For instructions about using Oracle WebCenter Sites Explorer, see the online help as well as sections in this manual that describe specific tasks requiring Oracle WebCenter Sites Explorer. For more information on Oracle WebCenter Sites Explorer and its features, see the Oracle WebCenter Sites Explorer online help.

8.3 CatalogMover

You use the CatalogMover tool to export and import WebCenter Sites database tables, including the `ElementCatalog` and `SiteCatalog` tables. For example, you can use CatalogMover to export page elements and content assets to one system, and load the same elements and assets into the database on another system. You can export and import database tables as either HTML files or ZIP files.

You can use CatalogMover through either the Windows interface described in the following sections, or the command line interface described in Section 8.3.7, "Command Line Interface."

> **Note:** In previous versions of WebCenter Sites, tables in the WebCenter Sites database were called *catalogs*. This term still applies to the names of some database tables as well as to the CatalogMover tool itself.

This section includes the following topics:

- Section 8.3.1, "Starting CatalogMover"
- Section 8.3.2, "Connecting to WebCenter Sites"
- Section 8.3.3, "CatalogMover Menu Commands"
- Section 8.3.4, "Catalog Menu"
- Section 8.3.5, "Exporting Tables"
- Section 8.3.6, "Importing Tables"
- Section 8.3.7, "Command Line Interface"

8.3.1 Starting CatalogMover

To start CatalogMover

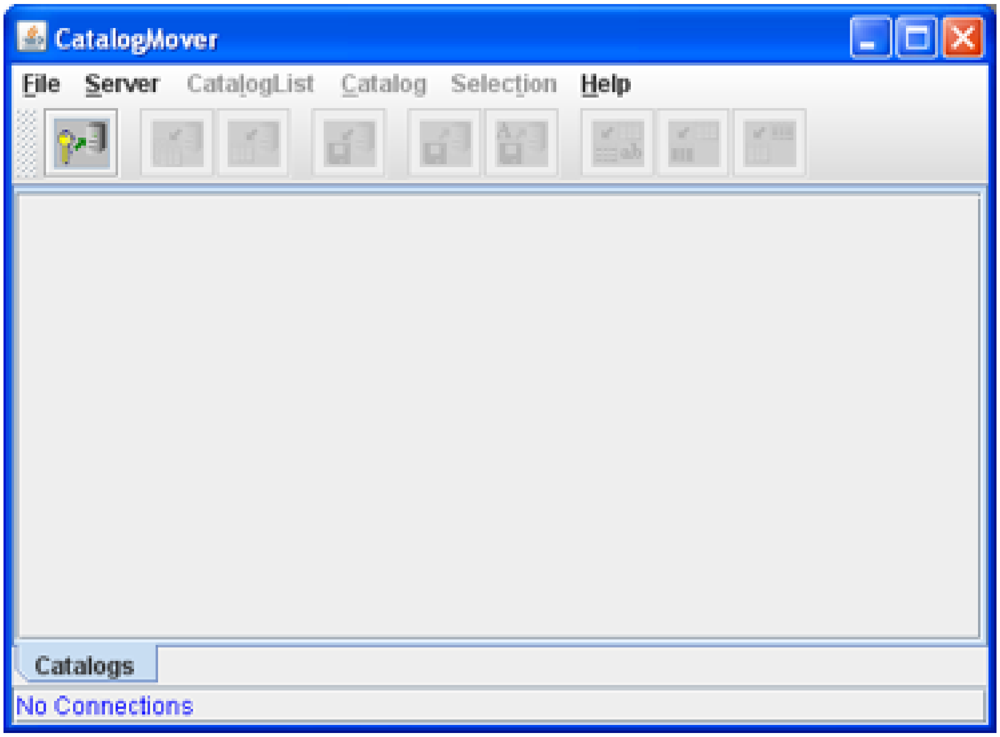
Execute the following scripts at the MS DOS prompt or in a UNIX shell:

- Windows: `catalogmover.bat`
- Solaris: `catalogmover.sh`

The following JAR files must be in the classpath, or be specified by the `-classpath` switch:

```
cs.jar
swingall.jar
commons-logging.jar
cs-core.jar
```

The CatalogMover utility appears:

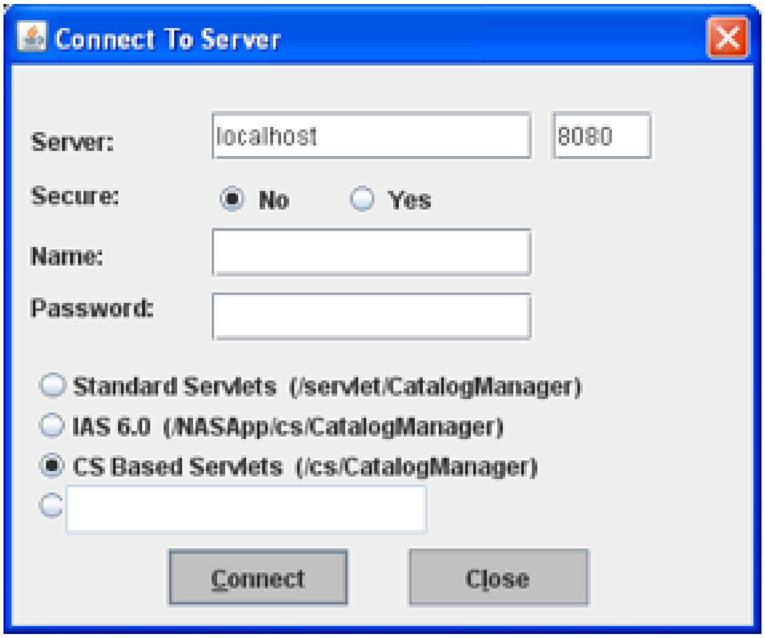


8.3.2 Connecting to WebCenter Sites

Before using CatalogMover, you must first connect to a WebCenter Sites system.

To connect to WebCenter Sites

1. Choose **Server** then **Connect**. The Connect to Server dialog displays.


Connect To Server

Server: localhost 8080

Secure: ● No ○ Yes

Name:

Password:

○ Standard Servlets (/servlet/CatalogManager)

○ IAS 6.0 (/NASApp/cs/CatalogManager)

● CS Based Servlets (/cs/CatalogManager)

○

2. In the **Server** field, enter the name of the HTTP server you want to connect to.

3. For the **Secure** option, select **No** (default port 80) or **Yes** (default port 443). If you are running on a different port than the default, enter the port on which the server is running in the port field.
4. In the **Name** field, enter your user name.
5. In the **Password** field, enter your password.
6. Select one of the following options:
   - **Standard Servlets**: To connect to a system using WebSphere or WebLogic.
   - **IAS 6.0**: To connect to a NAS-App system.
   - **CS Based Servlets**: To connect to the CS-based servlets.
   - **Custom**: To connect to a different application server, enter the following value in the text box (referencing the .sh or .bat script):

     ```
     <ft.approot><ft.cgipath>/catalogmanager.
     ```
7. Click **Connect**.

8.3.3 CatalogMover Menu Commands

CatalogMover includes the following menu commands:

File Menu

- **Exit**: Disconnect from WebCenter Sites and close CatalogMover.

Server Menu

- **Connect**: Display the Connect to Server dialog box.
- **Reconnect**: Display the Connect to Server dialog box and renew the current WebCenter Sites connection.
- **Disconnect**: Disconnect from WebCenter Sites.
- **Purge Temporary Tables**: Purge imported tables before committing.
- **Commit Individual Tables**: Commit imported tables to the database.
- **Normalize Filenames on Export**: Enable CatalogMover's file name normalization behavior, which changes the names of files that are being moved to names that match their corresponding ID numbers. If this feature is not enabled, file names are not altered.

CatalogList Menu

- **Load**: Display a list of all tables in the database.

8.3.4 Catalog Menu

- **Load**: Load into local memory a table from the list. The following figure shows a loaded `ElementCatalog` table:

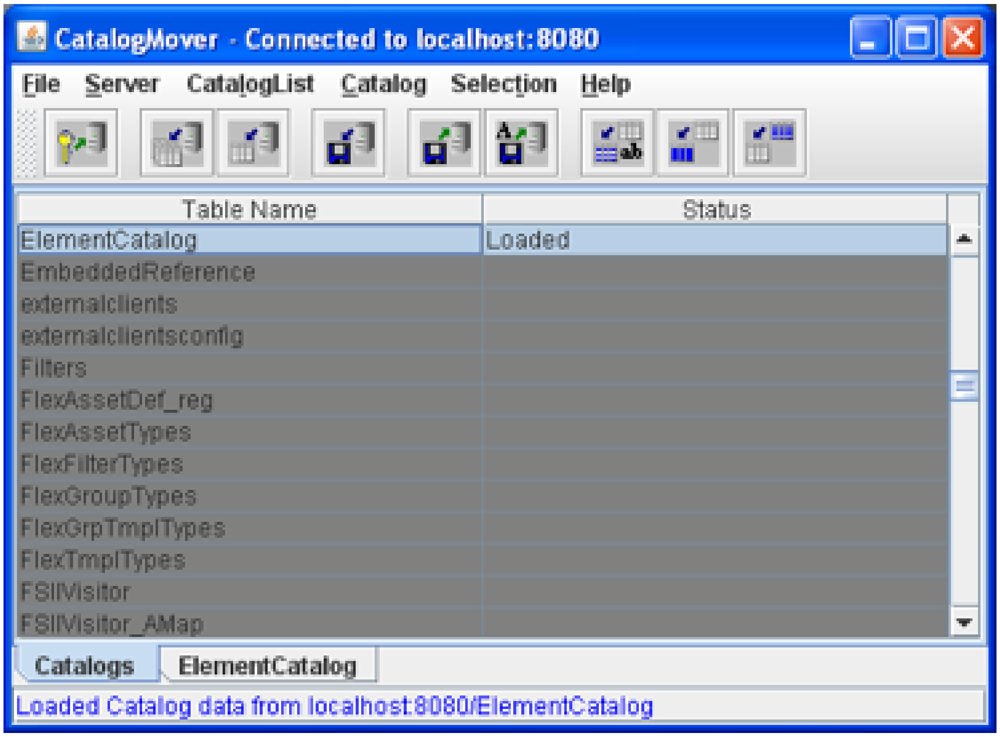


Table Name	Status
ElementCatalog	Loaded
EmbeddedReference	
externalclients	
externalclientsconfig	
Filters	
FlexAssetDef_reg	
FlexAssetTypes	
FlexFilterTypes	
FlexGroupTypes	
FlexGrpTmplTypes	
FlexTmplTypes	
FSIIVisitor	
FSIIVisitor_AMap	

Loaded Catalog data from localhost:8080/ElementCatalog

Click the **Element Catalog** tab to view all rows in the table, and to select specific rows for export.

- **Refresh**: Update the loaded tables from the WebCenter Sites database.
- **Auto Import Catalog(s)**: Import a previously exported ZIP file.
- **Import Catalog**: Import into the local database a table that was exported from another WebCenter Sites database.
- **Export Catalog Rows**: Export the selected rows in the loaded table.

Selection Menu

- **Select All Rows**: Select all rows in the currently displayed table.
- **Deselect All Rows**: Deselect all rows in the currently displayed table.
- **Select Rows By SubString**: Select rows in the currently displayed table by typing a portion of any field value string that uniquely identifies a set of rows.

Help Menu

- **About**: Display version information about the WebCenter Sites installation.

8.3.5 Exporting Tables

Exporting is the process of retrieving table rows and their content from the database and saving them in local HTML files and associated data directories. CatalogMover creates one HTML file per table.

This section includes the following topics:

- Section 8.3.5.1, "Exporting Selected Table Rows"
- Section 8.3.5.2, "Selecting Rows for Export"
- Section 8.3.5.3, "Exporting to a ZIP File"

8.3.5.1 Exporting Selected Table Rows

To export selected table rows

1. Connect to WebCenter Sites as described in Section 8.3.2, "Connecting to WebCenter Sites."
2. Choose **CatalogList** then **Load** to display a list of all tables in the database
3. Choose **Catalog** then **Load** to load a table, and select rows as described in Section 8.3.5.2, "Selecting Rows for Export" below.
4. Choose **Catalog** then **Export Catalog Rows**.

   A dialog box appears prompting you to specify a directory for the HTML file containing the exported rows.
5. Navigate to your directory of choice, and click **Save**.

   CatalogMover exports the selected rows to your selected directory.

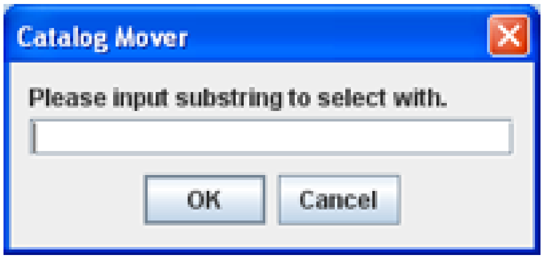
8.3.5.2 Selecting Rows for Export

You can select specific rows for export in a loaded table by clicking on them, or you can search for specific rows by substring.

To search for and select rows according to a substring

1. Choose **Selection** and then **Select Rows By SubString**.

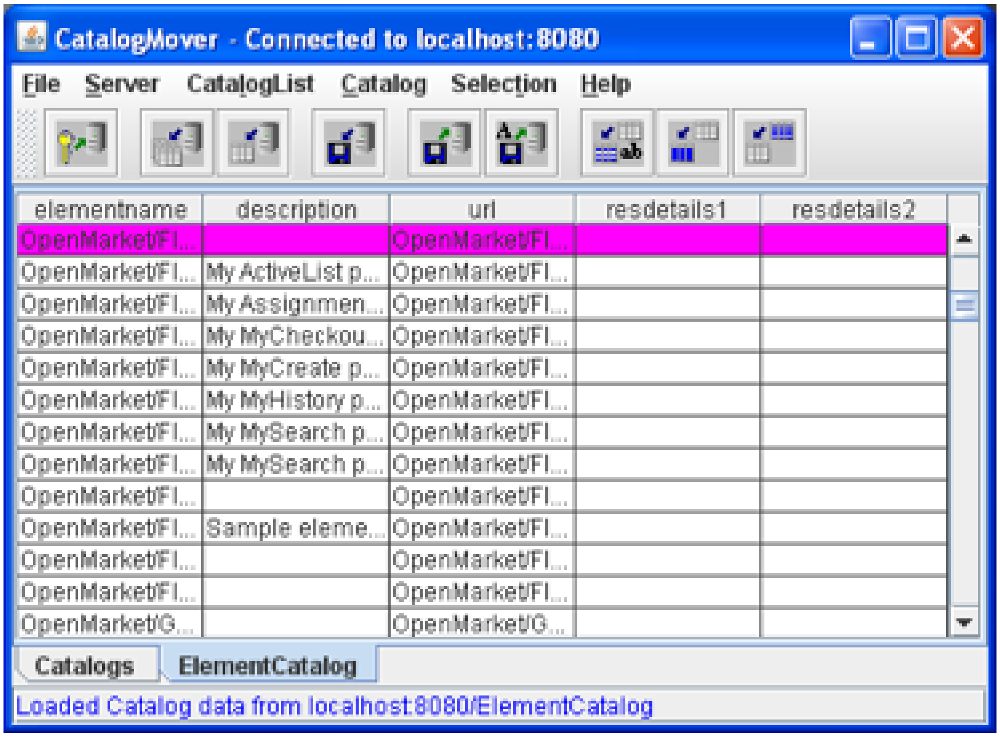
   The following dialog box appears:



Please input substring to select with.

2. In the text field, enter the substring you want to locate. For example, if you wanted to search the ElementCatalog primary key for all rows with *folder* in the element name, enter `folder` and click **OK**.

   CatalogMover searches the table and selects the rows that match your substring query against the primary key for the table, as shown in the following figure:



elementname	description	url	resdetails1	resdetails2
OpenMarket/Fl...		OpenMarket/Fl...		
OpenMarket/Fl...	My ActiveList p...	OpenMarket/Fl...		
OpenMarket/Fl...	My Assignmen...	OpenMarket/Fl...		
OpenMarket/Fl...	My MyCheckou...	OpenMarket/Fl...		
OpenMarket/Fl...	My MyCreate p...	OpenMarket/Fl...		
OpenMarket/Fl...	My MyHistory p...	OpenMarket/Fl...		
OpenMarket/Fl...	My MySearch p...	OpenMarket/Fl...		
OpenMarket/Fl...	My MySearch p...	OpenMarket/Fl...		
OpenMarket/Fl...		OpenMarket/Fl...		
OpenMarket/Fl...	Sample eleme...	OpenMarket/Fl...		
OpenMarket/Fl...		OpenMarket/Fl...		
OpenMarket/Fl...		OpenMarket/Fl...		
OpenMarket/G...		OpenMarket/G...		

Loaded Catalog data from localhost:8080/ElementCatalog

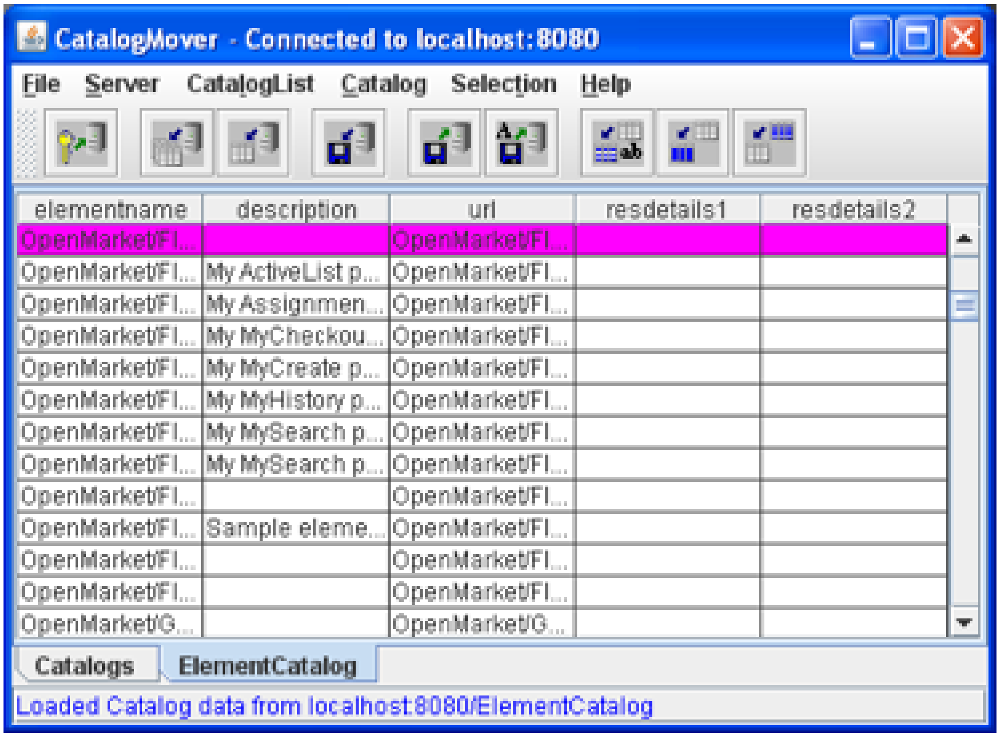
Note: Selecting rows by substring only works for the left-most column in the table. However, you can change column positions so that any column can become the left-most column. To do this, simply click and drag the column header.

8.3.5.3 Exporting to a ZIP File

You can select several rows from several tables and export them to a ZIP file on the local computer from which you are running CatalogMover. Once you create the ZIP file, you can import the contents of the file into server tables.

To export a ZIP file with CatalogMover

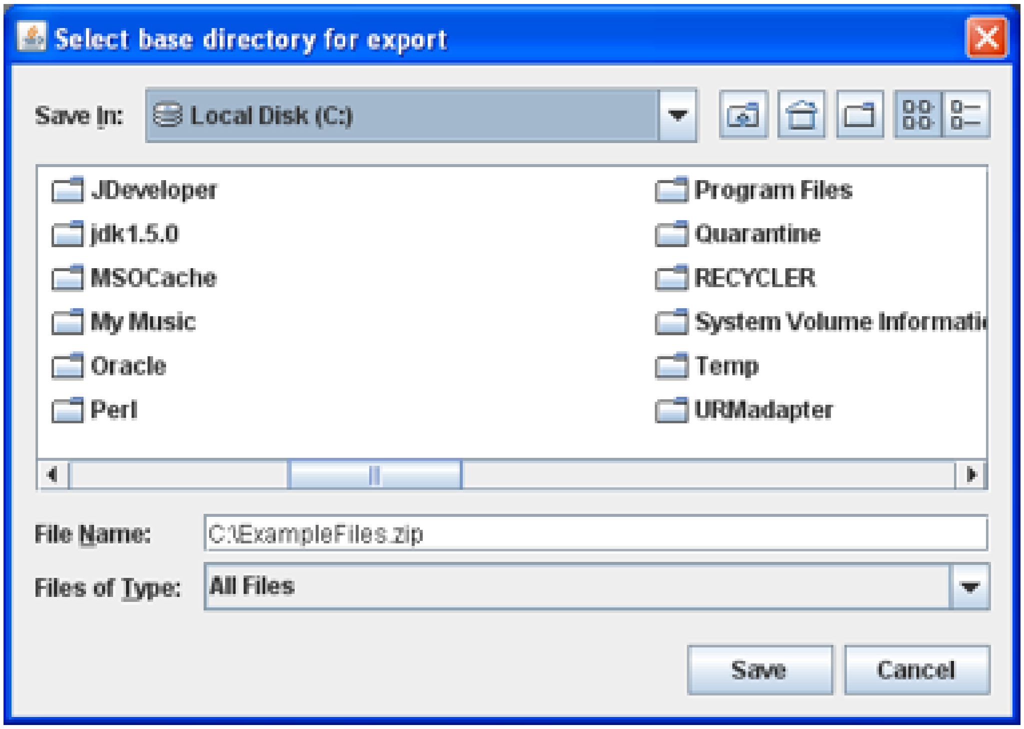
1. Choose **CatalogList** then **Load** to display a list of all tables in the database.
2. Choose **Catalog** then **Load** to load a table, and select rows as described in Section 8.3.5.2, "Selecting Rows for Export."



elementname	description	url	resdetails1	resdetails2
OpenMarket/Fl...		OpenMarket/Fl...		
OpenMarket/Fl...	My ActiveList p...	OpenMarket/Fl...		
OpenMarket/Fl...	My Assignmen...	OpenMarket/Fl...		
OpenMarket/Fl...	My MyCheckou...	OpenMarket/Fl...		
OpenMarket/Fl...	My MyCreate p...	OpenMarket/Fl...		
OpenMarket/Fl...	My MyHistory p...	OpenMarket/Fl...		
OpenMarket/Fl...	My MySearch p...	OpenMarket/Fl...		
OpenMarket/Fl...	My MySearch p...	OpenMarket/Fl...		
OpenMarket/Fl...		OpenMarket/Fl...		
OpenMarket/Fl...	Sample eleme...	OpenMarket/Fl...		
OpenMarket/Fl...		OpenMarket/Fl...		
OpenMarket/Fl...		OpenMarket/Fl...		
OpenMarket/G...		OpenMarket/G...		

Loaded Catalog data from localhost:8080/ElementCatalog

3. Choose **Catalog** then **Export Catalog Rows**. The following dialog box appears:



Save In: Local Disk (C:)

File Name: C:\ExampleFiles.zip

Files of Type: All Files

4. Navigate to the directory where you want to save the ZIP file.

5. In the **File Name** field, enter a name for the files and type a ZIP file extension.

6. Click **Save**. The rows you selected from all of the tables are exported to a ZIP file in the directory you chose.

8.3.6 Importing Tables

Importing is the process of sending locally stored HTML files and the associated data to the server. You can select a particular HTML file to import, or you can choose to import all HTML files.

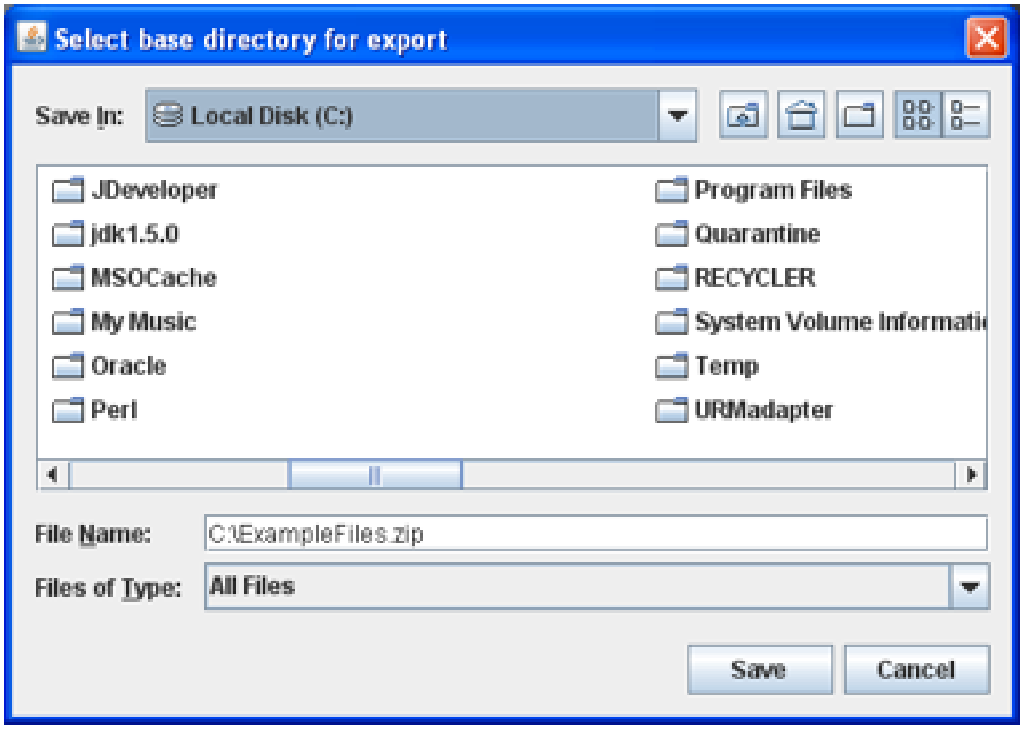
This section includes the following topics:

- Section 8.3.6.1, "Importing HTML Files Previously Exported"
- Section 8.3.6.2, "Importing a Previously Exported ZIP File"
- Section 8.3.6.3, "Merging Existing CatalogMover Files"
- Section 8.3.6.4, "Replacing Existing CatalogMover Files"

8.3.6.1 Importing HTML Files Previously Exported

To import HTML files that have been previously exported from another table

1. Connect to the WebCenter Sites installation you want to import the HTML files to, as described in Section 8.3.2, "Connecting to WebCenter Sites."
2. Choose **CatalogList** then **Load** to display a list of all tables in the database.
3. Choose **Catalog** then **Import Catalog**.
4. Navigate to the HTML file containing the previously exported table rows.
5. Select the HTML file and click **Open**. The following dialog box appears:



File Name: C:\ExampleFiles.zip

Files of Type: All Files

6. If you are importing new table rows that do not currently exist, enter the information in the **Catalog Data Directory** and the **Catalog ACL List** fields.

   If you are replacing existing table rows with the imported table rows, leave these fields blank.

7. Click **OK**. The table rows contained in the previously export HTML file are imported into the WebCenter Sites database to which you are connected.

   A dialog box appears, listing the table rows that were imported.

> **Note:** If you import tables that do not exist on the server to which you are connected, the new tables are automatically created as they are imported.

8.3.6.2 Importing a Previously Exported ZIP File

You can import table rows stored in an exported ZIP file to your server using CatalogMover.

To import a previously exported ZIP file

1. While connected to your database, choose **Catalog** then **Auto Import Catalogs**.
2. In the resulting dialog box, navigate to the directory where you previously exported the table rows. To see the ZIP file, change the **Files by Type** menu to **all files**.
3. Select the ZIP file and click **Save**. The rows contained in the ZIP file are automatically imported to your database.

8.3.6.3 Merging Existing CatalogMover Files

To merge CatalogMover files

1. Connect to the WebCenter Sites installation you want to import the HTML files to, as described in Section 8.3.2, "Connecting to WebCenter Sites."
2. Choose **CatalogList** then **Load** to display a list of all tables in the database.
3. Choose **Catalog** then **Load** to load a table, and select the rows that you want to merge into another file, as described in Section 8.3.5.2, "Selecting Rows for Export."
4. Choose **Catalog** then **Export Catalog Rows.**
5. Navigate to the HTML file you want to merge the rows with. Click **Save**. The Overwrite dialog box displays.
6. Click **Update existing exported data**. CatalogMover merges the exported rows into the HTML file you selected.

8.3.6.4 Replacing Existing CatalogMover Files

To replace CatalogMover files

1. Connect to the WebCenter Sites installation you want to import the HTML files to, as described in Section 8.3.2, "Connecting to WebCenter Sites."
2. Choose **CatalogList** then **Load** to display a list of all tables in the database.
3. Choose **Catalog** then **load** to load a table, and select the rows that you want to merge into another file, as described in Section 8.3.5.2, "Selecting Rows for Export."
4. Choose **Catalog** then **Export Catalog Rows**.
5. Navigate to the HTML file you want to merge the rows with. Click **Save**. The Overwrite dialog box appears

6. Click **Replace existing exported data**. CatalogMover replaces rows in the HTML file you selected with the exported rows.

8.3.7 Command Line Interface

The following parameters allow CatalogMover to perform functions without displaying a GUI. The parameter is followed by a space followed by the value:

Table 8–1 Command Line Interface Parameters

Parameters	Description
`-h`	display command line parameters
`-u username`	username
`-p password`	password
`-s servername`	servername to connect

`-b baseurl`	base URL: either
	`http://($host)/cgi-bin/gx.cgi/AppLogic+FTCatalogManager (NAS)`
	or
	`http://($host)/servlet/CatalogManager(WebLogic)`

`-t table`	table name: Used when exporting to designate tables to export, use multiple -t parameters to export multiple tables
`-x function`	function to perform: Legal values are `import`, `import_all`, `export`, `export_all`
`-d directory`	directory: When exporting, directory to contain exported tables. When importing all, directory containing all tables to import.
`-f filename`	file containing table to import: Can either be an HTML file or a ZIP file generated by export.
`-c directory`	upload directory to be used if creating a table
`-a aclone,acltwo,...`	ACL list: Comma-separated list of ACLs to be used if creating a table

8.4 Property Editor

The Property Editor tool provides an easy-to-use Windows interface that lets you view, modify, and add properties in the WebCenter Sites `futuretense.ini` file.

This section includes the following topics:

- Section 8.4.1, "Starting the Property Editor"
- Section 8.4.2, "Setting Properties"
- Section 8.4.3, "Merging Property Files"

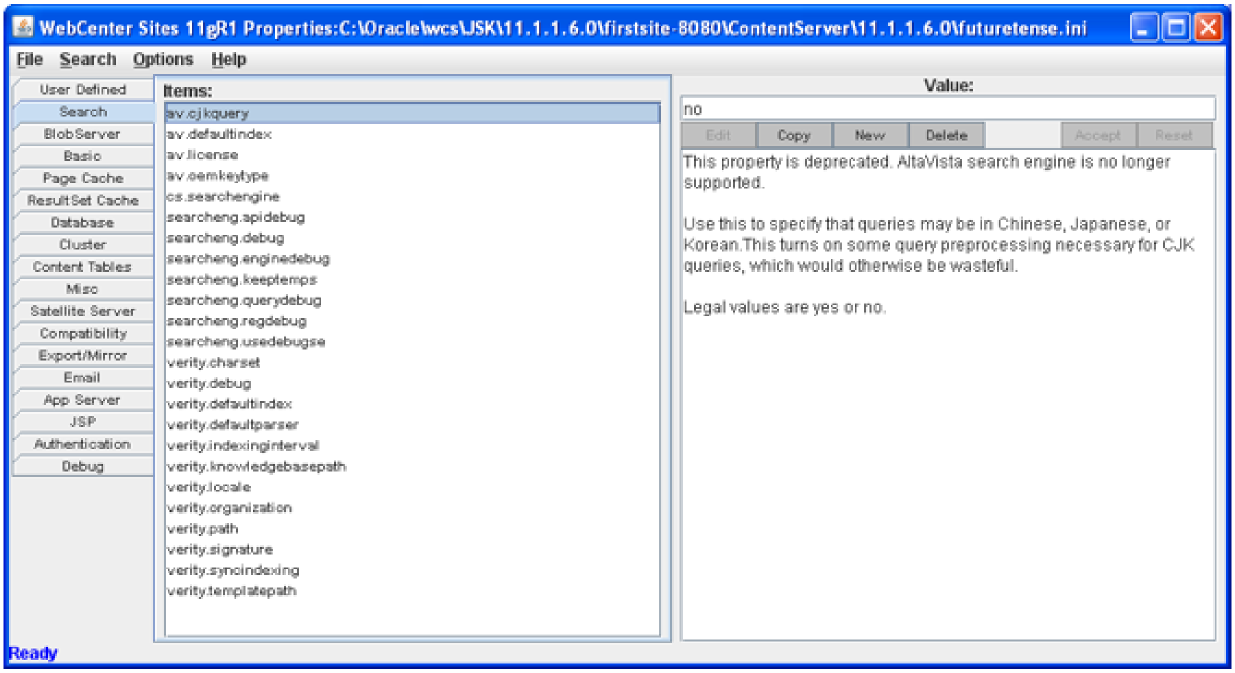
8.4.1 Starting the Property Editor

To start the Property Editor

Run the following scripts:

- On Windows NT: `propeditor.bat`
- On Solaris: `propeditor.sh`

The Property Editor appears:



av.cjkquery
av.defaultindex
av.license
av.oemkeytype
cs.searchengine
searcheng.apidebug
searcheng.debug
searcheng.enginedebug
searcheng.keeptemps
searcheng.querydebug
searcheng.regdebug
searcheng.usedebugse
verity.charset
verity.debug
verity.defaultindex
verity.defaultparser
verity.indexinginterval
verity.knowledgebasepath
verity.locale
verity.organization
verity.path
verity.signature
verity.syncindexing
verity.templatepath

no

This property is deprecated. AltaVista search engine is no longer supported.

Use this to specify that queries may be in Chinese, Japanese, or Korean.This turns on some query preprocessing necessary for CJK queries, which would otherwise be wasteful.

Legal values are yes or no.

The Property Editor displays properties in functional groups, such as Database and Caching, on the left side of the window.

The **Items** pane lists the properties in the selected functional group.

The **Value** pane lists the current value for the selected item, a brief description of the item, and the acceptable values for it.

> **Note:** The `futuretense.ini` file contains a release number string, `ft.version`, which contains a value such as 4.0.0. that is set by WebCenter Sites.
>
> Do not modify this property, it is for reference only.

8.4.2 Setting Properties

To set WebCenter Sites properties on your system

1. If necessary, start the Property Editor.
2. Choose **File**, then **Search** and open the `.ini` file you want to edit.
3. Select a properties group from the tabs on the left side of the window. The Property Editor displays the properties in the **Items:** pane.
4. Select a property in the **Items:** pane. The Property Editor displays the current property value and a brief description in the **Values:** pane.
5. In the **Values:** pane, enter the new value in the text field.
6. In the **Values:** pane, click the **Accept** button.
7. Repeat steps 3 through 6 for all the properties you want to change.
8. When you finish, choose **File**, then **Save** to save your changes.

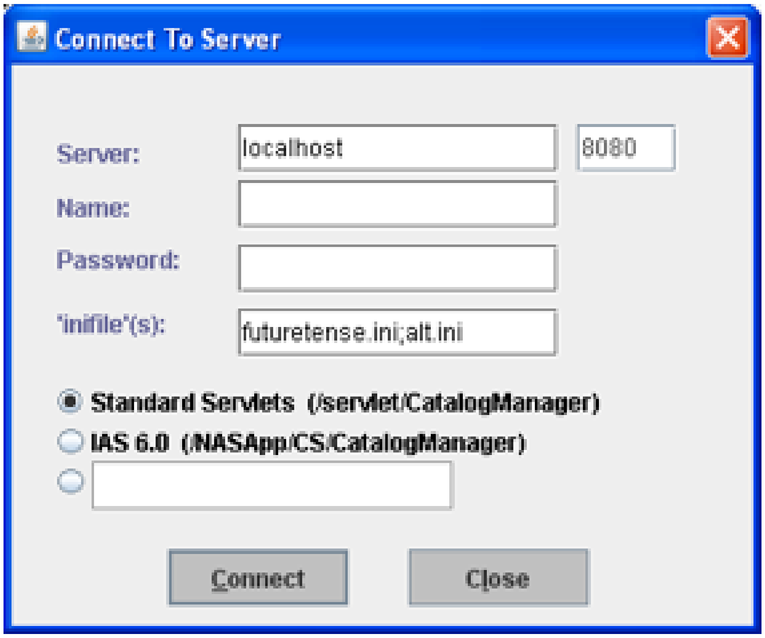
9. Click **OK** in the confirmation message box.
10. Choose File, then Save to save your changes and close the Property Editor.
11. Stop and restart the application server to apply the changes.

8.4.3 Merging Property Files

You ordinarily use the Property Editor to modify the `futuretense.ini` property file. You can also add properties to your property file from the property file on the server to which you are connecting. If you have the same properties with different values defined in multiple INI files, WebCenter Sites uses the values in the last property file that it loads.

To merge a property file with the `futuretense.ini` file, enter the names of the two property files separated by a semicolon in the **'inifile'(s)** field when you connect to WebCenter Sites. If you do not want to merge property files, leave this field blank.

For example, the following Connect to Server dialog box shows a merge between `futuretense.ini` and `alt.ini`:



Server: localhost 8080

Name:

Password:

'inifile'(s): futuretense.ini;alt.ini

● Standard Servlets (/servlet/CatalogManager)

○ IAS 6.0 (/NASApp/CS/CatalogManager)

○

8.5 XMLPost

The XMLPost utility imports data into the WebCenter Sites database. This utility is based on the WebCenter Sites `FormPoster` Java class and it is delivered with the WebCenter Sites base product. It imports data using the HTTP POST protocol.

To import assets, you use XMLPost with posting elements that are delivered with WebCenter Sites. For information about using XMLPost to import assets, see the following chapters:

- Chapter 20, "Importing Assets of Any Type"
- Chapter 21, "Importing Flex Assets"

9

Sessions and Cookies

This chapter explains how to use XML tags to manage sessions and cookies.

This chapter contains the following sections:

9.1 Understanding Sessions

Imagine a website containing two pages: `main` and `water`. Suppose a visitor sees `main` first and then moves on to `water`. HTTP is a stateless protocol. So, if a typical web server is managing this site, any knowledge gathered at `main` is lost when the visitor browses over to `water`. In other words, `water` cannot take advantage of any information that the visitor might have provided at `main`.

To get around this limitation, application servers detect when a visitor first enters a website. At that point, the application server starts a **session** for this visitor. In the preceding example, when the visitor requests the `main` page, the application server starts a session. The website designer can use `main` to gather information about the visitor and store that information in **session variables**. The information in session variables is available to all subsequent pages. So, for example, if Bob provides his age to `main`, and `main`'s designer wrote the age to a session variable, then `water` could easily access Bob's age.

Session variables contain values available for the duration of the session. When the session ends, the application server destroys the session variables associated with that session. Each session variable consumes memory on the application server, so creating unnecessary session variables can hurt performance.

WebCenter Sites automatically creates some session variables; the website developer can optionally create others.

The application server can maintain sessions on a cluster.

9.2 Session Lifetime

A session begins when a visitor first hits your website. The session ends when any of the following happens:

- The visitor terminates his browser.
- The session has timed out. The `cs.timeout` property is used by WebCenter Sites to set the session timeout value in the application server. If this property is set to 300, then a user session becomes invalid in 300 seconds, or 5 minutes.
- The system administrator stops the application server.

This section contains the following topics:

- Section 9.2.1, "Session Variables Maintained by WebCenter Sites"
- Section 9.2.2, "Logging In and Logging Out"

9.2.1 Session Variables Maintained by WebCenter Sites

Upon creating a session, WebCenter Sites automatically creates the following session variables:

Table 9–1 Session Variables

Session Variable	What it Holds
`SessionVariables.currentUser`	The `id` of the visitor logged in.
`SessionVariables.currentAcl`	The comma-separated list of all ACLs to which this visitor belongs. If the visitor has not explicitly logged in, the default ACL is `Browser`.
`SessionVariables.username`	The username under which this visitor is logged in. If the visitor has not explicitly logged in, the default username is `DefaultReader`.
`SessionVariables.iniFile`	The name of the file containing WebCenter Sites properties.

9.2.2 Logging In and Logging Out

When a visitor first hits the site, WebCenter Sites creates a session and implicitly logs in the visitor as `DefaultReader`. During the session, if the visitor explicitly logs in, WebCenter Sites automatically updates the values of `SessionVariables.currentUser`, `SessionVariables.currentAcl`, and `SessionVariables.username`. Logging in does not affect the values of any other session variables. In other words, if your pages create session variables prior to a login, then those values are still valid after the login. When a visitor explicitly logs out, the WebCenter Sites-generated session variables automatically revert to the values they held prior to login. For example, consider the following sequence:

1. A visitor first hits a page, so the value of `SessionVariables.username` is `DefaultReader`.
2. The visitor logs in as marilyn, so the value of `SessionVariables.username` is `marilyn`.
3. If marilyn logs out, the value of `SessionVariables.username` reverts to `DefaultReader`.

To trigger a logout, you call the `<CATALOGMANAGER>` tag with the `ftcmd=logout` modifier. When issuing this tag, you can optionally supply the `killsession` modifier, which destroys the current session. You can then create a new session by invoking the `<CATALOGMANAGER>` tag with the `ftcmd=login` modifier.

9.3 Sessions Example

Here's a simple session example, consisting of three very short elements:

Table 9–2 Sessions Example

Element	What it Does
`FeelingsForm`	Asks visitors to pick their current mood.
`SetFeelings`	Assigns the current mood to a session variable.
`Meat`	Evaluates the session variable.

This section contains the following topics:

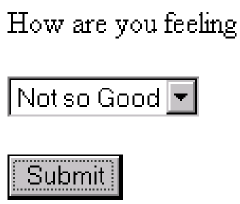
- Section 9.3.1, "FeelingsForm Element"
- Section 9.3.2, "SetFeeling Element"
- Section 9.3.3, "Meat Element"

9.3.1 FeelingsForm Element

The feelings form doesn't really involve sessions or variables; this element merely generates a form. The visitor's chosen mood is passed to the `SetFeeling` element:

```
<form action="ContentServer" method="post">
   <input type="hidden" name="pagename"
          value="CSGuide/Sessions/SetFeelings"/>

 <P>How are you feeling right now?</P>
 <P>
 <select name="Feeling" size="1">
    <option>Good</option>
    <option>Not so Good</option>
 </select>
 </P>

 <P><input type="submit" name="doit" value="Submit"/></P>
</form>
```

The resulting page looks like the following:



How are you feeling right now?

9.3.2 SetFeeling Element

Upon clicking the Submit button, the visitor is transported to `SetFeeling`. This element assigns the visitor's mood to a new session variable named `CurrentFeeling`.

```
<SETSSVAR NAME="CurrentFeeling" VALUE="Variables.Feeling"/>

<P>Welcome to our site.</P>

<P>Now proceed to
<A href="ContentServer?pagename=CSGuide/Sessions/Meat">
some meaty content.
</A></P>
```

The resulting page looks as follows:

```
Welcome to our site.
Now proceed to some meaty content.
```

If an element in this application asked the visitor to login, WebCenter Sites would have automatically set the `username` session variable to the visitor's login name. In that case, you could have personalized the welcome message in `SetFeeling` as follows:

```
<P>Welcome to our site, <CSVAR NAME="SessionVariables.username"/>
</P>
```

9.3.3 Meat Element

Upon clicking some meaty content, the visitor is transported to the `Meat` page. This page evaluates the session variable:

```
<IF COND="SessionVariables.CurrentFeeling=Good">
 <THEN>
     <P>Sessions are happiness.</P>
 </THEN>
 <ELSE>
     <P>Don't let sessions get you down.</P>
 </ELSE>
</IF>
```

A visitor in a not so good mood sees:

```
Don't let sessions get you down.
```

Notice how `CurrentFeeling` was available to `Meat`. In fact, `CurrentFeeling` is available to any other elements in the session.

9.4 What Is a Cookie?

A **cookie** is a string that your application writes to the visitor's browser. A cookie stores information about visitors that lasts between sessions. The visitor's browser writes this string to a special cookie file on the visitor's disk. When that visitor returns to your website, the visitor's browser sends a copy of the cookie back to the web server that set it. Once a cookie has been created, it is available as a variable to elements on a page.

For example, your application might store the visitor's favorite sports team in a cookie. Then, when the visitor returns, your application could retrieve the cookie and use its information to display the team logo in a banner.

When cookies are no longer needed, you can delete them.

This section contains the following topics:

- Section 9.4.1, "CookieServer"

- Section 9.4.2, "Cookie Tags"

9.4.1 CookieServer

CookieServer is a servlet that sets cookies for you. You access CookieServer by creating cookies with the `satellite.cookie` tag.

9.4.2 Cookie Tags

WebCenter Sites offers two tags for managing cookies:

Table 9–3 Cookie Tags

Tag	Use
`satellite.cookie`	Sets a cookie on the client's browser.
`REMOVECOOKIE`	Deletes a cookie from the client's browser.

There is no special tag to obtain the value of a cookie. Instead, when a visitor returns to the website, WebCenter Sites loads the value of the cookie as a regular variable.

When creating a cookie (by calling `satellite.cookie`), you can specify the following attributes:

Table 9–4 Cookie Attributes

Attribute	Value
`name`	Name of the cookie. This also serves as the name of the incoming variable containing the value of the cookie. **Important:** Cookies in the WebCenter Sites page context are treated as variables. Therefore, when a cookie and an asset attribute share the same name, they are treated as the same variable.
`expiration`	Time in seconds after which the cookie no longer is sent to the web server.
`security`	Optionally set security on the cookie.
`URL`	Restrict that the cookie only be sent on this URL
`Domain`	Restrict that the cookie only be sent to URLs in the specified domain.

Because they feel that cookies are a security threat, some visitors configure their browsers to reject cookies. If the information in the cookie is critical, your application must be prepared for this.

You must set or remove cookies before using any tags that stream content back to the visitor's browser. You must set or remove cookies even before the `<HTML>` tag.

9.5 Cookie Example

This example consists of several very short elements:

Table 9–5 Cookie Elements

Element	What it Does
`Start`	Determines whether a cookie is set. If cookie is set, call `DisplayWelcome`. If cookie is not set, call `GetColorPreference`.
`ColorForm`	Displays a form that asks visitor to pick her favorite color.
`CreateCookie`	Creates a cookie on this visitor's browser. Then, redirects visitor to `DisplayWelcome`.
`DisplayWelcome`	Displays a simple welcome message in the visitor's favorite color.

This section contains the following topics:

- Section 9.5.1, "Start.xml"
- Section 9.5.2, "ColorForm"
- Section 9.5.3, "CreateCookie"
- Section 9.5.4, "DisplayWelcome"
- Section 9.5.5, "Running the Cookie Example"

9.5.1 Start.xml

The `Start.xml` element determines whether the cookie has already been set. If the cookie has been set, WebCenter Sites stores its value inside a regular variable named `Variables.ColorCookie`. The code for `Start.xml` is as follows:

```
<IF COND="IsVariable.ColorCookie=true">
 <THEN>
    <CALLELEMENT NAME="CSGuide/Sessions/DisplayWelcome"/>
 </THEN>
 <ELSE>
    <CALLELEMENT NAME="CSGuide/Sessions/ColorForm"/>
 </ELSE>
</IF>
```

9.5.2 ColorForm

The `ColorForm.xml` element displays some an HTML form to gather the visitor's favorite color. The code for `ColorForm.xml` is as follows:

```
<form action="ContentServer" method="post">
   <input type="hidden" name="pagename" value="CSGuide/Sessions/CreateCookie"/>

 <P>What is your favorite color?</P>
 <P>
 <select name="FavoriteColor" size="1">
    <option>Red</option>
    <option>Green</option>
    <option>Blue</option>
 </select>
 </P>

<P><input type="submit" name="doit" value="Submit"/></P>
</form>
```

9.5.3 CreateCookie

The `CreateCookie.xml` element sends a cookie named `ColorCookie` to the visitor's browser. If the visitor has disabled cookies, the browser ignores the request to set a cookie. If the visitor has enabled cookies (the default), the browser writes the cookie to this system's cookie file.

The following is the code for `CreateCookie.xml`:

```
<satellite.cookie NAME="ColorCookie" VALUE="Variables.FavoriteColor"
TIMEOUT="31536000" SECURE="false"/>

<CALLELEMENT NAME="CSGuide/Sessions/DisplayWelcome"/>
```

The preceding code sets the value of the cookie to the visitor's favorite color. This cookie lasts for one year (31,536,000 seconds).

9.5.4 DisplayWelcome

By the time DisplayWelcome is called, the cookie has been set. The following code uses the value of the cookie to display a welcome message in the visitor's favorite color.

```
<H1><font color="Variables.ColorCookie"
     REPLACEALL="Variables.ColorCookie">
Displaying a Friendly Welcome.
</font></H1>
```

9.5.5 Running the Cookie Example

To run the cookie example, use your browser to go to the following pagename:

```
CSGuide/Sessions/Start
```

The first time you run this example, all four elements execute. After the first time, only `Start` and `DisplayWelcome` execute.

9.6 Tips and Tricks

The following suggestions might be useful:

- In a cluster, session state must be replicated across cluster members. In a cluster, try to keep session size to a minimum; don't store more than 2 Kilobytes of session data per client.
- Determine reasonable session timeout values. Setting timeouts that are too large tie up system resources. Setting them too small forces visitors to log in with annoying frequency.

9.7 Satellite Server Session Tracking

websites that present personalized content to visitors must track sessions. WebCenter Sites and Satellite Server both track sessions. Both set cookies in the visitor's browser; thus, two cookies (rather than one) each independently track a session. This redundancy is useful if a Satellite Server goes down; when a Satellite Server goes down, the WebCenter Sites session is maintained.

Though Satellite Server will only serve session-specific pagelets back to the person who originally requested them, explicitly flushing session-specific information on user logout is a wise way to conserve space in the Satellite Server cache.

The following sections describe how to flush session information from Satellite Server.

This section contains the following topics:

- Section 9.7.1, "Flushing a Session Via URL"
- Section 9.7.2, "Flushing Current Session Information"
- Section 9.7.3, "Flushing Other Session Information"

9.7.1 Flushing a Session Via URL

You can flush all data pertaining to a particular session. You do this from WebCenter Sites by posting a form to a URL in the following format:

```
https://host:port/servlet/
FlushServer?reset=true&username=username&password=password&
ssid=sessionID
```

where:

Table 9–6 Session Parameters

Parameter	Value
host	Specify the name of the Satellite Server host whose cache is to be flushed
port	Specify 80 (the default) unless you re-configured Resin to run on a different port.
username	Specify the value assigned to the username property.
password	Use the value assigned to the password property.
sessionID	Specify the session ID (the one maintained by WebCenter Sites, and not by Satellite Server) representing the session to be removed.

9.7.2 Flushing Current Session Information

To flush the information for the Satellite Server session that you are currently in, use the `FlushServer` URL with the current session's ID. The current session ID (`ssid`) is stored in a session variable with a name that is dependent upon your application server. You can see this name by looking at the session variable `HTTP_COOKIE`.

The following java code flushes the information for the current session:

```
String value;
String name = "WebLogicSession";
value = ics.GetVar(name);

String sFlushSessionUrl ="http://mysatellite:80/servlet/
FlushServer?username=ftuser&password=ftuser&
reset=true&ssid=" + value;"

String sSatTest1Results = Utilities.readURL(sFlushSessionUrl);
```

9.7.3 Flushing Other Session Information

To flush information from a session other than the one you are in

1. Add the following tag to the container page that contains the pagelets that you want to flush:

   ```
   <satellite.page
   pagename="QA/Satellite/Functional/xml/pagelet4"
   cachecontrol="session:0:00:00 */*/*"/>
   ```

   The `cachecontrol` value of `"session:0:00:00 */*/*"` means that every session that requests this page creates a pagelet that can only be viewed by subsequent requests by that session. Once the session for a given page expires, that page cannot be viewed again. The container page will expire from the cache at midnight each day.

2. After setting the cachecontrol parameter for the container page, use the Inventory servlet with the keys parameter to get its session ID (`ssid`). The `ssid` is the string that precedes the protocol and server name. For example, if the Inventory servlet displays:

   ```
   OuCOTrh9yporWfgu8Uthttp://myserver:80/servlet/
   ContentServer?pagename=QA/Satellite/Functional/xml/pagelet4
   ```

   then the `ssid` is `OuCOTrh9yporWfgu8U`.

3. Flush information from the session by using the `ssid` you found with the FlushServer URL. For example:

   ```
   http://myserver:80/servlet/
   FlushServer?username=ftuser&password=ftuser&reset=true&
   ssid=OuCOTrh9yporWfgu8U
   ```

> **Note:** You should have session affinity enabled if you want to flush information from a session other than the one you are in.

10

Error Logging and Debugging

WebCenter Sites provides several options for logging error messages and debugging source code. This chapter gives you information about general error logging and debugging techniques that apply throughout the WebCenter Sites development environment.

WebCenter Sites can log its activity in a log file, which in a new installation, is named `sites.log`, located in the `logs` folder. The type and volume of information that is written to the log file is controlled by the loggers that you choose to enable or define. See Section 10.1, "Logging to the WebCenter Sites Log File" for information about loggers.

WebCenter Sites also has a reserved variable that is used by JSP and XML tags for returning an error code if the tag did not successfully complete its task. See Section 10.2, "Using Error Codes with Tags" for more information.

This chapter contains the following sections:

10.1 Logging to the WebCenter Sites Log File

Logging is based on either the commons-logging framework (using the `commons-logging.properties` file) or the Apache log4j framework.

In new WebCenter Sites installations, log4j is the WebCenter Sites logging system. When log4j is set up, the `log4j.properties` file is created to specify how information must be logged and which information will be logged. In addition, the WebCenter Sites Admin interface provides the **Configure log4j** tool in the **System Tools** node, on the Admin tab. Using **Configure log4**j, you can view current loggers directly from the Admin interface. You can also dynamically add new loggers and change logger levels. Changes will persist upon system restart if you copy the text version of the loggers from the interface to the `log4j.properties` file. For more information about the **Configure log4j** tool, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

For information about the `log4j.properties` and `commons-logging.properties` files, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*

To define your own loggers or write your own messages to WebCenter Sites's log file, use the `ics:logmsg` tag. The following example writes a warning message to WebCenter Sites's log file.

```
<ics:logmsg msg="This is a warning message"
  name="com.fatwire.logging.cs.jsp" severity="warn"/>
```

For more information about ics:logmsg, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

Note: It is recommended that you set loggers to a level that agrees with the type of system on which logging is implemented. On development and content management systems, logging levels can be set to a greater severity (such as INFO or DEBUG), which provides a large amount of information. On delivery systems, loggers can be either disabled or set to low severity (WARN or ERROR) to avoid performance setbacks and making system information available on a publicly accessed environment.

10.2 Using Error Codes with Tags

WebCenter Sites has a reserved variable named Variables.errno which most JSP and XML tags use for returning an error code (generally referred to as an **errno**) if the tag did not successfully complete its task.

For example, the <CALLELEMENT> XML tag sets Variables.errno as follows:

- -10 if you specified a nonexistent element.
- -12 if you specified an existing element that WebCenter Sites could not evaluate.

On success, <CALLELEMENT> does not modify the value of Variables.errno.

Note: For revision tracking operations, the reserved variable named Variable.errdetails provides additional information about the error.

You typically use the following strategy with tags that use Variables.errno:

1. Initialize Variables.errno to 0 before calling the tag.
2. Call the tag.
3. Evaluate Variables.errno.

Tag Examples Using Error Codes

For example, the following code performs all three steps:

```
<SETVAR NAME="errno" VALUE="0"/>
<SETCOUNTER NAME="pi" VALUE="3.14159"/>
  <IF COND="Variables.errno=-501">
    <THEN>
      <p>Bad value of pi</p>
    </THEN>
  </IF>
```

Running this code yields the following HTML because SETCOUNTER cannot handle floating-point values:

```
<p>Bad value of pi</p>
```

The ASSET, RENDER, and SITEPLAN tags clear errno before they execute. You do not need to set errno to 0 when you use these tags. For example, after you use an ASSET tag, just check the value of errno to determine whether it has changed:

```
<ASSET.LOAD NAME="topArticle" TYPE="Article"
OBJECTID="Variables.cid"/>
  <IF COND="IsError.Variables.errno=false">
    <THEN>
      <ASSET.CHILDREN NAME="topArticle" LIST="listOfChildren"/>
    </THEN>
  </IF>
```

At the end of template elements, you can include error checking code such as this:

```
<IF rendermode="preview">
  <THEN>
    <IF COND="IsError.Variable.errno=true">
      <THEN>
        <FONT COLOR="#FF0000">
         Error <CSVAR NAME="Variables.errno"/>
         while rendering <CSVAR NAME="pagename"/>
         with asset ID <CSVAR NAME ="Variables.cid"/>.
        </FONT>
      </THEN>
    </IF>
  </THEN>
</IF>
```

Java Interface

After making calls to WebCenter Sites, the String variable `errno` can be retrieved and tested for success or failure. Here's an example:

```
cs.clearErrno();

IList rslt = cs.SelectTo(SYSTEMUSERS_TABLE, ALL_FIELDS, USERNAME,
             null, NO_LIMIT, null, CACHE_RESULTS, errstr);

errno = cs.GetVar("errno");

if (errno.compareTo(ERRNO_SUCCESS) == 0)
    {
     ...
```

Error Number Rules

Error numbers are always integers. The following table briefly summarizes error numbering rules for `Variables.errno`.

See the *Oracle Fusion Middleware WebCenter Sites Tag Reference* for specific error numbers for each tag.

Table 10–1 Error Number Rules

Number	Significance
Negative integers	Failure
0 (zero)	Success
Positive integers in a tag other than a revision tracking tag.	Information
Positive integers in a revision tracking tag.	Failure

11

Data Design: The Asset Models

The WebCenter Sites servlets are the operating system that runs your WebCenter Sites content management system, but the WebCenter Sites database is the brains of the system. It stores the system information that makes the WebCenter Sites applications run, the content that you are using the WebCenter Sites applications to manage (that is, assets), and the structural information that provides the format and business logic for displaying your content to the visitors of your online sites.

For the most part, data design means asset design. However, developers frequently need to create tables that hold supporting data for their assets. Determining the need for those tables and then designing them is also a part of data design.

This chapter contains the following sections:

Note: Designing and creating tables that do not hold assets is discussed in Chapter 12, "The WebCenter Sites Database."

11.1 Asset Types and Asset Models

An **asset** is an object that is stored in the WebCenter Sites database, an object that can be created, edited, inspected, deleted, duplicated, placed into workflow, tracked through revision tracking, searched for, and published to your delivery (live) site.

An **asset type** is a definition or specification that determines the characteristics of asset objects of that type.

Developers design and create asset types while designing your content management system and your online sites. Content providers then create and edit assets of those types

In general, assets perform one of the following three roles:

- Provide **content** that visitors read and examine on your online sites

- Provide the **formatting** logic or code for displaying the content
- Provide **data structure** for storing the content in the WebCenter Sites database

The developer's job is to design asset types that are easy for content providers to work with on the management system and that can be delivered efficiently to visitors from the delivery system.

This section contains the following topics:

- Section 11.1.1, "Two Data Models"
- Section 11.1.2, "Default Core Asset Types"
- Section 11.1.3, "Which Asset Model Should You Use to Represent Your Content?"

11.1.1 Two Data Models

The WebCenter Sites provides two data models for the assets types that you design: **basic** and **flex**.

- **Basic** asset types have a **simple data structure**: they have one primary storage table and simple parent-child relationships with each other.

  Basic asset types are separate, standalone asset types that represent individual kinds of content: an article, an image file, a page, a query, and so on. You use the AssetMaker utility (located on the **Admin** tab in the WebCenter Sites interface) to create new basic asset types.

- **Flex** asset types have a **complex data structure** with several database tables and the ability to support many more fields than do basic asset types. Additionally, they can have more than one parent, any number of grandparents, and so on, that they can **inherit** attribute values from.

  Flex asset types comprise families of asset types that define each other and assign attribute values to each other. You use the Flex Family Maker utility (located on the **Admin** tab) to create a family of flex asset types.

11.1.2 Default Core Asset Types

Several core asset types are delivered by WebCenter Sites and Engage. Because WebCenter Sites has a stack architecture, the core asset types are made available as follows:

- WebCenter Sites delivers the template, query, collection, SiteEntry, CSElement, Link, and page asset types. All of the other modules and products use the template and page asset types.
- WebCenter Sites delivers the attribute editor asset type. It supports any flex attribute asset types that you create.
- Engage delivers the visitor attribute, history attribute, history definition, segment, recommendation, and promotion asset types.

Assets of these types provide format or logic for the display of asset types that hold your content by retrieving, ordering, organizing, and formatting those assets. In other words, you use the core asset types to organize and format the content on your online site.

Asset Types Delivered with WebCenter Sites

Asset types delivered with WebCenter Sites provide basic site design logic. You can create as many individual assets of these types as you need, but you cannot modify the asset types themselves:

- **Query** stores queries that retrieve a list of assets based on selected parameters or criteria. You use query assets in page assets, collections, and recommendations. The database query can be either written directly in the New or Edit form for the query asset as a SQL query, or written in an element (with WebCenter Sites query tags or a as a search engine query) that is identified in the New or Edit form.
- **Collection** stores an ordered list of assets of one type. You build collections by running one or more queries, selecting items from their resultsets, and then ranking (ordering) the items that you selected. This ranked, ordered list is the collection. For example, you could rank a collection of articles about politics so that the article about last night's election results is number one.
- **Page** stores references to other assets. Arranging and designing page assets is how you represent the organization or design of your site. You design page assets by selecting the appropriate collections, articles, imagefiles, queries, and so on for them. Then, you position your page assets on the **Site Plan** tab that represents your site in the tree on the left side of the WebCenter Sites interface.

  Note that a page asset and a WebCenter Sites page are quite different. The page asset is an organizational construct that you use in the Site Plan tab as a site design aid and that you use to identify data in your elements. A WebCenter Sites page is a rendered page that is displayed in a browser or by some other mechanism.
- **Template** stores code (XML or JSP and Java) that renders other assets into WebCenter Sites pages and pagelets. Developers code a standard set of templates for each asset type (other than CSElement and SiteEntry) so that all assets of the same type are formatted in the same way.

  Content providers can select templates for previewing their content assets without having access to the code itself or being required to code.
- **CSElement** stores code (XML or JSP and Java) that does not render assets. Typically, you use CSElements for common code that you want to call from more than one template (a banner perhaps). You also use CSElements to provide the queries that are needed to create DynamicList recommendations in Engage.
- **SiteEntry** represents a WebCenter Sites page or pagelet and has a CSElement assigned as the root element that generates the page. Template assets do not have associated SiteEntry assets because they represent both an element and a WebCenter Sites page.
- **Link** stores a URL to an external website. You use this asset to embed an external link within another asset.
- **Attribute Editor** is an attribute editor that specifies how data is entered for a flex attribute when that attribute is displayed on a New or Edit form for a flex asset or a flex parent asset. It is similar to a Template asset. However, unlike a Template asset, you use it to identify the code that you want WebCenter Sites to use when it displays an attribute in it's interface, not when it displays the value of an attribute on your online site.

Because the data needs of each organization using a WebCenter Sites content management system are different, there are no default asset types that represent content. However, the sample sites deliver sample content asset types that you can examine and modify for use on your sites.

Oracle WebCenter Sites: Engage

The Engage application delivers several core asset types that you use to gather visitor information so that you can personalize the product placements and promotional offerings that are displayed for each visitor:

- **Visitor attribute** holds types of information that specify one characteristic only (scalar values). For example, you can create visitor attributes named "years of experience," "job title," or "number of children."
- **History attributes** are individual information types that you group together to create a vector of information that Engage treats as a single record. This vector of data is the **history definition**. For example, a history type called "purchases" can consist of the history attributes "SKU," "itemname," "quantity," and "price."
- **Segments** are assets that divide visitors into groups based on common characteristics (visitor attributes and history types). You build segments by determining which visitor data assets to base them on and then setting qualifying values for those criteria. For example, a segment could define people who live in Alaska and own fly fishing gear, or it could define people who bought a personal computer in the past six months, and so on.

After you define and categorize the visitor data that you want to collect, you use the following asset types to select, organize, and display the flex assets that represent your content on your online site:

- **Recommendation** is something like an advanced collection. It collects, assesses, and sorts flex assets (products or articles, perhaps) and then recommends the most appropriate ones for the current visitor, based on the segments that visitor belongs to.
- **Promotion** is a merchandising asset that offers some type of value or discount to your site visitors based on the flex assets (products, perhaps) that the visitor is buying and the segments that the visitor qualifies for.

> **Note:** Oracle WebCenter Sites: Engage interacts with assets that are built using the flex asset model only. You cannot program recommendations and promotions to work with assets that use the basic asset model.

11.1.3 Which Asset Model Should You Use to Represent Your Content?

During the process of designing your online site with the WebCenter Sites content management system, you and others on your team create the asset types that you need to represent the content for your site. The WebCenter Sites template and page asset types provide the formatting framework for the asset types that represent your data, whether you use the basic data model or the flex data model.

The asset data model (basic or flex) that you should choose to represent the data that you want to display on your online site depends on the nature of that data, as described in the following two sections.

11.1.3.1 When to Use the Basic Model

The basic model is a good choice when your data has the following characteristics:

- It is fixed, predictable: there will be no need to add attributes to the asset type.
- It is homogenous: all assets of the same type have similar attributes.

- It has a moderate number of attributes. You are limited by your database as to how many columns/attributes you can have in the asset type table for a basic asset.
- You want to use the static publishing method. There are very limited applications of the flex asset model in which it makes sense to use the static publishing method.
- Visitors browse your online site by navigating from link to link.

When the data for an asset type can be imagined as a spreadsheet, as a simple flat table where each asset of that type is a single record and every record has the same columns, that asset type should use the basic asset model.

11.1.3.2 When to Use the Flex Model

The flex model is the right choice when your data has the following characteristics:

- It has lots of attributes. For example, products can have potentially hundreds of attributes. Because attribute values for the flex family member are stored as rows rather than columns, flex assets can physically have many more attributes than basic assets can.
- It can be represented in a hierarchy in which assets inherit attribute values from parent assets.
- You cannot predict what attributes might be necessary in the future and your data might need additional attributes periodically.
- Asset instances of the same type can vary widely. That is, not all assets of that type should have the same attributes. For example, a bath towel product asset would have attributes that a toaster product asset would not, but both the bath towel and the toaster are product assets.
- Visitors browse your online site by navigating through "drill-down" searches that are based on the attribute values of your data.
- You want to use Oracle WebCenter Sites: Engage.

For example, products fit into the flex asset model because markets are constantly changing. You cannot always predict what products you will be selling next year or what attributes those products will have.

If your business needs will require you to make modifications to your asset types such as adding or changing their attributes, the flex data model is probably the right choice for you. The flex asset model gives you the extensibility that you need to represent data whose characteristics cannot be predicted.

11.2 The Basic Asset Model

WebCenter Sites delivers the basic asset model. In general, the data model for basic asset types is one database table per asset type. All basic assets of the same type have the exact same fields (properties) and all assets of a single type are stored in the same database table. Most of the core WebCenter Sites asset types use the basic data model.

To create new basic asset types, you use the AssetMaker utility. You code XML files called **asset descriptor files** using a custom tag named `PROPERTY` and then upload the file with AssetMaker. A **property** is both a column and a field. A `PROPERTY` statement defines a column in the table that stores assets of that type and defines how data is to be entered into the corresponding field for that column in the WebCenter Sites forms.

For information about coding asset descriptor files and creating new basic asset types, see Chapter 15, "Designing Basic Asset Types."

This section contains the following topics:

- Section 11.2.1, "Basic Asset Types from the Burlington Financial Sample Site"
- Section 11.2.2, "Relationships Between Basic Assets"
- Section 11.2.3, "Category, Source, and Subtype"
- Section 11.2.4, "Basic Asset Types and the Database"

11.2.1 Basic Asset Types from the Burlington Financial Sample Site

If you installed the Burlington Financial sample site, WebCenter Sites installed five asset types that represent content. These sample site asset types use the basic asset model delivered with WebCenter Sites:

- **Article** stores the text of an article and information about it. It has fields for headline, byline, credit line, body, and so on. Note that this is a custom asset type that was **not created with AssetMaker.**
- **ImageFile** stores an image file as an uploaded binary large object (blob). These image files can be associated with other assets such as a page or an article. This asset type was created with AssetMaker.

This sample site also provides an example of how you can create additional formatting asset types, if necessary, with the following asset type:

- **StyleSheet** stores style sheet files of any format (CSS, XSL, and so on). You create the style sheet in a text editor and then upload it into WebCenter Sites as a style sheet asset. When you store style sheets as assets, you can assign a workflow to them, use revision tracking, and so on. This asset type was created with AssetMaker.

Burlington Financial installs the following asset types, but does not use them:

- **Linkset** stores a group of links to either the URLs of related assets or the URLs of external websites. Assets of this type can be associated with other assets like a page or an article.
- **Image** stores the URL for an image file that can be associated with other assets like a page or an article.

These asset types were used by a previous sample site. They are included for backward compatibility.

11.2.2 Relationships Between Basic Assets

Basic asset types have very simple parent-child relationships. You use these relationships to associate or link assets to each other. Then, when you design the online pages for your online sites you code template elements that identify, extract, and then display an asset's children or parent assets in appropriate ways.

The relationships that basic assets can have with each other are called **associations** and **unnamed relationships**. When these relationships occur between individual assets, they are written to the `AssetRelationTree` table.

11.2.2.1 Associations

Associations are defined, asset-type-specific relationships that are represented as fields in the WebCenter Sites asset forms. After you create an asset type with AssetMaker, you use the Association form for that asset type to create association fields.

You use associations to set up relationships that make sense for the asset types in your system and then you use the names of these relationships to identify the related assets and display them in appropriate ways on your site pages.

For example, the Burlington Financial sample asset named article has three associations with the imagefile asset type: Main ImageFile, Teaser ImageFile, and SpotImageFile. The Burlington Financial article templates are coded to display the imagefiles that are linked to articles through these associations. The association is what enables the template to determine which imagefile is the correct one to display for an individual article asset.

When a content provider selects an image asset in the **Main Image** field of the New and Edit article forms, the selected imagefile asset becomes a child of the article asset. (Note that this same imagefile asset can also be a child of other articles.)

When you create a new association between asset types, WebCenter Sites creates a row for that type of association in the `Association` table. Then, when you create an asset and specify the name of another asset in an association field, that relationship is written to the `AssetRelationTree` table.

11.2.2.2 Unnamed Relationships

Unnamed relationships occur when you build a collection, the items in the collection become children of the collection.

11.2.3 Category, Source, and Subtype

There are three additional ways to organize or categorize basic assets: category, source, and subtype. Categories and subtypes are specific to an asset type. Source, however, applies to all the asset types in a content management site. In other words, source is site-specific.

11.2.3.1 Category

`Category` is a default column and field that you can use to categorize assets according to a convention that works for your sites. Although all basic asset types have a `category` column by default, you do not have to use it (it is not a required field).

For example, the Burlington Financial sample site has categories named Personal Finance, Banking and Loans, Rates and Bonds, News, and so on. Articles identified with these categories are selected by queries that use "category" as a selection criterion and displayed on specific site pages, as appropriate.

When you create a new basic asset type, AssetMaker creates one category code for assets of that type. You then use the Category form for your new asset type to create additional categories if you want to use this feature.

New categories are written to the `Category` table, which serves as the lookup table for the **Category** field on the New and Edit asset forms for asset types that use the basic asset model.

The purpose of the **Category** field and column is for site design. You can use category, or not, in your queries and query assets for your online site. The WebCenter Sites application does not base any of its functions on category codes. (With the exception that you can Search for assets based on this field, if you are using it.)

11.2.3.2 Source

`Source` is a column and field that you can use to identify where an asset originated. Although WebCenter Sites provides administrative support (through the Source form) for you to use this feature in the design of your online site, the `source` column does not exist by default in the primary storage tables for basic asset types other than Article. If you want to use source with your basic asset types, you must include a property statement in your asset descriptor file for it.

For example, the Burlington Financial sample site has sources named WireFeed, Asia Pulse, UPI, and so on. Certain online pages select stories to display based on the results of queries that search for articles based on the value in their source column.

After you create a new basic asset type, you add new sources in the Source form on the **Admin** tab, if necessary. New sources are written to the `Source` table, which serves as the lookup table for the **Source** field on the New and Edit asset forms for basic-style assets.

11.2.3.3 Subtype

The **subtype** concept provides a way to further classify an asset type. In the flex asset data model, the definition asset types create subtypes of flex assets and flex parent assets. In the basic asset data model, the concept of subtype is implemented through the `subtype` column in the primary storage table for the asset type.

The WebCenter Sites application uses the value of an asset's **Subtype** in many ways:

- For Template assets, subtype means the type of asset that the template formats. Templates that format articles are a different subtype of template than templates that format images. When you create an article asset, only the templates that format articles appear as options in the **Template** field on that asset's New or Edit form.

  In addition, you can use the WebCenter Sites Contributor interface to specify a subtype that will be displayed using a given template. For example, if your website uses two subtypes of article asset, Sports and News, you can create a template that only displays articles with the Sports subtype.

- For query assets, subtype means the type of asset that the query returns. Query assets that return articles are a different subtype of query asset than those that return imagefiles.
- For collection assets, subtype means the type of asset that the collection holds. Collections that hold articles are a different subtype of collection asset than those that hold imagefiles.
- For the basic asset types that you design, subtype is designed to classify an asset based on how it is rendered. You can define a default template for each subtype of an asset type for each of your publishing targets.

If you do not need to assign a different template to assets of a specific type based on the publishing target for the asset, you do not need to create new subtypes.

If you create any subtypes for an asset type, the New and Edit forms for assets of that type display a field named **Subtype**. The drop-down list in the field displays all the possible subtypes for that asset type.

Note: In the flex asset model, the definition asset types serve as subtypes. For example, in the GE Lighting sample site, there is one product definition: lighting. This means that there is one subtype for product assets: the lighting subtype.

For some asset types, the subtype is set implicitly and cannot be changed. Other asset types allow users to choose a subtype for the asset using the WebCenter Sites Contributor interface. The following table lists the WebCenter Sites asset types according to whether they have configurable subtypes:

Implicit Subtypes	Configurable Subtypes
▪ All flex assets ▪ Query assets ▪ Collection assets ▪ Template assets	▪ All custom basic assets (made with AssetMaker) ▪ Article assets ▪ Image assets ▪ Linkset assets ▪ Recommendation assets ▪ Link assets ▪ Page assets

For information about setting configurable subtypes, see Chapter 15, "Designing Basic Asset Types."

11.2.4 Basic Asset Types and the Database

Although there is one primary storage table for basic asset types, WebCenter Sites keeps other kinds of supporting information for basic assets in other tables. When you create a new asset of a basic type, WebCenter Sites writes to the following database tables:

- The primary database table that holds assets of its type. For example, each page asset has a row in the `Page` table and each article asset has a row in the `Article` table.

  These tables store all of the asset's attribute or field values, such as the asset's name, its object ID, who created it, which template it uses, and so on. The name of this table always matches the name of the asset type.

  When you create a new basic asset type, the AssetMaker utility creates the primary storage table (a WebCenter Sites object table) for the asset type as a part of that process.

- The `AssetRelationTree` table, if the asset has associations with other assets. The relationships that basic assets can have are described in Section 11.2.2, "Relationships Between Basic Assets."

- The `AssetPublication` table, which specifies which content management sites (publications) give you access to the asset. If the asset is shared among sites (publications), there is a row entry for each pubid. A **pubid** is a unique value that identifies a site (publication).

- The `SitePlanTree` table, if the asset is a page asset. This table stores information about the page asset's hierarchical position in your site plan.

When you develop the templates that display the assets that represent your content, you code elements with XML or JSP tags that extract and display the information from the tables in the preceding list.

Be sure to examine the New and Edit forms for the various sample asset types and to use the Oracle WebCenter Sites Explorer tool to examine the tables in your WebCenter Sites database.

Note: Do **not** use Oracle WebCenter Sites Explorer tool to modify the data in any of these tables. All editing of assets and their related tables should be done only through the WebCenter Sites interface.

11.2.4.1 Template Asset Type and the Database

Although the Template asset type is a core asset type, it does not use the basic asset model. It is a complex asset type with entries in the following database tables:

- The `Template` table, its primary storage table
- The `SiteCatalog` table
- The `ElementCatalog` tables

When you create a new Template asset, WebCenter Sites automatically creates entries in both the `SiteCatalog` and `ElementCatalog` tables for it.

11.2.4.2 Default Columns in the Basic Asset Type Database Table

WebCenter Sites needs several default columns for its basic functionality and so AssetMaker creates each of the following columns (as shown in the following table) in the asset type's primary storage table in addition to the columns defined in the asset descriptor file for that asset type.

Note that you do not need to code your asset descriptor files to include property statements for the columns in this list:

Table 11–1 Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
`id`	A unique ID for each asset, automatically generated by WebCenter Sites when you create the asset. You cannot change the value in this field.	Forms: ■ Inspect ■ Edit ■ Status ■ search forms
`name`	A unique name for the asset. Names are limited to 64 alphanumeric characters.	Forms: ■ New ■ Edit ■ Inspect, ■ Status Also in the search results lists.
`description`	A short description of the asset that offers more information than just the name.	Forms: ■ New ■ Edit ■ Inspect ■ Status Also in the search results lists.

Table 11–1 (Cont.) Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
`status`	The status of the asset, one of the following status codes obtained from the `StatusCode` table: PL: created ED: edited RF: received (from XMLPost, for example) UP: upgraded from Xcelerate 2.x VO: deleted (void) WebCenter Sites controls the value in this field; it cannot be edited manually.	Forms: **Status**, if the status of an asset is either PL (created) or ED (edited) Note that assets with a status of VO (deleted) are not displayed anywhere in the WebCenter Sites Windows interface.
`createdby`	The identity of the user who originally created the asset. This user name is obtained from the `SystemUsers` table. WebCenter Sites controls the value in this field; it cannot be edited manually.	Forms: Status Also, if revision tracking is enabled for assets of this type, the Revision History list.
`createddate`	The date and time that the asset was written to the database for the first time. WebCenter Sites controls the value in this field; it cannot be edited manually.	Forms: Status Also, if revision tracking is enabled for assets of this type, the Revision History list.
`updatedby`	The identity of the user who most recently modified the asset in any way. This user name is obtained from the SystemUsers table. WebCenter Sites controls the value in this field; it cannot be edited manually.	Forms: Status Also, if revision tracking is enabled for assets of this type, the Revision History list.
`updateddate`	The date on which the information in the status field was changed to its current state. WebCenter Sites controls the value in this field; it cannot be edited manually.	Forms: Status Also, if revision tracking is enabled for assets of this type, the Revision History list.

Table 11–1 (Cont.) Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
`startdate`	Promotion assets (a Engage asset) have durations during which they can be displayed on the visitor pages on your live system. This column stores the start time of the promotion's duration. The promotion asset type is the only default asset type that uses this column. If you want to use the startdate and enddate fields for your asset types, see Section 15.2.3.8, "Example: Enabling path, filename, startdate, and enddate."	Forms: - **Duration**, **Edit**, and **Inspect** for promotion assets. - **New**, **Edit**, **Inspect**, and **Status** if you enable it for other asset types.
`enddate`	For promotion assets (a Engage asset), this column stores the end time of the promotion's duration The promotion asset type is the only default asset type that uses this column.	Forms: - **Duration**, **Edit**, and **Inspect** for promotion assets - **New**, **Edit**, **Inspect**, and **Status** if you enable it for other asset types
`subtype`	The value of the asset's subtype. The subtype is set in different ways for different assets. For more information, see Section 11.2.3.3, "Subtype."	Forms: - **New**, and **Edit** for Template assets (**Asset Type** field) - **New**, and **Edit** for query assets (**Result of Query** field) - **New**, and **Edit** for any asset type that has subtypes configured for it - **Set Default Templates**
`filename`	The name to use for the file created for this asset during the Export to Disk publishing method. The page and article asset types are the only asset types that have this field enabled by default. If you want to use the filename field for your asset types, see Section 15.2.3.8, "Example: Enabling path, filename, startdate, and enddate."	Forms: - **New** and **Edit** for page and article assets, by default - **New** and **Edit** for any other asset type that has the field enabled

Table 11–1 (Cont.) Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
`path`	The directory path to use for exported page files that are generated from child assets of this asset when the Export to Disk publishing method renders that asset into a file. The page and article asset types are the only asset types that have this field enabled by default. If you want to use the filename field for your asset types, see Section 15.2.3.8, "Example: Enabling path, filename, startdate, and enddate."	Forms: ▪ **New** and **Edit** for page and article assets, by default ▪ **New** and **Edit** for any other asset type that has the field enabled
`template`	The template for the asset. This is the template that is used to render the asset when it is either published with Export to Disk or rendered on a live dynamic delivery system. This template is also used to calculate the dependencies when the asset is approved for the Export to Disk publishing method, unless the asset type has subtypes and there is a default approval template assigned for the asset based on its subtype.	Forms: ▪ **New** ▪ **Edit** ▪ **Inspect** ▪ **Status**
`category`	The category code of the category assigned to the asset, if any. If you decide to use the category field to organize assets, you add category codes in the Asset Types forms on the Admin tab.	Forms: ▪ **New** ▪ **Edit** ▪ **Inspect** ▪ **Status**
`urlexternaldoc`	If the asset was entered with the Sites Desktop interface rather than the WebCenter Sites interface, stores the external document that is the source for the asset. WebCenter Sites controls the value in this field; it cannot be edited manually.	not applicable
`externaldoctype`	The mimetype of the file held in the `urlexternaldoc` field. WebCenter Sites controls the value in this field; it cannot be edited manually.	not applicable

Table 11–1 (Cont.) Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
`urlexternaldocxml`	Reserved for future use.	not applicable

11.3 The Flex Asset Model

The flex asset model has the following main characteristics:

- Flex assets are defined by flex definitions; an asset type that determines which flex attributes make up an individual flex asset. Flex definitions create subtypes of the flex asset type.
- The definition asset types create subtypes of flex and flex parent assets, which allows individual instances of a flex asset or flex parent asset type to vary widely.
- Flex attributes are assets. The flex data model lets you add flex attributes to (or remove them from) existing flex asset types at any time.
- Flex filters can take the data from one flex attribute, transform or assess it in some way, and then store the results in another flex attribute when you save the flex asset. The resulting value from a flex filter action is called a "derived" attribute value. For more information about flex filters, see Chapter 17, "Flex Filters."
- Flex assets can inherit attribute values, even derived values, from their flex parents, which means that you can represent your data in hierarchies.

You do not create individual flex asset types as you do basic asset types; instead, you create a flex family of asset types.

This section contains the following topics:

- Section 11.3.1, "The Flex Family"
- Section 11.3.2, "Sample Site Flex Families"
- Section 11.3.3, "Flex Attributes"
- Section 11.3.4, "Flex Parents and Flex Parent Definitions"
- Section 11.3.5, "Flex Assets and Flex Definition Assets"
- Section 11.3.6, "Flex Families and the Database"

11.3.1 The Flex Family

The flex asset data model can be thought of in terms of a family of asset types. There are six asset types in a flex family. Five are required, the sixth is optional, as indicated in the table below.

Table 11–2 Flex Family Members

Flex Family Member	Number Per Family
flex attribute asset type	one
flex parent definition asset type	one or more
flex definition asset type	one or more
flex parent asset type	one or more
flex asset type	one or more
flex filter asset type	none or more

Whereas some of the asset types are used exclusively by developers to create the other asset types in the data model, the flex asset type is always used by the content providers to create assets of that type. (When necessary, authorized users can be given access to additional flex family members.)

To create a flex family, you access the Add New Flex Family form by double-clicking **Add New Family**, located on the **Admin** tab under the **Flex Family Maker** node in the WebCenter Sites Admin interface. In the Add New Flex Family form, you name each of the asset types in the family. For example, one of the flex families in the GE Lighting sample site is the product family. The flex asset is called the product asset, the flex attribute is called the product attribute, and so on.

The key member of a flex family is the **flex asset**. The flex asset is the unit of data that you extract from the database and display to the visitors of your online site (delivery system). All of the other members in the family contribute to the flex asset member in some way.

While the flex asset is the key, the **attributes** are the foundation of the flex asset model. An attribute is an individual component of information. For example, color, height, author, headline. You use attributes to define the flex assets and the **flex parents**. Flex assets inherit attribute values from their parents who inherit attribute values from their parents and so on.

You decide which attributes describe which flex assets and which flex parents by creating "templates" with the **flex definition** and **flex parent definition** asset types. Flex parents and their definitions implement the inheritance of attribute values.

Note that a flex parent or a flex asset cannot be defined by attributes of two types. The GE sample site has two kinds of attributes: product attributes and content attributes. A product asset (the flex asset member in the product flex family) can be defined by product attributes only, its definition cannot include content attributes.

A **flex filter** enables you to configure some kind of action to take place on the value of an attribute and then save the results of the action when the flex asset is saved. For example, you can configure a filter that converts the text in a Word file into HTML code.

In summary, the flex asset member of a flex family is the reason for the family, the unit of content that you want to display. The other members of a flex family provide data structure for the flex asset. However, because all of the members in the family are assets, you can take advantage of the standard WebCenter Sites features like revision tracking, workflow, search, and so on.

11.3.1.1 Parent, Child, and Flex Assets

When you are using the flex asset data model, the phrase "parent-child" relationship refers to the relationship between a flex asset and its flex parent asset(s). This is a different parent-child relationship than the ones that basic assets have through asset associations.

Although it is possible for flex assets to have the kinds of parent-child relationships that basic assets do, it is unlikely for the following reasons:

- WebCenter Sites provides the `ASSETSET` and `SEARCHSTATE` tag families, which you use instead of the collection and query asset types to select the flex assets that you want to display. For more information about this tag family, see Section 11.4, "Assetsets and Searchstates."
- Flex assets have no need for associations. For example, if you want to assign an image file to a flex asset like a product, you can create an attribute that identifies the image file and assign it to the definition for the flex asset.

11.3.2 Sample Site Flex Families

If the GE sample site is installed on your development system, there are two flex asset families that you can examine: the product family and the content family.

To better understand the following descriptions of the sample flex asset types, examine some of the product, article, and image assets in the WebCenter Sites interface as you read this section.

11.3.2.1 The Product Family

The product family provides the data structure for the lighting products that are sold from the GE Lighting sample site. It creates an online catalog of lighting products.

These are the asset types in the product family:

- **Product attribute** is the flex attribute asset type used to define the products and product parents in the GE Lighting sample catalog. For example, there are product attributes named wattage, voltage, bulb size, ballast type, and so on.
- **Product** is the flex asset member of the product family. Product assets represent the lighting products that are sold from the GE Lighting sample site. In this online catalog, product names are numbers similar to a SKU number.
- **Product definition** is the flex definition asset type in the product family. It is used to create one subtype of products: lighting. The lighting definition formats (defines) all of the light bulbs in this online catalog.
- **Product parent** is the flex parent asset type in the product family. Product parents represent categories of products such as Compact Fluorescent, Fluorescent, Halogen, and so on.
- **Product parent definition** is the flex parent definition asset type in the product family. It is used to create subtypes of product parents. There are two: Category and Subcategory.

The product attribute, product definition, and product parent definition assets are listed on the **Design** tab because you use them for data design. The product and product parent assets are located on the sample site's **Product** tab.

The product asset is the reason for the product family: the GE Lighting sample site sells products.

11.3.2.2 The Content Family

The content family provides the data structure for the articles that describe and images that illustrate the products that are sold from the GE Lighting sample site.

This is the content family:

- **Content attribute** is the flex attribute asset type used to define the articles (flex) and images (flex) that illustrate the products sold from the GE Lighting sample site.
- **Article (flex)** is a flex asset type that stores the text of an article and information about it. It has attributes such as byline, headline, subheadline, body, and so on.
- **Image (flex)** is a flex asset type that stores the URL of an image file. Although the GE Lighting sample site makes this asset type available, it does not use it.
- **Content definition** is the flex definition asset type in the content family. It is used to create one subtype of the article (flex) asset type called "story."

- **Content parent** is the flex parent asset type in the content family. Although the GE Lighting sample site makes this asset type available, it does not use it.
- **Content parent definition** is the flex parent definition asset type in the content family. Although the GE Lighting sample site makes this asset type available, it does not use it.

Notice that there are two flex asset types in the GE sample site's content family. They share attributes, parents, definitions, and parent definitions.

The content attribute, content definition, and content parent definition assets are listed on the **Design** tab because you use them for data design. The image (flex), article (flex), and content parent assets are located on the sample site's **Content** tab.

11.3.3 Flex Attributes

Flex attributes are the foundation of the flex asset model. An attribute represents one unit of information. You use attribute assets to define flex assets and flex parents. They are then displayed as fields in the New and Edit forms for your flex assets and their parents.

An attribute is similar to a property for a basic asset. As does a property, an attribute defines the kind of data that can be stored in a column in a WebCenter Sites database table and describes a field in the forms. However, while a property defines one column in an asset type's database table, an attribute is an asset with database tables of its own.

This data structure (attributes as assets rather than columns) is a one of the main reasons why flex assets are so flexible.

Once again, a flex parent or a flex asset cannot be defined by attributes of two types. For example, the GE Lighting sample site product asset can be defined by product attributes only, its definition cannot include content attributes.

> **Note:** If you remove a flex attribute from a flex definition, be aware that the data stored by those attributes is not deleted from the database (queries will continue to return this data).

11.3.3.1 Data Types for Attributes

The data types for your attributes are defined by the WebCenter Sites database properties located in the `futuretense.ini file`, with the exception of the `money` data type, which is defined by a property in the `gator.ini` file (which is the name of the `.ini` file).

Table 11–3 lists the data types for flex attributes, the properties that define the data types, and the files where the properties are located:

Table 11–3 Data Types for Flex Attributes

Type	Property	.ini file
`date`	`cc.datetime`	`futuretense.ini`
`float`	`cc.double`	`futuretense.ini`
`integer`	`cc.integer`	`futuretense.ini`
`money`	`cc.money`	`gator.ini`
`string`	`cc.varchar`	`futuretense.ini`

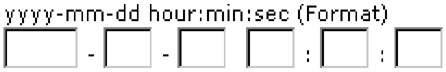
Table 11–3 (Cont.) Data Types for Flex Attributes

Type	Property	.ini file
`text`	`cc.bigtext`	`futuretense.ini`
`asset`	`cc.bigint`	`futuretense.ini`
`blob`	`cc.bigint`	`futuretense.ini`

11.3.3.2 Default Input Styles for Attributes

When a flex attribute is displayed as a field on a New or Edit form, it has default input styles based on its data types. The following list presents the default input styles for flex attributes:

- Date: input boxes that look like this:



- Float: text field with decimal position enforced.
- Integer: text field.
- Money: text field with currency format enforced.
- String: text field that accepts up to 255 characters.
- Text: text box. The number of characters that it accepts depends on the database and database driver you are using.
- Asset: drop-down list of all the assets of the type that was specified.
- Blob: a text field with a **Browse** button.

If you do not want to use the default input style for a flex attribute, you can create an **attribute editor** and assign it to the attribute. Attribute editors are assets but they are also similar to the `INPUTFORM` statement in an asset descriptor file for a basic asset: they specify how data is entered into the attribute field. For more information about attribute editors, see Chapter 18, "Designing Attribute Editors."

11.3.3.3 Foreign Attributes

You can have flex attributes that are stored in foreign tables, that is, foreign attributes. They are subject to the following constraints:

- The foreign table must be registered with WebCenter Sites. That is, the foreign table must be identified to WebCenter Sites in the `SystemInfo` table. For information, see Section 12.3.4, "Registering a Foreign Table."
- The foreign table must have a column that holds an identifier that uniquely identifies each row. The identifier must have fewer than 20 characters.
- The foreign table must have a column that is reserved for the attribute data value, which can be of any appropriate data type. For example, if the attribute is of type string, the data type must be appropriate for a string.

11.3.4 Flex Parents and Flex Parent Definitions

Flex parents and their **flex parent definitions** are organizational constructs that do two things:

- Implement the inheritance of attribute values. The parent definitions set up (describe) the rules of inheritance and the parents pass on attribute values to the flex assets according to those rules of inheritance.
- Determine the position of a flex asset on the tabs that display your assets in the WebCenter Sites interface. The hierarchy of the parents and the flex assets on the tabs in that tree are based on the hierarchy set up with the parent definitions.

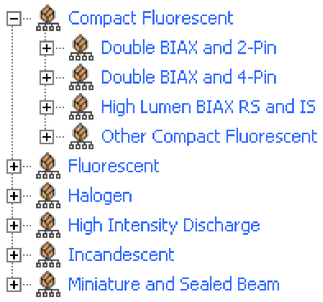
Each parent asset type has its own set of attributes, as specified in its parent definition. The parent definition creates a form that you see in the WebCenter Sites interface.

You use parents to organize or manage the flex assets by passing on attribute values that are standard and do not need to vary for each individual child asset of that parent.

Parent asset types affect how you and the content providers see and interact with the data within the WebCenter Sites interface.

For example, in the GE Lighting sample site there are two parent definitions: Category and SubCategory. Their sole purpose is to create structure on the sample site's **Product** tab in the tree (in the WebCenter Sites interface).

In the GE Lighting site, when the product parent's definition is Category, the product parent is displayed at the top level on the **Product** tab. When the product parent's definition is SubCategory, the product parent is displayed at the second level and it has a parent of its own:



- Compact Fluorescent
  - Double BIAX and 2-Pin
  - Double BIAX and 4-Pin
  - High Lumen BIAX RS and IS
  - Other Compact Fluorescent
- Fluorescent
- Halogen
- High Intensity Discharge
- Incandescent
- Miniature and Sealed Beam

For example, in the GE Sample site, there are several top-level product parents: Compact Fluorescent, Halogen, and so on. They were created with the Category definition. The next-level product parents, such as Double BIAX and 2-Pin, Double BIAX and 4-pin, and so on were created with the Subcategory definition.

11.3.4.1 Business Rules and Taxonomy

The purpose of parent definitions and parent assets is not only to express the taxonomy of your data; they also allow you to apply business rules (logic) without risk of input error from end users. If, by creating a flex asset of a specific definition, there are dependencies that it should inherit, that flex asset should have a parent.

For example, here is a simple product, a toaster with five attributes:

- SKU = 1234
- Description = toaster
- Price = 20
- CAT1 = Kitchen
- CAT2 = Appliances

When the value of CAT2 is "Appliances," the value of CAT1 can only be "Kitchen." In other words, there is a business rule dependency between the value of CAT1 and the value of CAT2.

In this kind of case, there is no reason to require the content providers to fill in both fields. Because every field whose data has to be entered manually is a field that might hold bad data through input error, you would use inheritance to impose the business rule:

- Make CAT1 and CAT2 parent definitions.
- Make Kitchen a parent created with the CAT1 definition and Appliances a parent created with the CAT2 definition.
- Make Kitchen the flex parent of Appliances.

Now, when content providers create products, if they select Appliances for CAT2, the value for CAT1 is determined automatically through inheritance.

11.3.5 Flex Assets and Flex Definition Assets

A **flex asset** is the reason for the flex family. It is the asset type that represents the end goal; a product, a piece of content that is displayed, and so on. For example, in the GE sample site there are three flex asset types:

- Product, which represents an individual saleable unit
- Article (flex), an asset that holds text
- Image (flex), an asset that holds the URL of a picture file

All of the other members in the family contribute to the flex asset member in some way.

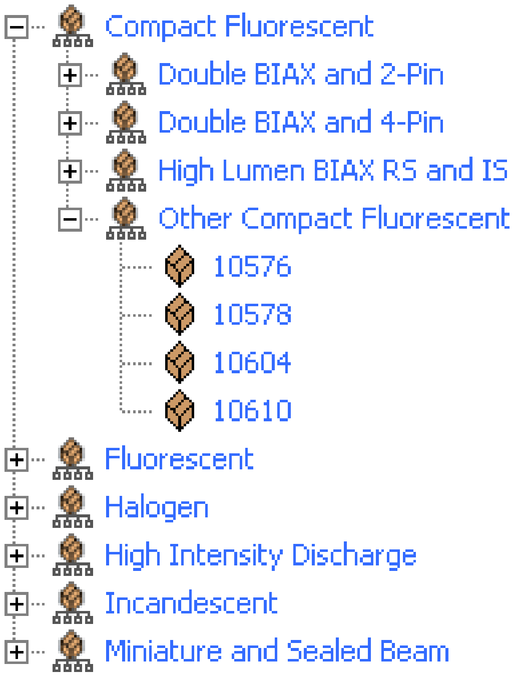
A **flex definition asset** describes one kind of flex asset in a flex family; for example, a shoe, a toaster, a bowling ball, a brochure, a newsletter, an article, and so on. A flex definition asset is a template in that it directly affects a form that you see in the WebCenter Sites interface.

Although the GE sample site has only one flex definition for products (lighting) and one flex definition for articles and images, you can create as many flex definitions as you need.

For example, if you were designing a product catalog that offered both toasters and linens, you would certainly create a flex definition asset for toasters and a different flex definition asset for linens.

Individual flex assets can be created according to only one flex definition asset. You could not create a product that used both the toaster definition and the linens definition.

A flex asset has not only the attributes assigned directly to it when it was created, it also has the attributes that it inherits from a parent. It can have more than one flex parent and whether the parents have parents depends on the hierarchical structure that you design. The products in the GE sample site, for example, have three levels of hierarchy:



The Other Compact Fluorescent product parent has a parent of its own (Compact Fluorescent) and several children (10576, 10578, and so on).

11.3.6 Flex Families and the Database

Each asset type in a flex family has several database tables. For example, the flex asset member has six tables and a flex parent type has five. This data model enables the flex member in a flex family to support more fields than an asset type in the basic asset model can support.

The four most important types of tables in the flex model are as follows:

- The primary table for the asset type
- The `_Mungo` table, which holds attribute values for flex assets and flex parent assets only
- The `MungoBlobs` table, which holds the values of all the flex attributes of type blob.
- The `_AMap` table, which holds information about the inheritance of attribute values for flex asset and flex parents only

There are several other tables that store supporting data about the relationships between the flex assets as well as additional configuration information (details about search engines, the location of foreign attributes, publishing information, and, if revision tracking is enabled, version information).

Additionally, certain kinds of site information are held in the same tables that basic assets use. For example, the `AssetPublication` table specifies which content management sites the asset type is enabled for.

When you develop the templates that display the flex assets that represent your content, you code elements that extract and display information from the `_Mungo` tables and the `MungoBlobs` table.

11.3.6.1 Default Columns in the Flex Asset Type Database Table

As do basic asset types, each of the flex asset types has a primary storage table that takes its name from the asset type. For example, the primary table for the GE sample site asset type named product is called `Products`. The primary table for the product attribute asset type is called `PAttributes`.

Unlike the primary table for a basic asset type, the primary table for a flex asset type has only the default columns. This is because flex asset types that have attribute values do not store those values in the primary table, attribute values are stored in the `_Mungo` table for the asset type.

In general, the **default column types** in the primary table for a **flex asset type** are the **same** as the default columns in the primary storage table for a **basic asset type**. For the general list of default column types, see Section 11.2.4.2, "Default Columns in the Basic Asset Type Database Table."

However, there are, of course, exceptions and additions, as described in the following table:

Table 11–4 Default Columns in the Flex Asset Type Database Table

Column	Description
`category`	Category is **not** used in the flex asset model so there is no `category` column in any of the primary tables for flex asset types. Flex assets have no need for the category feature because queries for flex assets are based on the values of their flex attributes.
`template`	Only the table for the flex asset member in a flex family, product, article (flex), and image (flex), for example, holds values in this column. This is because only the flex asset member in the family can have a Template asset assigned to it and be displayed on your online site.
`renderid`	Holds the object ID of the Template asset assigned to a flex asset.
`attributetype`	An additional column in the primary table for flex attribute types. It holds the name of the attribute editor that formats the input style of the attribute when it is displayed in the New and Edit forms (if there is one).
`flextemplateid`	An additional column in the primary table for a flex asset type (the flex asset member of a flex family.) It holds the ID of the flex definition that the flex asset was created with.
`flexgrouptemplateid`	An additional column in the primary table for flex parent asset types. It holds the object ID of the parent definition that the flex parent asset was created with.

11.3.6.2 The _Mungo Tables

The flex asset and flex parent asset types have an AssetType`_Mungo` table, where AssetType is the name of the flex asset type (and matches the name of the main storage table). Its purpose is to store the attribute values assigned to an asset when an asset of this type is created. For example the GE sample site table `Products_Mungo` holds the attribute values for product assets.

Each attribute value has a separate row.

Each row in `_Mungo` table has a value in each of the following columns:

Table 11–5 _Mungo Table Rows

Column	Description
id	A unique ID for each attribute value, automatically generated by WebCenter Sites when the flex asset is saved and the row is created. This is the table's primary key.
ownerid	The ID of the flex asset that the attribute value belongs to. (From the flex asset table: `Product`, for example.)
attrid	The ID of the attribute. (From the attribute table: `PAttributes`, for example.)
assetgroupid	If the attribute value is inherited, the ID of the parent who passed on the value. (From the parent table: `ProductGroups`, for example.)

Each row in a `_Mungo` table also has all of the following columns but it will have a value (data) in only one of them, depending on the data type of the attribute:

Table 11–6 _Mungo Table Columns

Column	Description
floatvalue	If the attribute's data type is `float`, the value of the attribute.
moneyvalue	If the attribute's data type is `money`, the value of the attribute.
textvalue	If the attribute's data type is `textvalue`, the value of the attribute.
datevalue	If the attribute's data type is `date`, the value of the attribute.
intvalue	If the attribute's data type is `int`, the value of the attribute.
blobvalue	If the attribute's data type is `blob`, the ID of the row in the `MungoBlobs` table that holds the value of the attribute.
urlvalue	If the attribute's data type is `url`, the path or url entered for the attribute.
assetvalue	If the attribute's data type is `asset`, the ID of the asset.
stringvalue	If the attribute's data type is `float`, the value of the attribute.

Because the `_Mungo` tables have URL columns (see Section 12.2.3, "Indirect Data Storage with the WebCenter Sites URL Field"), a default storage directory (defdir) must be set for it. You use the `cc.urlattrpath` property in the `gator.ini` file to set the defdir for your `_Mungo` tables.

11.3.6.3 The MungoBlobs Table

There is one `MungoBlobs` table. It holds all the values for all flex attributes of type blob, for all the flex attribute types in your system. Each attribute value has a separate row in the table.

11.3.6.4 The _AMap Tables

Flex asset and flex parent asset types have an AssetType`_AMap` table. Its purpose is to map the asset to the attributes it inherits from its parents. Then when you create a template that displays the asset on a page in your online site, you can query for assets based on any of their attributes and display any of those attributes, whether they were inherited or were directly assigned.

The _AMap table has one row for each flex asset that has a value for the inherited attribute. (However, if an attribute has more than one value, the _Mungo table has a row for each value.)

An _AMap table has the following columns:

Table 11–7 _AMap Table Columns

Column	Description
id	A unique ID for each row, automatically generated by WebCenter Sites when the flex asset is saved and the row is created. This is the table's primary key.
inherited	The ID of the parent the attribute was inherited from, if it was inherited. (From the parent table: ProductGroups, for example.)
attributeid	The ID of the attribute. (From the attribute table: PAttributes, for example)
ownerid	The ID of the flex asset that the attribute value belongs to. (From the flex asset table: Product, for example.)

11.4 Assetsets and Searchstates

WebCenter Sites provides the ASSETSET and SEARCHSTATE method families for identifying the individual flex assets that you want to display on your online pages.

Assetset

An *assetset* is a group of flex assets or flex parent assets *only*, not flex attributes or flex definitions or flex parent definitions. For example, in the GE sample site, you can create assetsets that contain products, articles (advanced), or images (advanced). When you code your site pages, you code statements that create assetsets and then display the assets in them.

Searchstate

You identify which flex assets should be in an assetset by using the SEARCHSTATE method family in the templates for your flex assets. A *searchstate* is a set of search constraints that are applied to a list or set of flex assets. A constraint can be either a filter (restriction) based on the value of an attribute or based on another searchstate (called a nested searchstate).

A searchstate can search either the _Mungo tables in the database or the attribute indexes created by a search engine. This means that you can mix database and rich-text (full-text through an index) searches in the same query.

Because these tags search only the _Mungo table or attribute indexes for that flex asset type, using them to extract your flex assets is much more efficient than using the ASSET tags or the query asset.

Assetsets and Attribute Asset Types

WebCenter Sites cannot perform searches across attribute asset types. Because assetsets are created on the basis of attribute values, only assets that share the same attribute asset type can be included in the same assetset. This is an important point to consider when you design your flex families: if you create flex asset types that do not share a common attribute asset type, you have separated your data and ensured that assets from different types cannot be included in a common assetset. And displaying assetsets is the mechanism for displaying flex assets on your delivery system.

For example, you can have two types of flex assets in the same flex family. As long as they use the same type of attributes, you can create assetsets that include assets of both types. Keep in mind, though, that a search across two types of flex assets creates a join between their `_Mungo` tables, which can deprecate performance.

In the GE sample site there are two flex asset types: article (advanced) and image (advanced). They share the same attribute asset type ("Content Attributes") and the same definitions (content definition and content parent definition). However, it is the shared attribute asset type that enables them to be included in the same assetset, even though they are two different flex asset types.

However, because articles and images do not share an attribute asset type with the GE product asset type, you cannot create an assetset that includes products and articles.

11.5 Search Engines and the Two Asset Models

Because the data structure of the two asset models is so different, there is a key difference in the way the asset models interact with a search engine:

- A basic asset type is defined by an asset descriptor file and its primary storage table includes all of its properties as columns. To specify which fields of a basic asset type should be indexed, you must customize certain elements for the asset type. See Chapter 15, "Designing Basic Asset Types."
- Because "fields" for flex assets are flex attributes, which are assets, you decide which "fields" are indexed for rich-text search, attribute by attribute. Additionally, the WebCenter Sites application enables you to specify which attributes should be indexed with the **Search Engine** field on the attribute's New and Edit forms. You do not need to customize any elements to enable this feature.

11.6 Tags and the Two Asset Models

The ultimate goal of creating and managing assets is to move them to your delivery system, where the code in your elements can extract them from the database and display them to your site visitors. The WebCenter Sites applications offer various "toolsets," custom tag sets, in both XML and JSP.

The toolset you use to extract assets from the database in your templates depends on the kind of asset that you are working with.

- For assets with the basic asset model, you use the `ASSET` method family.
- For the flex asset member in a flex family, you use the `ASSETSET` and `SEARCHSTATE` method families. Note that you **should not** use the **ASSET.LOAD** tag for the **flex asset member** in a flex family (product, article, and image, for example). Using `ASSET.LOAD` tag for flex assets is extremely inefficient because it retrieves all of the information for that asset from all of its tables. The `SEARCHSTATE` methods queries only the `_Mungo` table for the asset type of the flex asset and the `MungoBlobs` table.
- For recommendation assets, you use the `COMMERCECONTEXT` method family.

There are many more method families available with these products as well as an extensive set of custom tags from WebCenter Sites itself and several APIs.

For information about all the tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

11.7 Summary: Basic and Flex Asset Models

This section summarizes the similarities and differences between the two asset models.

Where the Asset Models Intersect

Even though there are many differences in the way that the basic and flex asset models function, there are several points of intersection.

- No matter which asset model you are using, basic or flex, you use the template and page asset types that are delivered with the WebCenter Sites application.
- All asset types have a status code, which means that all assets, whether they are flex or basic, can be searched for with queries based on status.
- All asset types, whether they are flex or basic, have the following configuration or administrative traits in common:
  - They must be enabled by site.
  - They must have Start Menu items configured for them before anyone can create individual instances of those types.
  - Individual instances of them can be imported with the XMLPost utility.

Where the Asset Models Differ

The following table summarizes the major differences between the asset models:

Table 11–8 Major Differences Between the Asset Models

Feature	Basic Asset Model	Flex Asset Model
Number of database tables	One	Several
Adding fields to an asset type	Requires a schema change.	Does not require a schema change.
Links to other assets	Through associations and unnamed relationships.	Through flex family relationships.
Subtypes	Usually available through the Subtype item on the **Admin** tab. For more information on how subtypes are set, see Section 11.2.3.3, "Subtype."	Through flex definitions and flex parent definitions.
Search engine indexing	Must customize certain elements for the asset type.	Use the Search Engine field in the flex attribute form.
Main tag families	`ASSET`, `SITEPLAN`, and `RENDER`	`ASSETSET`, `SEARCHSTATE`, and `RENDER`
Publishing methods	Export to Disk Mirror to Server	Export to Server is possible, but is atypical for the flex model. Mirror to Server

11.8 Summary: Asset Types

The following table lists all the asset types delivered by the WebCenter Sites modules, products, and sample sites:

Table 11–9 Asset Types

Name of asset type	Product or Sample Site
page	WebCenter Sites
template	WebCenter Sites
collection	WebCenter Sites
query	WebCenter Sites
CSElement	WebCenter Sites
SiteEntry	WebCenter Sites
link	WebCenter Sites
article	WebCenter Sites, Burlington Financial
imagefile	WebCenter Sites, Burlington Financial
stylesheet	WebCenter Sites, Burlington Financial
linkset	WebCenter Sites, Burlington Financial (for backward compatibility only)
image	WebCenter Sites, Burlington Financial (for backward compatibility only)
HelloArticle	WebCenter Sites, HelloAssetWorld
HelloImage	WebCenter Sites, HelloAssetWorld
attribute editor	WebCenter Sites
product	WebCenter Sites, GE Lighting
product attribute	WebCenter Sites, GE Lighting
product definition	WebCenter Sites, GE Lighting
product parent	WebCenter Sites, GE Lighting
product parent definition	WebCenter Sites, GE Lighting
article (flex)	WebCenter Sites, GE Lighting
image (flex)	WebCenter Sites, GE Lighting
content attribute	WebCenter Sites, GE Lighting
content definition	WebCenter Sites, GE Lighting
content parent	WebCenter Sites, GE Lighting
content parent definition	WebCenter Sites, GE Lighting
visitor attribute	Engage
history attribute	Engage
history definition	Engage
segment	Engage
recommendation	Engage
promotion	Engage
DrillHierarchy	Engage, Burlington Financial Extension
PDF	Engage, Burlington Financial Extension

12 The WebCenter Sites Database

Just about everything in WebCenter Sites, its modules, and WebCenter Sites products is represented as a row in a database table.

This chapter describes the various kinds of tables and columns in the WebCenter Sites database and presents procedures for creating tables. The WebCenter Sites modules and products (Engage, for example) deliver most of the tables that you need. However, if you are using WebCenter Sites to develop your own application or you need to use a table that does not hold assets (a lookup table, for example) you create that table using one of the methods described in this chapter.

This chapter contains the following sections:

- Section 12.1, "Types of Database Tables"
- Section 12.2, "Types of Columns (Fields)"
- Section 12.3, "Creating Database Tables"
- Section 12.4, "How Information Is Added to the System Tables"
- Section 12.5, "Property Files and Databases"

For information about managing the data in non-asset tables, see Chapter 13, "Managing Data in Non-Asset Tables."

> **Note:** WebCenter Sites database tables used to be called *catalogs* and there are still remnants of that terminology throughout the application in table names, servlet names (CatalogManager), and the Java interfaces that you use to work with data in the database.

12.1 Types of Database Tables

There are five types of tables in the WebCenter Sites database:

- Object tables, which hold data as objects and provides a unique identifier, automatically, for each row in the table
- Tree tables, which hold the hierarchical information about relationships between objects in object tables
- Content tables, which hold flat data and do not provide a unique identifier for each row
- Foreign tables, which can be either of the following:
  - Tables that are outside of the WebCenter Sites database but that WebCenter Sites has access to.

  - Tables that are in the WebCenter Sites database but that WebCenter Sites did not create.
- System tables, which are core WebCenter Sites application tables whose schema cannot be modified

WebCenter Sites can cache the resultsets from queries against any table in the WebCenter Sites database, including foreign tables.

This section contains the following topics:

- Section 12.1.1, "Object Tables"
- Section 12.1.2, "Tree Tables"
- Section 12.1.3, "Content Tables"
- Section 12.1.4, "Foreign Tables"
- Section 12.1.5, "System Tables"
- Section 12.1.6, "Identifying a Table's Type"

12.1.1 Object Tables

Object tables store data as an object and can be represented in hierarchies. Those objects can be loaded, saved, and managed with the CatalogManager API. The asset type tables are object tables.

The primary key for object tables is always the ID (`id`) column and that cannot be changed. When you instruct WebCenter Sites to add an object table, it creates an ID column in that table. ID is a unique identifier that WebCenter Sites assigns by default to each row as it is added to the table. For example, when someone creates a new asset, WebCenter Sites determines the ID and assigns that value as the ID for that asset.

You cannot change the ID that WebCenter Sites assigns to objects (such as assets).

Note: When AssetMaker or Flex Family maker creates an object table for a new asset type, it creates several additional columns by default. For information about the default columns in basic asset tables, see Section 11.2.4.2, "Default Columns in the Basic Asset Type Database Table."

Anytime you need to store data and you want to ensure that each row of that data is uniquely identified, use an object table because WebCenter Sites handles ID generation for you.

Examples of object tables (catalogs)

- All tables that hold assets
- Many of the publishing tables
- The Engage tables that hold visitor data

12.1.2 Tree Tables

Tree tables store information about the hierarchical relationships between object tables. In other words, object tables can be represented in hierarchies, but the hierarchy itself is stored in a tree table (the hierarchy is the tree.)

For example, WebCenter Sites adds the following tables to the WebCenter Sites database:

- AssetRelationTree: which stores information about associations between assets. These associations create parent-child relationships. For information about asset associations, see Section 11.3, "The Flex Asset Model."
- SitePlanTree: which stores information about parent-child relationships between page assets and the assets that are referred to from those assets. This information is presented graphically on the **Site Plan** tab that is present in the WebCenter Sites interface.

Each row in a tree table is a node in that tree. Each node in a tree table points to two places:

- To an object in an object table, that is, to the object that it represents
- To its parent node in that tree table, unless it is a top-level node and has no parent

In other words, the object itself is stored in an object table. That object's relationships to other objects in the database (as described by the tree) are stored in the tree table as a node on a tree.

Note that children nodes point to parent nodes but parents do not point to children.

When you create a tree table, it has the following columns by default. You cannot add to or modify these columns:

Table 12–1 Default Columns

Column	Description
nid	The ID of the node. This is the primary key.
nparentid	The ID of the node's parent node.
nrank	A number that ranks peer or sibling nodes. For example, the AssetRelationTree table uses this column to determine the order of the assets that are in collections.
otype	The object type of the node. For example, in the SitePlanTree table, otype is either the asset type "page" or the name of a site ("publication"). In the AssetRelationTree table, otype is an asset type and is the name of the object table for assets of that type.
oid	The ID of the object that the node refers to.
oversion	Reserved for future use.
ncode	Holds a string that has meaning in the context of what the table is being used for. For example, in the SitePlanTree, ncode is set to "placed" or "unplaced" based on whether the page asset that the node refers to has been placed or not. In the AssetRelationTree, ncode holds the name of an association.

12.1.3 Content Tables

Content tables store data as flat data (rather than as objects) and that information cannot be organized in a hierarchy. You use content tables for simple lookup tables. For example, these are only a few of the content tables that adds to the WebCenter Sites database:

- Source, which holds strings that are used to identify the source of an article or image asset
- Category, which holds codes that are used to organize assets in several ways

- StatusCode, which holds the codes that represent the status of an asset

All three of these tables are lookup tables that the product uses to look up values for various columns in the asset type tables (object tables).

In another example, WebCenter Sites also adds a content table called MimeType. This table holds mimetype codes that are displayed in the **Mimetype** fields of the Burlington Financial sample site asset types named stylesheet and imagefile. The **Mimetype** fields for these asset types query the MimeType table for mimetype codes based on the keyword column in that table.

Setting the Primary Key for a Content Table

When you create a content table, an ID column is not created for you and the primary key is not required to be ID. This is another major difference between content tables and object tables.

The cc.contentkey property in the futuretense.ini file specifies the name of the default primary key for all content tables. When you create a new content table, you are responsible for defining a column with the name specified by the cc.contentkey property.

However, you can override the identity of the primary key for a specific content table by adding and setting a custom property in the futuretense.ini file. This property must use the following format:

```
cc.tablenameKey
```

For example, if you create a content table named Books and you want to override the default primary key so that it uses the ISBN column instead, you would add a property named cc.BooksKey and set it to ISBN.

12.1.4 Foreign Tables

A foreign table is one that WebCenter Sites does not completely manage. For example, perhaps your site pages perform queries against a table that is populated by an ERP system and WebCenter Sites displays that information to your site visitors.

WebCenter Sites can query foreign tables and cache the resultsets just as it does for its own object and content tables. However, you must first identify that foreign table to WebCenter Sites by adding a row for it in the SystemInfo table. This is the **only** time you should ever modify information in the SystemInfo table.

Additionally, you must be sure to flush the WebCenter Sites resultset cache with a CatalogManager flushcatalog tag whenever the external system updates the tables that you query. Otherwise, the resultsets cached against those tables might not be up-to-date.

For information about resultset caching, see Chapter 14, "Resultset Caching and Queries."

12.1.5 System Tables

System tables are core, WebCenter Sites tables whose schema is fixed. They are implemented in WebCenter Sites by their own classes and they do not follow the rules (for caching and so on) that the other tables follow.

You can add rows to some of the system tables (either using the WebCenter Sites Management Tools forms, found on the **Admin** tab of the WebCenter Sites interface, or the Oracle WebCenter Sites Explorer tool), but you cannot add or modify the columns in these tables in any way. You also cannot add system tables to the database.

The following table lists and defines the WebCenter Sites system tables:

Table 12–2 System Tables

Table	Description
`ElementCatalog`	Lists all the XML or JSP elements used in your system. An element is a named piece of code. For more information about the `ElementCatalog` table, see Chapter 23, "Creating Template, CSElement, and SiteEntry Assets."
`SiteCatalog`	Lists a page reference for each page or pagelet served by WebCenter Sites. For more information about the `SiteCatalog` table, see Chapter 23, "Creating Template, CSElement, and SiteEntry Assets."
`SystemACL`	Has a row for each of the access control lists (ACLs) that were created for your WebCenter Sites system. ACLs are sets of permissions to database tables. For information about creating ACLs, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*. For information about using ACLs to implement user management for your online site, Chapter 31, "User Management on the Delivery System."
`SystemEvents`	Has a row for each event being managed by WebCenter Sites. An event represents an action that takes place on a certain schedule. WebCenter Sites inserts a row in this table when you set an event by using either the `APPEVENT` or `EMAILEVENT` tags.
`SystemInfo`	Lists all the tables that are in the WebCenter Sites database and any foreign tables that WebCenter Sites needs to reference.
`SystemItemCache`	Holds information about specific items on pages that are cached (assets, for example): the identity of the item, the page it is associated with, and the time it was cached.
`SystemPageCache`	Holds information about pages that are cached: the folder that it is cached to, the query used to generate the file name, the time it was cached, and the time that it should expire.
`SystemSeedAccess`	Registers Java classes that are external to WebCenter Sites but that WebCenter Sites has access to (includes access control).
`SystemSQL`	Holds SQL queries that you can reuse in as many pages or pagelets as necessary. You can store SQL queries in this table and then use the `ics.CallSQL` method, `CALLSQL` XML tag, the `ics:callsql` JSP tag to invoke them. Then, if you need to modify the SQL statement, you only have to modify it once.
`SystemUserAttr`	Stores attribute information about the users such as their email addresses. Note that if you are using LDAP, this table is not used.
`SystemUsers`	Lists all the users who are allowed access to pages, functions, and tables. Note that if you are using LDAP, this table is not used.

12.1.6 Identifying a Table's Type

To determine the table type of any table in the WebCenter Sites database, examine the `SystemInfo` table, the system table that lists all the tables in the database.

To determine a table type

1. Open the Oracle WebCenter Sites Explorer and log in to the WebCenter Sites database.
2. Double-click on the **SystemInfo** table.
3. In the list of tables, examine the **systable** column. The value in this column identifies the type of table represented in the row:

Table 12–3 Determining Table Types

Value in systable column	Definition
`yes`	system table
`no`	content table
`obj`	object table
`tree`	tree table
`fgn`	foreign table

> **Note:** If you do not have the appropriate ACLs assigned to your user name, you cannot open and examine the SystemInfo table.

12.2 Types of Columns (Fields)

When you create new tables for the WebCenter Sites database, whether they hold assets or not, you can specify three general categories of field (column) types for the columns in those tables:

- Generic field types
- Database-specific field types
- The WebCenter Sites URL field

This section contains the following topics:

- Section 12.2.1, "Generic Field Types"
- Section 12.2.2, "Database-Specific Field Types"
- Section 12.2.3, "Indirect Data Storage with the WebCenter Sites URL Field"

12.2.1 Generic Field Types

Generic field types refers to field types that work in any DBMS that WebCenter Sites supports. They are mapped to be compliant with JDBC standards. Therefore, if your WebCenter Sites system changes to a different DBMS, your database is still valid.

When you use generic, JDBC-compliant field types, you can use the CatalogManager API (`CATALOGMANAGER` XML or JSP tags, or the `ics.CatalogManager` Java method) to modify and maintain the data in your tables.

The following table contains a complete list of the WebCenter Sites generic field types and the database properties (from the `futuretense.ini` file) that define their data types. Refer to this list whenever you create a new table with the WebCenter Sites Management Tools forms, found on the **Admin** tree in the WebCenter Sites interface, or the CatalogManager API:

Table 12–4 Field Types

Field Type	Description	Property
CHAR(n)	A short string of exactly n characters.	cc.char
VARCHAR(n)	A short string of up to n characters. For example, VARCHAR(32) means that this column can hold a string of up to 32 characters.	cc.varchar and cc.maxvarcharsize (The maximum value that you can set for cc.varchar depends on the value of the cc.maxvarcharsize property.)
DATETIME	A date/time combination.	cc.datetime
TEXT	A LONGVARCHAR, a variable-length string of up to 2,147,483,647	cc.bigtext
IMAGE	One binary large object (blob).	cc.blob
SMALLINT	A 16-bit integer, that is, an integer from -32,768 to +32,767.	cc.smallint
INTEGER	A 32-bit integer, that is, an integer from -2,147,483,648 to +2,147,483,647.	cc.integer
BIGINT	A 64-bit integer, that is, integers having up to 19 digits.	cc.bigint
NUMERIC(L,P)	A floating-point (real) number, having a total number of L significant digits of which up to P significant digits are fractional. For example, NUMERIC(5,2) could represent a number such as 806.35 but could not accurately represent a number such as 25693.2283	cc.numeric
DOUBLE	A double precision type.	cc.double

In addition to defining the column type, you must specify which of the following column constraints applies to the column:

Table 12–5 Column Constraints

Constraint	Description
NULL	It can hold a null value, that is, it can be left empty.
NOT NULL	It cannot hold a null value, that is, it cannot be left empty
UNIQUE NOT NULL	It must hold a value that is guaranteed to be unique in this table.
PRIMARY KEY NOT NULL	Marks the primary key column in a content table. You cannot set this column constraint for an object table.

When you use AssetMaker to create an object table for a new asset type or when you create new flex attributes, the data types for those items are different than the ones listed here.

For more information about the data types for columns for basic asset types, see Section 15.1.3.2, "Storage Types for the Columns." For information about the data types for flex attributes, see Section 11.3.3.1, "Data Types for Attributes."

12.2.2 Database-Specific Field Types

You can use database-specific field (column) types in your tables. However, if you use field types that are specific to one kind of DBMS (that is, types that have not been mapped to a JDBC standard), note the following:

- You may not be able to use the CatalogManager API on those tables.
- If you ever change your DBMS you must also modify your tables.

For a complete list of field types specific to the DBMS that you are using, consult your DBMS documentation.

12.2.3 Indirect Data Storage with the WebCenter Sites URL Field

Object and content tables in the WebCenter Sites database have a unique characteristic: columns can store their data indirectly, which means that you can store large bits of data externally to the DBMS but within the data repository.

To create such a column, you must use a column name that begins with the letters `url`. When you use the letters `url` as the first three letters of a column name, WebCenter Sites treats that column as an indirect data column.

Why use a URL field? For the following reasons:

- When the DBMS you are using does not support fields that are large enough to accommodate the size of the data that you want to store there
- If the DBMS you are using does not support enough fields in an individual table to contain the data that you want to store
- Because the performance of selecting data degrades with large field sizes

> **Note:** If the size of the data you wish to store in a URL column exceeds the value set for the `cc.maxvarcharsize` property in the `futuretense.ini` file, then that data is stored in the database, instead of being stored indirectly as a file that is referenced by a pointer in the database.

The Default Storage Directory (defdir)

Any table with a URL column must have a default storage directory specified for it. This directory is where the values entered into the column are actually stored.

The phrase "default storage directory" is shortened to the word **defdir** in several places in the product. For example, the `defdir` column in the `SystemInfo` table holds the name of the default storage directory for tables with URL columns; one of the forms for the AssetMaker utility presents a **defdir** field; and so on.

The value entered into a URL field is actually a relative path to a file. Why a relative path? Because the value in a URL field is appended to the value of the table's defdir setting.

The way that you set the defdir value for the tables that you create depends on the applications you have and what you are doing:

- If you create a new WebCenter Sites table with the WebCenter Sites Management Tools forms, found on the **Admin** tree in the WebCenter Sites user interface, and your table has a URL field, you enter the value for defdir in the **File Storage Directory** field in the "Add Catalog" (table) form.
- If you create a new WebCenter Sites table with the CatalogManager API, you use the `uploadDir` argument to set the value of defdir.
- If you create a new basic asset type, you specify the value of the defdir in the **defdir** field on the AssetMaker form. (Note that all tables that hold basic assets have a URL column and must have a defdir value set.)
- If you create a new flex asset type, you do **not** specify the value of the defdir for the URL column in the flex asset's `_Mungo` table. This value is obtained from a property that was set when your WebCenter Sites application was installed. **Never** change the value of that property.

> **Important:** After a table with a URL column is created, do not attempt to change or modify the `defdir` setting for the table in any way. If you do, you will break the link between the storage directory and the URL column, which means that your data can no longer be retrieved.

For information about creating URL fields, see the following procedures and examples:

- Section 12.3, "Creating Database Tables"
- The upload field examples for basic asset types, starting with Section 15.2.3.3, "Upload Example 1: A Standard Upload Field."
- The procedure for creating flex attributes of type blob in the section Section 16.3.6.2, "Creating Flex Attributes of Type Blob (Upload Field)."

12.3 Creating Database Tables

This section describes how to create object, tree, and content tables and how to register foreign tables (that is, identify them to WebCenter Sites). You cannot create or modify system tables.

This section contains the following topics:

- Section 12.3.1, "Creating Object Tables"
- Section 12.3.2, "Creating Tree Tables"
- Section 12.3.3, "Creating Content Tables"
- Section 12.3.4, "Registering a Foreign Table"

12.3.1 Creating Object Tables

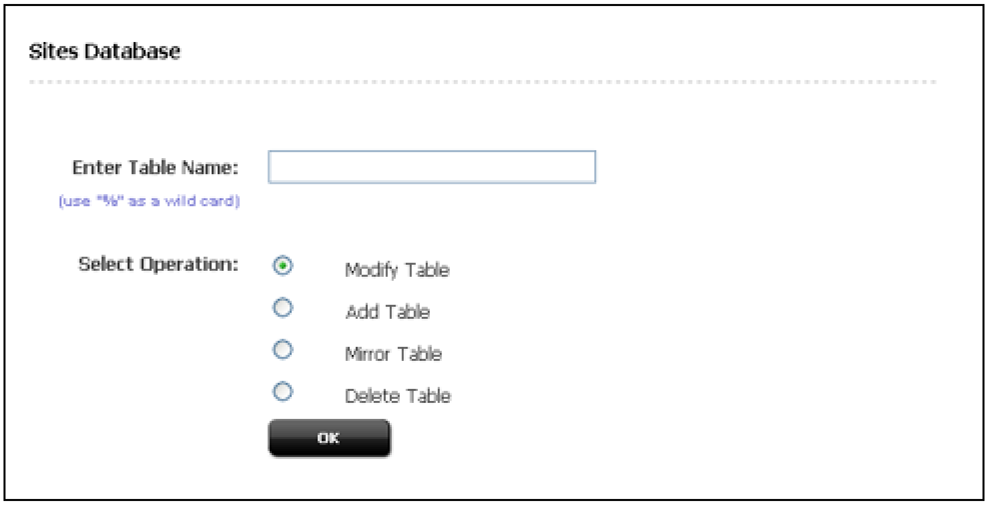
There are three ways to create object tables:

- Create tables that hold basic asset types. You must use AssetMaker, a WebCenter Sites utility located on the **Admin** tab. AssetMaker creates the object table for the asset type as well as the forms that you use to create assets of that type. For more information, see Chapter 15, "Designing Basic Asset Types."

- Create tables that hold flex asset types. You must use Flex Family Maker, a utility located on the **Admin** tab. For more information, see Chapter 16, "Designing Flex Asset Types."
- Create an object table that does **not** hold assets. Use the WebCenter Sites Management Tools, found on the **Admin** interface (or Oracle WebCenter Sites Explorer).

To create an object table that does not hold assets

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Select the **Admin** tab, then select **Management Tools**, and then **Sites Database**.



Sites Database

Enter Table Name:

(use "%" as a wild card)

Select Operation: ● Modify Table ○ Add Table ○ Mirror Table ○ Delete Table

5. In the **Sites Database** window, select **Add Table** and click **OK**.

   The **Add Table** form displays.

6. Click in the **Table Name** field and enter a name. Do not use the name of a table that already exists. You can enter up to 64 alphanumeric characters, including the underscore (_) character but not including spaces.
7. Click in the **Table Type** field and select **Content Table**.
8. If your table will have a URL column (an upload column), click in the **File Storage Directory** field (that is, the defdir) and enter the path to the file directory that will store the data from the URL column. If the directory does not exist yet, WebCenter Sites will create it for you.
9. Click in the **Access Privileges** field to select which ACLs (access control lists) a user must have in order to access this table. For information about ACLs, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.
10. Click in the **Field Name** column and enter the name of the field. Remember that to create a URL column that stores data as a file located in an external directory, you must start the field name with the letters `url`. If you are creating a URL column, be sure that you have specified the file storage directory (defdir) for the data stored in this field (see step 7 of this procedure).

11. Click in the **Field Type** column and specify both the data type and column constraint for the column. Include a space between the data type and the column constraint.

    For example: `VARCHAR(32) NULL` or `INTEGER NOT NULL`.

    For a list of the valid data types and the column constraints, see Section 12.2.1, "Generic Field Types."

12. Repeat steps 9 and 10 for each column in your new table.

13. Click the **Add** button.

    WebCenter Sites adds the table to the database.

To verify that your table has been added, open Oracle WebCenter Sites Explorer and examine the `SystemInfo` table. Your new table should be included in the list with its `systable` column set to `obj`. If you specified a file storage directory, it is listed in the `defdir` column.

Managing Data in Object Tables

There are several ways to modify and manage the data in object tables.

To create and modify **assets**, you use the WebCenter Sites and Engage applications. To extract assets from the database and then display them to the visitors of your delivery system, you use WebCenter Sites, Engage XML and JSP tags.

You can enter data into object tables that do not hold assets in one of the following ways:

- Programmatically, by coding forms with the `ics.CatalogManager` Java method or the `CATALOGMANAGER` XML and JSP tags, the `OBJECT` XML and JSP tags, and the WebCenter Sites SQL methods and tags, that prompt users for information and then to write that information to the database
- Manually by using either the Oracle WebCenter Sites Explorer tool or a form in the WebCenter Sites Management Tools forms to add rows to the table.

Chapter 13, "Managing Data in Non-Asset Tables" presents information about the CatalogManager API and examples of adding rows to tables that do not hold assets.

12.3.2 Creating Tree Tables

If you are using WebCenter Sites modules or products, it is unlikely that you would need to create a tree table.

Tree tables are managed by the TreeManager servlet. To create a tree table (catalog), you use the `ICS.TreeManager` Java method or the `TREEMANAGER` XML or JSP tags. You cannot create a tree table (catalog) with the WebCenter Sites Management Tools.

For example:

```
<TREEMANAGER>
<ARGUMENT NAME="ftcmd" VALUE="createtree"/>
<ARGUMENT NAME="treename" VALUE="ExampleTree"/>
</TREEMANAGER>
```

For a list of the columns that are created for tree tables, see Section 12.1.2, "Tree Tables." For information about the TreeManager methods and tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

12.3.2.1 Managing Data in Tree Tables

The WebCenter Sites modules and products manage all the data in their tree tables. You should never attempt to manually modify information in any of the tree tables.

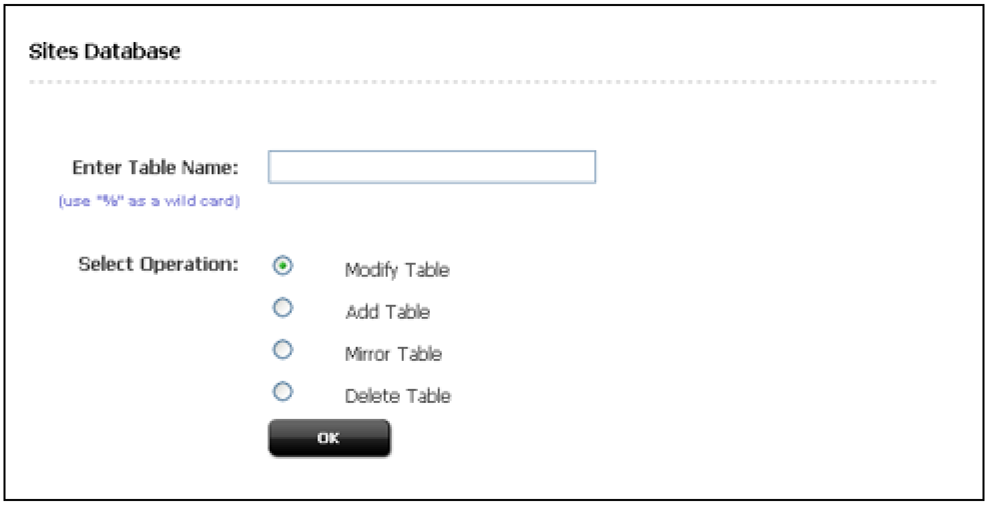
If you have any tree tables that you created to manage relationships between your own object tables (that is, object tables that do not store assets), you use the `ICS.TreeManager` Java method or the `TREEMANAGER` XML or JSP tags. These tags and methods use an `FTValList` parameter, which describes the tree operation to be performed.

The following chapter, Chapter 13, "Managing Data in Non-Asset Tables" presents information about the CatalogManager API and examples of adding rows to tables that do not hold assets.

12.3.3 Creating Content Tables

To create a content table, use the WebCenter Sites Management Tools

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Select the **Admin** tab, then select **Management Tools**, and then **Sites Database**.



Sites Database

Enter Table Name:
(use "%" as a wild card)

Select Operation:
● Modify Table
○ Add Table
○ Mirror Table
○ Delete Table

5. In the **Sites Database** window, select **Add Table** and click **OK**.

   The **Add Table** form displays.

6. Click in the **Table Name** field and enter a name. Do not use the name of a table that already exists. You can enter up to 64 alphanumeric characters, including the underscore (_) character but not including spaces.
7. Click in the **Table Type** field and select **Content Table.**
8. If your table will have a URL column, click in the **File Storage Directory** field and enter the path to the file directory that will store the data from the URL column. If the directory does not exist yet, WebCenter Sites will create it for you.
9. Click in the **Access Privileges** field to select which ACLs (access control lists) a user must have in order to access this table. For information about ACLs, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

10. Click in the **Field Name** field and enter the name of the field. Remember that to create a URL column that stores data as a file located in an external directory, you must start the field name with the letters `url`. If you are creating a URL column, be sure that you have specified the file storage directory (defdir) for the data stored in this field (see step 7 of this procedure).

11. Click in the **Field Type** column and specify both the data type and column constraint for the column. Include a space between the data type and the column constraint.

    For example: `VARCHAR(32) NULL` or `INTEGER NOT NULL`.

    Remember that you must specify a primary key column and that it must exactly match either the setting for the `cc.contentkey` property or a custom property specified for this table in the `futuretense.ini` file.

    For example: `INTEGER PRIMARY KEY NOT NULL`

12. Repeat steps 9 and 10 for each column in your new table.

13. Click the **Add** button.

    WebCenter Sites adds the table to the database.

To verify that your table has been added, open the Oracle WebCenter Sites Explorer tool and examine the `SystemInfo` table. Your new table should be included in the list with the value in its `systable` column set to `no`. If you specified a file storage directory, it is listed in the `defdir` column.

When you add non-asset tables to facilitate some function of your site, you then need to either customize your asset forms in WebCenter Sites or create your own forms to enter and manipulate that data.

Managing Data in Content Tables

There are several ways to modify and manage the data in content tables.

Most of the WebCenter Sites content tables have a WebCenter Sites form available from the Admin tab that you can use to edit or add data. For example, `Source` and `Category`. (`MimeType`, however, does not.)

You can enter data into your custom content tables in one of the following ways:

- If the table was created such that users or visitors supply data that is written into the table, you code forms with the `ics.CatalogManager` Java method or the `CATALOGMANAGER` XML and JSP tags along with the WebCenter Sites SQL methods and tags, to prompt users for information and to then write it to the database programmatically.
- If the table is a simple lookup table that facilitates some function on your site, you enter data into it manually by using either the Oracle WebCenter Sites Explorer utility or the WebCenter Sites Management Tools window to add rows to the table.

Chapter 13, "Managing Data in Non-Asset Tables" presents information about the CatalogManager API and examples of adding rows to tables that do not hold assets.

12.3.4 Registering a Foreign Table

Registering a foreign table means identifying the table to WebCenter Sites by adding a row for the table to the `SystemInfo` table. Note that this is the only condition in which you should ever add a row to the `SystemInfo` table or change information held in the `SystemInfo` table in any way.

To register a foreign table

1. Open Oracle WebCenter Sites Explorer and log in to the WebCenter Sites database.
2. Double-click on the **SystemInfo** table.
3. Right-click the header for the **tblname** column and then select **New** from the context menu.

   A new row appears.
4. In the new row, click in the **tblname** column and enter the name of the table.
5. Click in the **defdir** column and enter the path to the table.
6. Click in the **systable** column and enter `fgn`.
7. Click in the **acl** column and enter the names of the ACLs that have access to the table.
8. Select **File** and then **Save All**.

Managing Data in a Foreign Table

You can use the `ics.CatalogManager` Java method or the `CATALOGMANAGER` XML and JSP tags and the WebCenter Sites SQL methods and tags to interact with a foreign table. When you use these methods or tags to update data in the foreign table, WebCenter Sites can flush its resultset cache as needed.

If you use a method external to WebCenter Sites to update a foreign table, you must be sure to also use the `CATALOGMANGER` command `flushcatalog` to instruct WebCenter Sites to flush the resultset cache for that table.

12.4 How Information Is Added to the System Tables

You cannot create system tables and with very few exceptions, you should always use the WebCenter Sites Management Tools to add rows to the system tables that you are allowed to add rows to. The way that information is added to each system table varies, as described in the following table:

Table 12–6 Methods of Adding Information to System Tables

Table	Method of Adding Information
`SiteCatalog`	There are several ways that page entries are added to this table: ▪ When you create a Template asset, WebCenter Sites automatically creates a page entry for it in the `SiteCatalog` table. ▪ When you create a SiteEntry asset, WebCenter Sites automatically creates a page entry for it in the `SiteCatalog` table. ▪ You can use the Oracle WebCenter Sites Explorer tool or the Site form in the WebCenter Sites Management Tools interface. Note that if you want to set or modify page cache settings for page entries, it is easier to use forms in the WebCenter Sites interface than it is to use Oracle WebCenter Sites Explorer.

Table 12–6 (Cont.) Methods of Adding Information to System Tables

Table	Method of Adding Information
`ElementCatalog`	There are several ways that elements are added to this table: ■ When you create a Template asset, WebCenter Sites automatically creates an entry for it in the `ElementCatalog` table. ■ When you create a CSElement asset, WebCenter Sites automatically creates an entry for it in the `ElementCatalog` table. ■ You can use the Oracle WebCenter Sites Explorer tool to add non-asset elements. For information about coding elements and pages, see Chapter 28, "Coding Elements for Templates and CSElements."
`SystemACL`	The ACL form in the **WebCenter Sites Management Tools** node. For more information, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.
`SystemEvents`	WebCenter Sites adds a row to this table for each event that is designated when an `APPEVENT` tag, `EMAILEVENT` tag, or Java API equivalent is invoked from an element.
`SystemInfo`	Do **not** add or modify information to this table. The **only exception** to this rule is if you need to identify a foreign table to WebCenter Sites.
`SystemSQL`	The Oracle WebCenter Sites Explorer tool. For information about the various kinds of queries that are available, see Chapter 14, "Resultset Caching and Queries."
`SystemUsers`	The User form in the **WebCenter Sites Management Tools** node.
`SystemUserAttr`	The User form in the **WebCenter Sites Management Tools** node.

12.5 Property Files and Databases

There are several properties in the `futuretense.ini` file that control the connection to the WebCenter Sites database. These properties specify the configuration of the database and establish a privileged and non-privileged user connection between the database and the application server.

All of the database properties were configured for your system when your system was installed. By default, all commands operate on the WebCenter Sites database identified in the `futuretense.ini` file. The `futuretense.ini` file is located in this directory:

```
<installation_directory>/ContentServer/11.1.1.6.0/
```

13

Managing Data in Non-Asset Tables

This chapter describes how to interact with WebCenter Sites database tables that do not hold assets.

There are two ways to work with the data in your custom, non-asset tables:

- Programmatically, using the tags and methods for the CatalogManager API to code forms for data entry and management
- Manually, by using the Oracle WebCenter Sites Explorer tool or the Content form in the WebCenter Sites Management Tools to manually add rows and data to those rows.

To work with assets, you must log in to the WebCenter Sites interface and use the asset forms provided by the WebCenter Sites and Engage applications.

To add large numbers of assets programmatically, use the XMLPost utility, as described in Chapter 20, "Importing Assets of Any Type" and Chapter 21, "Importing Flex Assets."

This chapter contains the following sections:

- Section 13.1, "Methods and Tags"
- Section 13.2, "Coding Data Entry Forms"
- Section 13.3, "Managing the Data Manually"
- Section 13.4, "Deleting Non-Asset Tables"

13.1 Methods and Tags

This section provides an overview of the tags and methods that you use to program how you manage data in non-asset tables and how you interact with those tables in general.

This section contains the following topics:

- Section 13.1.1, "Writing and Retrieving Data"
- Section 13.1.2, "Querying for Data"
- Section 13.1.3, "Lists and Listing Data"

13.1.1 Writing and Retrieving Data

CatalogManager is the WebCenter Sites servlet that manages content and object tables in the database and the TreeManager servlet manages tree tables in the database.

- To access the CatalogManager servlet, you can use the `ics.CatalogManager` Java method, the `CATALOGMANAGER` XML tag, or the `ics:catalogmanager` JSP tag.
- To access the TreeManager servlet, you can use the `ics.TreeManager` Java method, the `TREEMANAGER` XML tag, or the `ics:treemanager` JSP tag.

These methods and tags take name/value pairs from arguments that specify the operation to perform and the table to perform that operation on.

13.1.1.1 CatalogManager

The `ics.CatalogManager` java method, the `CATALOGMANAGER` XML tag, and the `ics:catalogmanager` JSP tag support a number of attributes that operate on object and content tables. The key attribute is `ftcmd`. By setting `ftcmd` to `addrow`, for example, you tell CatalogManager to add one row to the catalog.

CatalogManager security, when enabled, prevents users with the `DefaultReader` ACL from accessing CatalogManager. You enable CatalogManager security by setting the `secure.CatalogManager` property, found in the `futuretense.ini` file, to `true`. Note that your session will be dropped if you attempt to log out of CatalogManager when CatalogManager security is enabled.

These are the main `CATALOGMANAGER` XML tag's attributes, passed as argument name/value pairs, that modify the contents of a row or a particular field in a row:

Table 13–1 CATALOGMANAGER XML Tag

argument name="ftcmd" value=	Description
`addrow`	Adds a single row to a table.
`addrows`	Adds more than one row to a table.
`deleterow`	Deletes a row from a table. You must specify the primary key column for the row.
`deleterows`	Deletes more than one row from a table. You must specify the primary key for the rows.
`replacerow`	Deletes the existing row in a table and replaces the row with the specified information.
`replacerows`	Replaces multiple rows in a table. If a value is not specified for a column, the column value is cleared
`updaterow`	Performs a query against a given table and displays records from a table. The rows displayed match the criteria specified by the value of the parameters.
`updaterow2`	Like `updaterow`, updates values in columns for a row in a table; however, where you cannot clear columns with updaterow, `updaterow2` lets you clear columns if there is no value for the specified column (for example, if there is no related field in the form).
`updaterows`	Modifies field values for multiple rows in a table.
`updaterows2`	Like `updaterows`, modifies field values for multiple rows in a table; however, where you cannot clear columns with `updaterows`, `updaterows2` lets you clear columns if there is no value for the specified column (for example, if there is no related field in the form).

Any requests going to CatalogManager with the command parameter (`ftcmd`) must be either a `POST` request or one of the following commands:

- `exportlog`
- `exportForm`
- `logout`
- `selectFromTable`
- `selectCount`
- `mirrorgetconfig`
- `listtables`
- `retrieve`
- `retrievebinary`
- `pingdb`
- `interrogatetbl`
- `checksession`
- `history`
- `retrieverevision`

For more information and a complete list of the CatalogManager commands, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*. For information about the `ics.CatalogManager` Java method, see the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.

13.1.1.2 Tree Manager

Here are the main `ics.TreeManager` commands. Note that these operations manipulate data in the tree table only, but do not affect the objects that the tree table nodes refer to.

Table 13–2 TreeManager Commands

Name	Description
`addchild`	Given a parent node, add a child node.
`addchildren`	Add multiple child nodes.
`copychild`	Copy a node and its children to a different parent. All copied nodes point to the same objects.
`createtree`	Create a tree table.
`delchild`	Delete a node and its child nodes.
`delchildren`	Delete multiple nodes.
`deletetree`	Delete a tree table.
`findnode`	Find a node in a tree.
`getchildren`	Get all child nodes.
`getnode`	Get node and optionally object attributes.
`getparent`	Get the nodes parent.
`listtrees`	Get the list of all tree tables.
`movechild`	Move node and its child nodes to a different parent.
`nodepath`	Return parent; child path to a node.

Table 13–2 (Cont.) TreeManager Commands

Name	Description
`setobject`	Associate a different object with the node.
`validatenode`	Verify that a node is in a tree.
`verifypath`	Verify that a given path exists in a tree.

For information about the `ics.TreeManager` method, see the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.

For information about the XML and JSP `TREEMANAGER` tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

13.1.2 Querying for Data

There are three methods, with XML and JSP tag counterparts, to help your code query for and select content:

Table 13–3 Querying for Data

Method	XML tag	JSP tag	Description
`ics.SelectTo`	`SELECTTO`	`ics:selectto`	Performs a simple select against a single table.
`ics.SQL`	`EXECSQL`	`ics:sql`	Executes an inline SQL statement (embedded in the code).
`ics.CallSQL`	`CALLSQL`	`ics:callsql`	Executes a SQL statement that is stored as a row in the SystemSQL table.

To use `ics.CallSQL` (or the tags), you code SQL statements and then paste them into the `SystemSQL` table.

By storing the actual queries in the `SystemSQL` table and calling them from the individual pages (like you call a pagename or an element), you keep them out of your code, which makes it easier to maintain the SQL used by your site. If you want to change the SQL, you do not have to fix it in every place that you use it, you can just edit it in the `SystemSQL` table and every element that calls it now calls the edited version.

The `ics.CallSQL` and `ics.SQL` methods can execute any legal SQL commands. If a SQL statement does not return a usable list, WebCenter Sites will generate an error. If you choose to use SQL to update or insert data, you must include code that explicitly flushes the resultsets cached against the appropriate tables using the `ics.FlushCatalog` method.

13.1.3 Lists and Listing Data

A number of `ICS` methods create lists. The `SelectTo` method, for example, returns the results of a simple SQL query in a list whose columns reflect the items in the WHAT clause and whose rows reflect matches against the table.

The IList interface can be used to access a list from Java. The lists are available by name using XML or JSP, and values can be iterated using the LOOP tag.

The lists created by WebCenter Sites point to underlying resultsets created from querying the database. Although the lists do not persist across requests, the resultsets do because if are cached.

> **Note:** Be sure to configure resultset caching appropriately. If the resultset of a query is cached, the list points to a copy of the resultset. If the resultset is not cached, the list points directly at the resultset which can cause database connection resource difficulties.

You can create your own list for use in XML or JSP by implementing a class based on the IList interface. Then your application or page can transform data prior to returning an item in a list or to create a single list from many lists.

The following methods manage lists:

Table 13–4 Methods that Manage Lists

Method	Description
`ics.GetList`	Returns an `IList`, given the name of the list.
`ics.CopyList`	Copies a list.
`ics.RenameList`	Renames an existing list.
`ics.RegisterList`	Registers a list by name with WebCenter Sites so that you can reference the list from an XML or JSP element or by using the `GetList` method.

For an example implementation of an IList, see SampleIList.java in the Samples folder on your WebCenter Sites system.

13.2 Coding Data Entry Forms

This section provides code samples that illustrate how to code forms that accept information entered by a user or visitor and to then write that information to the database using the WebCenter Sites methods and tags.

The examples in this section describe adding a new row, deleting a row, and querying for and then editing an existing row. Each example shows a version for XML, JSP, and Java.

This section contains the following topics:

- Section 13.2.1, "Adding a Row"
- Section 13.2.2, "Deleting a Row"
- Section 13.2.3, "Querying a Table"
- Section 13.2.4, "Querying a Table with an Embedded SQL Statement"

13.2.1 Adding a Row

A simple algorithm for adding a row is as follows:

1. Display a form requesting information for each of the fields in a row.
2. Write that form data to the table.

The following example adds a row to a fictitious table named `EmployeeInfo`. This table has the following columns:

Table 13–5 Example Adds Row to Table

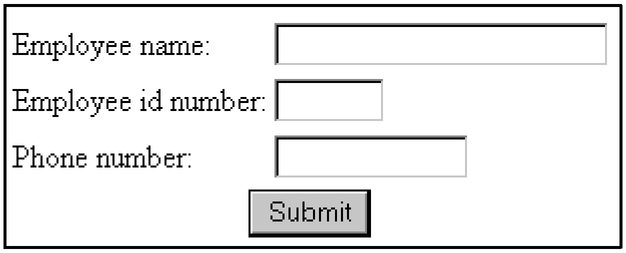
Field	Data type
id	VARCHAR(6)
phone	VARCHAR(16)
name	VARCHAR(32)

This example presents code from the following elements:

- `addrowFORM`, an XML element that displays a form that requests an employee ID number, phone number, and name.
- `addrowXML`, `addrowJSP`, and `addrowJAVA`, three versions of an element that writes the information entered by the employee to the `EmployeeInfo` table

13.2.1.1 The addrowFORM Element

The `addrowFORM` element displays a form that asks the user to enter information. It looks like this:



This is the code that creates the form:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/addrowFORM
-->

<form ACTION="ContentServer" method="post"
REPLACEALL="CS.Property.ft.cgipath">
<input type="hidden" name="pagename" value="Documentation/CatalogManager/addrow"/>

<table>
<tr>
<td>Employee name:</td>
<td><input type="text" value="" name="EmployeeName" size="22"
maxlength="32"/></td>
</tr>
<tr>
<td>Employee id number:</td>
<td><input type="text" value="" name="EmployeeID" size="6" maxlength="6"/></td>
</tr>
<tr>
<td>Phone number:</td>
<td><input type="text" value="" name="EmployeePhone" size="12"
maxlength="16"/></td>
</tr>
```

```
<tr>
<td colspan="2"><input type="submit" name="submit" value="Submit"/></td>
</tr>
</table>

</form>
</FTCS>
```

Notice that the maxlength modifiers in `<INPUT>` limit the length of each input to the maximum length that was defined in the schema.

The user fills in the form and clicks the **Submit** button. The information gathered in the form and the pagename of the `addrow` page (see the first `input type` statement in the preceding code sample) is sent to the browser. The browser sends the pagename to WebCenter Sites. WebCenter Sites looks it up in the `SiteCatalog` table and then invokes that page entry's root element.

13.2.1.2 Root Element for the addrow Page

The root element of the `addrow` page is responsible for adding the information passed from the `addrowFORM` element to the database. That is, for adding a row to the `EmployeeInfo` table and populating that row with the information passed from the `addrowFORM` element.

There can only be one root element for a WebCenter Sites page (that is, an entry in the `SiteCatalog` table). This section shows three versions of the root element for the `addrow` page:

- `addrowXML.xml`
- `addrowJSP.jsp`
- `addrowJAVA.jsp`

addrowXML

This is the code in the XML version of the root element:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/addrowXML
-->

<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
<ARGUMENT NAME="ftcmd" VALUE="addrow"/>
<ARGUMENT NAME="tablename" VALUE="EmployeeInfo"/>
<ARGUMENT NAME="id" VALUE="Variables.EmployeeID"/>
<ARGUMENT NAME="phone" VALUE="Variables.EmployeePhone"/>
<ARGUMENT NAME="name" VALUE="Variables.EmployeeName"/>
</CATALOGMANAGER>
errno=<CSVAR NAME="Variables.errno"/><br/>
</FTCS>
```

Note: The example code can use the `CATALOGMANAGER` tag because the fictitious table, `EmployeeInfo`, has WebCenter Sites generic field types. `addrowXML` might not work if `EmployeeInfo` has database-specific field types. For more information, see Section 12.2.1, "Generic Field Types."

addrowJSP

This is the code in the JSP version of the root element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/addrowJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<ics:setvar name="errno" value="0"/>
<ics:catalogmanager>
<ics:argument name="ftcmd" value="addrow"/>
<ics:argument name="tablename" value="EmployeeInfo"/>
<ics:argument name="id"
value='<%=ics.GetVar("EmployeeID")%>'/>
<ics:argument name="phone"
value='<%=ics.GetVar("EmployeePhone")%>'/>
<ics:argument name="name" value='<%=ics.GetVar("EmployeeName")%>'/>
</ics:catalogmanager>

errno=<ics:getvar name="errno"/><br/>

</cs:ftcs>
```

addrowJAVA

This is the code in the Java version of the root element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/addrowJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<%
ics.SetVar("errno","0");
FTValList vl = new FTValList();
vl.put("ftcmd","addrow");
vl.put("tablename","EmployeeInfo");
vl.put("id",ics.GetVar("EmployeeID"));
vl.put("phone",ics.GetVar("EmployeePhone"));
vl.put("name",ics.GetVar("EmployeeName"));
ics.CatalogManager(vl);
%>
errno=<%=ics.GetVar("errno")%><br />

</cs:ftcs>
```

13.2.2 Deleting a Row

The following example deletes a row from the fictitious EmployeeInfo table as described in Section 13.2.1, "Adding a Row."

This section presents code from the following elements:

- deleterowFORM, an XML element that displays a form that requests an employee name to delete from the EmployeeInfo table
- deleterowXML, deleterowJSP, and deleterowJAVA, elements that delete a row from the EmployeeInfo table based on the information sent to it from the deleterowFORM element

13.2.2.1 The deleterowFORM Element

The deleterowFORM element displays a form that asks the user to enter an employee name. This is the code that creates the form:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/deleterowFORM
-->

<form ACTION="ContentServer" method="post" REPLACEALL="CS.Property.ft.cgipath">
<input type="hidden" name="pagename"
value="Documentation/CatalogManager/deleterow"/>

<table>
<tr>
<td>Employee name:</td>
<td><input type="text" value="Barton Fooman" name="EmployeeName" size="22"
maxlength="32"/></td>
</tr>
<tr>
<td colspan="2"><input type="submit" name="submit" value="submit"/></td>
</tr>
</table>
</form>
</FTCS>
```

The user enters an employee name and clicks the **Submit** button. The employee name and the pagename for the deleterow page (see the first input type statement in the preceding code sample) are sent to the browser.

The browser sends the pagename to WebCenter Sites. WebCenter Sites looks it up in the SiteCatalog table and then invokes that page entry's root element.

13.2.2.2 Root Element for the deleterow Page

The root element of the deleterow page is responsible for deleting a row from the EmployeeInfo table, based on the employee name that is sent to it from the deleterowFORM element.

There can only be one root element for a WebCenter Sites page (that is, an entry in the SiteCatalog table). This section shows three versions of the root element for the deleterow page:

- deleterowXML.xml
- deleterowJSP.jsp

- deleterowJAVA.jsp

deleterowXML

This is the code in the XML version of the element:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/deleterowXML
-->
<SETVAR NAME="errno" VALUE="0"/>

<CATALOGMANAGER>
<ARGUMENT NAME="ftcmd" VALUE="deleterow"/>
<ARGUMENT NAME="tablename" VALUE="EmployeeInfo"/>
<ARGUMENT NAME="tablekey" VALUE="name"/>
<ARGUMENT NAME="tablekeyvalue" VALUE="Variables.EmployeeName"/>
</CATALOGMANAGER>

errno=<CSVAR NAME="Variables.errno"/><br/>
</FTCS>
```

deleterowJSP

This is the code in the JSP version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/deleterowJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<!-- user code here -->
<ics:setvar name="errno" value="0"/>
<ics:catalogmanager>
<ics:argument name="ftcmd" value="deleterow"/>
<ics:argument name="tablename" value="EmployeeInfo"/>
<ics:argument name="name" value='<%=ics.GetVar("EmployeeName")%>'/>
</ics:catalogmanager>

errno=<ics:getvar name="errno"/><br />

</cs:ftcs>
```

deleterowJAVA

This is the code in the Java version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/deleterowJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
```

```
<cs:ftcs>

<%
ics.SetVar("errno","0");
FTValList vl = new FTValList();
vl.put("ftcmd","deleterow");
vl.put("tablename","EmployeeInfo");
vl.put("name",ics.GetVar("EmployeeName"));
ics.CatalogManager(vl);
%>
errno=<%=ics.GetVar("errno")%><br />

</cs:ftcs>
```

13.2.3 Querying a Table

The following sample elements query the fictitious `EmployeeInfo` table for an employee's name, extract the employee name and displays it in a browser, prompts the user to edit the information, and then writes the edited information to the database.

This section presents code from the following elements:

- `SelectNameForm`, an XML element that displays a form that requests an employee's name.
- Three versions of the `QueryEditRowForm` element (XML, JSP, and Java), an element that locates the employee name and loads the information about that employee into a form that the employee can use to edit his or her information
- Three versions of the `QueryEditRow` element (XML, JSP, and Java), an element that writes the newly edited information to the database.

13.2.3.1 The SelectNameForm Element

The `SelectNameForm` element displays a simple form that requests the name of the employee who is altering his employee information. This is the code:

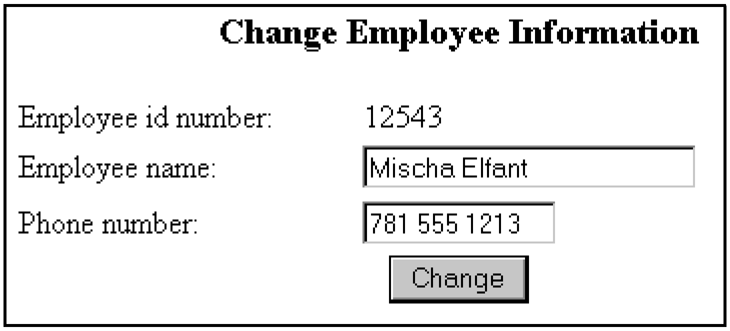
```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/SelectNameForm
-->

<form ACTION="ContentServer" method="post">
<input type="hidden" name="pagename"
value="Documentation/CatalogManager/QueryEditRowForm"/>
<TABLE>
<TR>
<TD>Employee name: </TD>
<TD><INPUT type="text" value="" name="EmployeeName" size="22"
maxlength="32"/></TD>
</TR>
<TR>
<TD COLSPAN="100%" ALIGN="CENTER">
<input type="submit" name="doit" value="Submit"/></TD>
</TR>
</TABLE>
</form>
</FTCS>
```

When the employee clicks the **Submit** button, the information gathered in the **Employee Name** field and the name of the QueryEditRowForm page (see the first input type statement in the preceding code sample) is sent to the browser.

The browser sends the pagename to WebCenter Sites. WebCenter Sites looks up the pagename in the SiteCatalog table, and then invokes that page entry's root element, QueryEditRowForm.

13.2.3.2 The Root Element for the QueryEditRowForm Page

The root element for the QueryEditRowForm page locates the row in the EmployeeInfo table that matches the string entered in the **Employee Name** field and then loads the data from that row into a new form. The employee can edit her name and phone number but cannot edit her id. The form looks like this:



Change Employee Information

Employee id number:	12543
Employee name:	Mischa Elfant
Phone number:	781 555 1213

There can only be one root element for a WebCenter Sites page (that is, an entry in the SiteCatalog table). This section shows three versions of the root element for the QueryEditRowForm page:

- QueryEditRowFormXML.xml
- QueryEditRowFormJSP.jsp
- QueryEditRowFormJAVA.jsp

QueryEditRowFormXML

This is the code in the XML version of the element:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/QueryEditRowFormXML
-->

<SETVAR NAME="errno" VALUE="0"/>
<SETVAR NAME="name" VALUE="Variables.EmployeeName"/>
<SELECTTO FROM="EmployeeInfo"
WHERE="name"
WHAT="*"
LIST="MatchingEmployees"/>

<IF COND="Variables.errno=0">
<THEN>
<form ACTION="ContentServer" method="post">
<input type="hidden" name="pagename"
value="Documentation/CatalogManager/QueryEditRow"/>
<input type="hidden" name="MatchingID" value="MatchingEmployees.id"
REPLACEALL="MatchingEmployees.id"/>
<TABLE>
```

```
<TR>
<TD COLSPAN="100%" ALIGN="CENTER">
<H3>Change Employee Information</H3>
</TD>
</TR>
<TR>
<TD>Employee id number: </TD>
<TD><CSVAR NAME="MatchingEmployees.id"/></TD>
</TR>
<TR>
<TD>Employee name: </TD>
<TD><INPUT type="text" value="MatchingEmployees.name" name="NewEmployeeName"
size="22" maxlength="32" REPLACEALL="MatchingEmployees.name"/></TD>
</TR>
<TR>
<TD>Phone number: </TD>
<TD><INPUT type="text" value="MatchingEmployees.phone" name="NewEmployeePhone"
size="12" maxlength="16" REPLACEALL="MatchingEmployees.phone"/></TD>
</TR>
<TR>
<TD colspan="100%" align="center">
<input type="submit" name="doit" value="Change"/></TD>
</TR>
</TABLE>
</form>
</THEN>
<ELSE>
<P>Could not find this employee.</P>
<CALLELEMENT NAME="Documentation/CatalogManager/SelectNameFormXML"/>
</ELSE>
</IF>
</FTCS>
```

When the employee clicks the **Change** button, the information gathered from the two fields and the name of the `QueryEditRow` page is sent to the browser.

The browser sends the pagename and the field information to WebCenter Sites. WebCenter Sites looks up the pagename in the `SiteCatalog` table, and then invokes that page entry's root element.

QueryEditRowFormJSP

This is the code in the JSP version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/QueryEditRowFormJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<ics:setvar name="errno" value="0"/>
<ics:setvar name="name" value='<%=ics.GetVar("EmployeeName")%>'/>
<ics:selectto table="EmployeeInfo"
where="name"
what="*"
listname="MatchingEmployees"/>
```

```
<ics:if condition='<%=ics.GetVar("errno").equals("0")%>'>
<ics:then>
<form action="ContentServer" method="post">
<input type="hidden" name="pagename"
value="Documentation/CatalogManager/QueryEditRow"/>
<input type="hidden" name="MatchingID" value="<ics:listget
listname='MatchingEmployees' fieldname='id'/>"/>
<TABLE>
<TR>
<TD COLSPAN="100%" ALIGN="CENTER">
<H3>Change Employee Information</H3>
</TD>
</TR>
<TR>
<TD>Employee id number: </TD>
<TD><ics:listget listname='MatchingEmployees
fieldname='id'/></TD>
</TR>
<TR>
<TD>Employee name: </TD>
<TD><INPUT type="text" value="<ics:listget
listname='MatchingEmployees' fieldname='name'/>"
name="NewEmployeeName" size="22" maxlength="32"/></TD>
</TR>
<TR>
<TD>Phone number: </TD>
<TD><INPUT type="text" value="<ics:listget
listname='MatchingEmployees' fieldname='phone'/>"
name="NewEmployeePhone" size="12" maxlength="16"/>
</TD>
</TR>
<TR>
<TD colspan="100%" align="center">
<input type="submit" name="doit" value="Change"/></TD>
</TR>
</TABLE>
</form>
</ics:then>
<ics:else>
<P>Could not find this employee.</P>
<ics:callelement element="Documentation/CatalogManager/
SelectNameForm"/>
</ics:else>
</ics:if>

</cs:ftcs>
```

When the employee clicks the **Change** button, the information gathered from the two fields and the name of the `QueryEditRow` page is sent to the browser.

The browser sends the pagename and the field information to WebCenter Sites. WebCenter Sites looks up the pagename in the `SiteCatalog` table, and then invokes that page entry's root element.

QueryEditRowFormJAVA

This is the code in the Java version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/QueryEditRowFormJAVA
```

```
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<%
ics.SetVar("errno","0");
ics.SetVar("name",ics.GetVar("EmployeeName"));
StringBuffer errstr = new StringBuffer();
IList matchingEmployees = ics.SelectTo("EmployeeInfo",// tablename
*", // what
"name", // where
"name", // orderby
1, // limit
null, // ics list name
true, // cache?
errstr); // error StringBuffer

if ("0".equals(ics.GetVar("errno")) && matchingEmployees!=null &&
matchingEmployees.hasData())
{
%>
<form action="ContentServer" method="post">
<input type="hidden" name="pagename"
value="Documentation/CatalogManager/QueryEditRow"/>
<%
String id = matchingEmployees.getValue("id");
String name = matchingEmployees.getValue("name");
String phone = matchingEmployees.getValue("phone");
%>
<input type="hidden" name="MatchingID" value="<%=id%>"/>
<TABLE>
<TR>
<TD COLSPAN="100%" ALIGN="CENTER">
<H3>Change Employee Information</H3>
</TD>
</TR>
<TR>
<TD>Employee id number: </TD>
<TD><%=id%></TD>
</TR>
<TR>
<TD>Employee name: </TD>
<TD><INPUT type="text" value="<%=name%>" name="NewEmployeeName" size="22"
maxlength="32"/></TD>
</TR>
<TR>
<TD>Phone number: </TD>
<TD><INPUT type="text" value="<%=phone%>" name="NewEmployeePhone" size="12"
maxlength="16"/></TD>
</TR>
<TR>
<TD colspan="100%" align="center">
<input type="submit" name="doit" value="Change"/></TD>
</TR>
</TABLE>
</form>
<%
}
else
{
%><P>Could not find this employee.</P>
<%
ics.CallElement("Documentation/CatalogManager/SelectNameForm",null);
}
%>
</cs:ftcs>
```

When the employee clicks the **Change** button, the information gathered from the two fields and the name of the `QueryEditRow` page is sent to the browser.

The browser sends the pagename and the field information to WebCenter Sites. WebCenter Sites looks up the pagename in the `SiteCatalog` table, and then invokes that page entry's root element.

13.2.3.3 The Root Element for the QueryEditRow Page

The root element for the `QueryEditRow` page writes the information that the employee entered into the **Employee Name** and **Phone number** fields and updates the row in the database.

There can only be one root element for a WebCenter Sites page (that is, an entry in the `SiteCatalog` table). This section shows three versions of the root element for the `QueryEditRow` page:

- `QueryEditRowXML.xml`
- `QueryEditRowJSP.jsp`
- `QueryEditRowJAVA.jsp`

QueryEditRowXML

This is the code in the XML version of the element:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/QueryEditRowXML
-->

<SETVAR NAME="errno" VALUE="0"/>

<CATALOGMANAGER>
<ARGUMENT NAME="ftcmd" VALUE="updaterow"/>
<ARGUMENT NAME="tablename" VALUE="EmployeeInfo"/>
<ARGUMENT NAME="id" VALUE="Variables.MatchingID"/>
<ARGUMENT NAME="name" VALUE="Variables.NewEmployeeName"/>
<ARGUMENT NAME="phone" VALUE="Variables.NewEmployeePhone"/>
</CATALOGMANAGER>

<IF COND="Variables.errno=0">
<THEN>
<P>Successfully updated the database.</P>
</THEN>
<ELSE>
<P>Failed to update the information in the database.</P>
</ELSE>
</IF>
</FTCS>
```

QueryEditRowJSP

This is the code in the JSP version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/QueryEditRowJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<ics:setvar name="errno" value="0"/>

<ics:catalogmanager>
<ics:argument name="ftcmd" value="updaterow"/>
<ics:argument name="tablename" value="EmployeeInfo"/>
<ics:argument name="id" value="<%=ics.GetVar("MatchingID")%>"/>
<ics:argument name="name"
value='<%=ics.GetVar("NewEmployeeName")%>'/>
<ics:argument name="phone"
value='<%=ics.GetVar("NewEmployeePhone")%>'/>
</ics:catalogmanager>

<ics:if condition='<%=ics.GetVar("errno").equals("0")%>'>
<ics:then>
<P>Successfully updated the database.</P>
</ics:then>
<ics:else>
<p>failed to update the information in the database. errno=<ics:getvar
name='errno'/></p>
</ics:else>
</ics:if>

</cs:ftcs>
```

QueryEditRowJAVA

This is the code in the Java version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/QueryEditRowJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<%
ics.SetVar("errno","0");

FTValList args = new FTValList();
args.put("ftcmd","updaterow");
args.put("tablename","EmployeeInfo");
args.put("id",ics.GetVar("MatchingID"));
args.put("name",ics.GetVar("NewEmployeeName"));
args.put("phone",ics.GetVar("NewEmployeePhone"));

ics.CatalogManager(args);

if("0".equals(ics.GetVar("errno")))
{
%><P>Successfully updated the database.</P><%
}
else
{
%><p>failed to update the information in the database. errno=<ics:getvar
name='errno'/></p><%
}
%>
</cs:ftcs>
```

13.2.4 Querying a Table with an Embedded SQL Statement

The following example shows another method of searching for a name in a table. This example also searches the fictitious EmployeeInfo table, returning the rows that match the string supplied by a user, but this time the code uses a SQL query rather than a SELECTTO statement.

This section presents code from the following elements:

- QueryInlineSQLForm, an XML element that displays a form that requests a movie title
- Three versions of the QueryInlineSQL element (XML, JSP, and Java), an element that searches the EmployeeInfo table for names that contain the string entered by the user in the preceding form

13.2.4.1 QueryInlineSQLForm

The QueryInlineSQL element displays a simple form that requests the name to use to search the EmployeeInfo table for. This is the code:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/QueryInlineSQLForm
-->
<form ACTION="ContentServer" method="post">
<input type="hidden" name="pagename"
value="Documentation/CatalogManager/QueryInlineSQL"/>

<table>
<tr>
<td>Employee Name:</td>
<td><input type="text" value="Foo,Bar" name="EmployeeName" size="22"
maxlength="32"/></td>
</tr>
<tr>
<td colspan="2"><input type="submit" name="submit" value="submit"/></td>
</tr>
</table>

</form>
</FTCS>
```

When the user clicks the **Submit** button, the information gathered in the **Employee Name f**ield and the name of the `QueryInlineSQL` page is sent to the browser.

The browser sends the pagename of the `QueryInlineSQL` page to WebCenter Sites. WebCenter Sites looks up the pagename in the `SiteCatalog` table, and then invokes that page entry's root element.

13.2.4.2 The Root Element for the QueryInlineSQL Page

The root element for the `QueryInlineSQL` page executes an inline SQL statement that searches the `EmployeeInfo` table for entries that match the string sent to it from the `QueryInlineSQLForm` element.

There can only be one root element for a WebCenter Sites page (that is, an entry in the `SiteCatalog` table). This section shows three versions of the root element for the `QueryInlineSQL` page:

- `QueryInlineSQLXML.xml`, which uses the `EXECSQL` XML tag to create the SQL query
- `QueryInlineSQLJSP.jsp`, which uses the `ics:sql` JSP tag to create the SQL query
- `QueryInlineSQLJAVA.jsp`, which uses the `ics.CallSQL` Java method to create the SQL query

QueryInlineSQLXML

This is the code in the XML version of the element:

```
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/QueryInlineSQLXML
-->

<SETVAR NAME="tablename" VALUE="EmployeeInfo"/>

<SQLEXP OUTSTR="MySQLExpression"
TYPE="OR"
VERB="LIKE"
STR="Variables.EmployeeName"
COLNAME="name"/>

<EXECSQL
SQL="SELECT id,name,phone FROM Variables.tablename WHERE
Variables.MySQLExpression"
LIST="ReturnedList"
LIMIT="5"/>

<table border="1" bgcolor="99ccff">
<tr>
<th>id</th>
<th>name</th>
<th>phone</th>
</tr>

<LOOP LIST="ReturnedList">
<tr>
<td><CSVAR NAME="ReturnedList.id"/></td>
<td><CSVAR NAME="ReturnedList.name"/></td>
<td><CSVAR NAME="ReturnedList.phone"/></td>
</tr>
</LOOP>
```

```
</table>

</FTCS>
```

Notice that the SQL statement is not actually embedded in the EXECSQL tag. Instead, a preceding SQLEXP tag creates a SQL expression which is passed as an argument to the EXECSQL call. The EXECSQL tag performs the search and returns the results to the list variable named ReturnedList.

Also notice that the first line of code in the body of the element creates a variable named tablename and sets the value to EmployeeInfo, the name of the table that is being queried. This enables CatalogManager to cache the resultset against the correct table.

QueryInlineSQLJSP

This is the code in the JSP version of the element:

```
<?xml version="1.0" ?>
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/QueryInlineSQLJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<ics:setvar name="tablename" value="EmployeeInfo"/>

<%
// no ics:sqlexp tag, must do in java
String sqlexp =
ics.SQLExp("EmployeeInfo","OR","LIKE",ics.GetVar("EmployeeName"),"name");
String sql = "SELECT id,name,phone FROM "+ics.GetVar("tablename")+" WHERE
"+sqlexp;
%>
<ics:sqltable='<%=ics.GetVar("tablename")%>'
sql='<%=sql%>'
listname="ReturnedList"
limit="5"/>

<table border="1" bgcolor="99ccff">
<tr>
<th>id</th>
<th>name</th>
<th>phone</th>
</tr>

<ics:listloop listname="ReturnedList">
<tr>
<td><ics:listget listname="ReturnedList" fieldname="id"/></td>
<td><ics:listget listname="ReturnedList" fieldname="name"/></td>
<td><ics:listget listname="ReturnedList" fieldname="phone"/></td>
</tr>
</ics:listloop>

</table>
```

```
</cs:ftcs>
```

Notice that the SQL statement is not actually embedded in the `ics:sql` tag. Instead, a preceding Java expression creates a SQL expression that is passed as an argument to the `ics:sql`call. (The code example uses Java because there is no JSP equivalent of the `SQLEXP` tag.) The `ics:sql` tag performs the search and returns the results to the list variable named `ReturnedList`.

Also notice that the first line of code in the body of the element creates a variable named `tablename` and sets the value to `EmployeeInfo`, the name of the table that is being queried. This enables CatalogManager to cache the resultset against the correct table.

QueryInlineSQLJava

This is the code in the Java version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/QueryInlineSQLJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<%

ics.SetVar("tablename","EmployeeInfo");

String sqlexp =
ics.SQLExp(ics.GetVar("tablename"),"OR","LIKE",ics.GetVar("EmployeeName"),"name");
String sql = "SELECT id,name,phone FROM "+ics.GetVar("tablename")+" WHERE
"+sqlexp;
StringBuffer errstr = new StringBuffer();

IList list = ics.SQL(ics.GetVar("tablename"),sql,null,5,true,errstr);

%>

<table border="1" bgcolor="99ccff">
<tr>
<th>id</th>
<th>name</th>
<th>phone</th>
</tr>

<%

while (true)
{
%>
<tr>
<td><%=list.getValue("id")%></td>
<td><%=list.getValue("name")%></td>
<td><%=list.getValue("phone")%></td>
</tr>
<%
if (list.currentRow() == list.numRows())
break;
list.moveTo(list.currentRow()+1);
}
%>

</table>
</cs:ftcs>
```

Notice that the SQL statement is not actually embedded in the `ics.SQL` statement. Instead, a preceding ics.SQLExp statement creates a SQL expression which is passed as an argument to the `EXECSQL` call. The `ics.SQL` statement performs the search and returns the results to the list variable named `ReturnedList`.

Also notice that this code also creates a variable named `tablename` and sets the value to `EmployeeInfo` (the name of the table that is being queried), before the code for the query. This enables CatalogManager to cache the resultset against the correct table.

13.3 Managing the Data Manually

You can add data to a table manually with either the Oracle WebCenter Sites Explorer tool or the forms in the WebCenter Sites Management Tools.

Oracle WebCenter Sites Explorer is the right choice in the following situations:

- If you are creating a page entry for a new page in the `SiteCatalog` table.
- If you are creating a row for an element in the `ElementCatalog` table and are coding that element with the editor in Oracle WebCenter Sites Explorer.
- If you need to add a small amount of data to a table that you have created to support some function of your site. That is, to add a small amount of data to a table that does not hold assets. (For example, to add rows to the `MimeType` table.)

Oracle WebCenter Sites Explorer has online help that you can use if you need information about adding, editing, or deleting rows. Additionally, Chapter 4, "Programming with Oracle WebCenter Sites" describes how to add page entries to the `SiteCatalog` table and elements to the `ElementCatalog` table.

The WebCenter Sites Management Tools are the right choice in the following situations:

- To add users or ACLs to the system.
- When you want to modify the cache settings for a page entry in the `SiteCatalog` table. Typically it is easier to complete this task in the ContentManagement form than it is to enter the information directly into the column using Oracle WebCenter Sites Explorer.

The WebCenter Sites Management Tools are documented in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

13.4 Deleting Non-Asset Tables

Note that if you delete a non-asset table that is being revision tracked from the database, the tracking table will not be removed. To prevent this, be sure that you disable revision tracking for the table before deleting it.

14 Resultset Caching and Queries

The CatalogManager servlet (and its API) maintains the resultset cache on your WebCenter Sites systems. This chapter describes resultset caching and how to create queries that allow CatalogManager to accurately cache resultsets and to then flush those resultsets from the cache. You or your system administrators set up resultset caching on all three systems (development, management, and delivery).

Whenever the database is queried, WebCenter Sites serves a resultset, either a cached resultset or an uncached resultset. Resultset caching reduces the load on your database and improves the response time for queries.

The `futuretense.ini` file provides global properties that set the size and timeout periods for all resultsets. You can add table-specific properties to the `futuretense.ini` file that override the default settings on a table-by-table basis. These custom properties enable you to fine-tune your systems for peak performance.

This chapter contains the following sections:

14.1 Caching Frameworks

By default, WebCenter Sites stores resultsets in the inCache framework. You have the option to switch to caching in hash tables, as described in "Section 14.7, "Switching Between Caching Frameworks."

When resultset caching over inCache is enabled, the **System Tools** node (on the **Admin** tab of the Admin interface) displays the resultset over inCache tool, which provides statistical information about resultset caches and their contents. Note that resultset caching over inCache functions independently of page and asset caching over inCache. For comprehensive information about the inCache framework, its caching models, and system tools, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

14.2 Database Queries

There are several ways to query the WebCenter Sites database for information. For example:

- With the `ics.SelectTo` Java method,`SELECTTO` XML tag, or `ics:selectto` JSP tag
- With the `selectrow` command of the `ics.CatalogManager` Java method, the `CATALOGMANAGER` XML tag, and the `ics:catalogmanager` JSP tag
- With the `ics.SQL` Java method, `EXECSQL` XML tag, or `ics:sql` JSP tag
- With the `ics.CallSQL` Java method, `CALLSQL` XML tag, or `ics:callsql` JSP tag
- Through the Search forms in the WebCenter Sites interface
- With a query asset
- With a `SEARCHSTATE` XML or JSP tag (flex assets only)

14.3 How Resultset Caching Works

When the database is queried, the resultset from the query is cached if resultset caching is enabled (by the properties described in Section 14.8.2, "Default Properties" and Section 14.8.3, "Table-Specific Properties"). Then, if someone runs the same query and the data in the table remains unchanged since the last time the query was run, WebCenter Sites serves the information from the resultset cache rather than querying the database again. Serving a resultset from cache is always faster than performing another database lookup.

The resultset cache is either a hash table or the inCache framework, depending on how the `rsCacheOverInCache` property in `futuretense.ini` is configured. The resultsets are organized by the name of the table that was associated with the query that generated the resultset. In other words, resultsets are cached against a table name.

Each time a table is updated (from either the WebCenter Sites interface or through a CatalogManager command in your custom elements), all the resultsets in the cache for that table are flushed. Resultsets are cached in the context of a single Java VM. Although Java VMs do not share resultsets, WebCenter Sites sends a signal to all the Java VMs in a cluster to flush the resultsets when they become invalid, as long as the synchronization feature has been enabled on all servers in the cluster.

14.4 Reducing the Load on the Database

Resultset caching reduces the load on your database in two ways:

- Serving a cached resultset does not open a database connection. WebCenter Sites attempts to obtain a resultset from the cache before it contacts the database. If the correct resultset exists, no contact is made with the database.
- When resultset caching is enabled but the appropriate resultset is not cached, WebCenter Sites obtains the resultset, stores it in the cache as an object, and then releases the database connection.

  When resultset caching is not enabled, WebCenter Sites cannot close the database connection until either the online page is completely rendered or the uncached resultset is explicitly flushed from the scope with a `flush` tag. When this occurs, your available database connections can be quickly used up (even on a relatively simple page).

As a general rule, resultset caching should be enabled for all of your database tables. Although there are times when you might need to limit either the number of resultsets that are cached or the length of time that they are cached for, it is rarely a good idea to disable resultset caching altogether.

Note: Never disable resultset caching on the ElementCatalog table. If you do, the performance of your system will suffer greatly, especially if you are using JSP in any of your elements.

14.5 Specifying the Table Name

There must always be a table name associated with a query so that the resultset can be cached against that table. Then, whenever that table is updated through the WebCenter Sites interface or your own custom elements, CatalogManager flushes all the resultsets associated with that table.

The way that the table name is specified for a resultset depends on the type of query you are running. The following sections describe the most commonly used methods for querying the database and how you specify the table name for such a query.

This section contains the following topics:

- Section 14.5.1, "SELECTTO"
- Section 14.5.2, "EXECSQL"
- Section 14.5.3, "CALLSQL"
- Section 14.5.4, "Search Forms in the WebCenter Sites Interface"
- Section 14.5.5, "Query Asset"
- Section 14.5.6, "SEARCHSTATE"

14.5.1 SELECTTO

When you use the ics.SelectTo Java method, SELECTTO XML tag, or ics:selectto JSP tag, you must specify the name of the table with a FROM parameter (clause). For example:

```
<SELECTTO FROM="EmployeeInfo"
         WHERE="name"
         WHAT="*"
         LIST="MatchingEmployees"/>
```

In this case, EmployeeInfo is the name of the table that is being queried and is the name of the table that the resultset is cached against. Whenever the EmployeeInfo table is updated, CatalogManager flushes all the resultsets cached against it.

14.5.2 EXECSQL

EXECSQL lets you execute an inline SQL statement. You specify the table or tables that you want to cache the resultset against using the TABLE parameter. If you specify multiple tables (by using a comma-separated list), the resultset will be cached against the first table in the list. Note that this means the resultset will be cached based on the resultset cache settings specified for the first table, including timeout and maximum size.

CatalogManager deletes outdated resultsets as the specified tables are updated.

For example, the following query caches the resultset against the article table:

```
<EXECSQL SQL="SELECT article.headline, images.imagefile FROM article,images WHERE
article.id='FTX1EE17FWB' AND images.id='FTK9384FWW'" LIST="sqlresult"
TABLE="article,images"/>
```

14.5.3 CALLSQL

When you use the `ics.CallSQL` Java method, `CALLSQL` XML tag, or `ics:callsql` JSP tag to invoke a SQL query that is stored in the `SystemSQL` table, the table name is set by the query's entry (row) in the `SystemSQL` table.

The `SystemSQL` table has a `deftable` column that identifies the table name that the resultset from the query should be cached against. You can specify multiple tables by putting a comma-separated list of tables in the `deftable` column. The first table in the list is the table that the query is cached against.

Each query stored in the table must have a value in the `deftable` column. If it does not, CatalogManager cannot store the resultsets accurately, which means they cannot be flushed when it is necessary. Note that the table name must identify an existing table. If you enter the name of a table that does not exist yet or if you misspell the name of the table, the resultset cannot be cached correctly.

14.5.4 Search Forms in the WebCenter Sites Interface

The Search forms that you use to look for assets in the WebCenter Sites interface search by asset type. The resultsets from the search form queries are stored against the primary storage table for assets of that type.

For example, for the Burlington Financial sample site asset named article, those resultsets are cached against the `Article` table; for page assets, it is the `Page` table; and so on.

14.5.5 Query Asset

Query assets can return assets of one type only. When you create a query asset, you specify what kind of asset the query asset returns in the **Result of Query** field: articles, or imagefiles and so on.

When that query asset is used on a page in the online site, WebCenter Sites stores the resultset against the table name of the primary storage table for the asset type that the query asset returns: `Article` or `Imagefile`, and so on.

14.5.6 SEARCHSTATE

The `SEARCHSTATE` XML and JSP tags create a set of search constraints that are applied to a list or set of flex assets (created with the `ASSETSET` tags). A constraint can be either a filter (restriction) based on the value of an attribute or based on another searchstate (called a nested searchstate).

You use the `SEARCHSTATE` and `ASSETSET` tags to extract and display flex assets or flex parent assets (not definitions or flex attributes) on your online pages for your visitors.

WebCenter Sites caches the resultsets of searchstates against the `_Mungo` table for the flex asset type. For example, if the searchstate returns the GE Lighting sample site flex asset named product, the resultset is cached against the `Products_Mungo` table.

When you configure the delivery system, be sure to add resultset caching properties for all of your `_Mungo` tables.

14.6 Flushing the Resultset Cache

In most cases, data is written to the database through the CatalogManager API, which flushes the resultset cache when it is appropriate to do so. For example:

- If you use WebCenter Sites Explorer to add a row to a table (the `SiteCatalog` table or the `ElementCatalog` table, for example), CatalogManager flushes all the resultsets cached against that table.
- If you use a form in the WebCenter Sites interface to add or edit an asset, a source, a category, a workflow process, a user, an ACL and so on, CatalogManager flushes the resultsets cached against the tables that are written to.
- If you use CatalogManager commands in an element of your own to update a single table, Catalog Manager automatically flushes the resultsets cached against that table.
- If you use CatalogManager commands in an element of your own to update multiple (joined) tables, Catalog Manager automatically flushes the resultsets cached against the joined tables.
- If you use the `CALLSQL` tag to execute a SQL statement that is stored in the `SystemSQL` table, Catalog Manager automatically updates the resultsets cached against the table or tables specified in the `deftable` column.

14.7 Switching Between Caching Frameworks

Resultset caching over inCache is enabled when the following conditions are met:

1. The `linked-cache.xml` configuration file is placed in the application server's classpath (`WEB-INF/classes` directory).
2. The `rsCacheOverInCache` property (in `futuretense.ini`) is set to `true`.

You can then switch between the inCache and hash table frameworks by setting the `rsCacheOverInCache` property to either `true` or `false`.

14.8 Configuring Resultset Caching

This section describes the process of planning and using resultset caching properties for all tables and specific tables.

This section contains the following topics:

- Section 14.8.1, "Planning Your Resultset Caching Strategy"
- Section 14.8.2, "Default Properties"
- Section 14.8.3, "Table-Specific Properties"

14.8.1 Planning Your Resultset Caching Strategy

Before you configure resultset caching for your database, create a spreadsheet of all the tables in your WebCenter Sites database, assemble a team of developers and database administrators, and discuss what the settings should be for all of your systems (development, management, testing, and delivery). One strategy is to identify a large group of similar tables for which you can use the default properties, and then add table-specific properties for the exceptions. To tune your delivery system for the best performance possible, however, it is likely that you will create custom properties for each table in the database on that system, at the very least, 50 to 100 of them.

Note: If you set the `com.fatwire.logging.cs.cache.resultset` property, debugging messages about the resultset cache are written to the WebCenter Sites log file. (Set the property in either the `commons-logging.properties` file, or in `log4j.properties`, depending on which logging framework you are using.)

14.8.2 Default Properties

Table 14–1 describes resultset caching properties in `futuretense.ini` that are assigned to *all* tables. The properties control the tables' resultset caches as long as no table-specific caching properties are assigned to the tables. The same properties are valid for resultset caching in both inCache and hash tables. To change these properties, open the `futuretense.ini` file with the Property Editor utility and modify them. For information about using the Property Editor, see Chapter 8, "WebCenter Sites Tools and Utilities."

Table 14–1 Default Properties That Control the Resultset Cache

property	description
`cc.cacheResults`	Specifies the default number of resultsets to cache in memory. Note that this does not mean the number of records in a resultset, but the number of resultsets. **Caution:** Unless you are debugging, do **not** set this property to `0` or `-1`. If you do, the WebCenter Sites interface will fail to save assets properly. (Setting this property to `0` or `-1` disables resultset caching for all tables that do not have their own caching properties configured.)
`cc.cacheResultsTimeout`	Specifies the number of minutes to keep a resultset cached in memory. Setting this property to `-1` means there is no timeout value for tables that do not have their own caching properties configured.
`cc.cacheResultsAbs`	Specifies how expiration time for resultsets in the resultset cache is calculated. ■ If this property is set to `true`, the expiration time for a resultset is absolute. For example, if `cc.cacheResultsTimeout` is set to 5 minutes, then 5 minutes after the resultset was cached, it is flushed from the cache. ■ If this property is set to `false`, the expiration time for a resultset is based on its idle time. For example, if `cc.cacheResultsTimeout` is set to 5 minutes, the resultset is flushed from the cache 5 minutes after the last time it was requested rather than 5 minutes since it was originally cached.

14.8.3 Table-Specific Properties

Table-specific properties override the default properties and enable you to fine-tune your systems for peak performance.

CatalogManager uses the default properties described in Table 14–1 and checks the `futuretense.ini` file to determine if it contains any table-specific resultset caching properties.

You can create three resultset caching properties for each table in the WebCenter Sites database. Table-specific properties work in the same way as the default properties (described in Table 14–1).

Syntax for table-specific properties is the following:

```
cc.<tablename>CSz=<number of resultsets>
cc.<tablename>Timeout=<number of minutes>
cc.<tablename>Abs=<true or false>
```

Note: If an asset type is enabled for revision tracking, and you wish to cache the resultsets of asset versions, use the properties above, but add `_t` after `<tablename>`:

```
cc.<tablename>_tCSz=<number of resultsets>
cc.<tablename>_tTimeout=<number of minutes>
cc.<tablename>_tAbs=<true or false>
```

For more information about resultset caching, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

Open the `futuretense.ini` file in the Property Editor utility and add table-specific properties for each table that you want to control. For information about using the Property Editor utility, see Chapter 8, "WebCenter Sites Tools and Utilities."

14.9 Summary

Resultset caching reduces the load on your database and improves the response time for queries. Be sure to do the following:

- Set the default resultset caching properties in the `futuretense.ini` file to values that make sense on each of your systems (development, management, testing, and delivery).
- Add table-specific resultset caching properties to the `futuretense.ini` file to fine-tune the performance of all of your systems (development, management, testing, and delivery).
- Provide the correct table name for all of your queries so the resultsets are cached correctly and can be flushed correctly.

15

Designing Basic Asset Types

As discussed in Chapter 11, "Data Design: The Asset Models," the data model for basic asset types is one database table per asset type. Each asset of that type is stored in that table.

You create new basic asset types with the AssetMaker utility. Typically you create them on a development system and then, when they are ready, you migrate your work from the development system to the management and delivery systems.

This chapter contains the following sections:

- Section 15.1, "The AssetMaker Utility"
- Section 15.2, "Creating Basic Asset Types"
- Section 15.3, "Deleting Basic Asset Types"

15.1 The AssetMaker Utility

AssetMaker is the utility that you use to create new basic asset types.

Your first step is to define the basic asset type outside WebCenter Sites by coding an `xml` file called an *asset descriptor file*, using AssetMaker XML tags. The asset descriptor file defines properties for the new asset type. The term *property* means both a column in a database table and a field in a WebCenter Sites entry form.

Your next step is to upload the file to WebCenter Sites and use AssetMaker to create two items: a database table for the new asset type, and the WebCenter Sites elements which generate the forms that you and others will use when working with assets of the new type (creating, editing, copying, and so on). Figure 15–1 shows the relationship of a database table to a form (a content-entry form, in this example) and how columns are interpreted as fields when the form is rendered.

This section contains the following topics:

- Section 15.1.1, "How AssetMaker Works"
- Section 15.1.2, "Asset Descriptor Files"
- Section 15.1.3, "Columns in the Asset Type's Database Table"
- Section 15.1.4, "Elements and SQL Statements for the Asset Type"

15.1.1 How AssetMaker Works

Using AssetMaker to create a new basic asset type involves four general steps:

1. Code the asset descriptor file.

This chapter describes asset descriptor files and coding them. The *Oracle Fusion Middleware WebCenter Sites Tag Reference* includes information that describes all of the AssetMaker tags.

2. Upload the file.

   When you upload the asset descriptor file, AssetMaker creates a row in the `AssetType` table and copies the asset descriptor file to that row.

3. Create the table.

   When you click the **Create Asset Table** button, AssetMaker does the following:

   – Parses the asset descriptor file.

   – Creates the primary storage table for assets of that type. The name of the table matches the name of the asset type identified in the asset descriptor file. The data type of each column is defined by statements in the file as well.

     In addition to the columns defined in the asset descriptor file, AssetMaker creates default columns that WebCenter Sites needs to function correctly.

   – Adds a row for the new table to the `SystemInfo` table.

     All asset tables are object tables so the value in the `systable` column is set to `obj`.

     All asset tables have URL columns so the value in the `defdir` column is set to the value that you specified either in the asset descriptor file or in the **DefDir** field in the **Create Asset Table** form when you create the asset type.

   – If you have checked the **Add 'General' category** option, Asset Maker adds one row to the `Category` table for the new asset type and names that category `General`.

4. Register the elements.

   When you register the elements, AssetMaker does the following:

   – Creates a subdirectory in the `ElementCatalog` table under `OpenMarket/Xcelerate/AssetType` directory for the new asset type.

   – Copies elements from the `AssetStubCatalog` table to the new subdirectory in the `ElementCatalog` table. These elements render WebCenter Sites forms for working with assets of this type and provide the processing logic for the WebCenter Sites functions.

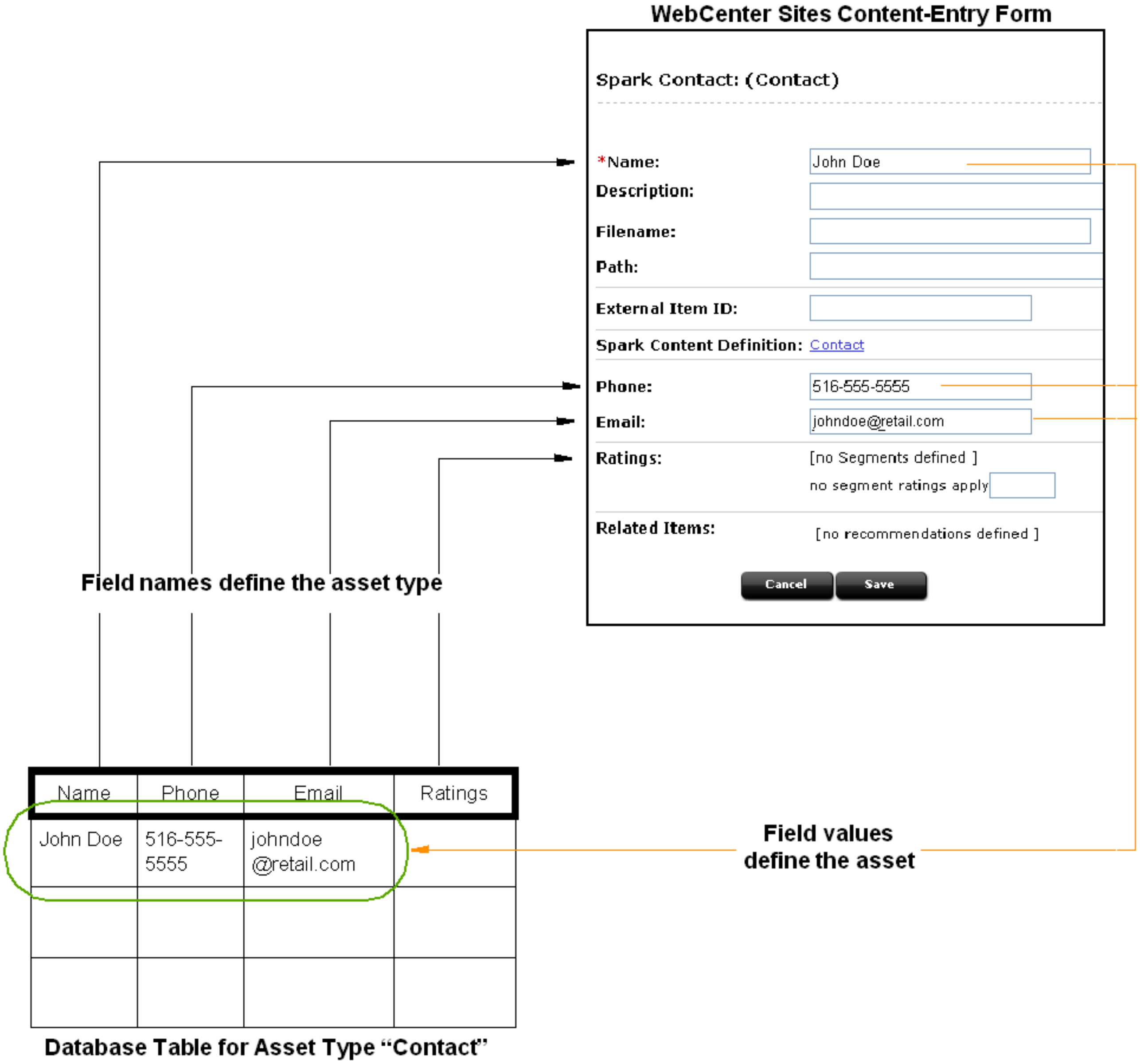
   – Creates SQL statements that implement searches on individual fields in the search forms. These statements are placed in the `SystemSQL` table.

When you use WebCenter Sites to work on an asset of this type (create, edit, inspect, and so on), AssetMaker parses the asset descriptor file, which is now located in the `AssetType` table, and passes its values to WebCenter Sites so that the forms are specific to the asset type. Statements in the asset descriptor file determine the input types of the fields, specify field length restrictions, and determine whether the field is displayed on search and search results forms.

Note that after you create an asset type, there are several configuration steps to complete before you can use it; for example, enabling the asset type on the sites that need to use the asset type, creating Start Menu shortcuts, and so on.

The flow chart in Figure 15–2 summarizes how AssetMaker works, and which database tables are involved when a basic asset type is created.

Figure 15–1 Asset Types: Database Tables and WebCenter Sites Forms



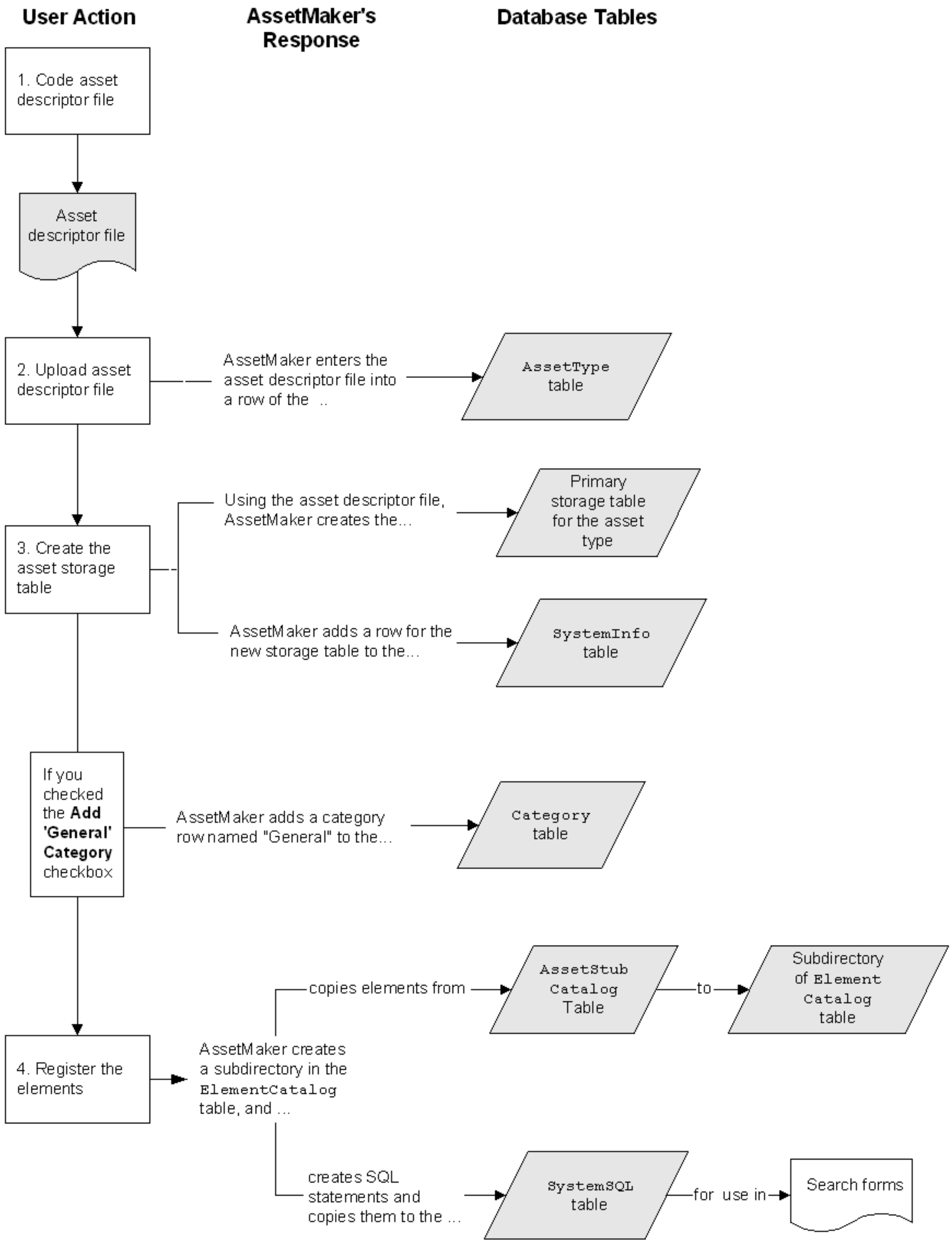
WebCenter Sites Content-Entry Form
Field names define the asset type
Field values define the asset
Database Table for Asset Type “Contact”
Cancel
Save

Spark Contact: (Contact)

*Name:	John Doe
Description:	
Filename:	
Path:	
External Item ID:	
Spark Content Definition:	Contact
Phone:	516-555-5555
Email:	johndoe@retail.com
Ratings:	[no Segments defined] no segment ratings apply
Related Items:	[no recommendations defined]

Name	Phone	Email	Ratings
John Doe	516-555-5555	johndoe @retail.com	

Figure 15–2 How AssetMaker Works


User Action
AssetMaker's Response
Database Tables
1. Code asset descriptor file
Asset descriptor file
2. Upload asset descriptor file
AssetMaker enters the asset descriptor file into a row of the ..
AssetType table
3. Create the asset storage table
Using the asset descriptor file, AssetMaker creates the...
Primary storage table for the asset type
AssetMaker adds a row for the new storage table to the...
SystemInfo table
If you checked the Add 'General' Category checkbox
AssetMaker adds a category row named "General" to the...
Category table
4. Register the elements
AssetMaker creates a subdirectory in the ElementCatalog table, and ...
copies elements from
AssetStub Catalog Table
to
Subdirectory of Element Catalog table
creates SQL statements and copies them to the ...
SystemSQL table
for use in
Search forms

15.1.2 Asset Descriptor Files

Using the AssetMaker XML tags, you code asset descriptor files that define the asset types you design for your systems.

This section contains the following topics:

- Section 15.1.2.1, "What Is an Asset Descriptor File?"
- Section 15.1.2.2, "Format and Syntax"
- Section 15.1.2.3, "Overview of the AssetMaker Tags"

15.1.2.1 What Is an Asset Descriptor File?

An asset descriptor file is a valid XML document in which developers use AssetMaker tags to define a basic asset type. An asset descriptor file does the following two things:

- Describes the asset type in terms of data structure. It specifies the name of the database table, the names of the columns, the columns' data types, and the sizes of the fields on the WebCenter Sites forms.
- Formats the HTML forms that are displayed by WebCenter Sites when users work with assets of the given type. Formatting an HTML form means naming the fields on the form, displaying the fields in required format (for example, check box, radio button, or drop-down list), accounting for field specifications (such as the number of characters that can be entered in to a text field), and so on.

AssetMaker uses the asset descriptor file to create a database table for the new asset type. When content providers work with assets of the given type (create, edit, and so on), AssetMaker parses the asset descriptor file, using the data in the file to customize the forms that WebCenter Sites displays.

> **Note:** For reference, sample AssetMaker descriptor code is provided on the WebCenter Sites installation medium, in the Samples folder. The same folder contains the `readme.txt` file that describes the sample descriptor files.

15.1.2.2 Format and Syntax

The basic format for every asset descriptor file is shown below. To the right of each AssetMaker tag is a brief description of the tag.

```
<?xml version="1.0" ?>
<ASSET ...> Names the asset type (storage table)
<PROPERTIES> Starts the properties specification section
  <PROPERTY ...> Specifies column and field name for the property
    <STORAGE .../> Specifies data type for the column
    <INPUTFORM .../> Specifies field format on New, Edit, Inspect forms
    <SEARCHFORM .../> Specifies field format on Advanced Search form
    <SEARCHRESULTS .../> Specifies which fields are shown in search results
  </PROPERTY>
  <PROPERTY ...>
    <STORAGE .../>
    <INPUTFORM .../>
    <SEARCHFORM .../>
    <SEARCHRESULTS .../>
  </PROPERTY>
  <PROPERTY ...>
...
</PROPERTIES> Ends the properties specification section
</ASSET> Ends the asset descriptor file
```

Shown next is the syntax of an asset descriptor file, indicating some of the parameters that an AssetMaker tag can take:

```
<?xml version="1.0" ?>
```

```
<ASSET NAME="assetTypeName" DESCRIPTION=""assetTypeName" ...>
<PROPERTIES>
  <PROPERTY NAME="fieldName1" DESCRIPTION="fieldName1"/>
    <STORAGE TYPE="VARCHAR" LENGTH="36"/>
    <INPUTFORM TYPE="TEXT" DESCRIPTION="fieldName1".../>
    <SEARCHFORM TYPE="TEXT" DESCRIPTION="fieldName1".../>
    <SEARCHRESULTS INCLUDE="TRUE"/>
  </PROPERTY>
  <PROPERTY NAME="fieldName2" DESCRIPTION="fieldName2"/>
    <STORAGE TYPE="INTEGER" LENGTH="4"/>
    <INPUTFORM TYPE="TEXT" DESCRIPTION="fieldName2".../>
    <SEARCHFORM TYPE="TEXT" DESCRIPTION="fieldName2".../>
   <SEARCHRESULTS INCLUDE="TRUE"/>
  </PROPERTY>
...
</PROPERTIES>
</ASSET>
```

An overview of the tags in the asset descriptor file is given in this section of the guide. Detailed information about the tags and their parameters, along with sample code, is given in the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

15.1.2.3 Overview of the AssetMaker Tags

- An asset descriptor file begins with the standard XML version tag:

  ```
  <?xml version="1.0"?>
  ```

- The ASSET tag, which follows the XML version tag, names the asset type and therefore its storage table in the WebCenter Sites database. The ASSET tag also sets some of the behavior and display attributes of assets of that type; for example, the ASSET tag determines what graphical notation designates that a field is required, and whether an asset can be previewed.

  The opening tag <ASSET> is always the first line of code and the closing tag <\ASSET> is always the last line of code in the asset descriptor file. Note that there is only one ASSET tag pair in each asset descriptor file because only one asset type per asset descriptor file can be created.

- The PROPERTIES tag marks the section of the file that holds the property descriptions. The opening tag <PROPERTIES> is always the second statement in the asset descriptor file. There is only one PROPERTIES tag pair in each asset descriptor file.

  Note: The PROPERTIES tag is required in every asset descriptor file, even if no PROPERTY tags are needed.

- The PROPERTY tags, nested within the PROPERTIES tag pair, specify the columns and fields for assets of this type. Each PROPERTY tag specifies the database name of the column that will hold the value(s) users will enter for this property, and the column's display name (i.e., its field name) as it will appear on the form that will be rendered for users who are to work with assets of this type.

- Nested inside each pair of PROPERTY tags are the following tags:

  – STORAGE: specifies the data type of the column that is being established by this property. Note that the data type in the STORAGE tag must map to one of the data types that is defined by the properties on the Database tab of the futuretense.ini file.

- INPUTFORM: specifies the name and format of the field on the New, Edit, and Inspect forms. For example: Is the field a drop-down list or a check box or a text field? The field's input type must be compatible with the data type of the database column, as specified by the STORAGE statement.
- SEARCHFORM: specifies the format of the field (property) when it appears on the Advanced Search form. If the SEARCHFORM statement is omitted from the PROPERTY section, the field being defined does not appear on the Advanced Search form.

  Note that if the value of the TYPE parameter is Table or Date, a drop-down list will appear on the Advanced Search form for the asset type, but not on the SimpleSearch form.
- SEARCHRESULTS: specifies which fields are displayed in the search results form after a search is run. The field value is also displayed if the INCLUDE parameter is set to true. (This tag is optional.)

  If you are modifying a standard field, do not set SEARCHRESULTS to true for name or description.

For detailed information about these tags and their parameters, see the "AssetMaker Tags" section of the *Oracle Fusion Middleware WebCenter Sites Tag Reference*. That section also provides information about dependencies and restrictions among the parameters STORAGE TYPE, INPUTFORM TYPE, and SEARCHFORM TYPE.

15.1.3 Columns in the Asset Type's Database Table

When AssetMaker creates the database table for a new asset type, it creates columns for all the properties defined by the PROPERTY tags in the asset descriptor file, and it creates default columns that are required by WebCenter Sites for its basic functionality. For a list of the default columns in each asset type's table, see Section 11.2.4.2, "Default Columns in the Basic Asset Type Database Table."

This section contains the following topics:

- Section 15.1.3.1, "The Source Column: A Special Case"
- Section 15.1.3.2, "Storage Types for the Columns"
- Section 15.1.3.3, "Input Types for the Fields"
- Section 15.1.3.4, "Data Types for Standard Asset Fields"

15.1.3.1 The Source Column: A Special Case

All of the asset type tables can also have a source column. WebCenter Sites provides a Source table and a **Source** form on the **Admin** tab that you use to add the rows to the Source table. You can use this feature to identify where an asset originated. The Burlington Financial sample site, for example, uses Source to identify which wire feed service provided an article asset.

However, unlike the columns listed in the preceding table, the source column is not automatically created when AssetMaker creates the asset type table. To add the source column to your table and have it displayed on your asset forms, you must include a PROPERTY description for it in the asset descriptor file. For an example, see Section 15.2.3.2, "Example: Adding the Source Column and Field."

15.1.3.2 Storage Types for the Columns

The STORAGE TYPE parameter specifies the data type of a column. The data types are defined by the WebCenter Sites database properties located in the futuretense.ini file.

The following table presents the possible data types for your asset type's table columns:

Table 15–1 STORAGE TYPE Parameter

Type (generic ODBC/JDBC data type)	Property
CHAR	cc.char
VARCHAR	cc.varchar
SMALLINT	cc.smallint
INTEGER	cc.integer
BIGINT	cc.bigint
DOUBLE	cc.double
TIMESTAMP	cc.datetime
BLOB	cc.blob
LONGVARCHAR	cc.bigtext

15.1.3.3 Input Types for the Fields

The INPUT TYPE parameter specifies how data can be entered in a field when it is displayed in the WebCenter Sites forms. The following table lists all the input types. Note that the input type for a field must be compatible with the data type of its column:

Table 15–2 INPUT TYPE Parameter

Input TYPE	Description
TEXT	A single line of text. Corresponds to the HTML input type named TEXT.
TEXTAREA	A text box, with scroll bars, that accepts multiple lines of text. Corresponds to the HTML input type named TEXTAREA. If you expect large amounts of text to be entered in the field, it is a good idea to create a text box that displays the contents of a URL column. To do so, you must specify a string for PROPERTY NAME that begins with url and set the STORAGE TYPE to VARCHAR. When a user clicks Save, the text entered into this kind of field is stored in the file directory specified as the default storage directory for this asset type. You can specify the default storage directory (defdir) in either the asset descriptor file, or in the AssetMaker form when you create the asset type. **Note:** ■ You can specify an unlimited size for a url field that is edited via a TEXTAREA field by not specifying a value for the MAXLENGTH parameter. ■ If you are specifying a string for PROPERTY NAME that begins with the letters url, do not use the following suffixes: _type, _size, _folder, and _file.

Table 15–2 (Cont.) INPUT TYPE Parameter

Input TYPE	Description
`UPLOAD`	A field that takes a file name (a URL) and presents a Browse button so that you can either enter the path to and name of a file or browse to it and select it. When you specify that a field is an upload field, set a string for `PROPERTY NAME` that begins with `url` and set `STORAGE TYPE` (the property's data type) to `VARCHAR`. You can also use the `BLOB` storage type for an upload field; in this case, the `PROPERTY NAME` string does not have to begin with `url`. When the user clicks Save, WebCenter Sites uploads the selected file and stores it in the file directory specified as the default storage directory for this asset type. You can specify the default storage directory (defdir) in either the asset descriptor file, or in the AssetMaker form when you upload the file. **Note:** ■ The size of a file that is selected in an upload field cannot exceed 30 megabytes. ■ If you are specifying a string for `PROPERTY NAME` that begins with the letters `url`, do not use the following suffixes: `_type`, `_size`, `_folder`, and `_file`.
`SELECT`	A field that presents a drop-down list of options that can be selected. You can either specify the options that are presented in the list or you can specify a query so that the options are selected from the database (or an external table) and presented dynamically. Corresponds to the HTML input type SELECT.
`CHECKBOX`	A check box field. You can specify the names of the check box options or you can specify a query so that the names are selected from the database (or an external table) and presented dynamically. This input type allows the user to select more than one option. Corresponds to the HTML input type `CHECKBOX`.
`RADIO`	A radio button control. You can either specify the names of the radio options or you can specify a query so that the names are selected from the database (or an external table) and presented dynamically. This input type allows the user to select only one option. Corresponds to the HTML input type `RADIO`.
`CKEDITOR`	A field whose contents you edit by using the CKEditor text editor. When you specify that a field is a CKEditor field, it is recommended that you make it a URL field. That is, set a string for PROPERTY NAME with the "url" prefix and set STORAGE TYPE (the property's data type) to VARCHAR. Note that if you are specifying a string form PROPERTY NAME that begins with the "url" prefix, do not use the following suffixes: _type, _size, _folder, and _file.

Table 15–2 (Cont.) INPUT TYPE Parameter

Input TYPE	Description
ELEMENT	Calls an element that you create to display a field on the **ContentForm**, **ContentDetails**, or **SearchForm** forms. The custom element must be found at one of the following locations: - For a field on the **ContentForm** form: `OpenMarket/Xcelerate/AssetType/`*`myAssetType`*`/ContentForm/`*`fieldname`* - For a field on the **ContentDetails** form: `OpenMarket/Xcelerate/AssetType/`*`myAssetType`*`/ContentDetails/`*`fieldname`* - For a field on the **SearchForm** form: `OpenMarket/Xcelerate/AssetType/`*`myAssetType`*`/SearchForm/`*`fieldname`* Where *`myAssetType`* is the asset type that you are creating the custom field for, and fieldname is the name of the custom field. An `ELEMENT` field can have any storage type, including `BLOB`.

15.1.3.4 Data Types for Standard Asset Fields

You can customize the appearance of WebCenter Sites's standard asset fields. All other changes are conditional on the type of field, as described below:

- All standard fields. You can change their display names.
- A standard field that is not a system field. You must not change its data type, with one exception: If the data type is `VARCHAR`, you can change only its length.
- System fields. You must not change the data type (including the length of a `VARCHAR` type of field).

  The following table lists the data types of standard fields (and indicates whether they are also system fields):

Table 15–3 Data Types for Standard Asset Fields

Standard Field	System Field	Data Type
ID	Yes	NOT NULL NUMBER(38)
NAME	n/a	NOT NULL VARCHAR(64)
DESCRIPTION	n/a	VARCHAR(128)
TEMPLATE	Yes	VARCHAR(64)
SUBTYPE	n/a	VARCHAR(24)
FILENAME	n/a	VARCHAR(64)
PATH	n/a	VARCHAR(255)
STATUS	Yes	NOT NULL VARCHAR(2)
EXTERNALDOCTYPE	Yes	VARCHAR(64)
URLEXTERNALDOCXML	Yes	VARCHAR(255)
URLEXTERNALDOC	Yes	VARCHAR2(255)
CREATEDBY	Yes	NOT NULL VARCHAR(64)
UPDATEDBY	Yes	NOT NULL VARCHAR(64)

Table 15–3 (Cont.) Data Types for Standard Asset Fields

Standard Field	System Field	Data Type
`CREATEDDATE`	Yes	`NOT NULL DATE`
`UPDATEDDATE`	Yes	`NOT NULL DATE`
`STARTDATE`	n/a	`DATE`
`ENDDATE`	n/a	`DATE`

15.1.4 Elements and SQL Statements for the Asset Type

After you upload an asset descriptor file, you "register" the elements. When you register elements, AssetMaker copies elements in the `AssetStubElementCatalog` table to a directory in the `ElementCatalog` table for this asset type.

Additionally, AssetMaker copies several SQL statements that implement the WebCenter Sites searches on the **Simple Search** and the **Advanced Search** forms for assets of this type.

If necessary, you can customize the SQL statements, the asset type-specific elements, or, in some cases, the elements in the `AssetStubElementCatalog` table.

> **Note:** **Under no circumstances** should you modify any of the other WebCenter Sites elements.

For information about customizing your elements, see Section 15.2.9, "Step 7: (Optional) Customize the Asset Type Elements."

This section contains the following topics:

- Section 15.1.4.1, "The Elements"
- Section 15.1.4.2, "The SQL Statements"

15.1.4.1 The Elements

AssetMaker places the elements for your new asset type to the `ElementCatalog` table according to the following naming convention:

```
OpenMarket/Xcelerate/AssetType/YourNewAssetType
```

For example, the elements for the sample asset type "ImageFile" are located here:

```
OpenMarket/Xcelerate/AssetType/ImageFile
```

The following table lists the elements that AssetMaker copies for each asset type:

Table 15–4 AssetMaker Elements

Element	Description
`ContentForm`	Renders the New and Edit forms for assets of this type. When the function is invoked, AssetMaker uses the `INPUTFORM` statements in the asset descriptor file to format these forms.
`ContentDetails`	Formats the Inspect form for assets of this type. When the function is invoked, AssetMaker uses the `INPUTFORM` statements in the asset descriptor file to customize these forms.

Table 15–4 (Cont.) AssetMaker Elements

Element	Description
`SimpleSearch`	Renders the Simple Search form for assets of this type. When the function is invoked, AssetMaker uses the `SEARCHFORM` statements in the asset descriptor file to format these forms.
`SearchForm`	Formats the Advanced Search form for assets of this type. When the function is invoked, AssetMaker uses the `SEARCHFORM` statements in the asset descriptor file to format these forms.
`AppendSelectDetails`	Builds the SQL queries on the individual fields in the Advanced Search form. When the Advanced Search form is rendered, AssetMaker uses the `SEARCHFORM` statements in the asset descriptor file to customize the form.
`AppendSelectDetailsSE`	Builds the SQL queries on the individual fields in the Advanced Search form when your system is using an external search engine. When this function is invoked, AssetMaker uses the `SEARCHFORM` statements in the asset descriptor file to create the SQL queries.
`IndexAdd`	The `IndexAdd` and `IndexReplace` elements establish which fields (columns) are indexed by the search engine when you are using a search engine. By default, only the standard fields are indexed. If you want other fields indexed, you must customize these forms. For more information, see Section 15.2.9, "Step 7: (Optional) Customize the Asset Type Elements."
`IndexReplace`	See the description of `IndexAdd`.
`Tile`	Formats the Search Results page, a page that lists the assets that meet the search criteria, for assets of this type. When the page is rendered, AssetMaker uses the `SEARCHRESULTS` statements in the asset descriptor file to display the results.
`LoadTree`	Determines how assets of this type appear when they are displayed on any tab in the tree other than the Site Plan tab.
`LoadSiteTree`	Determines how assets of this type appear when they are displayed on the Site Plan tab.
`PreUpdate`	Is called before a function that writes to the database is completed. In other words, before an asset is saved and during the create, edit, delete, or XMLPost functions, this element is called. This element takes no input from the asset descriptor file. However, you can customize it directly.
`PostUpdate`	Is called after a function that writes to the database is completed. In other words, after an asset is created, edited, deleted, or imported with XMLPost, this element is called. You can customize this element.

15.1.4.2 The SQL Statements

AssetMaker places the SQL statements in the `SystemSQL` table according to the following naming convention:

```
OpenMarket/Xcelerate/AssetType/YourNewAssetType
```

For example, the elements for the sample asset type "ImageFile" are located here:

```
OpenMarket/Xcelerate/ImageFile
```

The following table lists the SQL elements that AssetMaker creates:

Table 15–5 SQL Elements

Statement	Description
`SelectSummary`	A SQL statement that defines the query used in the Simple Search and Advanced Search form for assets of this type.
`SelectSummarySE`	Not used.

15.2 Creating Basic Asset Types

The length of time that it takes you to create a new asset type can range widely depending on the complexity of your asset type.

A simple asset type might require you to code one simple asset descriptor file and then upload it. A more complicated asset type might require you to modify the code in the elements that AssetMaker creates for your asset type or to add a database table to hold information that you want displayed in a drop-down list.

This section contains the following topics:

- Section 15.2.1, "Overview of Creating and Configuring a New Asset Type"
- Section 15.2.2, "Before You Begin"
- Section 15.2.3, "Step 1: Code the Asset Descriptor File"
- Section 15.2.4, "Step 2: Upload the Asset Descriptor File to WebCenter Sites"
- Section 15.2.5, "Step 3: Create the Asset Table"
- Section 15.2.6, "Step 4: Configure the Asset Type"
- Section 15.2.7, "Step 5: Enable the Asset Type on Your Site"
- Section 15.2.8, "Step 6: Fine-Tune the Asset Descriptor File"
- Section 15.2.9, "Step 7: (Optional) Customize the Asset Type Elements"
- Section 15.2.10, "Step 8: (Optional) Configure Subtypes"
- Section 15.2.11, "Step 9: (Optional) Configure Association Fields"
- Section 15.2.12, "Step 10: (Optional) Configure Categories"
- Section 15.2.13, "Step 11: (Optional) Configure Sources"
- Section 15.2.14, "Step 12: (Conditional) Add Mimetypes"
- Section 15.2.15, "Step 13: (Optional) Edit Search Elements to Enable Indexed Search"
- Section 15.2.16, "Step 14: Create and Assign Asset Type Icons (Contributor Interface Only)"
- Section 15.2.17, "Step 15: Code Templates for the Asset Type"
- Section 15.2.18, "Step 16: Move the Asset Types to Other Systems"

15.2.1 Overview of Creating and Configuring a New Asset Type

Following is an overview of the process for creating and configuring a new asset type. This chapter describes each of the steps, except as noted:

1. Code an asset descriptor file. See Section 15.2.3, "Step 1: Code the Asset Descriptor File."

2. Upload the asset descriptor file, using AssetMaker in WebCenter Sites's **Admin** tab. See Section 15.2.4, "Step 2: Upload the Asset Descriptor File to WebCenter Sites."
3. Create the database table and register the asset type elements by copying the asset type elements from the `AssetStubElementCatalog` table to the appropriate directory in the `ElementCatalog` table. See Section 15.2.5, "Step 3: Create the Asset Table."
4. Configure the asset type. See Section 15.2.6, "Step 4: Configure the Asset Type."
5. Enable the asset type for the site that you are using to develop assets on and create a **Start Menu** shortcut so that you can work with the asset type. See Section 15.2.7, "Step 5: Enable the Asset Type on Your Site."
6. Examine the **New**, **Edit**, **Inspect**, **Search**, and **Search Results** forms. If necessary, fine-tune the asset descriptor file, and re-register the asset type elements. See Section 15.2.8, "Step 6: Fine-Tune the Asset Descriptor File."
7. (Optional) If necessary, customize the asset type elements. See Section 15.2.9, "Step 7: (Optional) Customize the Asset Type Elements."
8. (Optional) Add **Subtype** entries for the new asset type. See Section 15.2.10, "Step 8: (Optional) Configure Subtypes."
9. (Optional) Create asset **Association** fields for the new asset type. See Section 15.2.11, "Step 9: (Optional) Configure Association Fields."
10. (Optional) Add **Category** entries for the new asset type. See Section 15.2.12, "Step 10: (Optional) Configure Categories."
11. (Optional) Add **Source** entries for the new asset type. See Section 15.2.13, "Step 11: (Optional) Configure Sources."
12. (Optional) Add **Mimetypes** for the new asset type. See Section 15.2.14, "Step 12: (Conditional) Add Mimetypes."
13. (Optional) If you are using a search engine rather than the WebCenter Sites database search utility to perform the logic behind the search forms and you want to use it on your new asset type, edit your search elements to enable indexed searching. See Section 15.2.15, "Step 13: (Optional) Edit Search Elements to Enable Indexed Search."
14. Create and assign the icon that will represent the asset type in the Contributor interface's navigation trees (**Site Tree**, **Content Tree**, and **My Work** tree). See Section 15.2.16, "Step 14: Create and Assign Asset Type Icons (Contributor Interface Only)."
15. Code templates for assets of this type. See Section 15.2.17, "Step 15: Code Templates for the Asset Type" and Chapter 28, "Coding Elements for Templates and CSElements."
16. Move the asset types to the other systems, management and delivery. See Section 15.2.18, "Step 16: Move the Asset Types to Other Systems." This allows your administrator to complete the final steps in creating the asset type, including setting up workflow and creating start menu items.

15.2.2 Before You Begin

Before you begin coding your asset descriptor files, you must plan your design and set up your development system, as described in the following sections.

This section contains the following topics:

- Section 15.2.2.1, "Plan the Asset Type Design"
- Section 15.2.2.2, "Set Up Your Development System"

15.2.2.1 Plan the Asset Type Design

Be sure to design your asset types on paper before you start coding an asset descriptor file. Consider the following kinds of details:

- What fields do you need?

  In general, try to minimize the number of fields that you use by organizing the information into useful units. When determining those units, consider both the information you plan to display on your online site and the data-entry needs of the content providers who will enter that data.
- What is the appropriate data type for each field?
- For fields with options, how will you supply the options? With a static list coded in the asset descriptor file or with a lookup table that holds the valid options?
- Which WebCenter Sites features will you use to organize or categorize assets of this type? For example, source, category, and asset associations. For each one, determine its name and plan how it will be used both on the management system and in the design of your online site.
- Does the implementation of your site design require assets of this type to use a different default template based on the publishing target that they are published to? If so, you will need to use the **Subtype** feature. Determine the names of the subtypes that you will need for assets of this type.

15.2.2.2 Set Up Your Development System

Also before you begin, be sure to set up your development system. For information about any of these preliminary steps, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

- Create the appropriate sites.
- Create a user name for yourself that has administrator rights and enable that user name on all of the sites on your development system. (Be sure that the TableEditor ACL is assigned to your user name or you will be unable to create new asset types.)

> **Note:** Without administrator rights, you do not have access to the **Admin** tab, which means that you cannot perform any of the procedures in this chapter. For the sake of convenience, assign the **Designer** and **GeneralAdmin** roles to your user name. That way you will have access to all the tabs and all of the existing **Start Menu** shortcuts for the assets in the sample site.

15.2.3 Step 1: Code the Asset Descriptor File

As described in Section 15.1.2, "Asset Descriptor Files," this is the basic format of an asset descriptor file:

```
<?xml version="1.0" ?>
<ASSET NAME="assetName"...>
<PROPERTIES>
  <PROPERTY.../>
    <STORAGE.../>
    <INPUTFORM.../>
    <SEARCHFORM.../>
    <SEARCHRESULTS.../>
  </PROPERTY>
  <PROPERTY... />
    <STORAGE.../>
    <INPUTFORM.../>
    <SEARCHFORM.../>
    <SEARCHRESULTS.../>
  </PROPERTY>
</PROPERTIES>
</ASSET>
```

To code your asset descriptor files, read the "AssetMaker Tags" section of the *Oracle Fusion Middleware WebCenter Sites Tag Reference* and use the tags described in that chapter to code the file. You can use the native XML editor in Oracle WebCenter Sites Explorer to code the file or you can use any other XML editor.

Note that you can customize the appearance of standard asset fields by including them in your asset descriptor file. Changing a field's storage type is conditional. For example, if the field is a system field, its storage type must not be changed. For the list of standard fields, their storage types, and allowed changes to storage type, see Section 15.1.3.4, "Data Types for Standard Asset Fields."

This section offers a sample asset descriptor file and several examples about coding specific kinds of properties.

This section contains the following topics:

- Section 15.2.3.1, "Sample Asset Descriptor File: ImageFile.xml"
- Section 15.2.3.2, "Example: Adding the Source Column and Field"
- Section 15.2.3.3, "Upload Example 1: A Standard Upload Field"
- Section 15.2.3.4, "Upload Example 2: A Text Box Field"

15.2.3.1 Sample Asset Descriptor File: ImageFile.xml

If the Burlington Financial sample site is installed on your system, you will find the `ImageFile.xml` asset descriptor file in the `AssetType` table. You can either start Oracle WebCenter Sites Explorer and open the file or you can examine it here:

```
<!-- this is the description of an asset -->
<ASSET NAME="ImageFile" DESCRIPTION="ImageFile"
 MARKERIMAGE="/Xcelerate/data/help16.gif" PROCESSOR="4.0"
 DEFDIR="c:\FutureTense\Storage\ImageFile">

<PROPERTIES>

  <PROPERTY NAME="source" DESCRIPTION="Source">
    <STORAGE TYPE="VARCHAR" LENGTH="24"/>
    <INPUTFORM DESCRIPTION="Source" TYPE="SELECT" TABLENAME="Source"
     OPTIONDESCKEY="description" OPTIONVALUEKEY="source" SOURCETYPE="TABLE"/>
    <SEARCHFORM DESCRIPTION="Source" TYPE="SELECT" TABLENAME="Source"
     OPTIONDESCKEY="description" OPTIONVALUEKEY="source" SOURCETYPE="TABLE"/>
  </PROPERTY>

  <PROPERTY NAME="urlpicture" DESCRIPTION="Image File">
    <STORAGE TYPE="VARCHAR" LENGTH="255"/>
    <INPUTFORM TYPE="UPLOAD" WIDTH="36" REQUIRED="NO" LINKTEXT="Image"/>
  </PROPERTY>
```

```
<PROPERTY NAME="urlthumbnail" DESCRIPTION="Thumbnail File">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM TYPE="UPLOAD" WIDTH="36" REQUIRED="NO" LINKTEXT="Image"/>
</PROPERTY>

<PROPERTY NAME="mimetype" DESCRIPTION="Mimetype">
  <STORAGE TYPE="VARCHAR" LENGTH="36"/>
  <INPUTFORM TYPE="SELECT" SOURCETYPE="TABLE" TABLENAME="MimeType"
   OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype"
   SQL="SELECT mimetype, description
   FROM MimeType
   WHERE keyword = 'image'
   AND isdefault = 'y'"
   INSTRUCTION="Add more options to mimetype table with isdefault=y
   and keyword=image"/>
  <SEARCHFORM DESCRIPTION="MimeType" TYPE="SELECT" SOURCETYPE="TABLE"
   TABLENAME="MimeType" OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype"
   SQL="SELECT mimetype, description
   FROM MimeType
   WHERE keyword = 'image'
   AND isdefault = 'y'"/>
</PROPERTY>

<PROPERTY NAME="width" DESCRIPTION="Width">
  <STORAGE TYPE="INTEGER" LENGTH="4"/>
  <INPUTFORM TYPE="TEXT" WIDTH="4" MAXLENGTH="4" REQUIRED="NO" DEFAULT=""/>
  <SEARCHFORM DESCRIPTION="Width is" TYPE="TEXT" WIDTH="4"
   MAXLENGTH="4" VERB="="/>
</PROPERTY>

<PROPERTY NAME="height" DESCRIPTION="Height">
  <STORAGE TYPE="INTEGER" LENGTH="4"/>
  <INPUTFORM TYPE="TEXT" WIDTH="4" MAXLENGTH="4" REQUIRED="NO" DEFAULT=""/>
  <SEARCHFORM DESCRIPTION="Height is" TYPE="TEXT" WIDTH="4"
   MAXLENGTH="4" VERB="="/>
</PROPERTY>

<PROPERTY NAME="align" DESCRIPTION="Alignment">
  <STORAGE TYPE="VARCHAR" LENGTH="8"/>
  <INPUTFORM TYPE="SELECT" SOURCETYPE="STRING"
   OPTIONVALUES="Left,Center,Right" OPTIONDESCRIPTIONS="Left,Center,Right"/>
  <SEARCHFORM DESCRIPTION="Alignment" TYPE="SELECT" SOURCETYPE="STRING"
   OPTIONVALUES="Left,Center,Right" OPTIONDESCRIPTIONS="Left,Center,Right"/>
</PROPERTY>

<PROPERTY NAME="artist" DESCRIPTION="Artist">
  <STORAGE TYPE="VARCHAR" LENGTH="64"/>
  <INPUTFORM TYPE="TEXT" WIDTH="36" MAXLENGTH="36" REQUIRED="NO" DEFAULT=""/>
  <SEARCHFORM DESCRIPTION="Artist contains" TYPE="TEXT"
   WIDTH="36" MAXLENGTH="64"/>
</PROPERTY>

<PROPERTY NAME="alttext" DESCRIPTION="Alt Text">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="255" REQUIRED="NO" DEFAULT=""/>
  <SEARCHFORM DESCRIPTION="Alt Text contains" TYPE="TEXT"
   WIDTH="48" MAXLENGTH="255"/>
</PROPERTY>
```

```
  <PROPERTY NAME="keywords" DESCRIPTION="Keywords">
    <STORAGE TYPE="VARCHAR" LENGTH="128"/>
    <INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="128" REQUIRED="NO" DEFAULT=""/>
    <SEARCHFORM DESCRIPTION="Keywords contain" TYPE="TEXT"
     WIDTH="48" MAXLENGTH="128"/>
  </PROPERTY>

  <PROPERTY NAME="imagedate" DESCRIPTION="Image date">
    <STORAGE TYPE="TIMESTAMP" LENGTH="8"/>
    <INPUTFORM TYPE="ELEMENT" WIDTH="24" MAXLENGTH="48" REQUIRED="NO"
     DEFAULT="" INSTRUCTION="Format: yyyy-mm-dd hh:mm"/>
    <SEARCHFORM DESCRIPTION="Image date" TYPE="ELEMENT"
     WIDTH="48" MAXLENGTH="128"/>
  </PROPERTY>

</PROPERTIES>
</ASSET>
```

Examine this asset descriptor file and then, if Burlington Financial is installed on your system, start WebCenter Sites, select the Burlington Financial site, examine the WebCenter Sites forms for the imagefile asset type, and compare the forms to the asset descriptor file.

Note the following about the `ImageFile` asset descriptor file:

- The `ASSET` tag provides a value for the `DEFDIR` parameter. All asset tables have at least two URL columns (upload fields) by default, which means you must set a value for the default storage directory (defdir) of any new asset type. (the imagefile asset type has two additional URL columns: `urlpicture` and `urlthumbnail`.)

  You can set the `defdir` value either with the `ASSET` tag's `DEFDIR` parameter, or with the **defdir** field in the AssetMaker **Create Asset Table** form.

  > **Note:** A `defdir` set using the **Create Asset Table** form overrides a defdir set in the asset descriptor file

  For more information about URL columns, see Section 12.2.3, "Indirect Data Storage with the WebCenter Sites URL Field."

- There are no `PROPERTY` statements for any of the default columns that AssetMaker creates in an asset type's database table. Those columns are listed in Section 11.2.4.2, "Default Columns in the Basic Asset Type Database Table."

- Not every property that has a `SEARCHFORM` statement has a matching `SEARCHRESULTS` statement. In other words, if you decide to put a field on the **Advanced Search** form, it does not mean that you have to display the data from that field on the **Search Results** form.

15.2.3.2 Example: Adding the Source Column and Field

The source column is not created by default even though WebCenter Sites has a Source feature on the **Admin** tab. In order to use the Source feature on your new asset types, you must include a property statement for the source column and field.

Note the following:

- `STORAGE TYPE` must be set to `VARCHAR` and `LENGTH` must be set to `24`.

- `INPUTFORM SOURCETYPE` must be set to `TABLE` and `TABLENAME` must be set to `Source`.

For example:

```
<PROPERTY NAME="source" DESCRIPTION="Source">
  <STORAGE TYPE="VARCHAR" LENGTH="24"/>
  <INPUTFORM TYPE="SELECT" TABLENAME="Source"
   OPTIONDESCKEY="description"
   OPTIONVALUEKEY="source" SOURCETYPE="TABLE"/>
  <SEARCHFORM DESCRIPTION="Source" TYPE="SELECT"
   TABLENAME="Source" OPTIONDESCKEY="description"
   OPTIONVALUEKEY="source" SOURCETYPE="TABLE"/>
</PROPERTY>
```

15.2.3.3 Upload Example 1: A Standard Upload Field

To create an upload field with a **Browse** button, code the `PROPERTY` statement as follows:

1. The string set for `PROPERTY NAME` must begin with the letters `url`.
2. The value for `STORAGE TYPE` must be set to `VARCHAR`.
3. The value for `INPUT TYPE` must be set to `UPLOAD`.

Here is a code snippet of an upload field from the `ImageFile` asset descriptor file:

```
<PROPERTY NAME="urlpicture" DESCRIPTION="Image File">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM TYPE="UPLOAD" WIDTH="36" REQUIRED="NO" LINKTEXT="Image"/>
</PROPERTY>
```

> **Note:** The size of a file that you can select in an upload field is limited to 30 megabytes.

15.2.3.4 Upload Example 2: A Text Box Field

To create an upload field with a text box that you can enter the text in (rather than with a **Browse** button that you use to select a file), code the `PROPERTY` statement as follows:

1. The string set for `PROPERTY NAME` must begin with the letters `url`.
2. The value for `STORAGE TYPE` must be set to `VARCHAR`.
3. The value for `INPUT TYPE` must be set to `TEXTAREA`.

The following code snippet creates a text area field for a `url` column:

```
<PROPERTY NAME="urlbody" DESCRIPTION="Article Body">
  <STORAGE TYPE="VARCHAR" LENGTH="256"/>
  <INPUTFORM TYPE="TEXTAREA" COLS="48" ROWS="25" REQUIRED="YES"/>
</PROPERTY>
```

15.2.3.5 Upload Example 2: A Text Box Field

To create an upload field with a text box that you can enter the text in (rather than with a **Browse** button that you use to select a file), code the `PROPERTY` statement as follows:

1. The string set for `PROPERTY NAME` must begin with the letters `url`.
2. The value for `STORAGE TYPE` must be set to `VARCHAR`.
3. The value for `INPUT TYPE` must be set to `TEXTAREA`.

The following code snippet creates a text area field for a `url` column:

```
<PROPERTY NAME="urlbody" DESCRIPTION="Article Body">
  <STORAGE TYPE="VARCHAR" LENGTH="256"/>
  <INPUTFORM TYPE="TEXTAREA" COLS="48" ROWS="25" REQUIRED="YES"/>
</PROPERTY>
```

15.2.3.6 Upload Example 3: A CKEditor Field

To create a CKEditor field, code the property statement as follows:

1. The `PROPERTY NAME` must begin with the letters `url`. In other words, you should use a URL column for the field. If you do not, you run the risk of making your field too small.
2. The value for `STORAGE TYPE` must be set to `VARCHAR`.
3. The value for `INPUT TYPE` must be set to `CKEDITOR`.

```
<PROPERTY NAME="urlbody" DESCRIPTION="Body">
  <STORAGE TYPE="VARCHAR" LENGTH="50"/>
  <INPUTFORM TYPE="CKEDITOR" WIDTH="300" HEIGHT="300" REQUIRED="YES"
   INSTRUCTION="Be concise! No more than 3 paragraphs."/>
</PROPERTY>
```

The length of the type `VARCHAR` is the length of the path to the file where the data is stored. This is typically less than 100. If an excessively large length is used (for example, "5000") this will cause issues with data storage as the database will store the data as a CLOB and the URL field will be corrupted as the data would be stored in the database rather than the file.

15.2.3.7 Upload Example 4: A Field That Uploads a Binary File

The following code creates a field where you can upload a blob. Note that if you do not specify the `MIMETYPE,` you may not be able to view the blob from the **Edit** and **Inspect** forms.

```
<PROPERTY NAME="type_binary" DESCRIPTION="Binary">
  <STORAGE TYPE="BINARY"/>
  <INPUTFORM TYPE="UPLOAD"
   WIDTH="24" MAXLENGTH="64"
   MIMETYPE="application/msword"
   LINKTEXT="Edit with Microsoft Word"
   INSTRUCTION="maps to cc.blob"/>
</PROPERTY>
```

15.2.3.8 Example: Enabling path, filename, startdate, and enddate

The path, filename, startdate, and enddate columns are special cases.

AssetMaker creates columns for path, filename, startdate, and enddate without requiring a `PROPERTY` statement for them. However, while these columns exist, their fields do not appear on your asset forms unless you include a `PROPERTY` statement for them in the asset descriptor file.

Note the following about these columns:

- For **path**, `STORAGE TYPE` must be set to `VARCHAR` and `LENGTH` must be set to `255`.
- For **filename**, `STORAGE TYPE` must be set to `VARCHAR` and `LENGTH` must be set to `128`.
- For **startdate**, `STORAGE TYPE` must be set to `TIMESTAMP`.
- For **enddate**, `STORAGE TYPE` must be set to `TIMESTAMP`.

> **Note:** If you include one of these standard columns in your asset descriptor file but your storage type does not match the one specified in this list, AssetMaker cannot create the asset type.

For example:

```
<PROPERTY NAME="path" DESCRIPTION="Path">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM DESCRIPTION="Path" TYPE="TEXT" LENGTH="255"/>
</PROPERTY>
```

15.2.3.9 Example: Using a Query to Obtain Options for a Drop-Down List

When the `INPUTFORM TYPE` of your property is `SELECT`, you can have WebCenter Sites populate the drop-down list for the select field with a static list of items that you provide with the `OPTIONDESCRIPTIONS` parameter, or with a list of items that WebCenter Sites obtains, dynamically, from a database table.

Another example of a select field that populates its drop-down list dynamically from a table is the Mimetype field on the imagefile forms, which queries the MimeType table for its options. Here's the code:

```
<PROPERTY NAME="mimetype" DESCRIPTION="Mimetype">

  <STORAGE TYPE="VARCHAR" LENGTH="36"/>
  <INPUTFORM TYPE="SELECT" SOURCETYPE="TABLE"
   TABLENAME="mimetype" OPTIONDESCKEY="description"
   OPTIONVALUEKEY="mimetype" SQL="SELECT mimetype, description
   FROM mimetype
   WHERE keyword = 'image'
   AND isdefault = 'y'"
   INSTRUCTION="Add more options to mimetype table
   with isdefault=y and keyword=image"/>

  <SEARCHFORM DESCRIPTION="Mimetype" TYPE="SELECT"
   SOURCETYPE="TABLE" TABLENAME="mimetype"
   OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype"
   SQL="SELECT mimetype, description
   FROM mimetype
   WHERE keyword ='image'
   AND isdefault = 'y'"/>

</PROPERTY>
```

This example shows a field that not only selects items from a database table, but, through an additional SQL query, further restricts which items are returned from that table, as well.

15.2.3.10 Example: Using a Query to Obtain Labels for Radio Buttons

If the `INPUTFORM TYPE` of your property is `RADIO`, you can input the label for each radio button using a static list of items that you provide with the `OPTIONDESCRIPTIONS` parameter, or with a list of items that WebCenter Sites obtains, dynamically, from a database table.

The following sample code creates radio buttons with labels drawn from the `CreditCard` table:

```
<PROPERTY NAME="sqlrbcc" DESCRIPTION="SQL RB Credit Card">
  <STORAGE TYPE="VARCHAR" LENGTH="4"/>
  <INPUTFORM TYPE="RADIO" SOURCETYPE="TABLE" TABLENAME="CreditCard"
   RBVALUEKEY="ccvalue" RBDESCKEY="ccdescription" />
  <SEARCHFORM TYPE="SELECT" DESCRIPTION="Credit Card:" SOURCETYPE="TABLE"
   TABLENAME="CreditCard" OPTIONVALUEKEY="ccvalue" OPTIONDESCKEY="ccdescription"
   SQL="SELECT ccvalue,ccdescription
   FROM CreditCard ORDER BY ccdescription"/>
</PROPERTY>
```

15.2.3.11 Example: Creating a Field with the ELEMENT Input Type

You can modify the fields on your asset forms by using the `ELEMENT` input type to call custom code to display the fields as you want them. You can use this method to create new asset fields, or to change the appearance of standard asset fields, though you cannot modify the storage type of a standard asset field.

An `ELEMENT` field can have any storage type, including `BLOB`.

When you set a field's input type to `ELEMENT`, WebCenter Sites calls a custom element to display the field. The custom element must be found at one of the following locations:

For a field on the **ContentForm** form:

```
OpenMarket/Xcelerate/AssetType/myAssetType/ContentForm/fieldname
```

For a field on the **ContentDetails** form:

```
OpenMarket/Xcelerate/AssetType/myAssetType/ContentDetails/fieldname
```

For a field on the **SearchForm** form:

```
OpenMarket/Xcelerate/AssetType/myAssetType/SearchForm/fieldname
```

Note that *`myAssetType`* is the asset type that you are creating the custom field for, and fieldname is the name of the custom field.

The following exerpt from an asset descriptor file uses the `ELEMENT` input type:

```
<PROPERTY NAME="imagedate" DESCRIPTION="Image date">
  <STORAGE TYPE="TIMESTAMP" LENGTH="8"/>
  <INPUTFORM TYPE="ELEMENT" WIDTH="24" MAXLENGTH="48" REQUIRED="NO" DEFAULT=""
   INSTRUCTION="Format: yyyy-mm-dd hh:mm"/>
  <SEARCHFORM DESCRIPTION="Image date" TYPE="ELEMENT"
   WIDTH="48" MAXLENGTH="128"/>
</PROPERTY>
```

Note that the input form uses a customized field, but the search form and content details forms display default fields.

The following code excerpt is the element that the descriptor file calls:

```
<!-- OpenMarket/Xcelerate/AssetType/ImageFile/ContentForm/imagedate
-
- INPUT
- Variables.AssetType
Variables.fieldname
Variables.fieldvalue- default or value for this field
Variables.datatype - from STORAGE tag in ADF for this field
Variables.helpimage - help icon
Variables.alttext - help text from INPUT tag in ADF
```

```
Other fields from input tag are in:
Variables.assetmaker/property/Variables.fieldname/inputform/[tag attribute]
- field name used in form should be Variables.AssetType:Variables.fieldname

- OUTPUT
-
-->
Enter date in the format yyyy-mm-dd hh:mm:ss<br/>

<setvar
 NAME="inputfieldsize"
 VALUE="Variables.assetmaker/property/Variables.fieldname/inputform/width"/>

<callelement NAME="OpenMarket/Xcelerate/Scripts/FormatDate"/>

<INPUT TYPE="text" SIZE="Variables.inputfieldsize"
 NAME="Variables.AssetType:Variables.fieldname"
 VALUE="Variables.fieldvalue"
 REPLACEALL="Variables.inputfieldsize,Variables.fieldvalue,Variables.fieldname,
 Variables.AssetType"onChange="padDate(this.form.elements['Variables.AssetType:
 Variables.fieldname'].value,this,'Variables.AssetType:Variables.fieldname'
 );"/>

</FTCS>
```

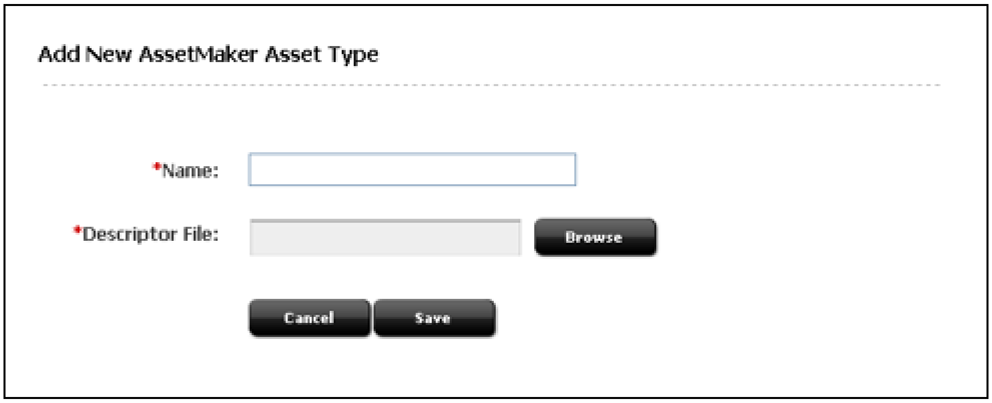
Note that you can customize as many fields as you want using the `ELEMENT` input type, but that you must write a separate element for each field that you want to modify.

15.2.4 Step 2: Upload the Asset Descriptor File to WebCenter Sites

After you have coded the asset descriptor file for your asset type, use AssetMaker to upload it and register the new elements:

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Select the **Admin** tab.
5. Expand **AssetMaker** and click **Add New**.

   The Add New AssetMaker Asset Type form appears:



Add New AssetMaker Asset Type

*Name:

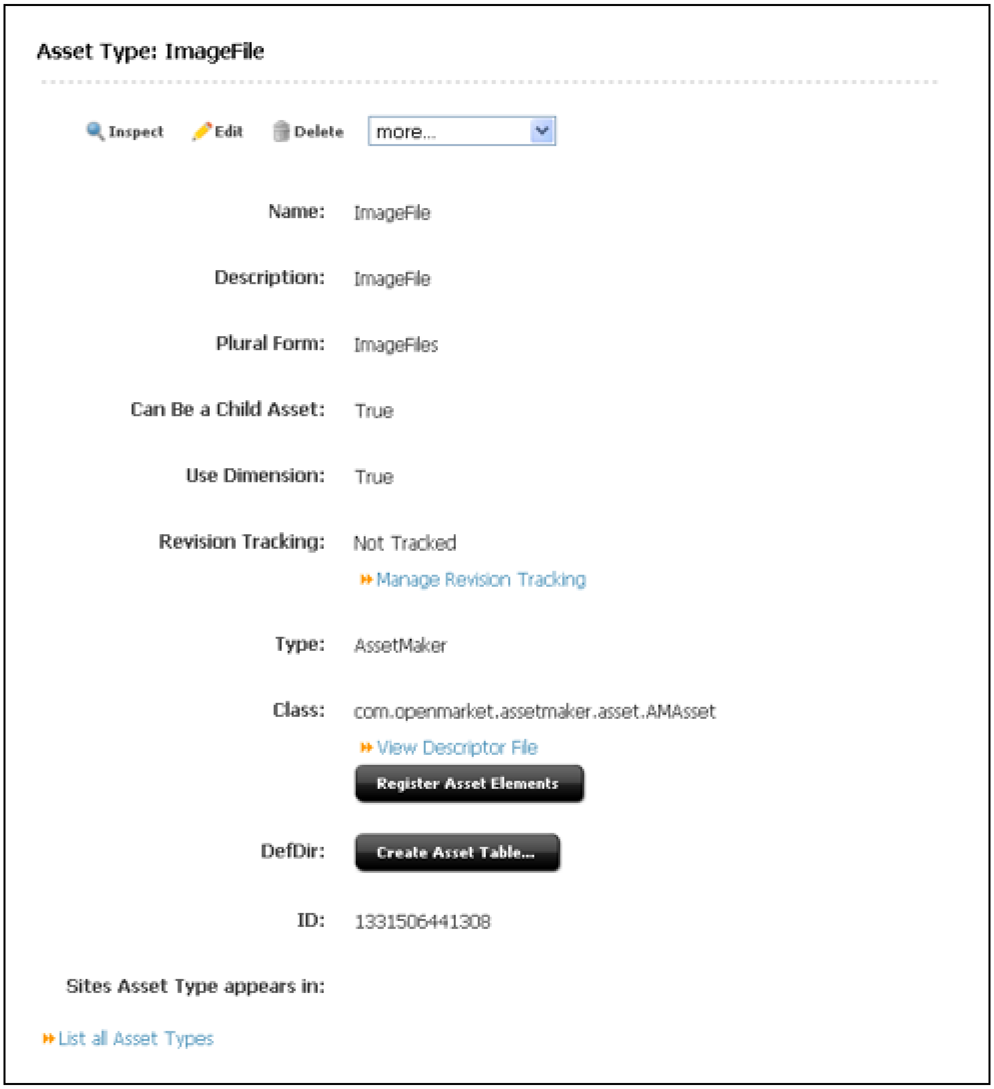
*Descriptor File:

6. Click in the **Name** field and enter the name of the new asset type. The string that you enter into this field must exactly match the string specified by the `ASSET NAME` parameter in the asset descriptor file that you are going to upload.

7. Click the **Browse** button next to the **Descriptor File** field and select the asset descriptor file.

8. Click **Save**.

   AssetMaker enters the file into the `AssetType` table (that is, it uploads the asset descriptor file to the default storage directory for the `AssetType` table), and then displays the Asset Type form:



Asset Type: ImageFile

Name:	ImageFile
Description:	ImageFile
Plural Form:	ImageFiles
Can Be a Child Asset:	True
Use Dimension:	True
Revision Tracking:	Not Tracked
Type:	AssetMaker
Class:	com.openmarket.assetmaker.asset.AMAsset
DefDir:	
ID:	1331506441308
Sites Asset Type appears in:	

15.2.5 Step 3: Create the Asset Table

This step continues from step 8 of the previous section (Section 15.2.4, "Step 2: Upload the Asset Descriptor File to WebCenter Sites").

1. Select **Create Asset Table**.

2. Examine the value in the **Defdir** field and change it if necessary. AssetMaker reads this value from the asset descriptor file. You must enter a value in this field if either of the following conditions exist:

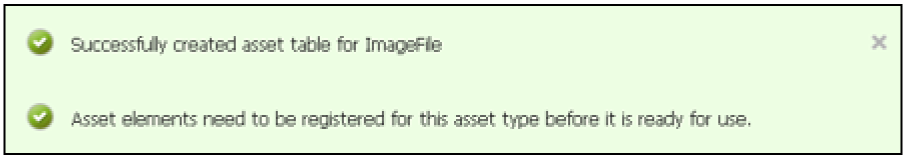
- If you did not provide a value with the `DEFDIR` parameter for the `ASSET` tag in the asset descriptor.
- If you want to change the default storage directory, which is typical when you are migrating the asset type to another system.

If you check the **Add 'General' category** option, AssetMaker adds one row to the `Category` table for the new asset type and names that category **General**.

3. Click **Create Asset Table**.

   AssetMaker creates the table.

   When it is finished, it displays a confirmation like this one:

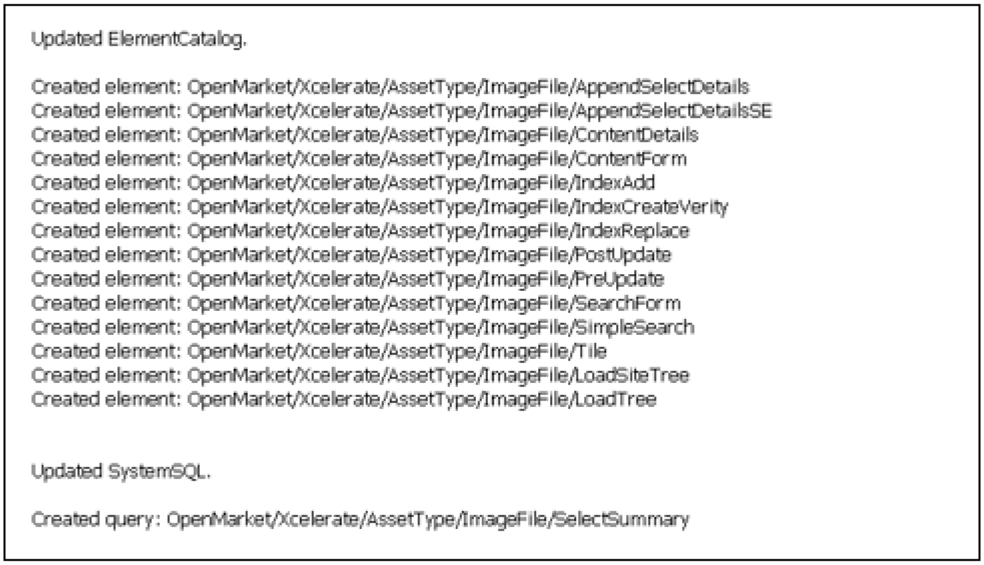


✓ Successfully created asset table for ImageFile

✓ Asset elements need to be registered for this asset type before it is ready for use.

4. Select **Register Asset Elements** and then click the **Register Asset Elements** button.

   AssetMaker copies the elements from the `AssetStubElementCatalog` table to the asset type's directory in the `ElementCatalog` table and copies the SQL statements in the SystemSQL table. When it is finished, it displays a confirmation like this one:



```
Updated ElementCatalog.

Created element: OpenMarket/Xcelerate/AssetType/ImageFile/AppendSelectDetails
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/AppendSelectDetailsSE
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/ContentDetails
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/ContentForm
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/IndexAdd
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/IndexCreateVerity
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/IndexReplace
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/PostUpdate
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/PreUpdate
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/SearchForm
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/SimpleSearch
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/Tile
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/LoadSiteTree
Created element: OpenMarket/Xcelerate/AssetType/ImageFile/LoadTree


Updated SystemSQL.

Created query: OpenMarket/Xcelerate/AssetType/ImageFile/SelectSummary
```

15.2.6 Step 4: Configure the Asset Type

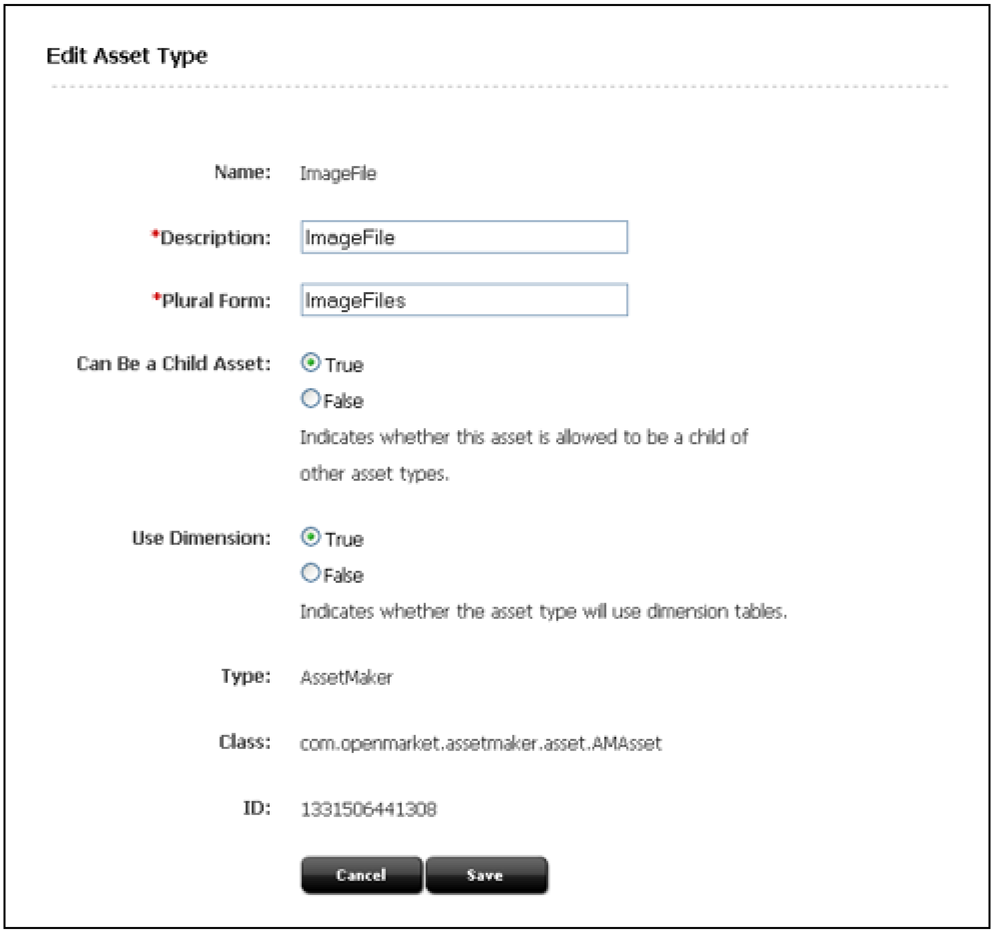
When AssetMaker created the new asset type (in Section 15.2.4, "Step 2: Upload the Asset Descriptor File to WebCenter Sites"), it also created an icon and administrative forms for configuring the new asset type, located on the **Admin** tab.

Complete the following configuration steps:

1. On the **Admin** tab, expand the **Asset Types** icon.
2. Under **Asset Types**, select your new asset type. (If you do not see it in the list, click the right mouse button, select **Refresh** from the context menu, and then select your new asset type.)

3. Click **Edit**.

   The following form appears:



Edit Asset Type

Name:	ImageFile
*Description:	ImageFile
*Plural Form:	ImageFiles
Can Be a Child Asset:	● True ○ False Indicates whether this asset is allowed to be a child of other asset types.
Use Dimension:	● True ○ False Indicates whether the asset type will use dimension tables.
Type:	AssetMaker
Class:	com.openmarket.assetmaker.asset.AMAsset
ID:	1331506441308

4. (Optional) Click in the **Description** field and change it, if necessary. The text in this field is the name that WebCenter Sites uses for this asset type on the forms and lists in the WebCenter Sites interface. By default, **Description** is set to the value of the `ASSET DESCRIPTION` statement in the asset descriptor file.

5. (Optional) Click in the **Plural Form** field and change it, if necessary. The text in this field is the text that WebCenter Sites uses in the WebCenter Sites interface when it is appropriate to refer to the asset type in the plural. By default, **Plural Form** is set to the value of the `ASSET DESCRIPTION` statement in the asset descriptor file plus the letter "s." You can also specify your own plural form in the asset descriptor file by setting the value of the `ASSET` tag's `PLURAL` parameter.

6. (Optional) Click in the **Can Be Child Asset** field and change the value, if necessary. By default, this field is set to **True**, which means that this asset type can be the child asset type in an association field for another asset type. Its name appears in the list of asset types in the **Child Asset Field** on the Add New Association forms.

7. Click **Save**.

15.2.7 Step 5: Enable the Asset Type on Your Site

Before you can examine the forms that the new elements render for the asset type, you must enable the asset on the site (or sites) that you are working with and create a simple **Start Menu** item for it.

For instructions on how to enable your asset types and create Start Menu items for them, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

15.2.8 Step 6: Fine-Tune the Asset Descriptor File

Create a new asset of your new type and examine the New, Edit, Inspect, and Search forms. (To create a new asset of this type, click **New** on the toolbar and select the **Start Menu** shortcut that you created in the preceding procedure.

After you examine the forms, you might need to modify the asset descriptor file.

You can make any of the following changes with relatively few steps:

- Re-ordering the fields that appear on the WebCenter Sites forms for the asset type
- Changing the name of a field (that is, the value of `PROPERTY DESCRIPTION`)
- Changing anything in an `INPUTFORM`, `SEARCHFORM`, or `SEARCHRESULTS` statement

If you want to make any of the changes in the preceding list, complete the following steps:

1. Use Oracle WebCenter Sites Explorer to open and modify the asset descriptor file that you uploaded in Section 15.2.4, "Step 2: Upload the Asset Descriptor File to WebCenter Sites."
2. Save your changes.
3. Re-register the elements for the asset type.

You cannot change the schema of the asset type's database table after it is created. The following changes are schema changes:

- Changing the name of a column (the `NAME` parameter in an existing `PROPERTY` statement)
- Changing the data type of the column (the `STORAGE TYPE` or `LENGTH` in an existing `PROPERTY` statement)
- Adding a new property (new `PROPERTY` statement), which adds a new column to the table
- Deleting a property (an existing `PROPERTY` statement), which deletes a column from the table

Therefore, if you want to make any of the changes in the preceding list, you must first delete the asset type, modify the asset descriptor file, and then create the asset type again.

Note that if you have customized the elements that AssetMaker copied from the `AssetStubElementCatalog` table to the asset type's directory in the `ElementCatalog` table, your changes are overwritten when you re-register the elements.

15.2.9 Step 7: (Optional) Customize the Asset Type Elements

There are two ways to customize asset type elements on your content management system:

- If you require changes that are specific to a certain asset type, you can modify the elements for that asset type in the `ElementCatalog` table, using Oracle WebCenter Sites Explorer. For information about modifying the elements, see Section 15.2.9.1, "About PreUpdate and PostUpdate Elements."
- If you require the same modifications for assets of all types, modify the source elements in the `AssetStubElementCatalog` table, using Oracle WebCenter Sites Explorer, before you create your asset types. That way you will have to customize only once.

> **Note:** Although customizing source elements in the `AssetStubElementCatalog` table might be necessary, it is not supported. If you plan to customize a stub element, consider that it will be overwritten under the following conditions:
>
> - Changing the stub elements requires you to re-register all of the asset elements that must take the new changes.When you re-register asset elements, AssetMaker moves new copies of the elements from the `AssetStubElementCatalog` table to the asset type's directory in the `ElementCatalog` table (the path is `ElementCatalog\OpenMarket\Xcelerate\AssetType`) and in the process overwrites the existing elements. If you customized elements in the `AssetType` directory, they will be overwritten.
> - The WebCenter Sites upgrade process typically installs new source elements in the `AssetStubElementCatalog` table. Code changes in the stub elements are overwritten when you re-register your asset types after upgrade.

Be meticulous about tracking all of your customizations at all times. That way you can re-create your work if necessary.

15.2.9.1 About PreUpdate and PostUpdate Elements

Actions or procedures that can be performed on assets are called **functions**. For example, **New**, **Edit**, and **Delete** are all functions that can be invoked by users of the Admin and Contributor interfaces in order to create, edit, and delete assets. Such functions also call the `PreUpdate` and `PostUpdate` elements.

`PreUpdate` and `PostUpdate` elements are used to contain logic that initiates various operations when the elements are called. The PreUpdate element is called when a function is invoked; the `PostUpdate` element is called after WebCenter Sites writes asset information to the database. Each element contains a variable whose name and value specify conditions that call the element. The variable is named `updatetype` if the element must be called from the Admin interface or from Form Mode of the Contributor interface or the XMLPost utility. The variable is named `servicesUpdateType` if the element must be called from Web Mode of the Contributor interface.

For example, your content managers are working with the Admin interface and their system is configured to import batches of articles from a wire service. You can have the `PreUpdate` element set the value for the **Source** field to **wirefeed** and the value for the **Byline** field to **API** just before import occurs if you code these operations in the element and set `updatetype=remotepost` as the condition under which the element will be called.

Note: PreUpdate and PostUpdate elements are always called when a WebCenter Sites user invokes the New, Edit, or Delete function in the Admin or Contributor interface, when assets are saved in Sites Desktop, or when assets are imported through XMLPost. Whether operations are performed via the PreUpdate and PostUpdate elements depends on how the elements are coded. By default, they are designed for no action.

PreUpdate and PostUpdate elements are accessible from Oracle WebCenter Sites Explorer, in the following path:

```
ElementCatalog\OpenMarket\Xcelerate\AssetType
```

Table 15–6 defines the values of the updatetype variable.

Note: The PreUpdate element is called twice if a user saves a new or edited asset in the Admin interface:

The first call occurs before the New or Edit form is rendered.

The second calls occurs after the user clicks **Save** in the New or Edit form, but WebCenter Sites has not yet written asset information to the database.

The condition above provides the opportunity to perform operations via PreUpdate at one or more points once the New or Edit function is invoked: before the New or Edit form is rendered, before asset information is written to the database, or both, depending on the value of the element's updatetype variable.

Table 15–6 Values of the UpdateType Variable

updatetype =	Description
setformdefaults	When a user invokes the **New** function in the Admin interface or in the Form Mode of the Contributor interface. ■ The PreUpdate element is called before the New form is rendered. ■ For PostUpdate, setformdefaults is not a legal value for the updatetype variable.
create	When a user saves a new asset in the Admin interface or in the Form Mode of the Contributor interface: ■ The PreUpdate element is called before the new asset is written to the database. ■ The PostUpdate element is called after the new asset is written to the database.
editfront	When a user invokes the **Edit** function in the Admin interface or in the Form Mode of the Contributor interface: ■ The PreUpdate element is called before the Edit form is rendered. ■ For PostUpdate, editfront is not a legal value for the pdatetype variable.

Table 15–6 (Cont.) Values of the UpdateType Variable

updatetype =	Description
`edit`	When a user saves an edited asset in the Admin interface or in the Form Mode of the Contributor interface: ■ The `PreUpdate` element is called before the edits are written to the database. ■ The `PostUpdate` element is called after the edits are written to the database.
`delete`	When a user deletes an asset from the Admin interface: ■ The `PreUpdate` element is called before WebCenter Sites deletes the asset. ■ The `PostUpdate` is performed after WebCenter Sites deletes the asset.
`remotepost`	When a user invokes the XMLPost function to import an asset: ■ The `PreUpdate` element is called before the asset is imported. ■ The `PostUpdate` element is called after the asset is imported.
`MSWord`	When a Sites Desktop user saves an asset created or edited in Microsoft Word: ■ The `PreUpdate` element is called before the new asset or edits are written to the database. ■ The `PostUpdate` element is called after the new asset or edits are written to the database.
`InSite`	When saved from Web Mode there will be a variable called `servicesUpdateType` which will have create, edit, or delete depending on the user operation. Table 15–7 defines the values of the `servicesUpdateType` variable.

Table 15–7 Values of the servicesUpdateType Variable (Contributor interface)

servicesUpdateType =	Description
`create`	When a user saves a new asset in the Contributor interface: ■ The PreUpdate element is called before the new asset is written to the database. ■ The PostUpdate element is called after the new asset is written to the database.
`edit`	When a user saves an edited asset in the Contributor interface: ■ The PreUpdate element is called before the edits are written to the database. ■ The PostUpdate element is called after the edits are written to the database.
`delete`	When a user deletes an asset in the Contributor interface: ■ The PreUpdate element is called before WebCenter Sites deletes the asset. ■ The PostUpdate element is called after WebCenter Sites deletes the asset.

15.2.10 Step 8: (Optional) Configure Subtypes

For the basic asset types that you design with AssetMaker, subtype means a subclass of the asset type based on how that asset is rendered. You can use subtypes to define a separate default approval template for an asset of that type and subtype on each publishing target.

This section contains the following topics:

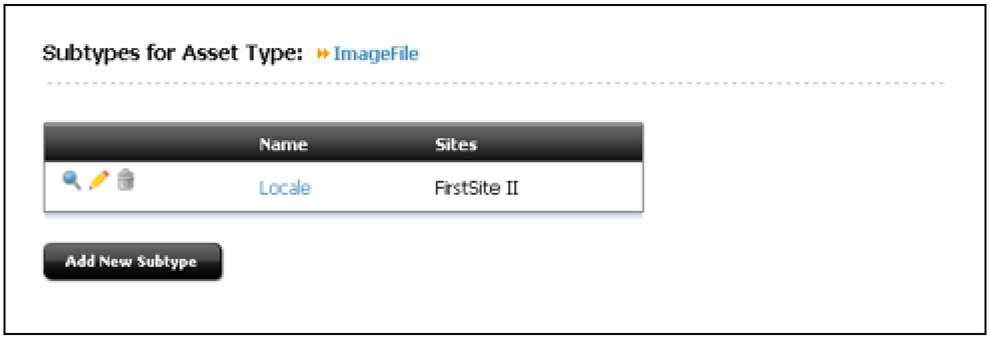
- Section 15.2.10.1, "Adding Subtypes"
- Section 15.2.10.2, "Deleting Subtypes"

15.2.10.1 Adding Subtypes

To create a subtype

1. On the **Admin** tab, expand the **Asset Types** option.
2. Under the **Asset Types** option, select the asset type that you want to create subtypes for.
3. Select the **Subtypes** option.

   The Subtypes form appears:



Subtypes for Asset Type: ImageFile

	Name	Sites
	Locale	FirstSite II

4. Click **Add New Subtype**.
5. In the next form, click in the first field in the **Name** column and enter the name of the subtype.
6. In the corresponding field in the **Sites** column, select the names of the sites that need this subtype.
7. Repeat these steps for up to five subtypes.
8. Click **Save**.

15.2.10.2 Deleting Subtypes

To delete a subtype

1. On the **Admin** tab, expand the **Asset Types** option.
2. Under the **Asset Types** option, select the asset type that you want to create subtypes for.
3. Select the **Subtypes** option.
4. In the Subtypes form, click the **Delete** (trash can) icon.
5. Click **Delete Subtype.**

15.2.11 Step 9: (Optional) Configure Association Fields

Associations are described in Section 11.2, "The Basic Asset Model." Briefly, associations are defined, asset-type-specific relationships that describe parent-child relationships or links between individual assets or asset sub-types. When you code your template elements, you use the names of these relationships to identify individual assets or sub-types, without having to refer to the object by its name.

For example, the Burlington Financial sample site associates imagefile assets with article assets. The template elements are coded to extract an associated imagefile asset by the name of the association rather than the name of the asset. That way, if you choose a different imagefile for an article, the template displays the new imagefile without your having to re-code the template.

Associations are represented as fields in the asset forms. These fields are not created from an asset descriptor file. Instead, you use the Asset Associations forms on the **Admin** tab.

Examples of association fields from the Burlington Financial sample site include the Main Image and Teaser Image associations between article assets and imagefile assets. When you create a Burlington Financial article asset, you can select an image asset from the **Main Image** and **Teaser Image** fields.

This section contains the following topics:

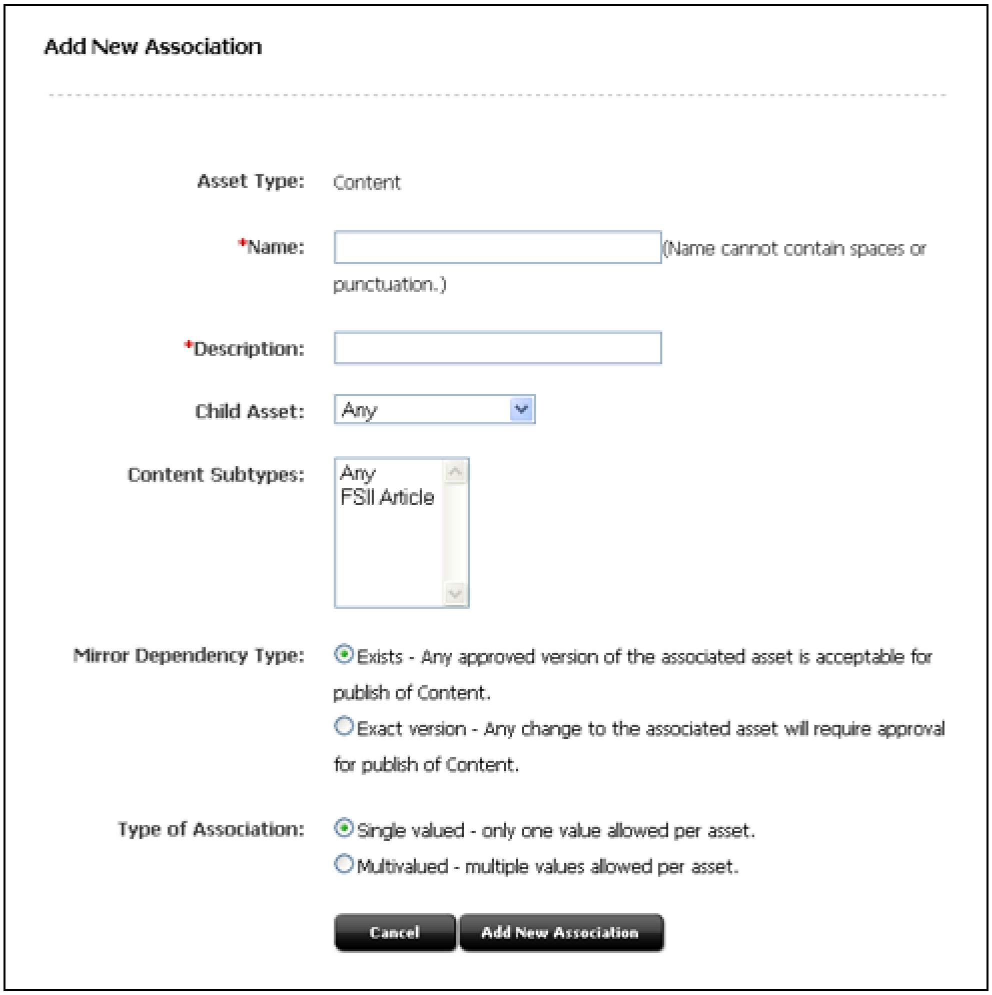
- Section 15.2.11.1, "Adding Association Fields"
- Section 15.2.11.2, "Deleting Association Fields"

15.2.11.1 Adding Association Fields

To add an association field

1. On the **Admin** tab, select **Asset Types.**
2. Under the **Asset Types** option, select the asset type that you want to create associations for.
3. Under the asset type you selected, select **Asset Associations** and then **Add New**.

   The Add New Association form appears:



Add New Association

Asset Type: Content

*Name: (Name cannot contain spaces or punctuation.)

*Description:

Child Asset: Any

Content Subtypes: Any
FSII Article

Mirror Dependency Type:
● Exists - Any approved version of the associated asset is acceptable for publish of Content.
○ Exact version - Any change to the associated asset will require approval for publish of Content.

Type of Association:
● Single valued - only one value allowed per asset.
○ Multivalued - multiple values allowed per asset.

4. Click in the **Name** field and enter the name of the association field, without using spaces, decimal points, or punctuation marks.

5. Click in the **Description** field and enter a short description of the field. Keep the description short because WebCenter Sites uses the text entered into this field as the name of the field when it is displayed on the new asset form.

6. Click in the **Child Asset** field and select the kind of asset type that will appear in this field. (It is called the **Child Asset** field because associations create parent-child relationships between assets.) You cannot specify the template or the page asset type in this field.

7. Select the subtype or subtypes for this association by highlighting them in the **Subtypes** field. To select multiple subtypes, press the **Control** key while you click your selection with the mouse.

8. Select the appropriate **Dependency Type** for this asset association. By default, it is set to **Exists**. The dependency type specified here is used by the approval system when your publishing method is Mirror to Server. For information about dependency types, see Section 28.2, "About Coding to Log Dependencies."

9. Click **Add**.

   WebCenter Sites creates a row in the `Association` table for this association. The name used in the row is the text you entered in the **Name** field in step 4.

15.2.11.2 Deleting Association Fields

Before you delete an association field, be sure to search for any assets that use it and clear the value in the field. Otherwise, those assets will still have the association when you delete the association field, but, because the field is no longer displayed in the WebCenter Sites interface, you will be unable to change it.

To delete an association field

1. On the **Admin** tab, select **Asset Types.**
2. Under the **Asset Types** option, select the asset type whose association field you want to delete.
3. Under the asset type you selected, select the association that you want to delete.
4. In the form on the right, click **Delete**.

The association field is now no longer displayed on forms for this asset type.

15.2.12 Step 10: (Optional) Configure Categories

You can use categories to organize your asset types according to some convention that works for your site design. For example, the Burlington Financial sample site uses queries based on category to determine which articles should be selected for various sections of the online site.

Although all basic asset types have a **Category** field (column) by default, it is not a required field and you do not have to use it.

This section contains the following topics:

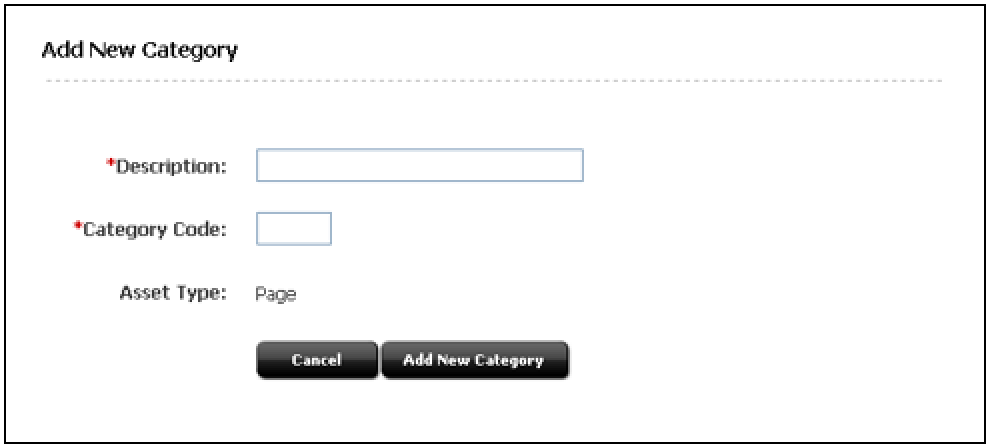
- Section 15.2.12.1, "Adding Categories"
- Section 15.2.12.2, "Deleting Categories"

15.2.12.1 Adding Categories

To add a category

1. On the **Admin** tab, select **Asset Types.**
2. Under the **Asset Types** option, select the asset type that you want to create categories for. (For example, select **Article**.)
3. Under that asset type, select **Categories** and then **Add New**.

   The following form appears:



Add New Category

*Description:	
*Category Code:	
Asset Type:	Page

4. Click in the **Description** field and enter a short description of the category. Keep the description short because WebCenter Sites uses the text that you enter in this field in the site tree and in the drop-down list for the **Category** field on the forms for assets of this type.
5. In the **Category Code** field, enter a two-character code for your new category.
6. Click the **Add** button.
7. Repeat steps 2 through 6, as needed, to finish creating the categories for this asset type.

The categories you created now appear in the drop-down lists in the **Category** fields on the New and Edit asset forms.

15.2.12.2 Deleting Categories

To delete a category

1. On the **Admin** tab, select **Asset Types**.
2. Under the **Asset Types** option, select the asset type that you want to delete a category for.
3. Under that asset type, select the category that you want to delete.
4. Click **Delete**.

15.2.13 Step 11: (Optional) Configure Sources

Sources apply to all the asset types in all the sites on your system. Therefore, if you are using Source, you need to add your sources only once (not for each asset type).

This section contains the following topics:

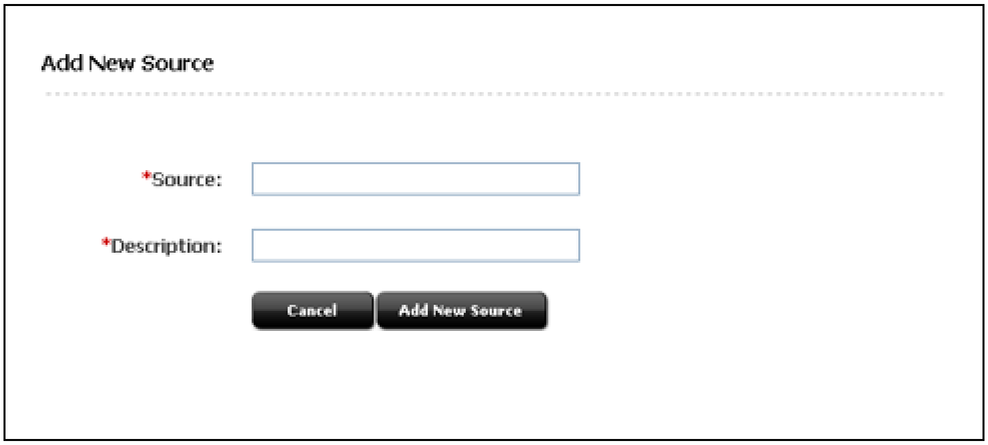
- Section 15.2.13.1, "Adding Sources"
- Section 15.2.13.2, "Deleting Sources"

15.2.13.1 Adding Sources

To create a source

1. On the **Admin** tab, select **Sources**, and then **Add New**.

   The Add Source form appears:



Add New Source

*Source:

*Description:

2. Click in the **Source** field and enter the name of a source.
3. Click in the **Description** field and enter a short description of the source. Keep this description short because WebCenter Sites uses the text from this field in the drop-down list for the **Source** field on the forms for assets.
4. Click **Add**.

   The source is written to the `Source` table.
5. Repeat steps 1 through 4 for each source that you need for your asset types.

15.2.13.2 Deleting Sources

To delete a source

1. On the Admin form, select **Sources**.
2. Under the **Sources** option, select the name of the source that you want to delete.
3. In the form for this source, click **Delete**.

15.2.14 Step 12: (Conditional) Add Mimetypes

The `MimeType` table holds mimetype codes that can be displayed in mimetype fields. You must add mimetypes for your asset if you reference the `MimeType` table in your asset descriptor file.

For example, both the imagefile and stylesheet asset types have upload fields with **Browse** buttons next to them. After you select a file in the upload field, you specify the mimetype of the file you selected from the **Mimetype** field.

The **Mimetype** fields for the imagefile and stylesheet asset types query the MimeType table for mimetype codes based on the `keyword` column:

- Mimetype codes with their `keyword` set to `stylesheet` appear in the drop-down list of the **Mimetype** field in the Stylesheet form.
- Mimetype codes with their `keyword` set to `image` appear in the drop-down list of the **Mimetype** field in the ImageFile form.

By default, stylesheet files can be `CSS` files and imagefile files can be `GIF` or `JPEG` files. You can add mimetype codes for these asset types, for your own custom asset types, or for any other reason.

To add mime types to a Mimetype drop-down list

1. Open Oracle WebCenter Sites Explorer.
2. Expand the **MimeType** table.
3. Do one of the following:
   – If you are adding a mimetype for the imagefile asset type, select the **image** folder in the **MimeType** table.
   – If you are adding a mimetype for the stylesheet asset type, select the **text** folder in the **MimeType** table
   – If you are adding a mimetype for a custom asset type with an upload field or for any other reason, select the appropriate location in the **MimeType** table.
4. Right-click in the frame on the right and then select **New** from the drop-down list.

   Oracle WebCenter Sites Explorer creates a new row in the table.

5. In the **mimetype** field, enter the name of the mimetype. For example: `XSL`.
6. In the **extension** field, enter the extension for mime types of this type. For example: `.xml`.
7. In the **description** field, enter a short description of this mimetype.
8. In the **isdefault** field, do one of the following:
   - If you want to specify more than one extension for the same mimetype, enter `n`. For example, if a mimetype named JPG has `.jpg` and `.jpeg` extensions, set **isdefault** to `n`.
   - If this is the only extension for the mimetype, enter `y`.
9. Click in the **keyword** field and do one of the following:
   - If you are adding a mimetype for the imagefile asset type, enter `image`.
   - If you are adding a mimetype for the stylesheet asset type, enter `stylesheet.`
   - If you are adding a mimetype for a custom asset type with an upload field or for any other reason, enter the appropriate keyword.
10. Select **File** and then **Save all**.

    Oracle WebCenter Sites Explorer saves the row.

    If you added a mimetype code with the keyword of `image`, that mimetype is now displayed in the **Mimetype** field of the ImageFile form. If you added a mimetype code with the keyword of stylesheet, that mimetype is now displayed in the **Mimetype** field of the Stylesheet form.

15.2.15 Step 13: (Optional) Edit Search Elements to Enable Indexed Search

WebCenter Sites and Engage have its own database SQL search mechanism that runs the Simple and Advanced searches. However, you can set up your management system to one of the supported third-party search engines instead. For configuration information, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

When you are using a search engine on your management system, each asset is indexed when it is saved after being created or edited. By default, only the default fields are indexed (for a list, see Section 11.2.4.2, "Default Columns in the Basic Asset Type Database Table"). If you want the fields that you created with `PROPERTY` statements in your asset descriptor file to be indexed, you must add statements for them in the following elements:

- `OpenMarketXcelerate/AssetType/YourAssetType/IndexAdd.xml`
- `OpenMarketXcelerate/AssetType/YourAssetType/IndexReplace.xml`

To add the asset type's custom fields to these elements, use the WebCenter Sites `INDEX` tags and follow the convention illustrated in these elements.

15.2.16 Step 14: Create and Assign Asset Type Icons (Contributor Interface Only)

Create and assign the icon that will represent the asset type in the Contributor interface's navigation trees (**Site Tree**, **Content Tree** and **My Work** tree). Icons can be of any type of image file (for example, PNG, GIF, and so on). In this example, we create an image file of type PNG.

1. Create an image no larger than 20x20 pixels representing the asset type.

2. Name the file using the syntax assetType.png. The file name determines the asset type for which the icon will be displayed. The name is case-sensitive.
3. Place the file in the following directory:

   ```
   <cs_app_dir>/Xcelerate/OMTree/TreeImages/AssetTypes/
   ```

   where `<cs_app_dir>` is the directory of the deployed WebCenter Sites application on your application server.
4. Restart your application server for the icons to appear in the Contributor interface's **Site Tree**, **Content Tree**, and **My Work** tree.

15.2.17 Step 15: Code Templates for the Asset Type

Creating your asset types and coding the templates for assets of that types is an iterative process. Although you need to create asset types before you can create templates for assets of that type, it is likely that you will discover areas that need refinement in your data design only after you have coded a template and tested the code.

For information about coding templates, see Chapter 28, "Coding Elements for Templates and CSElements."

15.2.18 Step 16: Move the Asset Types to Other Systems

When you have finished creating all of your new asset types (including creating templates for them), migrate them to the management and delivery systems. System administrators will then configure the asset types for the management system. They will enable revision tracking where appropriate, create workflow processes, create Start Menu shortcuts, and so on.

For information about this step, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

15.3 Deleting Basic Asset Types

When you delete an asset type that you created with AssetMaker, WebCenter Sites can either delete components of the asset type that you select, or **delete all traces of the asset type from the database**. A list of asset type components follows:

- The database table and **all the data in it**
- The elements in the `ElementCatalog` table
- The SQL statements in the `SystemSQL` table
- The row in the `AssetType` table
- Any rows in the `Association` table (optional)
- Any rows in the `Category` table (optional)
- Any rows in the `AssetPublication` table
- Any rows in the `AssetRelationTree` table

To delete an asset type that you created with AssetMaker

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.

4. Select the **Admin** tab.
5. Expand **AssetMaker** and then select the asset type that you want to delete.

   The Asset Type form appears:
6. Select the **Delete Asset** option and click **Submit**.

   WebCenter Sites displays a confirmation message.
7. Click **OK**.

16

Designing Flex Asset Types

As discussed in Chapter 11, "Data Design: The Asset Models," the data model for flex asset types can be thought of in terms of a family of asset types, with each asset type in the family having several database tables.

You create new flex asset types with the Flex Family Maker utility. However, when working with the flex asset model, developers not only create the flex asset types, they also create the individual data structure assets of those types. That is, flex attributes, flex parent definitions, flex definitions, and flex parent assets.

Typically, you design the flex asset types and create the data structure assets on a development system. Then when your data model is ready, you migrate your work from the development system to the management and delivery systems.

This chapter contains the following sections:

- Section 16.1, "Design Tips for Flex Families"
- Section 16.2, "The Flex Family Maker Utility"
- Section 16.3, "Creating a Flex Asset Family"
- Section 16.4, "Editing Flex Attributes, Parents, and Definitions"
- Section 16.5, "Using Product Sets"

16.1 Design Tips for Flex Families

Your job when designing your flex family is to create a data structure that meets the needs of two audiences:

- The visitors to your online site (that is, the users of your delivery system)
- The content providers who enter data into the WebCenter Sites database (that is, the users of the management system)

This section contains the following topics:

- Section 16.1.1, "Visitors on the Delivery System"
- Section 16.1.2, "Users on the Management System"
- Section 16.1.3, "How Many Attribute Types Should You Create?"
- Section 16.1.4, "Designing Flex Attributes"
- Section 16.1.5, "How Many Definition Types Should You Create?"
- Section 16.1.6, "Designing Parent Definition and Flex Definition Assets"
- Section 16.1.7, "Summary"

16.1.1 Visitors on the Delivery System

The experience of the visitors to your online site is based on the following asset types:

- flex asset
- flex attribute

Your online site pages display flex assets (assetsets) for the visitors through queries that are based on attribute values (searchstates). For more information see Section 11.4, "Assetsets and Searchstates." You use attribute values as the basis for drill-down searches that can give the appearance of a hierarchy on your online site if that is the look and feel that you want.

16.1.2 Users on the Management System

The users of your management system navigate through a visual hierarchical structure that you create for them with the following flex asset types:

- flex parent definition
- flex definition
- flex parent

Although the organizational structure that you create with these asset types does affect the data, it determines which attribute values are inherited by which flex assets, its biggest impact is on the users of the management system.

You are not required to use flex parents and flex parent definitions, but their inheritance properties make them a valuable tool for users who are maintaining a large amount of data such as an online catalog:

- Changing an attribute value at the parent level changes that value for all the flex assets who are children of that parent, which means you only have to change the value once.
- Inherited attribute values are values that aren't subject to user error during data entry, which means less data cleanup is required.

The inheritance tree that you create for your content providers has no bearing on how your site visitors navigate the online site you are designing. For example, if content is entered into your management system through some completely automated process (perhaps it is bulk loaded from an ERP system) you would have no need for parent asset types at all, yet you can still create drill-down searches on your online site.

16.1.3 How Many Attribute Types Should You Create?

As described in Section 11.4, "Assetsets and Searchstates," only the flex assets that share a common attribute type can belong to the same assetset because queries (searchstates) are based on attributes and not on the organizational constructs of parent definitions and flex definitions.

Therefore, when you design your data structure, remember that if you organize your data to use separate types of attributes, you might create a nicely delineated interface on the management system, but that data cannot be synthesized well on the delivery system and that is rarely what you want.

As a general rule, you should **create one type of attribute** for your system. If you need to, you can create more than one version of the rest of the family members (the flex asset type, flex definition type, flex parent type, and flex parent definition type), but they should still share the same pool of attributes.

For example, if the GE Lighting sample site had been designed such that the product family and the content family shared the same attribute type, you would be able to create assetsets that contained a product and a corresponding article about that product.

16.1.4 Designing Flex Attributes

Before you begin creating attributes, design them on paper. Determine all the attributes you need and decide where they will appear, with flex assets or the flex parents.

Start by planning out the bottom level of your hierarchy (that is, the individual instances of flex asset types like products) and determine the attributes you need for each item at that level. For example, if you plan to create flex filter assets, determine which attributes need to be created and assigned to the definitions as the input and output attributes for your filters.

You must determine all of the flex attributes that you need before you begin creating them because the way you plan to use them creates dependencies that you must account for when you create them.

16.1.4.1 Which Data Types

Assess the data types that are available for attributes and the default input types for those data types. Determine which data types will work best for which attributes. If you want to change the default input style for an attribute, you create an attribute editor for it before you create the attribute. See Chapter 18, "Designing Attribute Editors."

When you create a flex asset that uses an attribute of type `blob`, the format of the value entered for the attribute on an **Inspect** form depends on its type. For example, a text file shows the first 200 bytes in the file. An image file appears as a thumbnail image. And some files cannot be displayed at all. In this case, WebCenter Sites displays the message "filename **not displayable**" but the file location is still successfully recorded.

16.1.4.2 Using Attribute Editors

The default input type for an attribute depends on the data type that you select for it. If you do not want to use the default input type, you can create an attribute editor for the attribute.

Creating flex assets and their attribute editors is an iterative process. You can create the attribute editors first or you can create the attributes first and then go back and assign the attribute editors after you have created them. The process of creating attribute editors is described in Chapter 18, "Designing Attribute Editors."

Attributes of Type Blob

The default input style of an attribute of type `blob` is a text field with a **Browse** button. You use the **Browse** button to locate and select a file and WebCenter Sites uploads it to the default storage directory. You cannot use the WebCenter Sites forms to edit the contents of the file.

If you want to be able to enter content directly into the external file through the WebCenter Sites forms, you must assign an attribute editor to the attribute:

- If you use an attribute editor that uses the `TEXTAREA` input style, you can create a field that can hold up to 2,000 characters (entered through the forms); when saved, that content is written to the default storage directory.

- If you have CKEditor, you can use a CKEDITOR field to edit the contents of the external file that the attribute represents.

Attributes of Type Asset

The default input style for an attribute of type asset is a drop-down list of all the assets of the type specified. An unfiltered drop-down list is not recommended if you have more than 20 assets of that type.

In general, whenever you create an attribute of type asset, you should assign it an attribute editor.

- An attribute editor that uses the PICKASSET style checks to find out whether the tree is toggled on or off in the WebCenter Sites interface. If the tree is on, the user can select an asset from a tab in the tree. If the Tree is toggled off, the attribute editor displays a dialog box that lists the assets from the **Bookmarks** and **History** tabs.
- Another option is to use the PULLDOWN style but to supply a query asset that limits the options that appear in the list.
- If the number of assets that are valid choices is small, you can also use the CHECKBOXES or the RADIOBUTTONS input style, both of which require a query asset to identify the assets.

16.1.4.3 Where Will Each Attribute Be Used?

After you have determined the list of attributes, determine whether you plan to use them in a flex definition or a flex parent definition. Sort them logically by using the following guidelines:

- If an attribute's value is unique to an individual flex asset (product, article, image, for example), the attribute belongs at the bottom of the tree, with the flex asset.
- If an attribute's value is the same for multiple flex assets, the attribute belongs in a parent. (Of course there are always exceptions. For example, even if a toaster costs the same amount as a bowling ball, it is unlikely that they would inherit their prices from a common parent.)
- Based on that attribute distribution, you can determine how many flex definitions you need and how many parent definitions you need.

Remember that there is both a physical limit (based on your DBMS) and a psychological limit (user satisfaction) as to how many attributes you can or should use in an individual flex asset or flex parent. Someone has to enter all those values. Be sure to create and then assign to the definitions only those attributes that you really plan to use. It is very easy to add attributes in the future if you decide that you need additional ones.

16.1.4.4 Dependencies Imposed by Hierarchy

After you know where an attribute will be used, you can determine whether hierarchical concerns add requirements to the attribute. For example, if an attribute is to used by a flex parent and your data structure allows flex assets to have more than one parent, the attribute must be configured to hold multiple values because a flex asset might inherit more than one value for it.

In general, try not to make the inheritance structure too complex.

16.1.5 How Many Definition Types Should You Create?

The appearance and input of data on the management system is based on the flex asset definitions and the flex parent definitions. Parents and flex assets appear on tabs in the tree in the WebCenter Sites interface based on the hierarchy that you create through the definitions.

In general, it is best to create a separate set of definition types for each flex asset member in a family.

For example, the GE Lighting sample site has two flex asset members in the content family: article (flex) and image (flex). They share parents, parent definitions, and flex definitions. This means that some attributes are left blank for the image assets because they don't apply and some attributes are left blank for the article assets because they don't apply.

It would be better to have article parents, article definitions, and article parent definitions that are different from image parents, image definitions, and image parent definitions. But they should absolutely share the same attribute type, which they do.

16.1.6 Designing Parent Definition and Flex Definition Assets

The hierarchy on the tabs in the tree in the WebCenter Sites interface is created through the flex parent definitions and flex definitions:

- To set a hierarchy three levels deep, you need at least two parent definitions and at least one flex definition.
- To specify a hierarchy two levels deep, you need at least one parent definition and at least one flex definition.

Be sure to consider the basic tenets of usability when you set up a structural hierarchy with the flex definitions and flex parent definitions. For example:

- How deep can the hierarchy go before the content providers feel lost in the tree?
- How many attribute values can be inherited to alleviate the possibility of user error during input?
- How many options can be comfortably displayed in a drop-down list?

If you create a system that is overly difficult to use, the content providers will complain.

16.1.6.1 Determining Hierarchical Place

Open WebCenter Sites, log in to the GE Lighting sample site, and examine the form for a new product parent definition or for a new product definition.

In the **Parent Definition** section of these forms, you determine two things:

- The hierarchical position of the assets that use this definition and determine
- The parents that they can inherit attributes from

Remember that although the hierarchical position has meaning only in the Contributor interface on the management system, the attributes that they inherit have meaning both on the management system and on your online site.

The text box named **Available** lists all the existing parent definitions. You use this section of the form to specify how many parents are possible by selecting parent definitions from the **Available** list and moving it to the **Selected** list.

When you create a parent asset or a flex asset, the **New** form displays a drop-down field for each definition that you selected from the **Available** list when you created the definition that you are using to create the new parent or flex asset. The drop-down list in the **New** form displays all the parents that were created with that definition.

If the parent that is selected in the **New** form has any attribute values, the asset inherits them.

How many possible parents should you allow? In general, it is best to keep this simple. The more parent definitions you select from the **Available** list, the more fields the content providers have to fill out when they create a new flex asset.

If you do not select a parent definition in the **Available** list, it means that assets created with this definition are positioned at the top level of the tree on the tab that displays your flex assets.

The best way to understand how parent definitions, flex definitions, parent assets, and flex assets interact is to examine the assets delivered with the GE Lighting sample site.

16.1.6.2 Determining Attribute Inheritance

You configure attribute inheritance in the **Attributes** section of the parent definition form. You use that section to specify the attributes that define the parents that are created with this definition.

When you create a parent with this definition, the values that are entered for these attributes are passed down to the flex assets that are children of the parent asset.

16.1.6.3 How Many Flex Parent Definition Assets?

The simple answer is "as many as you need." Be sure to consider usability when you decide how many flex parent definition assets you need, and how many parent assets of those definitions that you need.

If you create many parent definitions, it probably means that you will have fewer parents created with each definition, which leads to shorter drop-down lists in the new parent and new flex asset forms. Short drop-down lists make it easier for content providers to select the correct parent from the list.

However, if your data needs require you to have a small number of parent definitions and a large number of parents, create a tab that lists all the parents so the content providers can select the correct parent asset from the tab.

16.1.6.4 How Many Flex Definition Assets?

A general rule is this: create enough flex definitions so that fields (attributes) are not left blank on the **New** and **Edit** flex asset forms.

If you create too few definitions, you run the risk of creating long forms with lots of attribute fields, not all of which apply for each asset. When you have long forms with lots of attribute fields, not only do content providers have to sort through the form to determine which attributes apply to the asset they are currently creating, the form takes a long time to be rendered in the user's browser.

16.1.7 Summary

Keep the following rules in mind as you design the data structure with a flex family for your online site:

- Carefully planned, easy-to-use asset design (data design) makes content providers happy.

- Usable layout and efficient code makes site visitors happy.

And both user groups need efficient systems that perform well.

16.2 The Flex Family Maker Utility

When you create a flex family with Flex Family Maker, it does the following:

- Creates several database tables (the number depends on which flex asset types that you create).
- Writes information about the new flex family to the following tables:
  - `FlexAssetTypes`, which holds a row for each flex asset member type
  - `FlexGrpTmplTypes`, which holds a row for each flex parent definition type
  - `FlexGrpTypes`, which holds a row for each flex parent type
  - `FlexTmplTypes`, which holds a row for each flex definition type
- Creates new directories in the `ElementCatalog` table using the following naming convention:

  `OpenMarket/Xcelerate/AssetType/`NameOfYourAssetType
- Copies elements from the `ElementCatalog` table to the directories in created for your asset types. WebCenter Sites use these elements to format the **New**, **Edit, Inspect**, **Search**, and **Search Results** forms for assets of that type.

For information about the main database tables for flex assets and flex parent assets, see Section 11.3.6, "Flex Families and the Database." For information about all the database tables in a flex family, see the *Oracle Fusion Middleware WebCenter Sites Database Schema Guide*.

The Flex Asset Elements

When you create a new flex asset type, Flex Family Maker copies elements to the following location in the `ElementCatalog` table:

```
OpenMarket/Xcelerate/AssetType/NameOfAssetType
```

For example, the GE sample site product asset elements are in:

```
OpenMarket/Xcelerate/AssetType/Products
```

It also creates a SQL statement that the search elements use and places it in the `SystemSQL` table under `OpenMarket/Xcelerate/AssetType/`NameOfAssetType.

For a description of the elements and the SQL statement that Flex Family Maker copies for you, see Section 15.1.4, "Elements and SQL Statements for the Asset Type." The elements for flex assets are the same as the elements for the basic assets with the exception of the `AppendSelectDetailsSE` element.

16.3 Creating a Flex Asset Family

When you are using the flex asset data model to represent the content you want to display on your online site, you and the other developers do not create only the flex asset types. You also create the individual data structure assets of those types: flex attributes, flex parent definitions, flex definitions, and flex parent assets.

This section contains the following topics:

- Section 16.3.1, "Overview of Creating a New Flex Asset Type or Family of Flex Asset Types"

- Section 16.3.2, "Before You Begin"
- Section 16.3.3, "Step 1: Create a Flex Family"
- Section 16.3.4, "Step 2: (Optional) Create Additional Flex Family Members"
- Section 16.3.5, "Step 3: Enable the New Flex Asset Types"
- Section 16.3.6, "Step 4: Create Flex Attributes"
- Section 16.3.7, "Step 5: (Optional) Create Flex Filter Assets"
- Section 16.3.8, "Step 6: Create Parent Definition Assets"
- Section 16.3.9, "Step 7: Create Flex Definition Assets"
- Section 16.3.10, "Step 8: Create Flex Parent Assets"
- Section 16.3.11, "Step 9: Create and Assign Asset Type Icons (Contributor Interface Only)"
- Section 16.3.12, "Step 10: Code Templates for the Flex Assets"
- Section 16.3.13, "Step 11: Test Your Design (Create Test Flex Assets)"
- Section 16.3.14, "Step 12 (Optional): Create Flex Asset Associations"
- Section 16.3.15, "Step 13: Move the Asset Types to Other Systems"

16.3.1 Overview of Creating a New Flex Asset Type or Family of Flex Asset Types

Following is an overview of the process for creating a new flex asset type or family of flex asset types. Where you start in the process depends how many asset types you need to design. If you can base your data structure on either of the sample site flex families, you do not have to create an entire flex family, you can create only the new members that you need.

This chapter describes each of the following steps in the process, except as noted:

1. Create the new flex family (Section 16.3.3, "Step 1: Create a Flex Family").
2. Create additional flex family members, as necessary (Section 16.3.4, "Step 2: (Optional) Create Additional Flex Family Members").
3. Enable the new asset types on all the WebCenter Sites sites on the development system (Section 16.3.5, "Step 3: Enable the New Flex Asset Types") and create **Start Menu** shortcuts for all the new asset types. Also, place the new flex definition, flex parent definition, and attribute types on the **Design** tab, and create a tab for your new flex parent and flex asset types.
4. Create the flex attributes and design your attribute editors (Section 16.3.6, "Step 4: Create Flex Attributes"). For more information about attribute editors, see Chapter 18, "Designing Attribute Editors."
5. (Optional) Create the flex filter assets (Section 16.3.7, "Step 5: (Optional) Create Flex Filter Assets"). For more information about flex filters, see Chapter 17, "Flex Filters."
6. Create the flex parent definitions (Section 16.3.8, "Step 6: Create Parent Definition Assets").
7. Create the flex definition assets (Section 16.3.9, "Step 7: Create Flex Definition Assets").
8. Create the flex parent assets (Section 16.3.10, "Step 8: Create Flex Parent Assets").

9. Create and assign icons that will represent the members of your flex family in search results lists that are displayed in the Thumbnail view (Section 16.3.11, "Step 9: Create and Assign Asset Type Icons (Contributor Interface Only)").
10. Create templates for the flex assets, the flex member of the flex family (Section 16.3.12, "Step 10: Code Templates for the Flex Assets").
11. Test your design by creating enough flex assets to examine the data structure that you have designed (Section 16.3.13, "Step 11: Test Your Design (Create Test Flex Assets)").
12. (Optional). Create flex asset associations (Section 16.3.14, "Step 12 (Optional): Create Flex Asset Associations").
13. Move your asset types to other systems (management and delivery; Section 16.3.15, "Step 13: Move the Asset Types to Other Systems"). This step is described in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

16.3.2 Before You Begin

Set up your development system and get access to it, as follows:

- Create the appropriate WebCenter Sites sites.
- Create a user name for yourself that has administrator rights, and enable that user name on all of the sites on your development system. Note that without administrator rights, you do not have access to the **Admin** tab, which means that you cannot perform some of the procedures in this chapter.

  For the sake of convenience, assign the **Designer** and **GeneralAdmin** roles to your user name. That way you will have access to all the tabs in the WebCenter Sites interface and all of the existing **Start Menu** shortcuts for the assets in the sample site. (Be sure that the TableEditor ACL is assigned to your user name.)

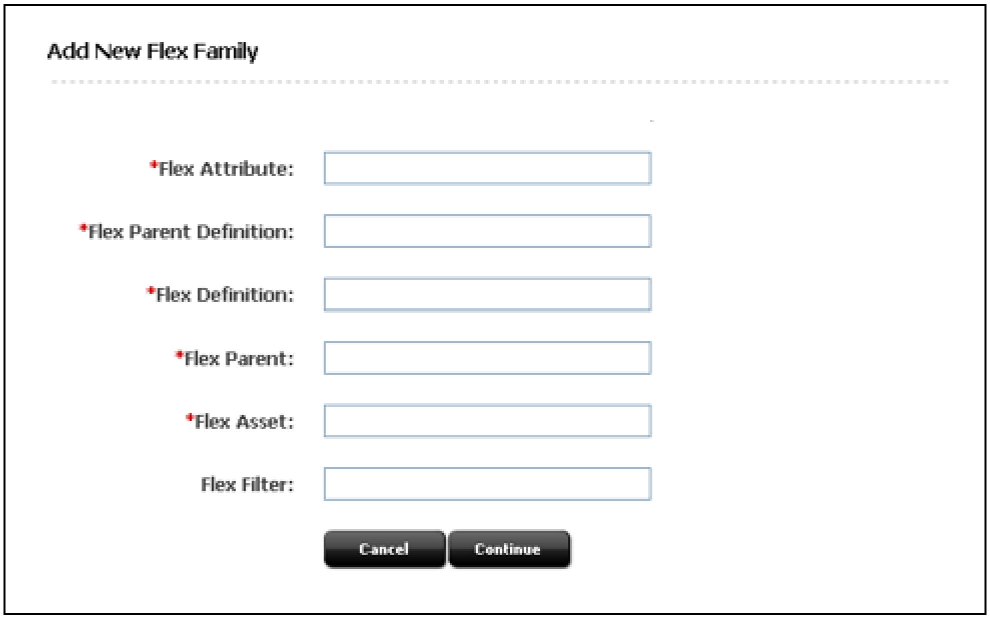
For information about these tasks, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

16.3.3 Step 1: Create a Flex Family

To create a new flex family

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Select the **Admin** tab.
5. Expand **Flex Family Maker** and click **Add New Family**.

   The Add New Flex Family form appears:



Add New Flex Family

*Flex Attribute:

*Flex Parent Definition:

*Flex Definition:

*Flex Parent:

*Flex Asset:

Flex Filter:

6. For **Flex Attribute**, enter the name of the new flex attribute asset type.

   The name you enter in this field is the internal name of the new attribute asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

7. For **Flex Parent Definition,** enter the name of the new flex parent definition asset type.

   The name you enter in this field is the internal name of the new parent definition asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

8. For **Flex Definition**, enter the name of the new flex definition asset type.

   The name you enter in this field is the internal name of the new flex definition asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

9. For **Flex Parent**, enter the name of the new flex parent asset type.

   The name you enter in this field is the internal name of the new parent asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

10. For **Flex Asset,** enter the name of the new flex asset type.

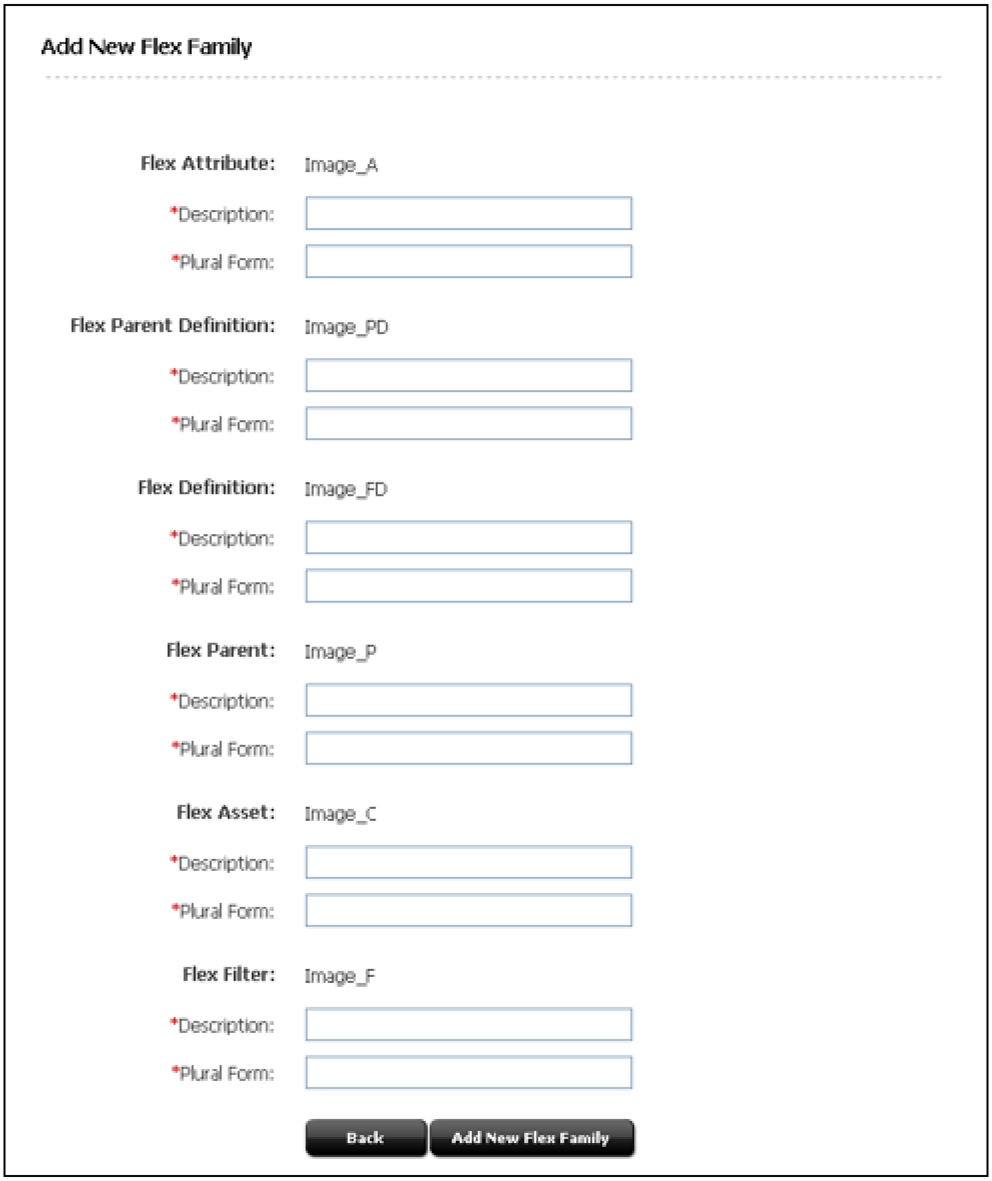
    The name you enter in this field is the internal name of the new flex asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

11. For **Flex Filter**, enter the name of the new flex filter asset type.

    The name you enter in this field is the internal name of the new flex filter asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

12. Click **Continue**.

    Flex Family Maker displays the following form:



Add New Flex Family

Flex Attribute: Image_A

*Description:

*Plural Form:

Flex Parent Definition: Image_PD

*Description:

*Plural Form:

Flex Definition: Image_FD

*Description:

*Plural Form:

Flex Parent: Image_P

*Description:

*Plural Form:

Flex Asset: Image_C

*Description:

*Plural Form:

Flex Filter: Image_F

*Description:

*Plural Form:

13. For each new member of the family, click in the **Description** field and enter the external name of the asset type, that is, the name of the asset type when it is displayed in WebCenter Sites. This is the name that appears on the forms (**New**, **Edit**, **Inspect**, and so on).

14. For each new member of the family, click in the **Plural** field and enter the plural version of its name. This version is used in status messages and so on when appropriate.

15. Click **Add New Flex Family**.

    Flex Family Maker creates the database tables that will store assets of these types. For information about these tables, see Section 11.3.6, "Flex Families and the Database."

    It also copies elements that format the forms for assets of these types to a directory with the name of the asset type in the `ElementCatalog` and `SystemSQL` tables.

16.3.4 Step 2: (Optional) Create Additional Flex Family Members

If you need to create additional flex family members (for example, if you need more than one flex asset type per member category), do the following:

1. In the **Admin** tab, expand the flex family you just created.
2. Drill down the flex family tree until you reach the category (flex parent for example) for which you want to create another member.
3. Under the desired member category, double-click **Add New** Member Category **Asset Type**.
4. WebCenter Sites displays the New Member Category Asset Type form.
5. In the form, fill out the required fields and click **Save**.

   WebCenter Sites displays a message confirming that the asset type was created.
6. Repeat this procedure for each additional flex family member you want to create.

16.3.5 Step 3: Enable the New Flex Asset Types

Before you can start creating assets (attributes, flex parent definitions, and so on), you must complete some steps on the **Admin** tab so that you have access to them. Note that your login must grant you administrator rights in order for you to have access to the **Admin** tab.

Complete the following steps:

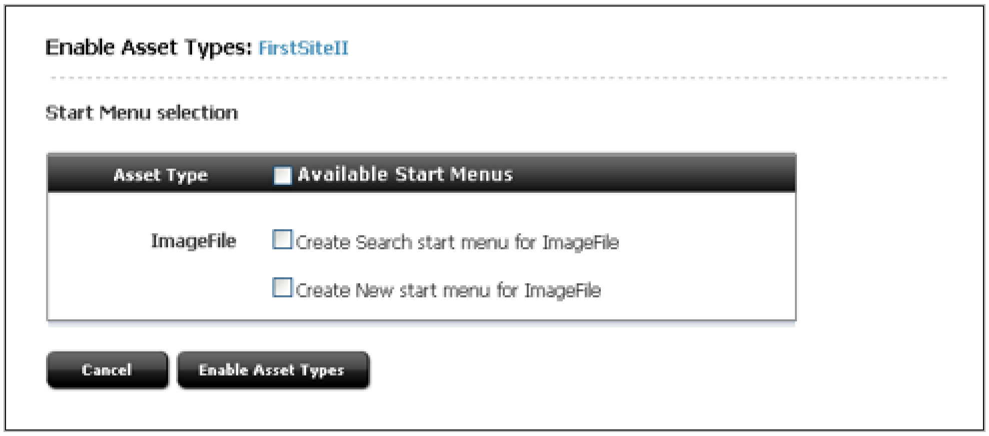
1. On the **Admin** tab, click the **Sites** icon and complete the following steps:

   a. Select the site that you are going to use to work with this asset type.

   b. Under that site, select **Asset Type** and then **Enable Asset Types**.

   c. Select your new asset types from the list and click **Enable Asset Types**.

   d. WebCenter Sites can automatically create a **New Start Menu Item** and/or a **Search Start Menu Item** for the Asset Types you are enabling. Check the box next to any available Start Menu Item that you would like WebCenter Sites to create.



Enable Asset Types: FirstSiteII

Start Menu selection

Asset Type	□ Available Start Menus
ImageFile	□ Create Search start menu for ImageFile
	□ Create New start menu for ImageFile

   If you choose not to generate these menu items at this time, you or your site administrator must manually create them later (no one can create assets of the enabled asset types until Start Menu items are created for them).

   e. Repeat steps a through d for each appropriate site.

2. Click **Enable Asset Types**.
3. The asset types are now enabled for the site(s). If you did not use WebCenter Sites to generate start menu items, you or your site administrator must now manually create them. As the developer of the asset types and the designer of the online site, your responsibility is to let the administrator know enough about your asset and site design that the site administrator can configure meaningful Start Menu items.

   You (the developers) must let the site and system administrators know which fields are used by the queries, collections, or other design elements for your online site so that they can create meaningful Start Menu items for the content providers. For more information about creating Start Menu items, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

After you or your administrator has created Start Menu items for your new asset types, you can create assets of these types. Note that even if you add your asset types to a tab, you will not be able to create new assets until you have created Start Menu shortcuts for them.

16.3.6 Step 4: Create Flex Attributes

Because the steps that you follow can differ significantly based on the data type that you select for your attribute, this section presents several procedures:

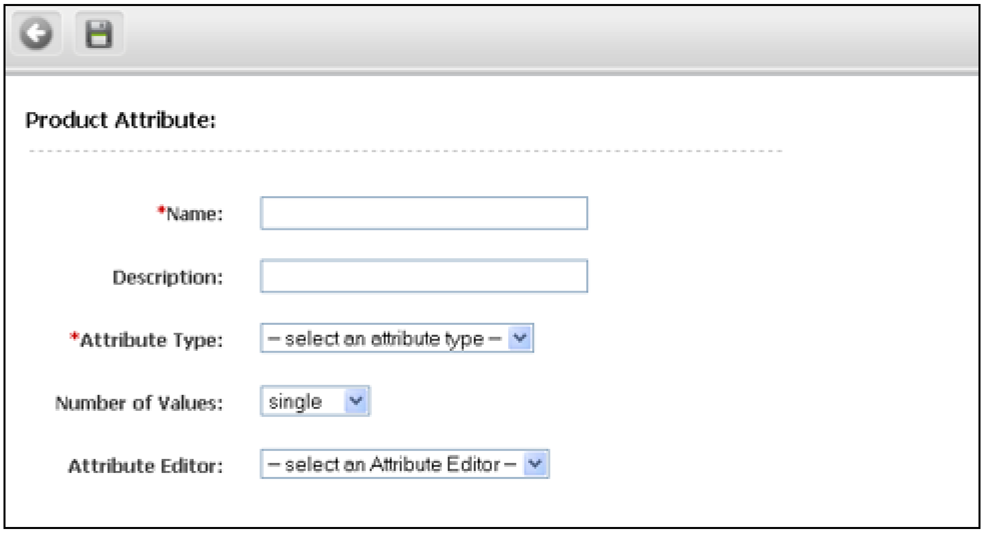
- A basic procedure for creating attributes of most data types
- Creating attributes of type `blob`
- Creating attributes of type `asset`
- Creating foreign attributes (that is, attributes that are stored in a foreign table). You must register the foreign table that contains the data you wish to use as a flex attribute. For information, see Section 12.3.4, "Registering a Foreign Table."

16.3.6.1 Creating Flex Attributes: Basic Procedure

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.

   Select the **Admin** interface.
3. Click **New** and select the name of your attribute type from the list of shortcuts. For example, **New Product Attribute**. If the workflow associate with this asset requires you to select assignees, select at least one user from each role and click **Set Assignees**.

   The new attribute form displays.



Product Attribute:

*Name:

Description:

*Attribute Type: – select an attribute type –

Number of Values: single

Attribute Editor: – select an Attribute Editor –

4. Click in the **Name** field and enter a name of up to 64 characters, excluding spaces.

5. Click in the **Description** field and enter a short, descriptive phrase that describes the use or function of the attribute.

6. Click in the **Value Type** field and select a data type for this attribute.

   – If you select **blob**, see Section 16.3.6.2, "Creating Flex Attributes of Type Blob (Upload Field)" and start with step 3.

   – If you select **asset**, see Section 16.3.6.3, "Creating Flex Attributes of Type Asset."

   – If you wish to create a foreign attribute, select a value other than **asset** or **blob**, complete step 7, and continue with step 3 in the Section 16.3.6.4, "Creating Foreign Flex Attributes."

   – If you select **text**: The WebCenter Sites Admin interface supports searches on attribute values of all types except **text**. For information about performing searches on attribute values, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.

   If you need help deciding which data type is appropriate for your attribute, see Section 11.3.3.1, "Data Types for Attributes."

7. Click in the **Number of Values** field and select either **single** or **multiple** from the drop-down list, as appropriate for the data type that you selected in the **Value Type** field.

   If this attribute is to be used by a flex parent and your data structure allows multiple flex parents for a flex asset, you must select **multiple** because the flex assets that inherit values for this attribute might inherit a value from more than one parent.

Note: When an attribute is configured to accept multiple values, it appears on the flex parent and flex asset forms as a field with an **Add Another attribute name** button.

If you want the attribute to accept multiple values for inheritance reasons but you do not want content providers to select more than one value for the attribute for individual parents or flex assets, assign the attribute an attribute editor that presents it as a single-value field (but select multiple in the **Value Type** field).

WebCenter Sites does not allow attributes of type `text` to have multiple values, due to the way these attributes are stored in the database. A message denoting this restriction appears if you attempt to save an attribute configured in such a way.

---

8. (Optional) If you do not want to use the default input type for this attribute (which is based on the data type that you selected in the **Value Type** field), click in the **Attribute Editor field** and select one from the drop-down list.

   If you need more information:

   For a list of the default input types (so you can determine whether you want to use an attribute editor instead), see Section 11.3.3.2, "Default Input Styles for Attributes."

   For information about creating attribute editors, see Chapter 18, "Designing Attribute Editors."

   For information about which attribute editors are appropriate for the data type of this attribute, see Section 18.1.2, "The Attribute Editor Asset."

9. (Optional) If you need to override the default ISO character set (ISO 8859-1), click in the **ISO Character Set** field and enter the one you want to use for this attribute.

10. Click **Save**.

16.3.6.2 Creating Flex Attributes of Type Blob (Upload Field)

To create an attribute of type `blob`

1. Complete steps 1 through 4 in Section 16.3.6.1, "Creating Flex Attributes: Basic Procedure."

2. Click in the **Value Type** field and select **blob**.

3. (Optional) Click in the **Folder** field and enter a path to the directory that you want to store the attribute values in. Note that the value that you enter in this field is appended to the value set as the default storage directory (defdir) for the `MungoBlobs` table.

4. Click in the **Number of Values** field and select **single** or **multiple**, as appropriate. For more information about this field, see Section 16.3.6.1, "Creating Flex Attributes: Basic Procedure."

5. (Optional) If you do not want to use the default input type (a **Browse** button), click in the **Attribute Editor** field and select one of the following:

   – An attribute editor that specifies the `TEXTAREA` input style. For information about attribute editors, see Section 18.1.2, "The Attribute Editor Asset."

- If your system is configured to use CKEditor, an attribute editor that specifies the CKEDITOR input style.

6. Complete steps 8 and 9 of Section 16.3.6.1, "Creating Flex Attributes: Basic Procedure."

16.3.6.3 Creating Flex Attributes of Type Asset

To create an attribute of type `asset`

1. Complete steps 1 through 4in Section 16.3.6.1, "Creating Flex Attributes: Basic Procedure."
2. Click in the **Value Type** field and select **asset**.
3. Click in the **Asset Type** field and select an option from the drop-down list.
4. Click in the **Mirror Dependency Type** field and select a dependency type.
5. Click in the **Number of Values** field and select either **single** or **multiple** from the drop-down list, as appropriate for the data type that you selected in the **Value Type** field.

   If this attribute is to be used by a flex parent and your data structure allows flex assets to have more than one flex parent, you must select **multiple** because the flex assets who inherit values for this attribute might inherit a value from more than one parent.
6. (Optional) If the number of assets of the type you selected in the **Number of Values** field is more than 20, click in the **Attribute Editor field** and select one. See Section 16.1.4.2, "Using Attribute Editors" for information about appropriate attribute editors.
7. Complete steps 8 and 9 of Section 16.3.6.1, "Creating Flex Attributes: Basic Procedure."

16.3.6.4 Creating Foreign Flex Attributes

If you keep data in another system (a price list, for example) that you also want to use for your flex assets, you can create a foreign attribute that points to the column in the foreign table whose data you want to use as a flex attribute.

To create a foreign attribute

1. Register the foreign table that contains the data you wish to use as a flex attribute. For information, see Section 12.3.4, "Registering a Foreign Table."
2. Complete steps 1 through 6 in Section 16.3.6.1, "Creating Flex Attributes: Basic Procedure." Note that you cannot select either *asset* or *blob* (or url) in the **Value Type** field.
3. (Optional) If you plan to use WebCenter Sites flex asset forms to enter values for the attribute into the foreign table and you do not want to use the default input type for the data type that you selected in the **Value Typ**e field, click in the **Attribute Editor** field and select an appropriate option.
4. Click in the **Editing Style** field and do one of the following:
   - If you want to use WebCenter Sites forms to enter values into this attribute's fields for the flex assets that use it, select **local**.
   - If you do not want users to be able to write values to this table through WebCenter Sites forms, select **external**.

5. Click in the **Storage Style** field and select **external** from the drop-down list.

6. Click in the **External ID** field and specify the name of the column that serves as the primary key for the table that holds this foreign attribute, that is, the column that uniquely identifies the attribute.

7. Click in the **External Table** field and enter the name of the table that stores this attribute.

8. Click in the **External Column** and enter the name of the column in the table specified in the **External Table** that holds the values for this attribute.

9. Complete steps 8 and 9 of Section 16.3.6.1, "Creating Flex Attributes: Basic Procedure."

16.3.7 Step 5: (Optional) Create Flex Filter Assets

This section contains general instructions for creating a flex filter asset whose functionality is defined by one of the default WebCenter Sites filter classes. For information about flex filter classes, and detailed instructions on creating flex filter assets, see Chapter 17, "Flex Filters."

Before you can create flex filter assets, the flex attributes that you plan to use as the input and output attributes must already be created. If the appropriate flex attributes do not exist yet, create them. Note the following requirements:

- For flex filters that use the Document Transformation filter type, the input and output attributes must be of type blob.

- For any flex filter, the input attribute, output attribute, and flex filter asset must all belong to the same flex family.

To create a flex filter asset

1. Log in to WebCenter Sites as an administrator.

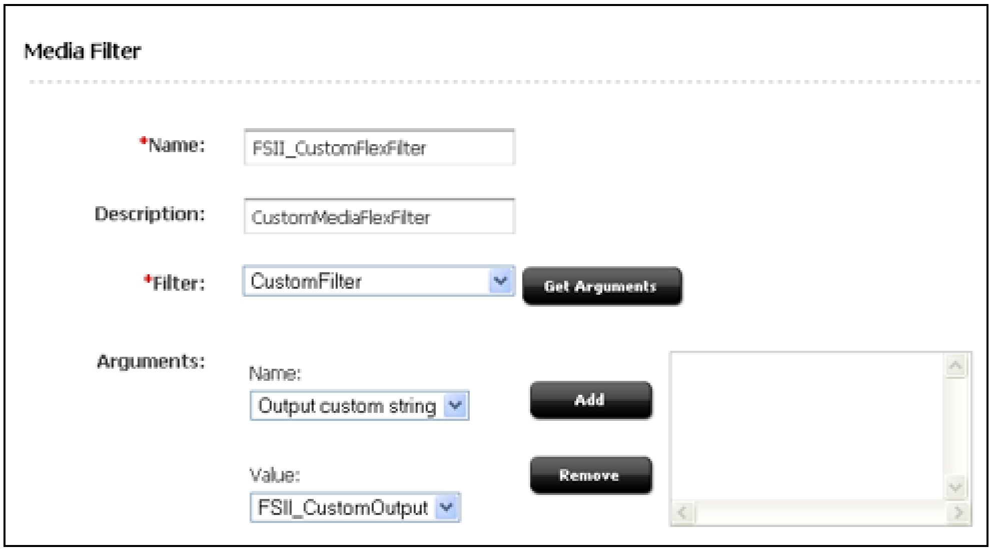
2. Select the site in which you want to work.

3. Select the **Admin** interface.

4. From the start menu options, click **New**.

5. In the list of asset types, select the type of filter you wish to create. For example, **New Media Filter**. If the workflow associated with this asset requires you to select assignees, select at least one user from each role and click **Set Assignees**.

   The new filter form appears:



Media Filter

*Name:	FSII_CustomFlexFilter
Description:	CustomMediaFlexFilter
*Filter:	CustomFilter
Arguments:	Name: Output custom string Value: FSII_CustomOutput

6. In the New form, fill in the following fields:
   - **Name**: Enter a unique name for this filter asset.
   - **Description**: Enter a brief description summarizing the filter's function.
   - **Filters**: Select the filter class that will define the functionality of the flex filter asset you are creating. By default, the filter options are Doc-Type, Document Transformation, FieldCopier, and ThumbnailCreator.

   > **Note:** If custom filter classes have been created for your system, they will also appear in this list. If you wish to create a custom flex filter class, see Section 17.2, "Defining a Flex Filter Class and Creating a Flex Filter Asset."

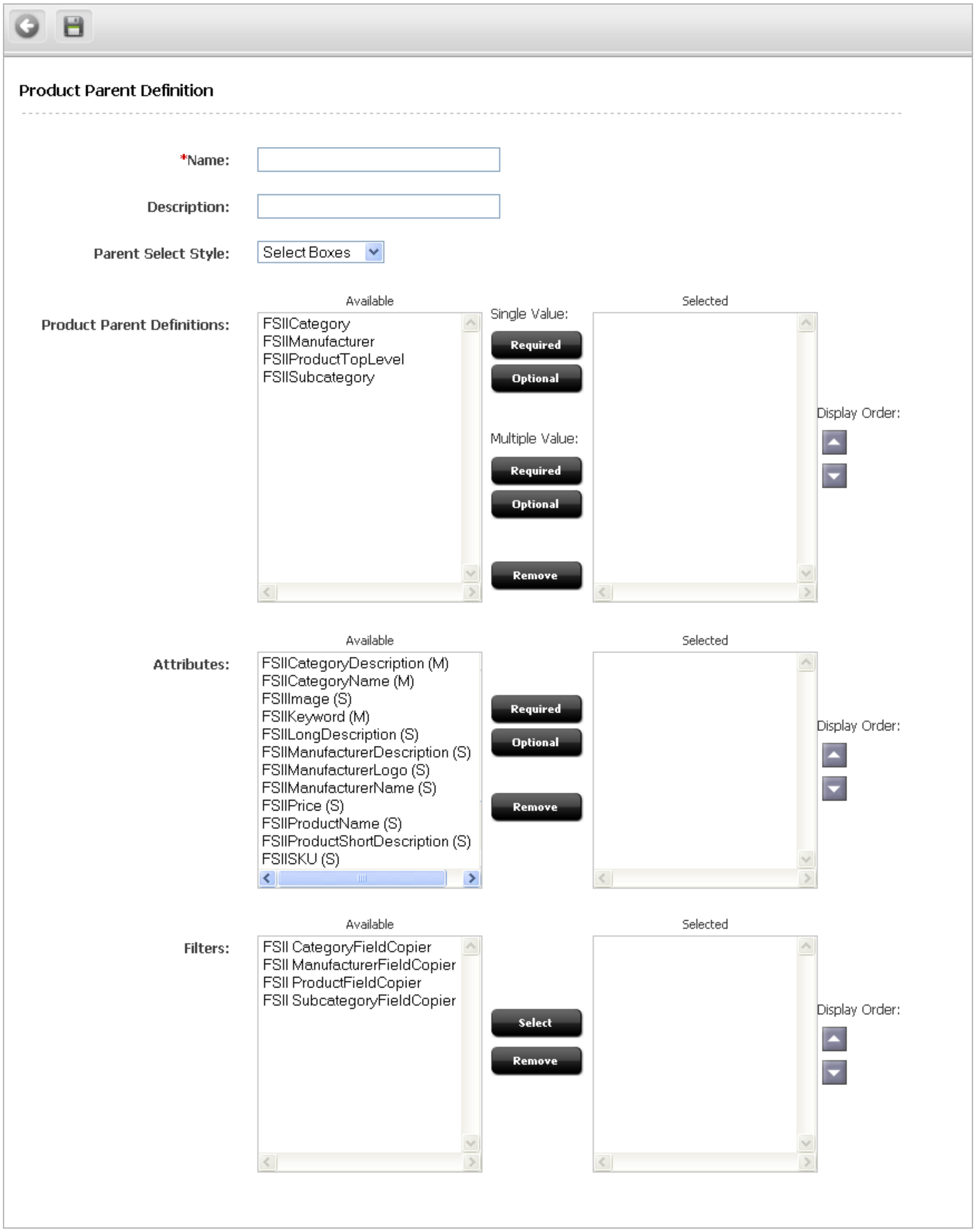
7. Click **Get Arguments**. In the Arguments field, specify the input and output arguments for the flex filter asset. Click **Add** to add the arguments to the filter.
8. Click **Save**.

16.3.8 Step 6: Create Parent Definition Assets

Complete the following steps:

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. From the start menu options, click **New**.
5. Select the name of your product definition asset from the list of shortcuts. For example, **New Product Parent Definition**. If the workflow associated with this asset requires you to select assignees, select at least one user from each role and click **Set Assignees**.

   The new form appears:



Product Parent Definition

*Name:

Description:

Parent Select Style: Select Boxes

Product Parent Definitions:

Available

FSIICategory
FSIIManufacturer
FSIIProductTopLevel
FSIISubcategory

Selected

Attributes:

Available

FSIICategoryDescription (M)
FSIICategoryName (M)
FSIIImage (S)
FSIIKeyword (M)
FSIILongDescription (S)
FSIIManufacturerDescription (S)
FSIIManufacturerLogo (S)
FSIIManufacturerName (S)
FSIIPrice (S)
FSIIProductName (S)
FSIIProductShortDescription (S)
FSIISKU (S)

Selected

Filters:

Available

FSII CategoryFieldCopier
FSII ManufacturerFieldCopier
FSII ProductFieldCopier
FSII SubcategoryFieldCopier

Selected

6. Click in the **Name** field and enter a name of up to 64 characters.

7. Click in the **Description** field and enter a short, descriptive phrase that describes the parent definition.

8. Click in the **Parent Select Style** field and determine how flex parents that use this definition will be selected on the parent asset forms. Do one of the following:

- If the number of parents of this type will be small, choose **Select Boxes**. Then, all the parents of this type will be displayed as options in a drop-down field on the flex asset forms.
- If the number of parents of this type will be large, choose **Pick From Tree**. Then, when you select a parent of this type on the flex asset form, you select it from the tree on the tab that displays your catalog data. For example, on the GE Sample site, the catalog data is displayed in a tree on the **Catalog** tab.

9. Select a parent definition from the **Available** list. For information about selecting parent definitions, see Section 16.1.6.1, "Determining Hierarchical Place."
10. Click the appropriate arrow, as described in this table:

Table 16–1 Buttons in Parent Definition Form

Button in parent definition form	Creates a field in the New parent form that does the following:
Single Value (Required)	Forces you to select one parent for the field.
Single Value (Optional)	Lets you select only one parent for the field.
Multiple Value, (Required)	Forces you to select at least one parent asset for the field.
Multiple Value (Optional)	Lets you select more than one parent asset for the field.

WebCenter Sites moves the parent definition from the **Available** list to the **Selected** list.

1. Repeat steps 6 and 7 as many times as necessary. Remember that the corresponding **New** parent form will include a field for each item that you select in the **Available** list on this parent definition form.
2. In the **Attributes** section, select the appropriate attributes. Note that if you are going to assign a flex filter asset to this parent definition, you must include the input and output attributes that the flex filter uses.
3. Do one of the following:
   - Click the **Required** button to specify that the attribute is required, that is, all flex parents created with this definition must have a value for this attribute.
   - Click the **Optional** button to specify that the attribute is optional.
4. (Optional) If you did not select the attributes in the order in which you want them to appear on the parent form for parents of this type, use the arrows to the right of the **Selected** box to order them.
5. (Optional) In the **Filters** section, select any flex filter assets that are appropriate for this parent definition.
6. Click **Save**.
7. Repeat this procedure for each parent definition asset that you need to create.

16.3.9 Step 7: Create Flex Definition Assets

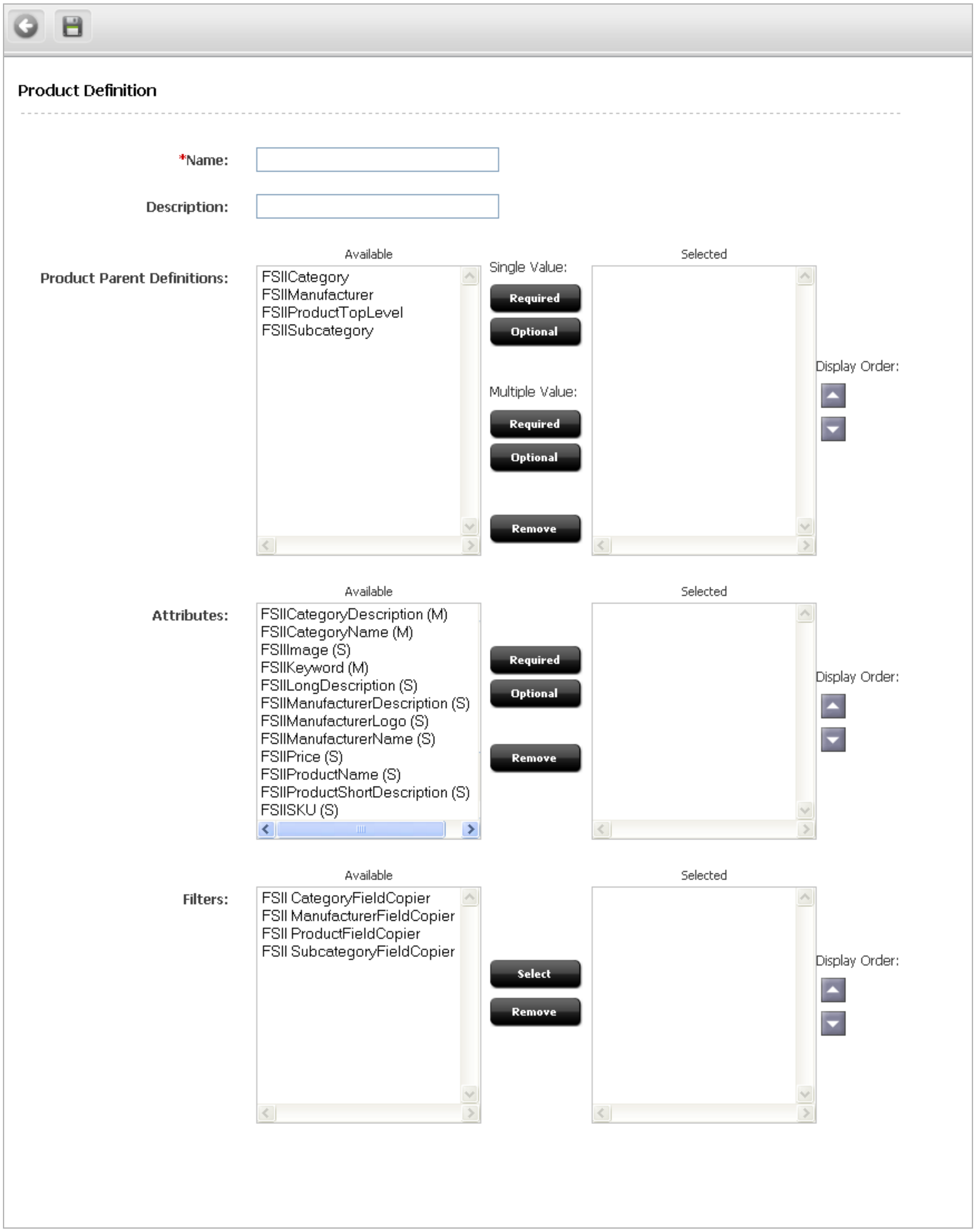
Complete the following steps:

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.

4. From the start menu options, click **New**.

5. Select the name of your flex definition asset type from the list of shortcuts. For example, **New Product Definition**. If the workflow associated with this asset requires you to select assignees, select at least one user from each role and click **Set Assignees**.

   The new form appears:



Product Definition

*Name:

Description:

Product Parent Definitions:

Available: FSIICategory, FSIIManufacturer, FSIIProductTopLevel, FSIISubcategory

Attributes:

Available: FSIICategoryDescription (M), FSIICategoryName (M), FSIIImage (S), FSIIKeyword (M), FSIILongDescription (S), FSIIManufacturerDescription (S), FSIIManufacturerLogo (S), FSIIManufacturerName (S), FSIIPrice (S), FSIIProductName (S), FSIIProductShortDescription (S), FSIISKU (S)

Filters:

Available: FSII CategoryFieldCopier, FSII ManufacturerFieldCopier, FSII ProductFieldCopier, FSII SubcategoryFieldCopier

6. Click in the **Name** field and enter a name of up to 64 characters.

7. Click in the **Description** field and enter a short, descriptive phrase that describes the parent definition.
8. Select a parent definition from the **Available** list. For information about selecting parent definitions, see Section 16.1.6.1, "Determining Hierarchical Place."
9. Click the appropriate arrow, as described in the following table:

Table 16–2 Buttons in Flex Definition Form

Button in flex definition form	Creates a field in the New flex asset form that does the following:
Single Value (Required)	Forces you to select only one parent in the field.
Single Value (Optional)	Lets you select only one parent in the field.
Multiple Value (Required)	Forces you to select at least one parent asset in the field.
Multiple Value (Optional)	Lets you select more than one parent asset in the field.

WebCenter Sites moves the parent definition from the **Available** list to the **Selected** list.

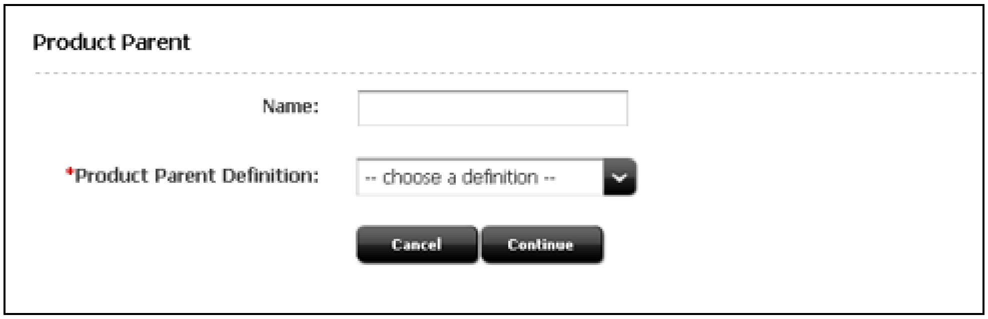
1. Repeat steps 5 and 6 as many times as is necessary. Remember that the corresponding **New** flex asset form will include a field for each item that you select in the **Available** list on this flex definition form.
2. In the **Attributes** section, select an attribute. Note that if you are going to assign a flex filter asset to this flex definition, you must include the input and output attributes that the flex filter uses.
3. Do one of the following:
   - Click the **Required** button to specify that the attribute is required; that is, that all flex assets created with this definition must have a value for this attribute.
   - Click the **Optional** button to specify that the attribute is optional.
4. (Optional) If you did not select the attributes in the order in which you want them to appear on the **New** and **Edit** forms for flex assets created with this definition, use the arrows to the right of the **Selected** box to order them.
5. (Optional) In the **Filters** section, select any flex filter assets that are appropriate for this flex definition.
6. Click **Save**.
7. Repeat this procedure for each flex definition that you need to create.

16.3.10 Step 8: Create Flex Parent Assets

To create flex parent assets

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Contributor** interface.
4. In the menu bar, select **Content**, then **New**, and then *type of flex parent asset you wish to create*.

   A tab opens displaying the Create view of the flex parent asset you wish to create.



Product Parent

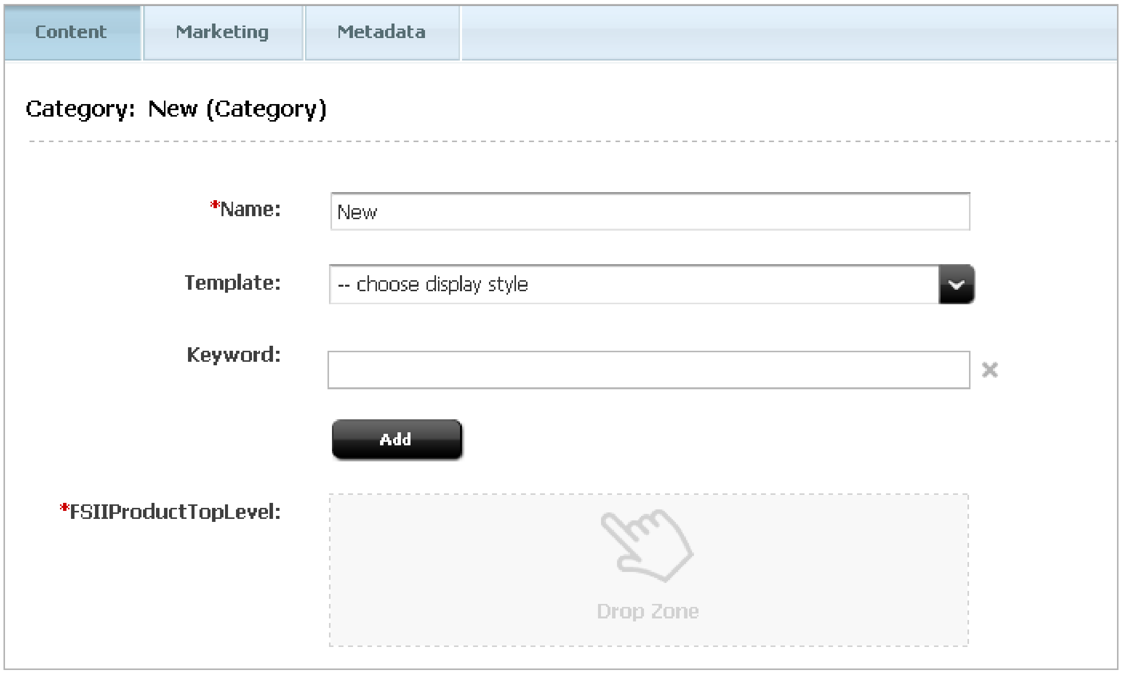
Name:

*Product Parent Definition: -- choose a definition --

5. In the **Name** field, enter a name of up to 64 characters.

6. In the **Parent Definition** field, select the desired parent definition from the drop-down list. The definition you select formats the next sections of the form (the sections you fill out to define this parent asset).

   The **Content** section of the form is displayed:



Category: New (Category)

*Name: New

Template: -- choose display style

Keyword:

*FSIIProductTopLevel:

7. Fill in the fields in the **Content** section of the form.

   Note the following about the kinds of fields that might appear in this section:

   – If there is an asterisk (*) next to the field, then the field is required.

   – In this example, the field in which you designate a parent asset is called `FSIIProductTopLevel`. The parent that you drag and drop into this field becomes the grandparent of any flex assets you designate as children of the parent you are creating in this procedure. If this field is not required, and you do not select any parents (grandparents), the parent you are creating will be a top-level parent in the **Content Tree**.

   – To determine whether the field is single or multi-valued, point to the drop zone associated with the field. A tooltip is displayed showing the type of assets this field accepts along with information about whether the field is single or multi-valued.

8. Use the form section selector to switch to the next sections of the form (**Marketing** and **Metadata** sections). If a field has an asterisk (*) next to it, it is a required field.

   The fields displayed in these sections are based on the parent definition you chose for this parent. The values that you enter into these fields are inherited by any flex assets that have this parent asset as their parents.

9. In the asset's toolbar, click the **Save** icon.

   WebCenter Sites writes the new parent to the database. All the information other than the attribute values are written to the FlexParent, FlexParent`_AMap`, and FlexParent`_Extension` tables, where FlexParent represents the internal name of your flex parents. The attribute values are written to the FlexParent`_Mungo` table.

16.3.11 Step 9: Create and Assign Asset Type Icons (Contributor Interface Only)

When a user performs a search in the Contributor interface and displays the results of the search in the **Thumbnail** view, each asset in the search results list is represented by a thumbnail image. By default, the name, asset type, modification date, and locale of the assets in the search results list are displayed below the thumbnail image representing the asset or the asset's type. Images can either be assigned per asset or per asset type. The focus of this section is to assign a thumbnail image for each asset type.

You must create and assign images that uniquely identify the members of your new flex family by completing the steps below.

1. For each flex family member, do the following:

   a. Create two image files of type JPG, one to be displayed in the docked search results list, and the other to be displayed in the undocked search results list. The standard size of the image that will be displayed in the docked search results list is 96x96 pixels. The standard size of the image that will be displayed in the undocked search results list is 170x170 pixels.

   b. Name the image files as follows:

      – To name the image that will be displayed for the asset type in the docked search results list, use the syntax *`assetType`*`.jpg`.

      – To name the image that will be displayed for the asset type in the undocked search results list, use the syntax *`assetType_large`*`.jpg`.

        You must save the images as JPGs. The file name determines the asset type for which the icon will be displayed. The name is case-sensitive.

   c. Place the image files in the appropriate directory:

      ```
      <cs_app_dir>/images/search
      ```

      where `<cs_app_dir>` is the directory of the deployed WebCenter Sites application on your application server.

2. Restart your application server for the icons to appear in the **Thumbnail** view of the search results list in the Contributor interface.

16.3.12 Step 10: Code Templates for the Flex Assets

Creating your flex asset definitions and coding the templates for the flex assets that use those definitions is an iterative process. Although you need to create definitions and flex assets before you can create templates for your flex assets, it is likely that you will discover areas that need refinement in your data design only after you have coded a template and tested the code.

For information about coding elements for your templates, see Chapter 23, "Creating Template, CSElement, and SiteEntry Assets" and Chapter 28, "Coding Elements for Templates and CSElements."

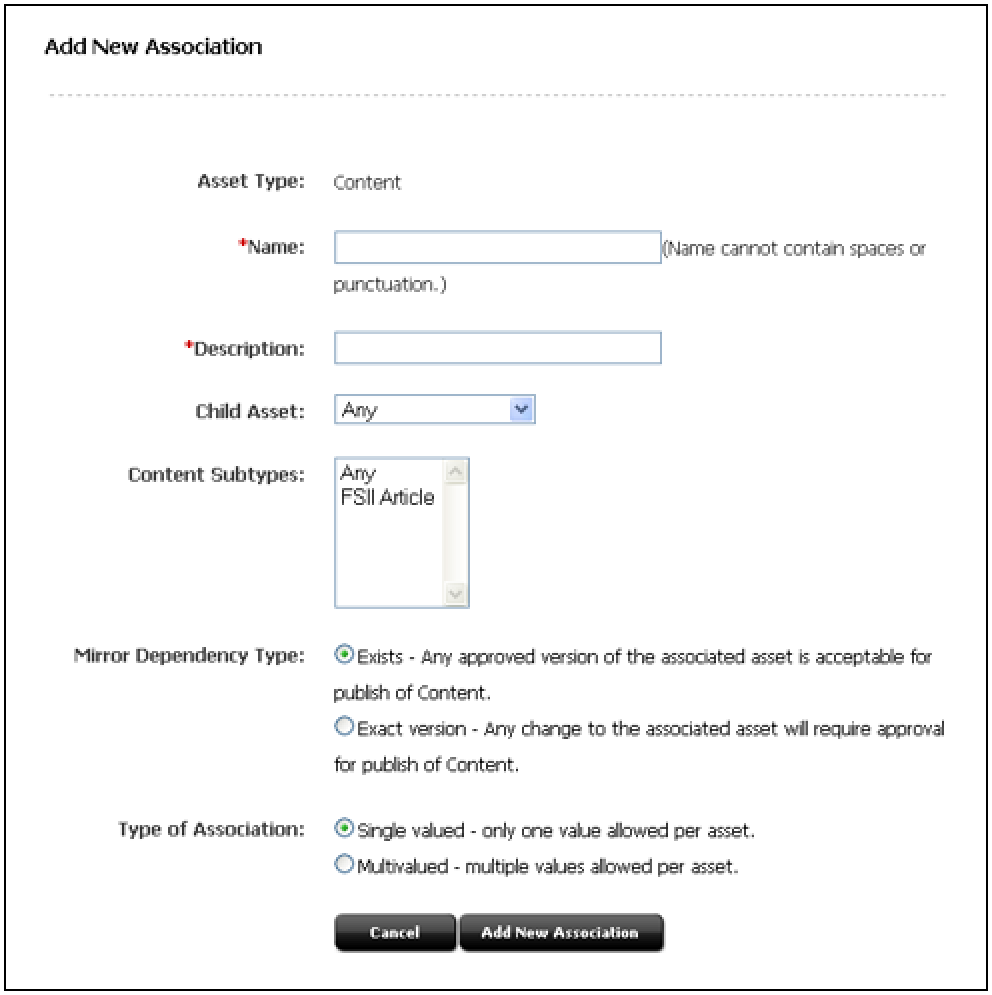
16.3.13 Step 11: Test Your Design (Create Test Flex Assets)

To thoroughly test your design, you must create some flex assets so that you can examine where they appear on the tree, what their forms look like, how long it takes to load their forms, and so on.

16.3.14 Step 12 (Optional): Create Flex Asset Associations

In most cases, you should use a flex asset's attributes to form associations. In the rare case that your associations must work across flex definitions, create associations between flex assets by completing the following steps:

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. On the **Admin** tab of the tree, click the **Asset Types** node.
5. Click on the plus sign next to asset type you wish to create an association for.
6. Click on the plus sign for the **Asset Associations** node.
7. Click **Add New**. The following screen appears:



Add New Association

Asset Type: Content

*Name: (Name cannot contain spaces or punctuation.)

*Description:

Child Asset: Any

Content Subtypes: Any
FSII Article

Mirror Dependency Type: ● Exists - Any approved version of the associated asset is acceptable for publish of Content.
○ Exact version - Any change to the associated asset will require approval for publish of Content.

Type of Association: ● Single valued - only one value allowed per asset.
○ Multivalued - multiple values allowed per asset.

8. Click in the **Name** field and enter a name.
9. Click in the **Description** field and enter a description of the association.
10. Select a child asset to associate with this asset using the **Child Asset** drop-down select box.
11. Select one or more sub-types using the **Subtypes** field.
12. Choose a dependency type for the associated flex asset using the **Mirror Dependency Type** options.
13. Click **Add New Association** to associate the flex asset types.

16.3.15 Step 13: Move the Asset Types to Other Systems

When you have finished creating your flex family, which includes creating the new flex asset types with Flex Family Maker, creating the data structure assets (including attribute editors), and coding templates for the flex asset type, you move them to the management and delivery systems.

Then, the system administrators configure the asset types for the management system. They enable revision tracking where appropriate, create workflow processes, create Start Menu shortcuts, and so on.

For information about moving your asset types to the management and delivery systems, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

16.4 Editing Flex Attributes, Parents, and Definitions

Editing most of the flex asset types requires careful planning because certain edits cause schema changes and **schema changes cause data loss**.

This section presents tips and advice about editing flex family asset types and contains the following topics:

- Section 16.4.1, "Editing Attributes"
- Section 16.4.2, "Editing Parent Definitions and Flex Definitions"
- Section 16.4.3, "Editing Parents and Flex Assets"

16.4.1 Editing Attributes

Note the following when editing a flex attribute:

- You can change the **Name** without causing a schema change. However, if you are using XMLPost to import flex assets into your WebCenter Sites database, you must edit your XMLPost files if you change the name of an attribute.
- You can change the **Description** without causing data loss.
- If you change the data type in the **Value Type** field, you **lose all data** associated with the attribute in the _Mungo table(s) that use this attribute type.
- If the attribute's data type is **asset** and you change the asset type, **all existing data for the attribute is invalid.**
- If you change the **Folder** field for a blob attribute, WebCenter Sites will no longer be able to find any existing data for that attribute. If you absolutely must change this value, you need to move the file system to match the new value that you set.
- You can change the **Number of Values** from single to multiple without causing data loss or complications.
- If you change the **Number of Values** from multiple to single, WebCenter Sites cannot determine which of the values in any existing rows are the values to keep.
- You can change the **Search Engine** and **ISO Character Set** without causing data loss.

16.4.2 Editing Parent Definitions and Flex Definitions

Note the following when editing a parent definition or a flex definition:

- You can change the **Name** without causing a schema change. However, if you are using XMLPost to import flex assets into your WebCenter Sites database, you must edit your XMLPost files if you change the name of a parent definition.
- You can change the **Description** and the **Parent Select Style** fields without causing data loss.
- If you change the parent selections:
  - Adding parents is allowed
  - Removing parents can cause assets to no longer have valid data.
  - Changing parents from optional to required can cause problems because parents or flex assets who do not have one of the newly required parents are no longer valid.
  - Changing parents from required to optional is allowed.

  - Changing parents from single value to multiple value is allowed.
  - Changing parents from multiple value to single value causes unpredictable results because WebCenter Sites can not determine which of the previously acceptable multiple values is the one to keep and which ones to remove.
- If you change the attribute selections:
  - Adding optional attributes is allowed.
  - Adding required attributes causes existing parents or flex assets without them to be invalid.
  - Removing attributes causes existing parents or flex assets with such an attribute value to be invalid.

16.4.3 Editing Parents and Flex Assets

Note the following when editing a flex or parent asset:

- You can change the **Name** without causing a schema change. However, if you are using XMLPost to import flex assets into your WebCenter Sites database, you must edit your XMLPost files if you change the name of a parent definition.
- You can change the **Description** without causing data loss.
- If you change parents, WebCenter Sites corrects all the inherited attribute values.
- You cannot change the definition that you used to create the parent or flex asset.
- Changing the value of an attribute is allowed. If you change the value of an attribute for a parent, WebCenter Sites corrects that attribute for all the assets that inherited it from this parent. Changing the attribute value for a flex asset is allowed.

16.5 Using Product Sets

When you are using WebCenter Sites to manage an online catalog, there is a special feature that you can use with product assets called a **product set**. Product sets allow you to group products that are actually the same product except that they are packaged and sold differently.

This section contains the following topics:

- Section 16.5.1, "What Is a Product Set?"
- Section 16.5.2, "Creating Product Sets"

16.5.1 What Is a Product Set?

For example, a book is the same book whether it is the paperback version or the hard-cover version. And a soft drink is the same soft drink whether it is sold in individual cans, as a six-pack, in a 2-liter bottle, or a case.

Product sets allow you to group products like these together so that they can be displayed together (in the same form) on the management system, yet remain individual saleable units, identified as such by their SKUs.

The model for the product set feature is as follows:

- The product set is a product parent that takes on the characteristics of a product asset. The product set (parent) has all of the attributes that define the core product.

- The product assets are SKUs. That is, they have only those attributes that describe the packaging or are the unique identifiers for members of the set: the SKU, the bottle size, and so on.
- The product set (parent) has an attribute that marks it as a product set and the value of this attribute is unique among all the product sets. This attribute is called GAProductSet and is a reserved name. The products in the set inherit this attribute and, by this inheritance, are marked as members of that product set (that is, children of that product parent).

16.5.2 Creating Product Sets

To create a product set

1. Create a product attribute named `GAProductSet`. This is a reserved name and your attribute name must match it exactly.
2. Create a new product parent definition and select the `GAProductSet` attribute.
3. Create a new product definition and designate that the parents created with the definition that you created in step 2 can be parents of products created with this product definition.
4. Create a new product parent from the definition you created in step 2.
5. Using the product definition that you created in step 3, create the products in the set and designate that the parent that you created in step 4 is their product parent.

Now, when you inspect or edit the product set (product parent), each product (SKU) in the set is listed on the **Product Parent** form, presenting a representation of the product set relationship.

There can be only one **GAProductSet** attribute in the WebCenter Sites database. If you have more than one WebCenter Sites site and you want to create product sets in more than one site, you must share the **GAProductSet** attribute to the sites that you want to use it in.

17

Flex Filters

This chapter provides information about flex filters. It also contains instructions on defining a flex filter class and creating a flex filter asset.

This chapter contains the following sections:

- Section 17.1, "Understanding Flex Filter Classes and Assets"
- Section 17.2, "Defining a Flex Filter Class and Creating a Flex Filter Asset"
- Section 17.3, "Document Transformation"

17.1 Understanding Flex Filter Classes and Assets

A flex filter performs post-processing operations on flex assets to which that filter is assigned. The filter is called when its associated asset is saved. The filter is not called if the asset's edit operation is canceled.

Multiple flex filters can be assigned to a flex asset. When the asset is saved, its filters perform their designated tasks in the order in which they are defined in the "Filters" section of the asset's definition. The task a flex filter performs is determined by the flex filter class that defines the filter. For example, flex filters that are defined by the Doc-Type filter class are enabled by that class to determine an asset's MIME-type and to extract a file type from a URL path.

Each flex filter is associated with the attributes that it updates with the data it processes for an asset. These attributes are called *derived attributes. Derived attributes must be created* in the same flex family to which the filter is assigned. However, derived attributes must not be specified in an asset definition because they are strictly a flex filter's output; they are added to an asset only when the asset is saved. When multiple filters are assigned to an asset, none of the filters must be dependent on attributes that are output by any other filters associated with that asset.

When a filter is defined (in the `Filters` table in the WebCenter Sites database) it can be applied to any asset type's child definition. (Filters cannot be applied to an asset type's parent definition.) If a filter is created for a specific purpose that is related to a specific asset type, the filter should be associated with only that asset type's child definition. For example, if a filter is designed to process an image type for "Media" assets, it cannot function when applied to text-based assets.

This section contains the following topics:

- Section 17.1.1, "Flex Filter Classes"
- Section 17.1.2, "Flex Filter Assets"

17.1.1 Flex Filter Classes

Flex filter classes implement the functionality of flex filter assets. These classes are listed in the `Filters` table in the WebCenter Sites database. When you create a flex filter asset, you select a flex filter class for it.

WebCenter Sites delivers the following flex filter classes:

- **Doc-Type**: Extracts components from an asset containing one or more MIME file types and maps each file type to an individually named attribute.
- **Thumbnail Creator**: Converts an image to a thumbnail.
- **Field Copier**: Copies the contents of a system-defined attribute into a user-defined attribute.
- **Document Transformation**: Converts a document from one file type into another by invoking a registered transformation engine (an engine that is specified in the `SystemTransforms` table). The transformation engine functions as a wrapper that forwards calls to a document transformer, which then performs document conversion.

You can create a custom flex filter class by defining it in the WebCenter Sites `Filters` database table. For instructions on creating a custom flex filter class, see Section 17.2.2, "Defining a Custom Flex Filter Class."

Doc-Type Filter Class

A Doc-Type filter takes a document and extracts the MIME-type data associated with the file into its individual components. For example, an uploaded file containing text and a GIF photo would be filtered into two generated attributes, one containing TXT formatted data and one containing GIF formatted data.

The Doc-Type class is defined by the following arguments:

- **Attribute to Hold Derived File Name**: (Optional) Enter the flex attribute that stores the output file name.
- **Attribute to Hold Derived File Type**: The file extension is stored in this attribute.
- **Attribute to Hold Derived MIME Type**: (Optional) MIME files can have several types such as plain text, attachment, media file, and so on.
- **Input Attribute Name**: The attribute name stored by WebCenter Sites corresponds to the uploaded file name.

Thumbnail Creator Filter Class

With the thumbnail creator filter, a content provider can upload an original graphic and WebCenter Sites will create a new thumbnail sized GIF graphic file.

The Thumbnail Creator class is defined by the following arguments:

- **Input Attribute Name**: The name of the uploaded file.
- Display values for the large version of the thumbnail graphic "Output Attribute for Main Height," "Output Attribute for Main Width," and "Enter Maximum Pixel Size."
- Attributes that define the thumbnail to be created "Output Attribute Name," "Output Attribute for Thumb Height," "Output Attribute for Image Aspect," and "Output Attribute for Thumb Width."

Field Copier Filter Class

The field copier filter copies the contents of a system-defined attribute into a user defined flex attribute.

The Field Copier class is defined by the following arguments:

- **Name**: The name of the system-defined attribute you wish to copy.
- **Value**: The name of the flex attribute into which you are copying the system-defined attribute's value.

Figure 19–2 illustrates an advanced example of how to implement a field copier filter, using the "Media" flex family of the FirstSiteII sample site. The purpose of the field copier filter in this example is to categorize image assets by the names of their parent assets.

Document Transformation Filter Class

The Document Transformation filter class is defined by the following arguments:

- **Document Transformer Name**: The name of a registered transformation engine exactly as it is listed in the `SystemTransforms` table. By default, four values are listed in the `SystemTransforms` table:
  - `CS: Convert to HTML`

    Initiates the conversion of binary files to HTML.
  - `CS: Convert to HTML fragment`

    Initiates the conversion of binary files into files with HTML coding but without the header tags.
  - `CS: Convert to XML`

    Initiates the conversion of binary files to XML.
  - `CS: Convert to Raw Text`

    Initiates the conversion of a binary file to TXT format.

  Note: The following document transformer is available with WebCenter Sites:

  `com.fatwire.transformer.tika.DocumentTransformerImpl`

  The document transformer is coded to convert documents to raw text files when it is invoked by the **CS: Convert to Raw Text** engine. Converting to any other file type requires writing a document transformer for that file type, registering the corresponding transformation engine (unless it is already registered), and registering the document transformer with WebCenter Sites. For information about implementing default and document transformation solutions, see Section 17.3, "Document Transformation."

- **Input Attribute Name**: The name of the flex attribute whose contents are to be converted by the flex filter. For the Document Transformation filter, the input attribute must be of type `blob` because it expects to find a file in that attribute.
- **Fail on Transform Error**: Choose whether the system will display an error message if the transformation does not complete properly.

- **Output Attribute Name**: The name of the flex attribute that stores the results of the document transformation. For the Document Transformation filter, the output attribute must be of type `blob` because it stores the results of the transformation as a file.

  The data stored in the output attribute (field) is read-only because it is derived from the data in the input attribute. This data is regenerated from the source data in the input attribute each time the asset is saved.

- **Output Document Extension**: The file extension to be assigned to the resulting file. Enter a document extension appropriate to the selected **Document Transformer Name**. For example, when you specify that the document transformation engine is **CS: Convert to HTML**, the document extension must be either **htm** or **html**.

17.1.2 Flex Filter Assets

Flex filter assets are defined by all of the following criteria:

- A flex filter class registered in the `Filters` table
- The information that is passed to that filter class (through arguments). These arguments specify which data to use, any constraints on the filter's action, and where to store the results of the filtering process when the asset is saved.

When you create a flex filter asset, you select the flex filter class to be used, then enter values for the arguments that the filter class needs in order to perform its action.

After you create a flex filter asset, you assign it to the appropriate flex definition assets. Then, whenever content providers save a flex asset of that definition, the filter automatically performs its assigned action. For detailed instructions, see Section 17.2.3, "Creating Flex Filter Assets."

17.2 Defining a Flex Filter Class and Creating a Flex Filter Asset

To use a flex filter you must first define a custom flex filter class in the WebCenter Sites database. You then activate the custom flex filter class by creating a flex filter asset whose functionality is defined by that class.

These are your basic steps:

1. Create the Java class that provides the implementation code for the custom flex filter class. For instructions, see Section 17.2.1, "Implementing a Flex Filter Class."
2. Define the new flex filter class in the `Filters` table in the WebCenter Sites database. For instructions, see Section 17.2.2, "Defining a Custom Flex Filter Class."
3. Create a flex filter asset that is defined by the custom flex filter class:
   a. Create the attributes that will be referenced in the filter's arguments.
   b. Create the flex filter asset and assign it to the desired flex filter class.
   c. Add the new filter asset to a child definition.
   d. Re-save all related assets associated with the definition to which you added the filter asset.

   For instructions, see Section 17.2.3, "Creating Flex Filter Assets."

This section contains the following topics:

- Section 17.2.1, "Implementing a Flex Filter Class"

- Section 17.2.2, "Defining a Custom Flex Filter Class"
- Section 17.2.3, "Creating Flex Filter Assets"

17.2.1 Implementing a Flex Filter Class

To implement a custom flex filter, create a new Java class for it by extending the **AbstractFlexFilter** class. This filter class contains all default functionality required to handle filter requests. A working example of a custom flex filter class that extends the AbstractFlexFilter class, and whose purpose is to access a flex attribute and set a derived attribute value, is provided by the source file `SampleFlexFilter.java`.

A flex filter's functionality is implemented by parameters called abstraction interfaces. Since flex filters only need to know about certain aspects of the assets they are filtering, abstraction interfaces provide filters with access to only the asset information necessary for them to perform their function on a given flex asset. Table 17–1 lists the abstraction interfaces required to implement a flex filter, and the asset information each abstraction interface passes to the filter. For detailed information about the various abstraction interfaces used to implement a custom flex filter class, see the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.

Table 17–1 Abstraction Interfaces (`com.openmarket.gator.interfaces` Package)

Abstraction Interface	Description
`IFilterEnvironment`	Provides methods to obtain information about the environment that is supporting the filter.
`IFilterableAssetInstance`	Provides methods to manipulate the asset that is being filtered.
`IFilterDescription`	Provides methods to describe all the potential derived attributes that will be modified by the filter during its execution.
`IFilterDependencies`	Provides methods to log the dependencies against the asset instance to be filtered. A dependency refers to another asset by type and identifier that shares a relationship with the asset to be filtered. When a dependency is declared, it is either exact or exists.

Note: Standard asset attributes can be obtained by the `IFilterableAssetInstance.get` method. For example, to get the standard asset description, you would add the line:

```
String description = instance.get(description);
```

When building the Java code for the new flex filter class, define a constructor with a single `FTValList` parameter (located in the `COM.FutureTense.Interfaces` package). This parameter provides a list of arguments obtained from the filter's definition in the `Filters` database table. These arguments are passed to the filter in the form of key/value pairs. If there are no predefined arguments for the filter class in the `Filters` database table, the `FTValList` is null.

17.2.1.1 Extending the 'AbstractFlexFilter' Class

The **AbstractFlexFilter** class can be extended to build your implementation of a flex filter class. This class is located in the `com.openmarket.gator.flexfilters` package. When you create the new flex filter's Java class, you can call the required methods necessary for your filter's functionality from the AbstractFlexFilter class' java code. This simplifies the amount of code necessary to create a custom implementation.

When a method is called, it is provided with the argument `String filterIdentifier`, which is the asset identifier for the filter. When you associate the same filter with different flex asset families, the filter identifier reflects the filter definition for the associated family. This argument is useful when you are implementing a filter that will be used by different asset families in order for the filter to know which flex family association is being used at the moment the filter is invoked.

For detailed information about the various abstract methods used to implement a flex filter class, see the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.

Required Abstract Methods

The following abstract methods are required by all flex filter implementations:

```
public void filterAsset(IFilterEnvironment env,
      String filterIdentifier,
      FTValList filterArguments,
      IFilterableAssetInstance instance)
      throws AssetException;
```

These lines are the main method to process asset post processing when a new or pre-existing asset is saved. This method is not called if the edit is canceled. It does the work that represents the filter's purpose. A list of arguments (`FTValList`) is provided so the filter can obtain input and/or output attribute definitions, and any other information valid to the filter. The `filterArguments` list is defined when the filter is created in the WebCenter Sites interface.

```
public FTValList getLegalArguments(IFilterEnvironment env,
      String filterIdentifier)
      throws AssetException;
```

These lines are the method that is called to return a list of legal filter arguments. The WebCenter Sites interface will call this during filter creation or editing to populate the drop-down list after selecting the filter and pressing the **Get Arguments** button.

Optional Abstract Methods

The following abstract methods can be used to override those within the AbstractFlexFilter class. These methods are optional because the default implementations provided by the AbstractFlexFilter class are usually sufficient for most filters:

```
public void describeDerivedAttributes(IFilterEnvironment env,
      String filterIdentifier, FTValList filterArguments,
      String defTypeName, String parentDefTypeName,
      IFilterDescription descriptionObject) throws AssetException
```

This method describes all the potential derived attributes, group affinities, and recommendations that the filter might set. When the filter plans to output to attributes this method must identify the attributes that will be modified. This is called whenever the flex asset is viewed, to anticipate the editing of the asset.

```
public void getDependencies(IFilterEnvironment env,String filterIdentifier,
   FTValList filterArguments, String assetTypeName, String parentTypeName,
   IFilterDependencies filterdeps) throws AssetException
```

This method is called to describe the filter's asset dependencies. Filter dependencies are set to either **exists** or **exact**.

```
public String[] getArgumentLegalValues(IFilterEnvironment env,
   String filterIdentifier, String argumentName) throws AssetException
```

This method is called to return a list of acceptable values for a specified argument. If the return list is null then any value is accepted. This method is called by the WebCenter Sites Admin interface when a filter asset is being created or edited to validate the argument values that are specified for the filter asset.

17.2.2 Defining a Custom Flex Filter Class

This section provides instructions for defining a custom flex filter class in the `Filters` table in the WebCenter Sites database. In this example, we create a flex filter class named **CustomFilter**.

To define a custom flex filter class

1. Copy the `.jar` or `class` file containing the implementation code for the custom flex filter class into the directory that holds the WebCenter Sites product `jars`:

   > **Note:** For information about creating a Java class that provides the implementation code for your custom flex filter class, see Section 17.2.1, "Implementing a Flex Filter Class."

   – For WebLogic:
   `app-server-install-dir/bea/path-to-domain/domain-name/applications/WEB-INF/lib`

   – For WebSphere:
   `WebSphere-Installation-Directory/InstalledApps/WEB-INF/lib`

2. Open Sites Explorer and add a row to the `Filters` table for the new filter class:

   a. In the tree, expand the **Tables** node, and then select the `Filters` table.

   b. Select **File**, then **New**, and then **Record**.

   c. Define the filter in the database by filling in the following columns:

      – **name**: Enter the name of the filter as it will be displayed in the WebCenter Sites interfaces.

      – **description**: (Optional) Enter a short summary about the purpose of the filter class.

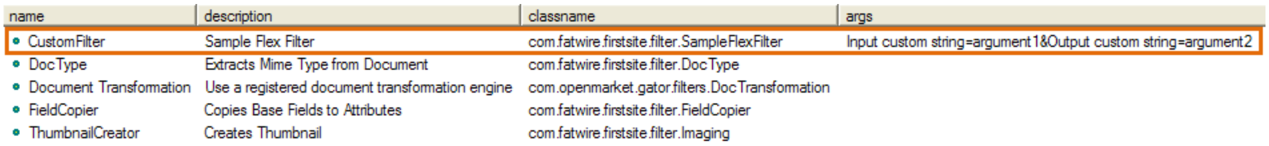
      – **classname**: Enter the exact classname of the filter class' implementation (for example, `com.fatwire.firstsite.filter.SampleFlexFilter`). This name must be available for loading in the WebCenter Sites classpath.

      – **args**: (Optional) Enter the input and output arguments for the filter. Argument key/value pairs are delimited by an ampersand (&) character. (for example, `arg1=argument1&arg2=argument2`). These arguments are passed to the filter constructor and their use is left up to the filter's developer.

   > **Note:** If you do not define the flex filter class' input and output arguments in the `Filters` table, you can define the arguments in the flex filter class' code. For information, see Section 17.2.1, "Implementing a Flex Filter Class."

d. Select **File** and then **Save**.

Your new filter entry will look similar to the following:



name	description	classname	args
● CustomFilter	Sample Flex Filter	com.fatwire.firstsite.filter.SampleFlexFilter	Input custom string=argument1&Output custom string=argument2
● DocType	Extracts Mime Type from Document	com.fatwire.firstsite.filter.DocType	
● Document Transformation	Use a registered document transformation engine	com.openmarket.gator.filters.DocTransformation	
● FieldCopier	Copies Base Fields to Attributes	com.fatwire.firstsite.filter.FieldCopier	
● ThumbnailCreator	Creates Thumbnail	com.fatwire.firstsite.filter.Imaging	

The filter class is now displayed as an option in the **Filter** drop-down list in the New and Edit forms of filter assets.

17.2.3 Creating Flex Filter Assets

This section provides instructions for creating a flex filter asset. In this example, we create a flex filter asset named **FSII_CustomFlexFilter** for the "Media" flex family of the FirstSiteII sample site.

Before you can create a filter asset, the flex attributes that you want to use as the input and output attributes must exist. To ensure that the values of preexisting attributes are not overwritten, you can create new attributes to be referenced by the arguments for your custom flex filter. For this example, we created two "Media" attributes of type `string`; **FSII_CustomInput** to be used as the input attribute and **FSII_CustomOutput** to be used as the output attribute.

To create a flex filter asset

1. Log in to the Admin interface as a general administrator, and select the site for which you wish to create a flex filter asset (FirstSiteII sample site in this example).

2. Create the attributes (if not already defined) that will be referenced by the filter's input and output arguments. Note the following requirements:

   – For flex filters that use the Document Transformation filter class, the input and output attributes must be of type blob.

   – For any flex filter, the input attribute, output attribute, and flex filter must all belong to the same flex family.

   For instructions on creating flex attributes, see Section 16.3.6, "Step 4: Create Flex Attributes."

3. Create a flex filter asset for the associated flex family ("Media" flex family in this example):

   a. From the start menu items, click **New**.

   b. In the list of asset types, select the type of filter you wish to create. In this example, **New Media Filter**.

   The New filter form is displayed.

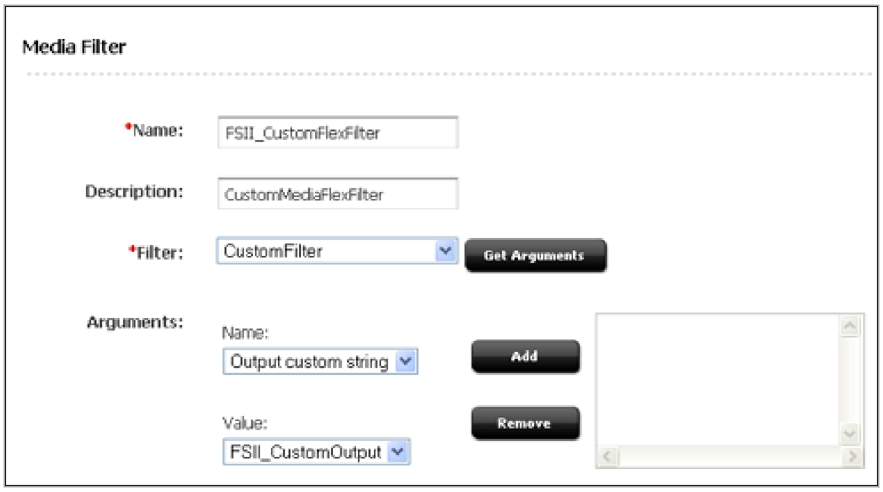
   c. Fill in the following fields:

      a. In the **Name** field, enter a unique name for this filter (in this example, we use **FSII_CustomFlexFilter**).

      b. In the **Description** field, enter a brief description summarizing the filter's function.

c. In the **Filters** drop-down list, select the filter definition that matches the name you assigned to the custom filter (in this example, we select **CustomFilter**). Then click **Get Arguments**.

d. In the Arguments field, specify the input and output arguments for the flex filter asset. Click **Add** to add the argument(s) to the filter.



Media Filter

*Name: FSII_CustomFlexFilter

Description: CustomMediaFlexFilter

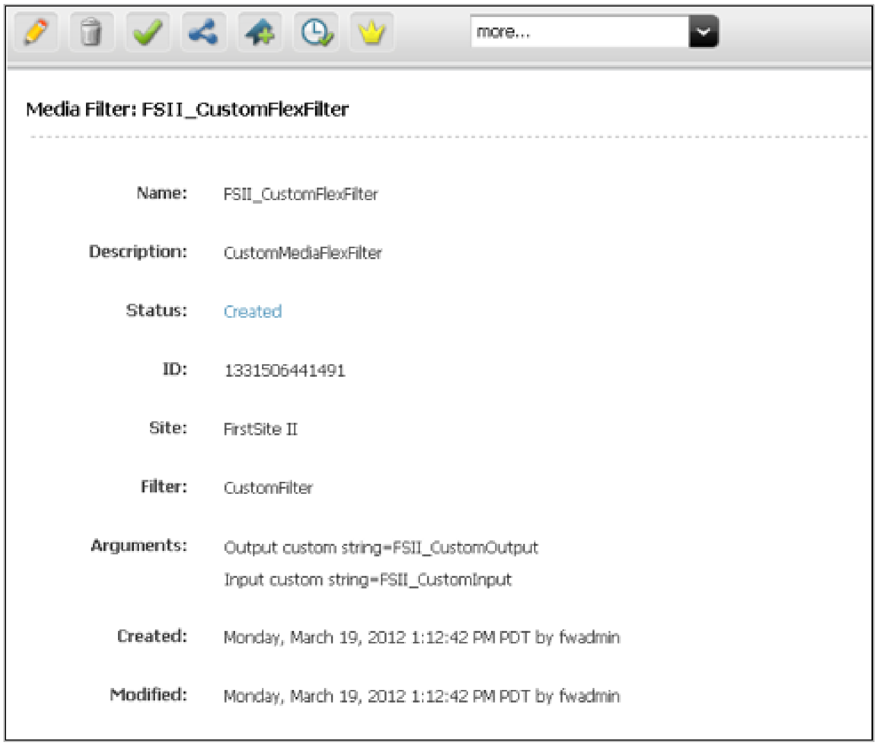
*Filter: CustomFilter

Arguments:

Name: Output custom string

Value: FSII_CustomOutput

d. Click **Save** to save the filter.



Media Filter: FSII_CustomFlexFilter

Name: FSII_CustomFlexFilter

Description: CustomMediaFlexFilter

Status: Created

ID: 1331506441491

Site: FirstSite II

Filter: CustomFilter

Arguments: Output custom string=FSII_CustomOutput
Input custom string=FSII_CustomInput

Created: Monday, March 19, 2012 1:12:42 PM PDT by fwadmin

Modified: Monday, March 19, 2012 1:12:42 PM PDT by fwadmin

4. Find the desired child definition to which you wish to add the new filter asset (in this example, we add the filter to the "Media" child definition **FSII_Image**).

   a. From the start menu items, click **Search**.

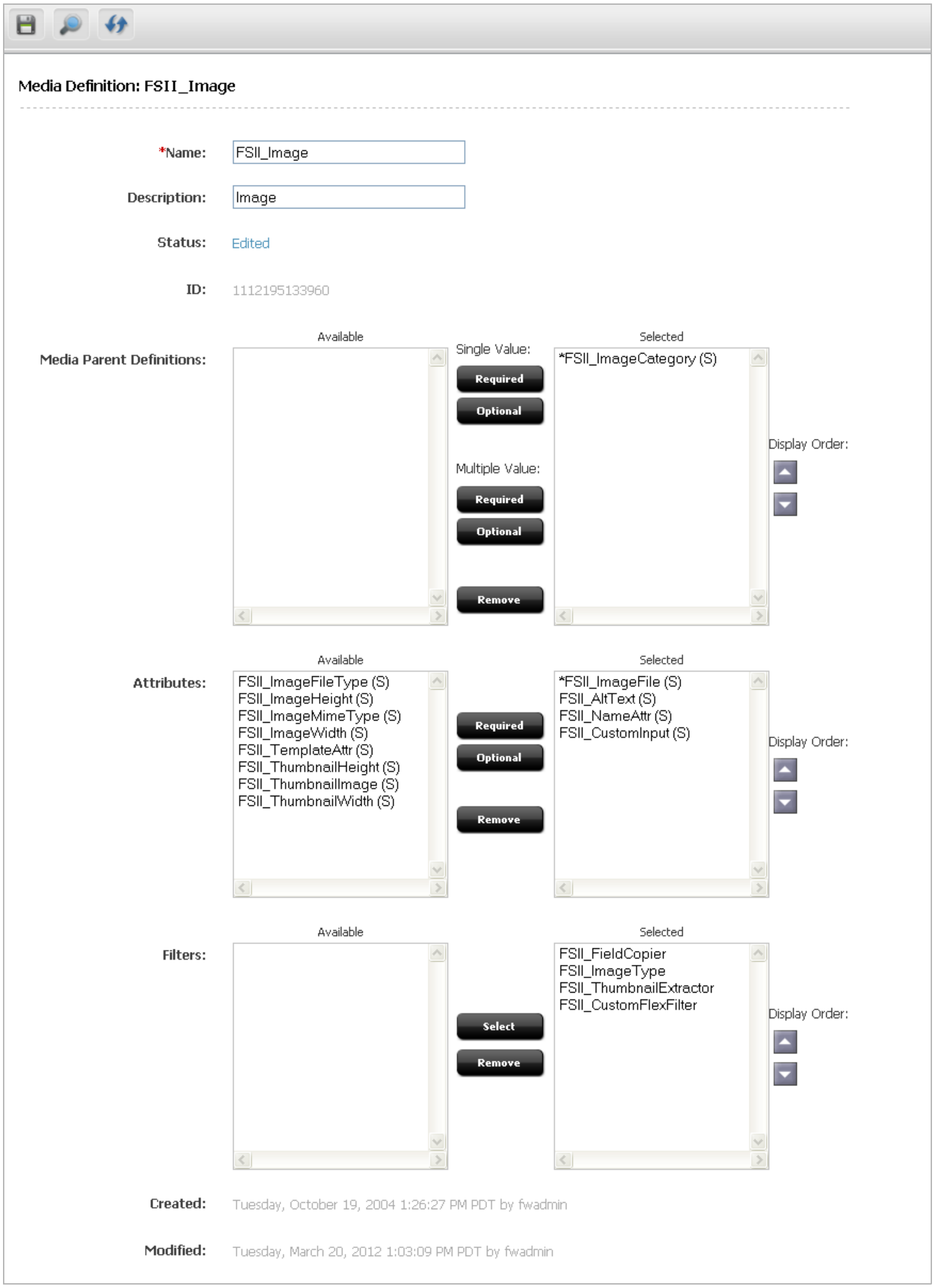
   b. In the list of asset types, select the desired type of asset definition to which you wish to add the filter (**Find Media Definition** in this example).

c. In the search field, enter the name of the definition to which you wish to add the filter (**FSII_Image** in this example).

d. Click **Search**.

e. In the list of search results, navigate to the desired definition and click its **Edit** icon (**FSII_Image** in this example).

The Edit form of the definition opens:



Media Definition: FSII_Image

*Name: FSII_Image

Description: Image

Status: Edited

ID: 1112195133960

Media Parent Definitions:

Available:

Selected:
*FSII_ImageCategory (S)

Attributes:

Available:
FSII_ImageFileType (S)
FSII_ImageHeight (S)
FSII_ImageMimeType (S)
FSII_ImageWidth (S)
FSII_TemplateAttr (S)
FSII_ThumbnailHeight (S)
FSII_ThumbnailImage (S)
FSII_ThumbnailWidth (S)

Selected:
*FSII_ImageFile (S)
FSII_AltText (S)
FSII_NameAttr (S)
FSII_CustomInput (S)

Filters:

Available:

Selected:
FSII_FieldCopier
FSII_ImageType
FSII_ThumbnailExtractor
FSII_CustomFlexFilter

Created: Tuesday, October 19, 2004 1:26:27 PM PDT by fwadmin

Modified: Tuesday, March 20, 2012 1:03:09 PM PDT by fwadmin

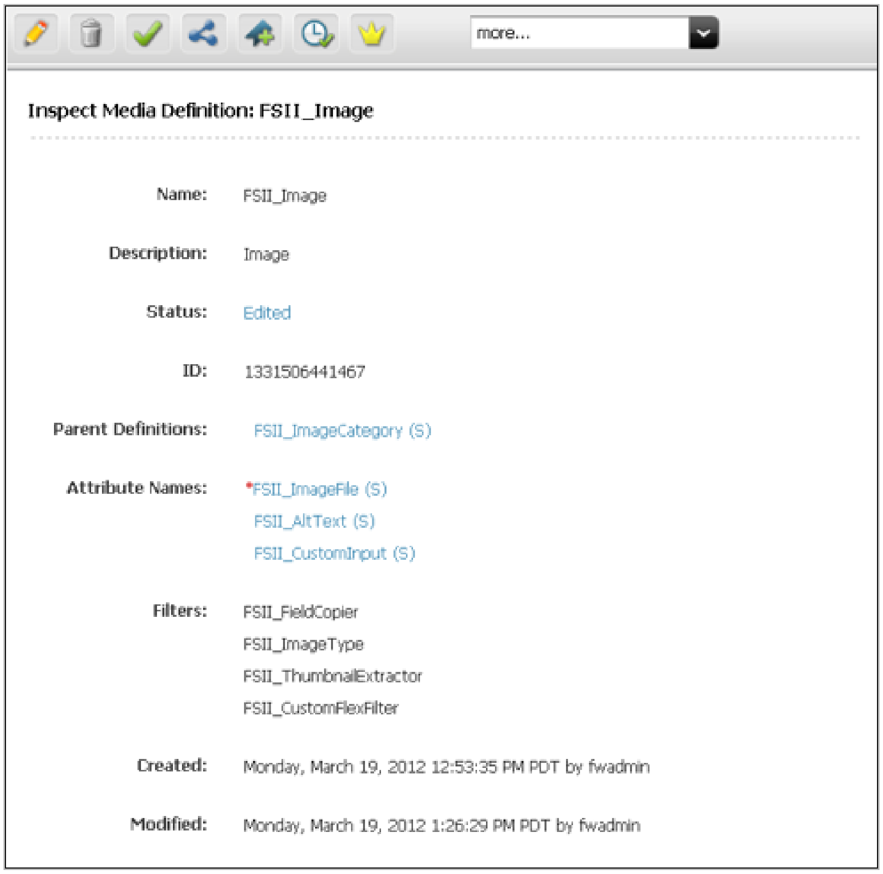
5. In the Edit form, add the filter and input argument to the definition:

   a. In the Attributes field, highlight the input attribute in the **Available** list and move it to the **Selected** list (**FSII_CustomInput** in this example).

   b. In the Filters field, highlight the desired flex filter in the **Available** list and move it to the **Selected** list (**FSII_CustomFlexFilter** in this example).

   > **Note:** Add the new filter after any other filters that will create or modify attributes which the filter you are adding depends on or shares in common.

   c. Click **Save**.



Inspect Media Definition: FSII_Image

Name: FSII_Image

Description: Image

Status: Edited

ID: 1331506441467

Parent Definitions: FSII_ImageCategory (S)

Attribute Names: *FSII_ImageFile (S)
FSII_AltText (S)
FSII_CustomInput (S)

Filters: FSII_FieldCopier
FSII_ImageType
FSII_ThumbnailExtractor
FSII_CustomFlexFilter

Created: Monday, March 19, 2012 12:53:35 PM PDT by fwadmin

Modified: Monday, March 19, 2012 1:26:29 PM PDT by fwadmin

6. Find and re-save all preexisting assets associated with the definition to which you added the filter. This enables the filter to populate the output attribute (**FSII_CustomOutput**) with the derived value from the input attribute (**FSII_CustomInput**). For example:

   a. In the applications bar, click the **Contributor** icon to switch to the WebCenter Sites Contributor interface.

   b. In the **Search** field, click the down-arrow to open the "Search Type" menu. In the "Search Type" menu, select the type of asset associated with the definition you modified in step 5 (**Find Media** in this example). Then click the **magnifying glass** button.

   A "Search" tab opens displaying the results of your search.

c. Re-save the asset. In the asset's toolbar, click the **Save** icon.

d. Inspect the asset. In the asset's toolbar, click the **Inspect** icon.

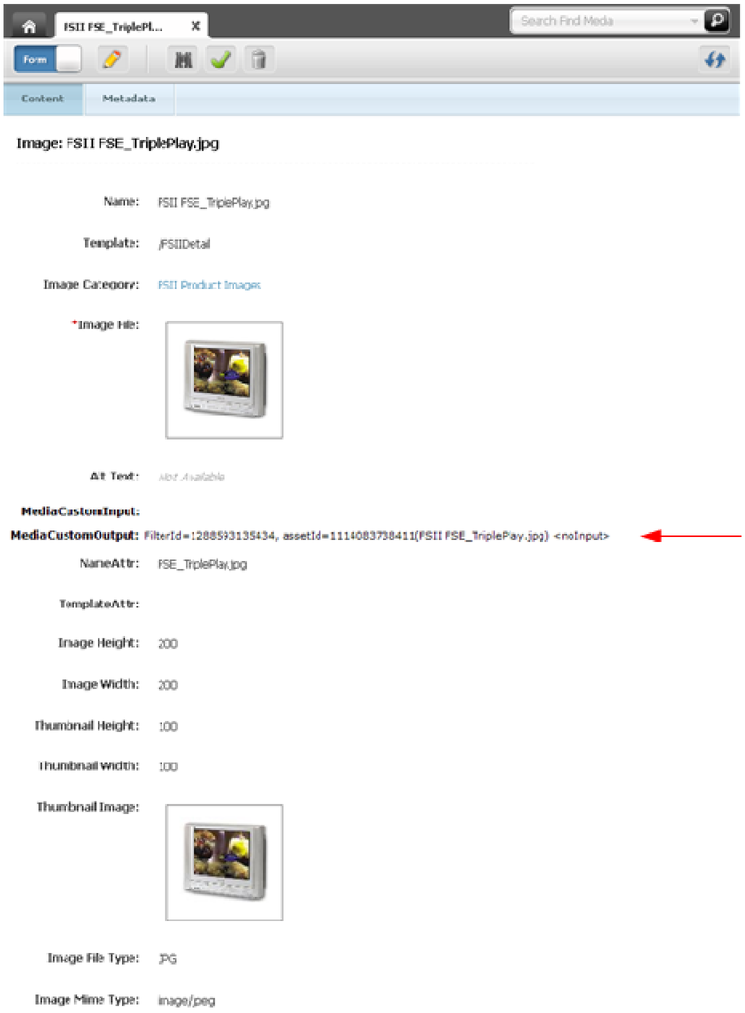


Image: FSII FSE_TriplePlay.jpg

Name: FSII FSE_TriplePlay.jpg

Template: /FSIIDetail

Image Category: FSII Product Images

*Image File:

Alt Text: Not Available

MediaCustomInput:

MediaCustomOutput: FilterId=1288593135434, assetId=1114083738411(FSII FSE_TriplePlay.jpg) <noInput>

NameAttr: FSE_TriplePlay.jpg

TemplateAttr:

Image Height: 200

Image Width: 200

Thumbnail Height: 100

Thumbnail Width: 100

Thumbnail Image:

Image File Type: JPG

Image Mime Type: image/jpeg

Output attribute containing
the derived string value of
the input method.

The above image is the result of editing a flex asset that invokes several filters, including the filter you created in this section. By default, "Media" assets call the "FSII_FieldCopier," "FSII_ImageType," and "FSII_ThumbnailExtractor" filters. In this example, the "Media" asset also calls the **FSII_CustomFlexFilter** filter, which takes the value of the input attribute (**MediaCustomInput**) and inserts a derived string value into the output attribute (**MediaCustomOutput**).

17.3 Document Transformation

Implementing a Document Transformation flex filter requires the following components:

- A transformation engine that is registered in the `SystemTransforms` table and named to indicate the target file type; for example, `CS:Convert to Raw Text`.
- A document transformer, which is a custom class that performs document conversion (for example, converting binary files to raw text files). The document transformer class implements the following interface: `com.fatwire.transformer.common.DocumentTransformer` interface

- The `transformer-formats.xml` file, which is used to associate the transformation engine with the document transformer. The file, located in WebCenter Sites's `WEB-INF/classes` folder, specifies the target file type and the document transformer.

When the Document Transformation flex filter is invoked, the transformation engine functions as a wrapper. The engine forwards calls (via the `transformer-formats.xml` file) to the document transformer, which then performs file conversion.

This section contains the following topics:

- Section 17.3.1, "Default Solution"
- Section 17.3.2, "Custom Solutions"
- Section 17.3.3, "Customizing Document Transformation"
- Section 17.3.4, "Next Steps"

17.3.1 Default Solution

WebCenter Sites provides a default solution that can be used to convert documents to raw text files. The default components are:

- The `CS:Convert to Raw Text` transformation engine, registered in the `SystemTransforms` table.
- A document transformer named `com.fatwire.transformer.tika.DocumentTransformerImpl`, which is coded to output raw text files once it is invoked by the `CS:Convert to Raw Text` engine.
- The `transformer-formats.xml` file, which is configured to associate the `CS:Convert to Raw Text` engine with the document transformer class named above.

Using the default solution requires you to implement a corresponding Document Transformation flex filter, which makes the transformation engine accessible from WebCenter Sites's interface as a document transformation option.

17.3.2 Custom Solutions

If you need a document transformation solution other than the default solution described above, you will have to create and customize the following components:

1. Write and deploy a document transformer for the target file type.
2. Register the transformation engine for the target file type.
3. Configure the `transformer-formats.xml` file to specify the document transformer and the target file type. (The xml file supports multiple document transformers.)
4. Implement the Document Transformation flex filter as described in this chapter.

For detailed steps, see Section 17.3.3, "Customizing Document Transformation."

17.3.3 Customizing Document Transformation

WebCenter Sites provides a default document transformer, `com.fatwire.transformer.tika.DocumentTransformerImpl`, which is coded to output raw text files once it is invoked by the `CS:Convert to Raw Text` engine. If the target file type must be other than raw text, you will have to complete the following steps to create and register the flex filter's supporting components.

Before implementing a Document Transformer flex filter:

- Section 17.3.3.1, "Writing and Deploying a Document Transformer"
- Section 17.3.3.2, "Registering the Transformation Engine"
- Section 17.3.3.3, "Registering the Document Transformer"

17.3.3.1 Writing and Deploying a Document Transformer

1. Write an implementation of the `com.fatwire.transformer.common.DocumentTransformer` interface. You will implement the following methods:

   ```
   public String getOutputDocument(String filename,
      TransformerFormat outputformat);
   ```

   and

   ```
   public String getOutputDocument(String filename,String inputFileExt,
      TransformerFormat outputformat);
   ```

   You can return `null` in the following method, as this method has been deprecated:

   ```
   public InputStream getBytesAsStream(String filename,
      TransformerFormat outputformat);
   ```

2. Copy the document transformer's `jar` or `class` file to the WebCenter Sites web application `lib` folder or `classes` folder.

   – For WebLogic:

     ```
     app-server-install-dir/bea/path-to-domain/domain-name/applications/WEB-INF/
     lib
     ```

   – For WebSphere:

     ```
     WebSphere-Installation-Directory/InstalledApps/WEB-INF/lib
     ```

3. Continue to Section 17.3.3.2, "Registering the Transformation Engine."

17.3.3.2 Registering the Transformation Engine

Note: If your document transformer is written to output files other than HTML, HTML fragment, or XML, skip to step 1 below this note. Otherwise, continue reading this note.

Using a Default Transformation Engine

WebCenter Sites provides the following transformation engines, in addition to CS:Convert to Raw Text:

- CS: Convert to HTML
- CS: Convert to HTML fragment
- CS: Convert to XML

All of the engines are registered by default in the `SystemTransforms` table. If your document transformer is written to output files of type HTML, or HTML fragment, or XML, use the corresponding engine. Do the following:

- Open Sites Explorer.
- Select the `SystemTransforms` table.
- Locate the required engine. For example: `CS:Convert to HTML`

  Note the value of the engine's `target` field. You will enter this value for `<mime-type>` in the `transformer-formats.xml` file, in Section 17.3.3.3, "Registering the Document Transformer." In the `args` field, set the arguments that are appropriate for this transformation engine. For example: `exporttype=HTML`
- Continue to Section 17.3.3.3, "Registering the Document Transformer."

To register a transformation engine

1. Open Sites Explorer and add a row to the `SystemTransforms` table for the new transformation engine, as follows:

   a. Select the `SystemTransforms` table and click in the table's workspace.

   b. Select **File**, then **New**, and then **Record** and fill in the new row as follows:

      a. In the `name` column, enter the name of the transformation engine. For example: ConvertToPDF

      b. In the `description` column, enter a description of the engine. For example: Convert to PDF

      c. In the `target` column, enter `text/<filetype>`. For example: **text/PDF**

      Note: You will use the target value for the `<mime-type>` in the next section, Section 17.3.3.3, "Registering the Document Transformer."

      d. In the `classname` column, enter the engine's default class name: **com.fatwire.transformer.common.FWTransformer**

      e. In the `args` column, set any arguments that are appropriate for this transformation engine. For example: **exporttype=PDF**

c. Save your changes.

2. Continue to the next step, Section 17.3.3.3, "Registering the Document Transformer."

17.3.3.3 Registering the Document Transformer

Register your document transformer in the `transformer-formats.xml` file, located in WebCenter Sites's `WEB-INF/classes` folder. The default file is shown below for reference.

> **Note:** Set the `<mime-type>` to the value of the `target` field for the transformation engine. (the value for `target` was set in Section 17.3.3.2, "Registering the Transformation Engine").
>
> Specify the class name of your document transformer (created in Section 17.3.3.1, "Writing and Deploying a Document Transformer").
>
> If you wish to specify multiple document transformers, repeat the entries for each transformer. The value for `<mime-type>` determines which document transformer will be invoked by the transformation engine.

Example 17–1 Default transformer-formats.xml

```
<transformer-format>
<name>Text Format</name>
<!-- name of the output format supported by this repository -->
<mime-type>text/plain</mime-type>

<file-extension>txt</file-extension>
<transformer-options>
<!-- number of possible transformers available for this transformation -->
  <transformer>
     <name>TIKA</name>
       <properties>
         <property>
           <name>ClassName</name>
<!-- name of the transformer class that gets loaded by transformer factory -->
          <value>com.fatwire.transformer.tika.DocumentTransformerImpl</value>
        </property>
        <property>
           <name>InputFile_Exts</name>
<!-- allowed input file extensions..* means all file types supported by tika -->
           <value>*</value>
        </property>
       </properties>
  </transformer>
</transformer-options>
</transformer-format>
```

17.3.4 Next Steps

Once you have created and registered the document transformation components as shown in Section 17.3.3, "Customizing Document Transformation," you can implement the corresponding Document Transform flex filters. Use the procedures provided in this chapter.

18
Designing Attribute Editors

An attribute editor specifies how data is entered for an attribute when that attribute is displayed on a New or Edit form for a flex asset or a flex parent asset in the WebCenter Sites interface on the management system.

When you assign an attribute editor to an attribute, it replaces the default input mechanism (style) that would otherwise be used for that attribute. The default input style is based on the data type of the attribute.

Because attribute editors format the input mechanism for attributes, you design your attribute editors as you design your flex attributes. Attribute editors are assets, which means you can use the workflow and revision tracking features to manage them as you do for any other type of asset.

This chapter contains the following sections:

18.1 Understanding Attribute Editors

There are three parts to an attribute editor, with an optional fourth and fifth:

- The `presentationobject.dtd` file, located in the WebCenter Sites installation directory. (Required.) This is the DTD file that defines all the possible input styles (presentation objects) for flex attributes and their style tags.
- The attribute editor asset. (Required.) It holds or points to XML code that provides input options for the attribute it is associated with. You use the style tags defined in the DTD to create this XML code.
- An element that formats the attribute, or, displays an edit mechanism, when that attribute appears in a New or Edit form. (Required.) This element must be located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table to be able to find it and its name must exactly match the name of the style tag that invokes it from the attribute editor. (See below for more information.)
- An element that formats the attribute value when it appears in an Inspect form. (Optional.) This element must also be located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table.

The name of the element must use the convention DisplayStyleTag, where StyleTag represents and must exactly match the name of the style tag that invokes it from the attribute editor.

- An element that formats the attribute data before it is saved in the database (Optional.) This element must also be located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table.

  The name of the element must use the convention StyleTagFlexAssetGather, where StyleTag represents and must exactly match the name of the style tag that invokes it from the attribute editor.

WebCenter Sites provides the following items, by default, to support the development of your attribute editors:

- The `presentationobject.dtd` file. It defines several input styles (presentation objects) that you can use in your attribute editors. This means you do not have to define your own unless the nine that are included do not cover your needs.
- Nine text files with sample XML that you can use to create attribute editor assets. You can cut and paste the sample XML into your attribute editor assets. These files are located in the installation directory under `Samples/Attribute_Editors`.
- Ten display elements that work with the sample XML code for attribute editor assets. They are located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table.

Remember that attribute editors are not required. When you do not use attribute editors, WebCenter Sites uses default input styles for the attributes, based on their data types. For a list of the default styles, see Section 11.3.3.2, "Default Input Styles for Attributes." If the default input styles are sufficient for your attributes, you do not need to create attribute editors.

This section contains the following topics:

- Section 18.1.1, "The presentationobject.dtd File"
- Section 18.1.2, "The Attribute Editor Asset"
- Section 18.1.3, "The Attribute Editor Elements"
- Section 18.1.4, "Conventions for the Attribute Editor Elements"

18.1.1 The presentationobject.dtd File

The `presentationobject.dtd` file defines all of the input types (presentation objects) that you can implement through attribute editors. The default `presentationobject.dtd` file defines nine input style tags and the arguments that they can pass from the attribute editor to the display elements (described in Section 18.1.3, "The Attribute Editor Elements").

Following is the entire `presentationobject.dtd` file. It is located in the WebCenter Sites installation directory:

```
<!-- PRESENTATIONOBJECT: An editor
-- PRESENTATIONOBJECT defines the presentation object for
-- instances of Gator attribute types. A presentation object
-- defines the properties of an editor for one of the following
-- controls: Text field, Text area, Pulldown menu
-- For additional information, refer to
-- com.openmarket.gator.interfaces.IPresentationObject.
-- You must specify one of TEXTFIELD, TEXTAREA, or PULLDOWN
-- elements.
```

```
-->
<!ELEMENT PRESENTATIONOBJECT (TEXTFIELD | TEXTAREA | PULLDOWN |
RADIOBUTTONS | CHECKBOXES | PICKASSET | FIELDCOPIER | DATEPICKER |
IMAGEPICKER | CKEDITOR | FCKEDITOR | IMAGEEDITOR | TYPEAHEAD |
UPLOADER)>
<!ATTLIST PRESENTATIONOBJECT NAME CDATA #REQUIRED>
<!-- TEXTFIELD: A text field of a specific width
-- You must specify the x dimension; the maximum number of
-- allowable characters defaults to 255.
-->
<!ELEMENT TEXTFIELD ANY>
<!ATTLIST TEXTFIELD XSIZE CDATA #IMPLIED>
<!ATTLIST TEXTFIELD WIDTH CDATA #IMPLIED>
<!ATTLIST TEXTFIELD MAXCHARS CDATA "255">
<!ATTLIST TEXTFIELD BLANKED (YES | NO) "NO">
<!ATTLIST TEXTFIELD MAXVALUES CDATA #IMPLIED>
<!-- TEXTAREA: A text area of a specific size
-- The wrap style defaults to soft.
-->
<!ELEMENT TEXTAREA ANY>
<!ATTLIST TEXTAREA XSIZE CDATA #IMPLIED>
<!ATTLIST TEXTAREA YSIZE CDATA #IMPLIED>
<!ATTLIST TEXTAREA WIDTH CDATA #IMPLIED>
<!ATTLIST TEXTAREA HEIGHT CDATA #IMPLIED>
<!ATTLIST TEXTAREA WRAPSTYLE (OFF | SOFT | HARD) "SOFT">
<!ATTLIST TEXTAREA DEPTYPE CDATA #IMPLIED>
<!ATTLIST TEXTAREA MAXVALUES CDATA #IMPLIED>
<!ATTLIST TEXTAREA RESIZE CDATA #IMPLIED>
<!-- PULLDOWN: A pulldown menu with an enumeration of items
-- You can specify zero or more list items; the fontsize defaults
-- to relative fontsize 3.
-->
<!ELEMENT PULLDOWN ((ITEM)* | QUERYASSETNAME)>
<!ATTLIST PULLDOWN FONTSIZE CDATA #IMPLIED>
<!-- RADIOBUTTONS: Radio buttons with an enumeration of items
-- You can specify zero or more list items; the fontsize defaults
to relative fontsize 3.
-->
<!ELEMENT RADIOBUTTONS ((ITEM)* | QUERYASSETNAME)>
<!ATTLIST RADIOBUTTONS FONTSIZE CDATA #IMPLIED>
<!ATTLIST RADIOBUTTONS LAYOUT (HORIZONTAL | VERTICAL) "VERTICAL">
<!-- CHECKBOXES: Check boxes with an enumeration of items
-- You can specify zero or more list items; the fontsize defaults
-- to relative fontsize 3.
-->
<!ELEMENT CHECKBOXES ((ITEM)* | QUERYASSETNAME)>
<!ATTLIST CHECKBOXES FONTSIZE CDATA #IMPLIED>
<!ATTLIST CHECKBOXES LAYOUT (HORIZONTAL | VERTICAL) "HORIZONTAL">
<!-- ITEM: A list item
-- You can specify zero or more characters of text.
-->
<!ELEMENT ITEM (#PCDATA)*>
<!-- SQL: Query to populate list of items
-- You can specify zero or more characters of text. Query must
-- return a 'value' column.
-->
<!ELEMENT QUERYASSETNAME (#PCDATA)*>
<!ELEMENT CKEDITOR ANY>
<!ATTLIST CKEDITOR ALLOWEDASSETTYPES CDATA #IMPLIED>
<!ATTLIST CKEDITOR SCRIPT CDATA #IMPLIED>
```

```
<!ATTLIST CKEDITOR IMAGEPICKERID CDATA #IMPLIED>
<!ATTLIST CKEDITOR IMAGEASSETTYPE CDATA #IMPLIED>
<!ATTLIST CKEDITOR TOOLBAR CDATA #IMPLIED>
<!ATTLIST CKEDITOR DEPTYPE CDATA #IMPLIED>
<!ATTLIST CKEDITOR WIDTH CDATA #IMPLIED>
<!ATTLIST CKEDITOR HEIGHT CDATA #IMPLIED>
<!ATTLIST CKEDITOR MAXVALUES CDATA #IMPLIED>
<!ATTLIST CKEDITOR RESIZE CDATA #IMPLIED>
<!ATTLIST CKEDITOR CONFIG CDATA #IMPLIED>
<!ATTLIST CKEDITOR CONFIGOBJ CDATA #IMPLIED>
<!-- Deprecated in Oracle Web Center Sites 11gR1 release. -->
<!ELEMENT FCKEDITOR ANY>
<!ATTLIST FCKEDITOR XSIZE CDATA #IMPLIED>
<!ATTLIST FCKEDITOR YSIZE CDATA #IMPLIED>
<!ATTLIST FCKEDITOR LAZYLOAD CDATA #IMPLIED>
<!ATTLIST FCKEDITOR ALLOWEDASSETTYPES CDATA #IMPLIED>
<!ATTLIST FCKEDITOR SCRIPT CDATA #IMPLIED>
<!ATTLIST FCKEDITOR IMAGEPICKERID CDATA #IMPLIED>
<!ATTLIST FCKEDITOR TOOLBAR CDATA #IMPLIED>
<!ATTLIST FCKEDITOR DEPTYPE CDATA #IMPLIED>
<!ATTLIST FCKEDITOR WIDTH CDATA #IMPLIED>
<!ATTLIST FCKEDITOR HEIGHT CDATA #IMPLIED>
<!ATTLIST FCKEDITOR MAXVALUES CDATA #IMPLIED>
<!-- PICKASSET: When the tree is active, it's the "add from tree"
-- button.
-- When the tree is disabled, it's The Content Centre remember
-- widget. -->
<!ELEMENT PICKASSET ANY>
<!ATTLIST PICKASSET MAXVALUES CDATA #IMPLIED>
<!ATTLIST PICKASSET DISPLAYELEMENT CDATA #IMPLIED>
<!-- TYPEAHEAD: Type And Select the asset. -->
<!ELEMENT TYPEAHEAD ANY>
<!ATTLIST TYPEAHEAD MAXVALUES CDATA #IMPLIED>
<!ATTLIST TYPEAHEAD DISPLAYELEMENT CDATA #IMPLIED>
<!ATTLIST TYPEAHEAD PAGESIZE CDATA #IMPLIED>
<!-- UPLOADER: Upload a file from local disc. -->
<!ELEMENT UPLOADER ANY>
<!ATTLIST UPLOADER MAXVALUES CDATA #IMPLIED>
<!ATTLIST UPLOADER MAXFILESIZE CDATA #IMPLIED>
<!ATTLIST UPLOADER FILETYPES CDATA #IMPLIED>
<!ATTLIST UPLOADER MINWIDTH CDATA #IMPLIED>
<!ATTLIST UPLOADER MAXWIDTH CDATA #IMPLIED>
<!ATTLIST UPLOADER MINHEIGHT CDATA #IMPLIED>
<!ATTLIST UPLOADER MAXHEIGHT CDATA #IMPLIED>
<!-- FIELDCOPIER: A hidden field whose value is set from another
-- field.
-- ex. If you want an attribute whose value is always the name of
-- the asset:
-- <FIELDCOPIER SOURCEFIELD="name"/>
-->
<!ELEMENT FIELDCOPIER ANY>
<!ATTLIST FIELDCOPIER SOURCEFIELD CDATA #REQUIRED>
<!-- This describe the Date Picker -->
<!ELEMENT DATEPICKER ANY>
<!ATTLIST DATEPICKER COMPARETOFIELD CDATA #REQUIRED>
<!ATTLIST DATEPICKER MAXVALUES CDATA #IMPLIED>
<!-- This describe the ImagePicker -->
<!ELEMENT IMAGEPICKER ANY>
<!ATTLIST IMAGEPICKER ASSETTYPENAME CDATA #REQUIRED>
<!ATTLIST IMAGEPICKER ATTRIBUTETYPENAME CDATA #REQUIRED>
```

```
<!ATTLIST IMAGEPICKER ATTRIBUTENAME CDATA #REQUIRED>
<!ATTLIST IMAGEPICKER CATEGORYATTRIBUTENAME CDATA #IMPLIED>
<!ATTLIST IMAGEPICKER RESTRICTEDCATEGORYLIST CDATA #IMPLIED>
<!ATTLIST IMAGEPICKER MAXVALUES CDATA #IMPLIED>
<!-- Image Editor -->
<!ELEMENT IMAGEEDITOR ANY>
<!ATTLIST IMAGEEDITOR EDITORTYPE (oie | clarkii) "oie">
<!ATTLIST IMAGEEDITOR HEIGHT CDATA #REQUIRED>
<!ATTLIST IMAGEEDITOR WIDTH CDATA #REQUIRED>
<!ATTLIST IMAGEEDITOR FITIMAGE (true | false) "true">
<!ATTLIST IMAGEEDITOR SNAPSHOTPANEL (true | false) "false">
<!ATTLIST IMAGEEDITOR LIMITCROPPING (true | false) "false">
<!ATTLIST IMAGEEDITOR CROPHEIGHT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR CROPWIDTH CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ENABLEOIEFORMAT (true | false) "false">
<!ATTLIST IMAGEEDITOR LIMITSIZE (true | false) "false">
<!ATTLIST IMAGEEDITOR MAXHEIGHT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MAXWIDTH CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MINHEIGHT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MINWIDTH CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR AUTORESAMPLE (true | false) "false">
<!ATTLIST IMAGEEDITOR AUTORESAMPLEPROPORTIONAL (true | false)
"false">
<!ATTLIST IMAGEEDITOR DEFAULTTEXTFONT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR DEFAULTTEXTSIZE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR DEFAULTTEXTCOLOR CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ASSETTYPE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ATTRIBUTE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ATTRIBUTETYPE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR CATEGORYATTRIBUTE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR RESTRICTEDCATEGORYLIST CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ENABLEIMAGEPICKER (true | false) "false">
<!ATTLIST IMAGEEDITOR OIEASSETTYPE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIEATTRIBUTE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIEATTRIBUTETYPE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIECATEGORYATTRIBUTE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIERESTRICTEDCATEGORYLIST CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIEENABLEIMAGEPICKER (true | false) "false">
<!ATTLIST IMAGEEDITOR TAGEDIT (true | false) "false">
<!ATTLIST IMAGEEDITOR BASE64JPEGQUALITY CDATA "95">
<!ATTLIST IMAGEEDITOR ASKTOSAVELOCALLY (true | false) "false">
<!ATTLIST IMAGEEDITOR DEFAULTSAVINGTYPE (gif | jpg | jpe | png |
tif | bmp | oie) "gif">
<!ATTLIST IMAGEEDITOR ENABLEGIFSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR ENABLEJPEGSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR ENABLEPNGSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR ENABLETIFFSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR ENABLEBMPSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR GRIDVISIBLE (true | false) "false">
<!ATTLIST IMAGEEDITOR GRIDSNAP (true | false) "true">
<!ATTLIST IMAGEEDITOR GRIDSPACINGX CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR GRIDSPACINGY CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MAXTHUMBNAILHEIGHT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MAXTHUMBNAILWIDTH CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR THUMBNAILFORMAT (gif | jpg | jpe | png | tif
| bmp | oie) "gif">
<!ATTLIST IMAGEEDITOR MAXVALUES CDATA #IMPLIED>
```

18.1.1.1 Conventions for the `presentationobject.dtd` File

If you want to create custom attribute editors other than the ones made possible by default, you must first define an XML input style tag, a `PRESENTATIONOBJECT` tag, in the `presentationobject.dtd` file. To define a new `PRESENTATIONOBJECT` tag, you must do the following:

- Add the new tag (presentation object) to the list in the `<!ELEMENT PRESENTATIONOBJECT...>` statement.
- Add a `<!ELEMENT...>` section that defines the new tag (presentation object) and the arguments that it takes. Follow the normal syntax rules for a `.dtd` file and follow the conventions used in the `presentationobject.dtd` file.

18.1.2 The Attribute Editor Asset

The attribute editor asset either holds XML code or points to an `.xml` file.

That XML code does one thing: if the input type is one that provides options (check boxes, radio options, drop-down lists, and so on), it provides the values of those options.

Although WebCenter Sites provides nine text files with sample code that you can use to create new attribute editor assets, it does not provide any attribute editor assets because you need to customize the sample code so that any options are appropriate for your data.

When you create your attribute editors, you can either cut and paste the code from HTML version of this book (samples follow this section) or you can use the text files located in the `Samples` subdirectory of the installation directory on your system.

This section contains the following topics:

- Section 18.1.2.1, "The Syntax and the Default Tags"
- Section 18.1.2.2, "CHECKBOXES Example"
- Section 18.1.2.3, "CKEditor Example"
- Section 18.1.2.4, "PICKASSET Example"
- Section 18.1.2.5, "PULLDOWN Example"
- Section 18.1.2.6, "RADIOBUTTONS Example"
- Section 18.1.2.7, "TEXTAREA Example"
- Section 18.1.2.8, "TEXTFIELD Example"
- Section 18.1.2.9, "TYPEAHEAD Example"
- Section 18.1.2.10, "UPLOADER Example"

18.1.2.1 The Syntax and the Default Tags

The code in an attribute editor asset has the following basic format:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="SomeName">
...
...
...

</PRESENTATIONOBJECT>
```

The tag that describes the format of the input style (presentation object) is embedded between the pair of PRESENTATIONOBJECT tags and it can have additional nested tags in it. Although the NAME attribute is required for the PRESENTATIONOBJECT tag, it is not used yet; it is reserved for future use.

The name of any PRESENTATIONOBJECT tag that you include in the code for an attribute editor asset must be defined in the presentationobject.dtd file. This .dtd file has the following PRESENTATIONOBJECT tags defined by default (listed in alphabetic order):

- CHECKBOXES
- CKEDITOR
- DATEPICKER
- FIELDCOPIER
- IMAGEEDITOR
- IMAGEPICKER
- PICKASSET
- PICKFROMTREE (deprecated; use PICKASSET instead)
- PULLDOWN
- RADIOBUTTONS
- TEXTAREA
- TEXTFIELD
- TYPEAHEAD
- UPLOADER

Note that the PRESENTATIONOBJECT tag that you use in the attribute editor code must exactly match the name of the display element that you want to use for the attribute editor. Therefore, if you decide to define a new tag for a custom attribute editor, the element that you create must use the same name as the tag.

For a description of the elements, see Section 18.1.3, "The Attribute Editor Elements." For code samples for attribute editors, see the following sections:

- Section 18.1.2.2, "CHECKBOXES Example"
- Section 18.1.2.3, "CKEditor Example"
- Section 18.1.2.4, "PICKASSET Example"
- Section 18.1.2.5, "PULLDOWN Example"
- Section 18.1.2.6, "RADIOBUTTONS Example"
- Section 18.1.2.7, "TEXTAREA Example"
- Section 18.1.2.8, "TEXTFIELD Example"
- Section 18.1.2.9, "TYPEAHEAD Example"
- Section 18.1.2.10, "UPLOADER Example"

18.1.2.2 CHECKBOXES Example

The presentationobject.dtd defines a CHECKBOXES tag, an attribute editor that uses the tag invokes the CHECKBOXES element, which creates a set of check boxes for the attribute.

The `CHECKBOXES` tag takes the following parameters:

- `ITEM` or `QUERYASSETNAME`: the source of the names listed next to the check boxes. To specify the names, use the `ITEM` parameter. To specify a query asset that obtains the names dynamically from a database table, use the `QUERYASSET` parameter.

  Note the following:

  – You cannot use a SQL statement. You must use a query asset if you want to use a query.

  – The SQL in the query asset must return a "value" column. For example: `select name as value from shippingtype`

  – If the data type of the attribute using the attribute editor is "asset", the query must also return the assets' IDs. For example: `select name as value, id as assetid from Product where...`

- `LAYOUT`: whether the check boxes should be positioned in a vertical list or spread out in a horizontal row. Valid options are `HORIZONTAL` or `VERTICAL`. The default is `HORIZONTAL.`

The following attribute editor code specifies that the `CHECKBOXES` element should use the results of a query asset named `A Prods` for the names of a vertical list of check boxes:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT NAME="CheckBox">
 <CHECKBOXES LAYOUT="VERTICAL">
  <QUERYASSETNAME>A Prods</QUERYASSETNAME>
 </CHECKBOXES>
</PRESENTATIONOBJECT>
```

For example code that shows the use of the `ITEM` parameter, see Section 18.1.2.5, "PULLDOWN Example."

Notes About Data Types

A `CHECKBOXES` attribute editor is appropriate for attributes with the following data types:

- date
- float
- integer
- money
- string
- asset (if asset, you must supply the name of the query asset that returns the names of the assets)

18.1.2.3 CKEditor Example

The `presentationobject.dtd` defines a `CKEDITOR` tag. An attribute editor that uses the tag invokes the `CKEDITOR` element, which launches the CKEditor. The person creating the flex asset enters the value for the attribute in that window.

- You must have the CKEditor application installed and configured correctly.

- It is highly recommended that you use CKEditor only when the data type of the attribute is set to blob. If you use blob as the data type, you do not have to worry about sizing the field.

The `CKEDITOR` tag takes the following parameters:

The following code includes a `CKEDITOR` tag that creates a text box that is 400 pixels wide by 200 pixels high:

- `EMBEDDEDLINKS`: Lets users to create links to other assets, if set to YES. Default value: NO
- `ALLOWEDASSETTYPES`: Lets you define which asset types can be included or linked to by means of CKEditor. Default value: ALL
- `IMAGEASSETTYPE`: Specifies the asset type of the image to be used by Image Picker.
- `DEPTYPE`: The approval dependency between the main asset and an embedded asset. Valid options are EXISTS and EXACT. Default value: EXACT
- `MAXVALUES`: This parameter limits the number of values that can be added to a multi-valued attribute. For example, if MAXVALUES is set to 10, then only ten values can be added (using the Add button). No default value.
- `SCRIPT`: Allows scripts to be run inside the editor. Takes a true/false value. Default value: false
- `MAXLENGTH`: The number of characters in the CKEditor's source view (displayed when you click the CKEditor's Source button). Default value: -1 (unlimited)
- `INSTRUCTION`: Used to provide help for the field.

The `CKEDITOR` tag also takes parameters for customizing the appearance of CKEditor instances. One set of parameters is defined in WebCenter Sites. The other set is available on the CKEditor website.

Parameters defined in WebCenter Sites must be added directly to the XML field of a CKEditor instance in order to be recognized by WebCenter Sites:

- `TOOLBAR`: The name of the toolbar, which is read from the configuration file specified in the `CONFIG` parameter (for example, `CONFIG="myconfig.js"`).
- `WIDTH`: The text box width in pixels. If you do not specify a value for `WIDTH`, WebCenter Sites sets a default width. Sample value: `700px`.
- `HEIGHT`: The text box height in pixels. If you do not specify a value for HEIGHT, WebCenter Sites sets a default height. Sample value: `300px`.

> **Note:** The `WIDTH` and `HEIGHT` parameters replace the deprecated `XSIZE` and `YSIZE` parameters. Setting `XSIZE` and `YSIZE` has no effect.

- `RESIZE`: allows users to resize the text area, if set to `true`. Default value: `false`.

Parameters listed on the CKEditor website must be specified either in a configuration file, named in the CONFIG parameter, or in the CONFIGOBJ parameter. **Except for the TOOLBAR parameter, they will not be recognized by WebCenter Sites if they are added to the CKEditor's XML field.** The following is a description of the `CONFIG` and `CONFIGOBJ` parameters:

- `CONFIG`: Specifies the relative path to the default or custom CKEditor configuration file. For example, setting `CONFIG="myconfig.js"` will load the custom `myconfig.js` file from the CKEditor base path (CKEditor source folder). The base path is defined in the xcelerate.ckeditor.basepath property, in the futuretense_xcel.ini file.

Within the configuration file is a CKEDITOR.editorConfig function, where you specify the CKEditor parameters listed at http://docs.cksource.com/ckeditor_api/symbols/CKEDITOR.html#.editorConfig. The list includes the TOOLBAR parameter.

- CONFIGOBJ: Takes a JSON string, which can contain any of the parameters listed at http://docs.cksource.com/ckeditor_api/symbols/CKEDITOR.config.html. (The list includes the TOOLBAR parameter.)

  The JSON string format uses double quotes:

  ```
  '{foo1:"value1",foo2:"value2"}' ...
  ```

  For example:

  ```
  CONFIGOBJ='{width:"300px",height:"300px",toolbar:"SITES",fullPage:true}'
  ```

Note: CKEditor parameters are recognized and used in the following hierarchical order:

Parameters in the CKEditor's XML field override parameters in CONFIGOBJ, which override parameters in the configuration file named in the CONFIG parameter.

For example, the TOOLBAR parameter in a CKEditor's XML field overrides the toolbar parameter in CONFIGOBJ='{toolbar: "SITES",fullPage:true}', which overrides the toolbar parameter in the configuration file myconfig.js (specified as CONFIG='myconfig.js'). The fullPage parameter in CONFIGOBJ overrides the same parameter in myconfig.js.

To avoid potential conflicts, specify each parameter only once, in the place where it is recognized, either in the editor's XML field, in the CONFIGOBJ parameter, or in the configuration file that is specified in the CONFIG parameter.

The following code includes a CKEDITOR tag that creates a CKEditor box 400 pixels wide by 200 pixels high:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="CKEditorTest">
  <CKEDITOR WIDTH="400px" HEIGHT="200px">
  </CKEDITOR>
</PRESENTATIONOBJECT>
```

Notes About Data Types

The best choice for the data type of an attribute that uses a CKEDITOR attribute editor is blob. You can use string or text but it is problematic because it is hard to predict how large the data entered into the attribute's field will be because each HTML marker counts toward the limit. The string data type is limited to 256 characters and text is limited to 2000. It is recommended that you use blob as the data type for attributes that use CKEDITOR as their input mechanism.

18.1.2.4 PICKASSET Example

The presentationobject.dtd defines a PICKASSET tag. An attribute editor that uses the tag invokes the PICKASSET element, which creates a Drop Zone field into which a user can drag and drop an asset from a tree or search results list. When an asset is dropped into the Drop Zone, the name of the asset is displayed in a box outlined with a dashed border. When you point to the name of the asset, a tooltip opens displaying information about the asset. To change the behavior of the PICKASSET attribute editor, for example, to customize the look and feel of the tooltip.

This tag can take the MAXVALUES parameter. The MAXVALUES parameter limits the number of values that can be added to a multi-valued attribute. For example, if MAXVALUES is set to 10, then only ten values can be added (using the Add button). No default value.

If you wish to customize the PICKASSET attribute editor, see Part III, "Customizing the Contributor Interface."

Below is the code to create a PICKASSET attribute editor:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT NAME="PickAsset">

  <PICKASSET>
  </PICKASSET>
</PRESENTATIONOBJECT>
```

Notes About Data Types

A PICKASSET attribute editor is only appropriate for attributes with a data type of asset.

18.1.2.5 PULLDOWN Example

The presentationobject.dtd defines a PULLDOWN tag. An attribute editor that uses the tag invokes the PULLDOWN element, which formats a field with a drop-down list of values.

This tag takes the following parameters:

- ITEM or QUERYASSETNAME: the source of the names in the drop-down list. To specify the names, use the ITEM parameter. To specify a query asset that obtains the names dynamically from a database table, use the QUERYASSET parameter.

  Note the following:

  – You cannot use a SQL statement. You must use a query asset if you want to use a query.

  – The SQL in the query asset must return a "value" column. For example: select name as value from shippingtype

  – If the data type of the attribute using the attribute editor is "asset", the query must also return the assets' IDs. For example: select name as value, id as assetid from Product where...

The following attribute editor code specifies that the list holds the items red, green, and blue:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
```

```
<PRESENTATIONOBJECT NAME="PulldownTest">
<PULLDOWN>
  <ITEM>Red</ITEM>
  <ITEM>Green</ITEM>
  <ITEM>Blue</ITEM>
</PULLDOWN>
```

For example code that shows how to use the `QUERYASSETNAME` parameter rather than `ITEM`, see Section 18.1.2.2, "CHECKBOXES Example."

Notes About Data Types

A `PULLDOWN` attribute editor is appropriate for attributes with the following data types:

- date
- float
- integer
- money
- string
- asset

A drop-down list is the default input style for attributes of type asset. The list displays all the assets of that type. Use a `PULLDOWN` attribute editor when you want to further restrict the items in the drop-down list with a query asset so that the list doesn't display every asset of that type.

18.1.2.6 RADIOBUTTONS Example

The `presentationobject.dtd` defines a `RADIOBUTTONS` tag. An attribute editor that uses the tag invokes the `RADIOBUTTON`S element, which creates a set of radio options for the attribute.

The `RADIOBUTTONS` tag takes the following parameters:

- `ITEM` or `QUERYASSETNAME`: the source of the names listed next to the radio options. To specify the names, use the `ITEM` parameter. To specify a query asset that obtains the names dynamically from a database table, use the `QUERYASSET` parameter.

  Note the following:

  – You cannot use a SQL statement. You must use a query asset if you want to use a query.

  – The SQL in the query asset must return a "value" column. For example: `select name as value from shippingtype`

  – If the data type of the attribute using the attribute editor is "asset", the query must also return the assets' IDs. For example: `select name as value, id as assetid from Product where...`

- `LAYOUT`: whether the buttons should be positioned in a vertical list or spread out in a horizontal row. Valid options are `HORIZONTAL` or `VERTICAL`. The default is `HORIZONTAL.`

The following attribute editor code specifies that the `RADIOBUTTONS` element should use the results of a query asset named `A Prods` for the names of a vertical list of buttons:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
```

```
<PRESENTATIONOBJECT NAME="RadioButtonTest">
  <RADIOBUTTONS LAYOUT="VERTICAL">
    <QUERYASSETNAME>A Prods</QUERYASSETNAME>
  </RADIOBUTTONS>
</PRESENTATIONOBJECT>
```

For example code that shows the use of the ITEM parameter, see Section 18.1.2.5, "PULLDOWN Example."

Notes About Data Types

A RADIONBUTTON attribute editor is appropriate for attributes with the following data types:

- date
- float
- integer
- money
- string
- asset (if asset, you must supply the name of the query asset that returns the names of the assets)

> **Note:** RadioButton atttribute editor of type date would interpret the date values entered in the presentation.dtd in server's timezone (or the timezone set in the JVM). This value will be converted to user's timezone when the asset form is rendered.
>
> If users want to see the same date value irrespective of the timezone, they should consider using an attribute editor of type string.

18.1.2.7 TEXTAREA Example

The presentationobject.dtd defines a TEXTAREA tag. An attribute editor that uses the tag invokes the TEXTAREA element, which creates a text box field for the attribute, and a pair of radio buttons that allows users to specify whether or not that attribute should display embedded link buttons.

The TEXTAREA tag takes the following parameters:

- WIDTH: the text box width, in pixels. If you do not specify a value for WIDTH, WebCenter Sites sets a default width. Sample value: 700px
- HEIGHT: the text box height, in pixels. If you do not specify a value for HEIGHT, WebCenter Sites sets a default height. Sample value: 300px

> **Note:** The WIDTH and HEIGHT parameters replace the deprecated XSIZE and YSIZE parameters. Setting XSIZE and YSIZE has no effect.

- RESIZE: allows users to resize the text area, if set to true. The default value is false.
- WRAPSTYLE: whether the text in the box wraps at all, and, if it does whether it wraps automatically (soft) or only when the user presses the Enter key (a hard return). Valid options are SOFT, HARD, and OFF. The default is OFF .

- DEPTYPE: the approval dependency between the main asset and an embedded asset. Valid options are EXISTS and EXACT. The default is EXACT.
  - EXISTS: when the main asset is edited, approved, and re-published, the embedded asset does not need to be approved and re-published as long as a version of the asset already exists at the destination.
  - EXACT: when the main asset is edited, approved, and re-published, the embedded asset, if it was edited, must be approved and re-published as well.

The following attribute editor code defines the XSIZE as 40 pixels, the YSIZE as 5 pixels, disables text wrapping by setting WRAPSTYLE to OFF, and sets the approval dependency for embedded assets to EXISTS:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="TextAreaTest">
  <TEXTAREA XSIZE="40" YSIZE="5" WRAPSTYLE="OFF" DEPTYPE="EXISTS">
  </TEXTAREA>
</PRESENTATIONOBJECT>
```

Notes About Data Types

A TEXTAREA attribute editor is appropriate for attributes with the data type of text and blob. Use the text data type when you need to store up to 2000 characters. If you need to store more than 2000 characters, use the blob data type.

18.1.2.8 TEXTFIELD Example

The presentationobject.dtd defines a TEXTFIELD tag. An attribute editor that uses the tag invokes the TEXTFIELD element from the New and Edit forms, which creates a text field for the attribute. When the attribute is displayed on the Inspect form, however, it uses the DisplayTEXTFIELD element.

The TEXTFIELD tag takes the following parameters:

- XSIZE: the length of the field, in characters.
- MAXCHARS: the number of characters, up to 256, allowed in the field.
- BLANKED: whether the attribute's value is replaced with a string of asterisks when it is displayed in the Inspect form. For example, if you created a "password" attribute, you would not want the value of the password displayed in an Inspect form. Valid options are YES and NO. The default is NO.

  Because using the BLANKED parameter automatically means that you need the field to behave differently on the New and Edit forms than it does on the Inspect form, the TEXTFIELD tag is delivered with both of the two possible elements by default.

The following attribute editor code defines the XSIZE as 60 and the maximum number of characters as 80:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="TextFieldTest">
  <TEXTFIELD XSIZE="60" MAXCHARS="80">
  </TEXTFIELD>
</PRESENTATIONOBJECT>
```

Notes About Data Types

A TEXTFIELD attribute editor is appropriate for attributes with the following data types:

- float
- integer
- money
- string
- url

18.1.2.9 TYPEAHEAD Example

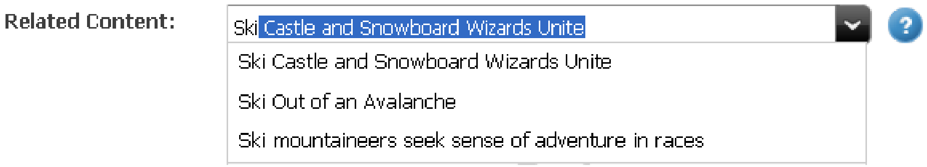
The presentationobject.dtd defines a TYPEAHEAD tag. An attribute editor that uses the tag invokes the TYPEAHEAD element, which creates a type ahead input box with a drop-down menu attribute. When the attribute is displayed in the New and Edit views of an asset, and the down-arrow in the drop-down field is clicked, the drop-down menu lists the names of the assets from which the user can choose.

The TYPEAHEAD tag takes the following parameters:

- PAGESIZE: limits the number of results that are shown in the drop-down menu at one time for each type ahead search. For example, if PAGESIZE is set to 5, then only five assets will be displayed in the drop-down menu at one time. The default value is 10.
- MAXVALUES: limits the number of values that can be added to a multi-valued attribute. For example, if MAXVALUES is set to 10, then only ten values can be added using the Add button). No default value.

The PAGESIZE parameter can be used to limit the number of results shown in the drop-down at one time. This paginates the results listed in the drop-down menu. For example, if the PAGESIZE parameter is set to show 10 assets at a time in the drop-down menu, a user can see the next 10 assets in the list by clicking **More choices**. Similarly, to go back to the last 10 assets that were listed, the user can click the **Previous choices** link.

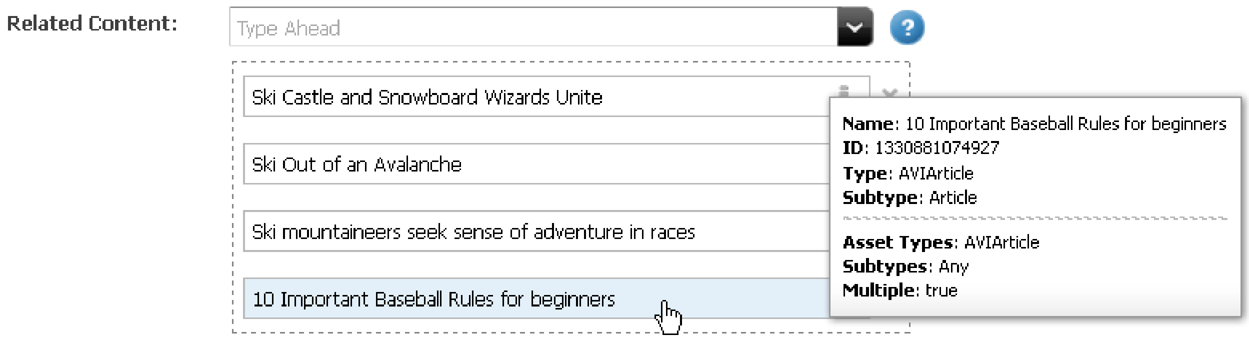
The list of results can also be filtered by typing the first couple of letters that the desired asset starts with into the input box. For example, if "Ski" is typed into the input box, the results that are displayed in the drop-down menu (in alphabetic order) will all start with "Ski." This is similar to running the query Ski*, which finds all assets that begin with the letters or word "Ski." As you type into the type ahead input box, it auto-fills with the first result in the drop-down menu (as shown below).



Related Content: Ski Castle and Snowboard Wizards Unite

Ski Castle and Snowboard Wizards Unite
Ski Out of an Avalanche
Ski mountaineers seek sense of adventure in races
?

Users can press **Enter** on their keyboards, if they want to select the auto-filled asset. Users can also select any result, one at a time, from the drop-down menu, either by selecting the desired asset (navigating the drop-down menu with the up and down arrow keys on the keyboard) and then pressing the **Enter** key or by using the mouse to click the desired asset. Once you select an asset, it is displayed below the drop-down menu. When a user points to a selected asset, a tooltip is shown (as shown below).



Ski Castle and Snowboard Wizards Unite

Ski Out of an Avalanche

Ski mountaineers seek sense of adventure in races

10 Important Baseball Rules for beginners

Name: 10 Important Baseball Rules for beginners
ID: 1330881074927
Type: AVIArticle
Subtype: Article

Asset Types: AVIArticle
Subtypes: Any
Multiple: true

To change the behavior of the TYPEAHEAD attribute editor, for example, to customize the look and feel of the tooltip, see Part III, "Customizing the Contributor Interface."

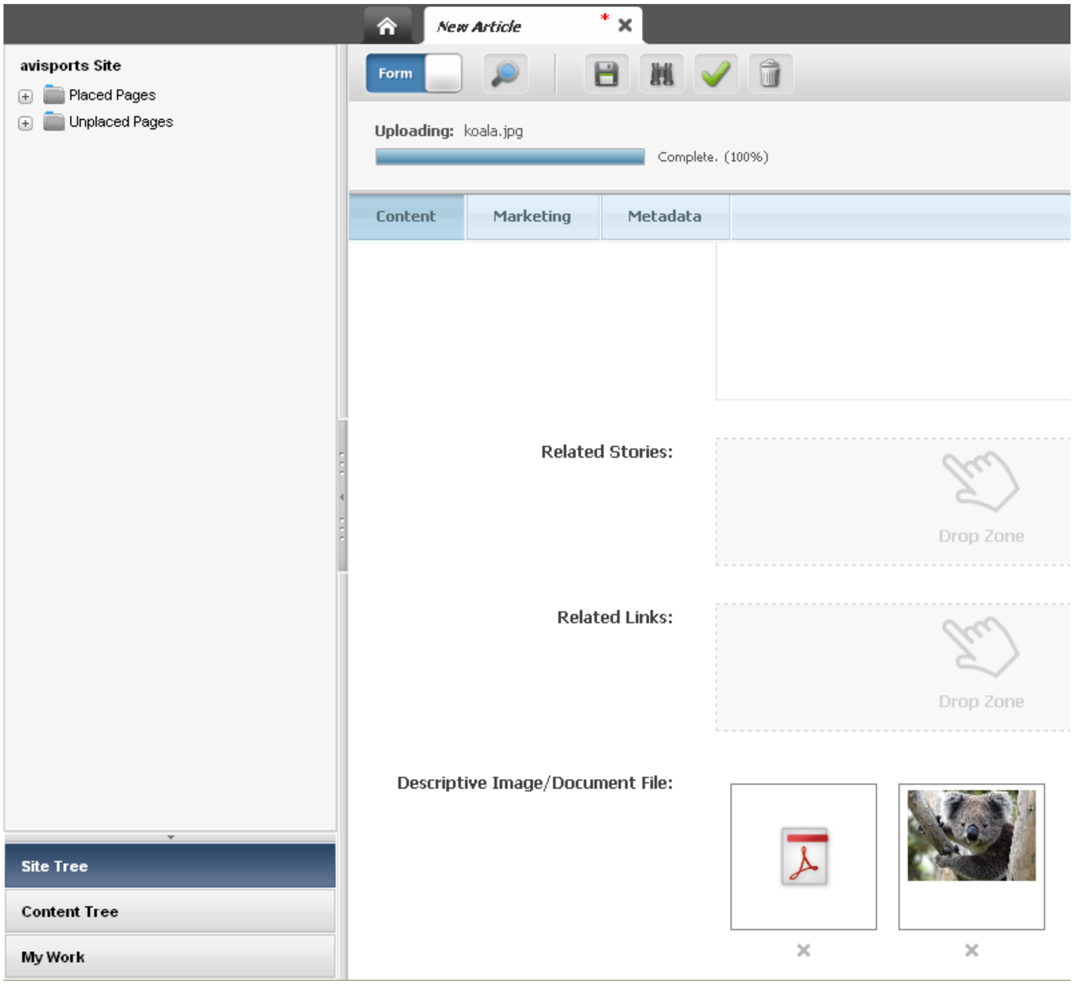
Notes About Data Types

A TYPEAHEAD attribute editor is appropriate only for attributes of data type asset.

18.1.2.10 UPLOADER Example

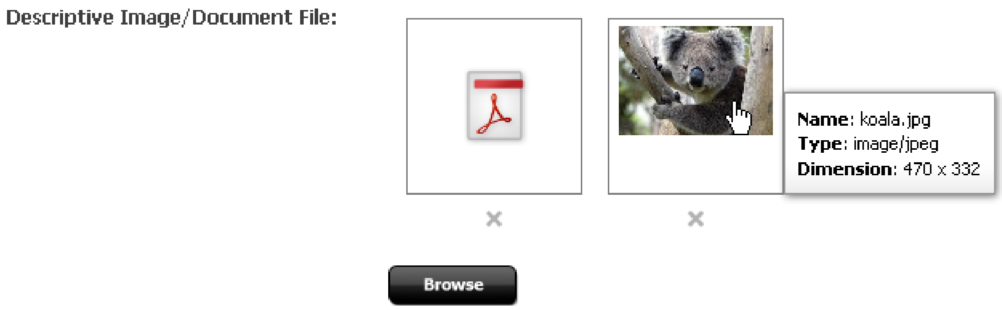
The presentationobject.dtd defines an UPLOADER tag. An attribute editor that uses the tag invokes the UPLOADER element, which creates an upload attribute. The upload field can upload files of any type. However, to restrict the types of files that can be uploaded, use the FILETYPES parameter.

When a user clicks to upload a file, the progress of the upload is shown in the asset's toolbar (as shown below).



A thumbnail image for each uploaded file is displayed in the Uploader field. When a user clicks the thumbnail of an uploaded image file, the image is displayed in an overlay at its actual size. If the uploaded files are not images, WebCenter Sites displays a thumbnail image specific to that file type in the Uploader field.If WebCenter Sites does not have a thumbnail image for a particular file type, an arbitrary image is shown, instead.

When a user points to the thumbnail image of an image file, the file's name, type, and dimensions are displayed in a tooltip. When a user points to the thumbnail of a file that is not an image, the file's name and type is displayed in a tooltip.



The UPLOADER tag takes the following parameters:

- MAXFILESIZE: lets you specify the maximum size of the files that can be uploaded. Valid units for this parameter are B, KB, MB, and GB. Default value: 1024MB. Default unit: KB
- FILETYPES: lets you specify the types of files that can be uploaded. For example, *.jpg*. If you do not specify a value for this parameter, the default value is any (*.*).
- MINWIDTH: lets you specify the minimum width (in pixels) of the files that can be uploaded. For example, if MINWIDTH is set to 700px, then users can only upload files with a width that is no less than 700px. If you do not specify a value for this parameter, the default value is any width. Valid unit: px
- MAXWIDTH: lets you specify the maximum width (in pixels) of the files that can be uploaded. For example, if MAXWIDTH is set to 700px, then users can only upload files with a width that is no more than 700px. If you do not specify a value for this parameter, the default value is any width. Valid unit: px
- MINHEIGHT: lets you specify the minimum height (in pixels) of the files that can be uploaded. For example, if MINHEIGHT is set to 700px, then users can only upload files with a height that is no less than 700px. If you do not specify a value for this parameter, the default value is any height. Valid unit: px
- MAXHEIGHT: lets you specify the maximum height (in pixels) of the files that can be uploaded. For example, if MAXHEIGHT is set to 700px, then users can only upload files with a height that is no more than 700px. If you do not specify a value for this parameter, the default value is any height. Valid unit: px
- MAXVALUES: limits the number of values that can be added to a multi-valued attribute. For example, if MAXVALUES is set to 10, then only ten values can be added (using the Add button). No default value.

Notes About Data Types

A UPLAODER attribute editor is appropriate only for attributes of data type blob.

18.1.3 The Attribute Editor Elements

The elements that take the input values passed to them from their attribute editor counterparts supply the logic behind the format and behavior of the attribute when it is displayed on a form. For example, it might perform a loop sequence for multivalue attributes so that additional values can be entered in the field.

Following are the default flex attribute display elements located in the ElementCatalog table under OpenMarket/Gator/AttributeTypes. The names of these elements match exactly the names of the custom XML tags defined in the presentationobject.dtd file:

Table 18–1 Attribute Editor Elements

Element	Description
CHECKBOXES	Formats the input style of the attribute as a set of check box options. The attribute editor must either define the names of the options or provide the name of a query asset to use to obtain the names.
CKEDITOR	Invokes the CKEditor text editor. The attribute editor must specify the height and width pixel dimensions for the text box.

Table 18–1 (Cont.) Attribute Editor Elements

Element	Description
PULLDOWN	Formats the input style of the attribute as a select field with a drop-down list. The attribute editor must either specify the items that are displayed in the list or provide the name of a query asset to use to obtain the values.
RADIOBUTTONS	Formats the input style of the attribute as a set of radio options. The attribute editor must define the names of the options or provide the name of a query asset to use to obtain the names.
TEXTAREA	Formats the input style of the attribute as a text box and displays radio buttons that allow the user to specify whether or not the text box will allow embedded links. The attribute editor must define the x and y dimensions of the box.
TEXTFIELD	Formats the input style of the attribute as a text field. The attribute editor must define the length of the field and the number of characters that are allowed in the field.
DisplayTEXTFIELD	Formats the appearance of the text field attribute's value when it is displayed on the Inspect form. If the attribute editor sets the BLANKED parameter to YES, this element displays the value from the field as a string of asterisks. Typically used for password fields.
PICKASSET	Formats the input style of the attribute as a Drop Zone field into which a user can drag and drop an asset from a tree or search results list.
PICKFROMTREE	Deprecated. Use PICKASSET.
TYPEAHEAD	Formats the input style of the attribute as a type ahead input box with a drop-down menu. When the asset is displayed in the New or Edit view of an asset, and the down-arrow in the drop-down field is clicked, the drop-down menu lists the names of the assets from which the user can choose. Use the PAGESIZE parameter to limit the number of results shown in the drop-down menu at one time.
UPLOADER	Creates an upload attribute. The upload field can upload files of any type. Use the FILETYPES parameter to restrict the types of files that can be uploaded.

18.1.4 Conventions for the Attribute Editor Elements

In order for WebCenter Sites to use an element for an attribute editor, that element must conform to the following rules:

- It must have the same name as the input style tag that calls it from the attribute editor code. For example, the default CHECKBOXES tag has a default CHECKBOXES.xml element.
- The element must be placed in the ElementCatalog using the following naming conventions: OpenMarket/Gator/AttributeTypes/name

If you want to create your own display elements to use with custom attribute editors, it is best to find one that is the closest to the attribute editor element that you want to create and then copy as much of it as possible.

For help, examine the code in the default attribute editor elements and read the following descriptions of the variables and syntax in them.

Variables

When WebCenter Sites loads a form that uses the attribute editor, it calls the element with the computer name. It passes the information in the following variables to the display element:

- `PresInst`: the instance of the current presentation object
- `AttrName`: the name of the current attribute
- `AttrType`: the data type of the current attribute
- `EditingStyle`: whether the attribute can take more than one value (based on the value in the **Number of Values** field for the attribute). This variable is set to either `single` or `multiple`.
- `RequiredAttr`: whether or not the attribute is required for the current asset. The variable is set to either `true` or `false`.
- `MultiValueEntry`: instructs WebCenter Sites how to handle the values for an attribute that can take more than one value.

  When this value is set to `yes`, the display element is called once, under the assumption that the widget created by the element enables the user to select more than one value in it (a multi-select drop-down list, for example).

  When this value is set to `no`, WebCenter Sites calls the display element once for each possible value for the attribute and displays one widget for each value that can be stored.

  Note that this value is always set to `yes` initially.
- `doDefaultDisplay`: whether to use the default input style for an attribute of this type. For a list, see Section 11.3.3.2, "Default Input Styles for Attributes." When WebCenter Sites calls the display element, this variable is initially set to `yes`. To use the input widget created by the element, the element must reset this variable to `no`.
- `AttrValueList`: the list of all the values for this attribute.
- `TempVal`: the value of a single attribute value.

Other Required Syntax

The code in the display element must also use the following conventions:

- It must store information about how to validate the attribute values in a variable named `RequireInfo`. WebCenter Sites passes this variable elements use JavaScript to validate the attribute values. Those elements are:

  ```
  OpenMarket/Gator/FlexibleAssets/FlexAssets/ContentForm1
  OpenMarket/Gator/FlexibleAssets/FlexGroups/ContentForm1
  ```

  This JavaScript performs prescribed error checking and validation based on the type of control, the data type, and other predictable characteristics. The information passed in the `RequireInfo` variable informs the JavaScript about the custom requirements for the attribute editor.
- The name of the widget in the display element (the `INPUT NAME`) must use the following convention:
  - For a single-value attribute, the name of the attribute.
  - For a multi-value attribute, it must use a 1-based counter prepend the attribute name for each attribute value (for example, 1color, 2color, 3color).

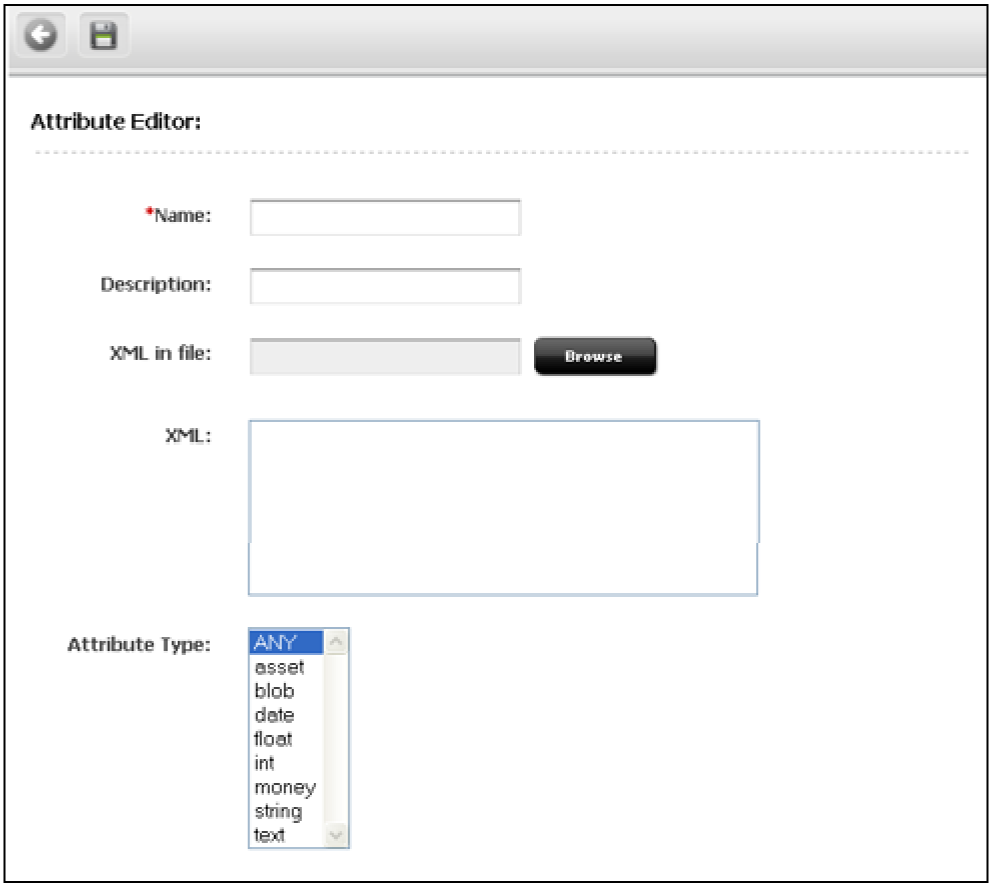
For an example, see Section 18.3, "Customizing Attribute Editors."

18.2 Creating Attribute Editors

To create an attribute editor using the sample XML code provided in this chapter or in the sample text files, complete the following steps:

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Click **New** and select **New Attribute Editor** from the shortcut list

   The **New Attribute Editor** form appears:



5. Click in the **Name** field and enter a unique name of up to 64 characters, excluding spaces.
6. Click in the **Description** field and enter a short phrase that describes the purpose of the attribute editor.
7. Click in the **XML** field. Either cut and paste the appropriate sample XML attribute editor code from the HTML version of this guide or from the sample text files provided in the `Samples` subdirectory of the installation directory.
8. Edit the code as needed. For example, if you are creating a `CHECKBOXES` or a `RADIOBUTTONS` attribute editor, you must provide names for the check boxes or radio buttons. If you are creating a `PULLDOWN` attribute editor, you must provide the values for the drop-down list.

See Section 18.1.2, "The Attribute Editor Asset" for more information about coding the attribute editor.

9. Click **Save**.

> **Note:** Another option is to code the XML for the attribute editor in a separate `.xml` file. In this case, rather than enter the code directly into the XML field, click the **Browse** button next to the XML in file field and select the file.

10. Before this attribute editor can be published to the management system, you must Approve it.

> **Note:** If you are using a query asset with this attribute editor, be sure to approve both the attribute editor and the query asset.
>
> Because the dependency between an attribute editor and its query asset is specified in the XML code in the attribute editor, the approval system can not detect the dependency and verify that the query asset exists on the management system.

18.3 Customizing Attribute Editors

If you need to create your own custom attribute editor, the best thing to do is to copy as much as you can from the sample attribute editor code and the sample display elements.

If you determine that you must create a new input style (you cannot use any of the default `PRESENTATIONOBJECT` tags), you must add a new `PRESENTATIONOBJECT` section to the `presentationobject.dtd` file that defines the attribute editor. For information about adding to this file, see Section 18.1.1, "The presentationobject.dtd File."

When you create a custom `PRESENTATIONOBJECT` tag, you must also supply the appropriate display elements for it:

- Required: An element that formats the attribute (displays an edit mechanism) when that attribute appears in a New or Edit form.
- Optional: An element that formats the attribute when it appears in the Inspect form.
- Optional: An element that formats the attribute data before it is saved in the database.

For information about the variables and conventions used in the display elements for an attribute editor, see Section 18.1.3, "The Attribute Editor Elements."

If you are creating or customizing an attribute editor into which a file can be uploaded (for example, if you are customizing the UPLOADER attribute editor), you can add custom logic to the attribute editor code which will ensure that the file being uploaded is valid. For more information, see Section 18.3.2, "Adding Custom Logic to Validate an Uploaded File."

18.3.1 Example: Customized Attribute Editor

This example demonstrates how you could customize the description of the `TEXTAREA` tag in the `presentationobject.dtd` file and the `TEXTAREA` element to create an attribute editor that disables a text box if the user does not have the proper permissions.

There are three steps:

- Step 1: Editing the description of the `TEXTAREA` tag in the `presentationobject.dtd` to support a new parameter named `PERMISSIONS`.
- Step 2: Writing the code for the attribute editor and creating the attribute editor.
- Step 3: Editing the `TEXTAREA` element to check the value of `PERMISSIONS`.

Step 1: Editing the presentationobject.dtd file

To support the new parameter, you add a single line of code to the `TEXTAREA` description in the presentationobject.dtd. The new line of code is line 8. Lines 1 through 7 are the default description of the `TEXTAREA` tag.

```
1. <!-- TEXTAREA: A text area of a specific size. You must specify
2. -- the x and y dimensions; the wrap style defaults to soft.
3. -->
4. <!ELEMENT TEXTAREA ANY>
5. <!ATTLIST TEXTAREA XSIZE CDATA #REQUIRED>
6. <!ATTLIST TEXTAREA YSIZE CDATA #REQUIRED>
7. <!ATTLIST TEXTAREA WRAPSTYLE (OFF | SOFT | HARD) "SOFT">
8. <!ATTLIST TEXTAREA PERMISSION CDATA>
```

Step 2: Example Code for the Example Attribute Editor

Here is the example code with the new parameter. It specifies that a user must have "Administrators" as the value for `PERMISSION` in order to see the field:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="TextAreaTest">
  <TEXTAREA XSIZE="40" YSIZE="10" WRAPSTYLE="SOFT" PERMISSION="Administrators">
  </TEXTAREA>
</PRESENTATIONOBJECT>
```

Step 3: Editing the TEXTAREA Element

The third step is editing the `TEXTAREA` element. Lines 56 through 70, 123 through 125, and 172 through 174 are the new code that enables or disables the field, based on the value of the `PERMISSION` parameter:

```
1. <?XML VERSION="1.0" ?>
2. <!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
3. <FTCS Version="1.1">
4. <!-- OpenMarket/Gator/AttributeTypes/TEXTAREA
5. --
6. -- INPUT
7. --
8. -- OUTPUT
9. --
```

```
-->

<!-- Display one TEXTAREA per attribute value -->
<IF COND="Variables.MultiValueEntry=no">
<THEN>

<!-- Don't want default display field  -->
<setvar NAME="doDefaultDisplay" VALUE="no"/>

<!-- Get all parameters from Attribute Editor xml -->
<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="FONTSIZE" VARNAME="FONTSIZE"/>
   <if COND="Variables.errno!=0">
<then>
<setvar NAME="FONTSIZE" VALUE="2"/>
</then>
</if>

<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="WRAPSTYLE" VARNAME="WRAPSTYLE"/>
<if COND="IsVariable.WRAPSTYLE!=true">
<then>
<setvar NAME="WRAPSTYLE" VALUE="OFF"/>
</then>
</if>

<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="XSIZE" VARNAME="XSIZE"/>
<if COND="IsVariable.XSIZE!=true">
<then>
<setvar NAME="XSIZE" VALUE="24"/>
</then>
</if>

<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="YSIZE" VARNAME="YSIZE"/>
<if COND="IsVariable.YSIZE!=true">
<then>
```

```
<setvar NAME="YSIZE" VALUE="20"/>
</then>
</if>

<setvar NAME="disableTextArea" VALUE="no"/>
<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="PERMISSION" VARNAME="PERMISSION"/>
<if COND="IsVariable.PERMISSION=true">
<then>
<setvar NAME="errno" VALUE="0"/>
<USERISMEMBER GROUP="Variables.PERMISSION"/>
<IF COND="Variables.errno=0">
<THEN>
<setvar NAME="disableTextArea" VALUE="yes"/>
</THEN>
</IF>
</then>
</if>

<tr>

<!-- Standard element to display attribute name or description
-->
<callelement NAME="OpenMarket/Gator/FlexibleAssets/Common
/DisplayAttributeName"/>
<td></td>
<td>

<!-- Single valued attributes -->
<if COND="Variables.EditingStyle=single">
<then>

<!-- Special case: TEXTAREA for URL attributes -->
<IF COND="Variables.AttrType=url">
<THEN>
<setvar NAME="errno" VALUE="0"/>
<BEGINS STR="AttrValueList.urlvalue"
WHAT="AttrValueList."/>
<IF COND="Variables.errno=1">
<THEN>
<setvar NAME="filename" VALUE="CS.UniqueID.txt"/>
```

```
</THEN>
<ELSE>
<setvar NAME="filename"
VALUE="AttrValueList.urlvalue"/>
</ELSE>
</IF>

<INPUT TYPE="hidden" NAME="Variables.AttrName_file"
VALUE="Variables.filename"
REPLACEALL="Variables.AttrName,Variables.filename"/>

<setvar NAME="errno" VALUE="0"/>
<BEGINS STR="AttrValueList.@urlvalue"
WHAT="AttrValueList."/>
<IF COND="Variables.errno=1">
<THEN>
<setvar NAME="MyAttrVal" VALUE="Variables.empty"/>
</THEN>
<ELSE>
<setvar NAME="MyAttrVal"
VALUE="AttrValueList.@urlvalue"/>
</ELSE>
</IF>
</THEN>
</IF>

<!-- Display a TEXTAREA with all parameters from Attribute
--Editor xml -->
<!-- The NAME of the input must be the attribute name -->
<IF COND="Variables.disableTextArea=yes">
<THEN>
<TEXTAREA DISABLED="yes" NAME="Variables.AttrName"
ROWS="Variables.YSIZE" COLS="Variables.XSIZE"
WRAP="Variables.WRAPSTYLE"
REPLACEALL="Variables.AttrName,Variables.XSIZE,
Variables.YSIZE,Variables.WRAPSTYLE,Variables.empty">

<!-- For most single valued attrs, the value is contained in
MyAttrVal -->
<csvar NAME="Variables.MyAttrVal"/>
</TEXTAREA>
</THEN>
```

```
<ELSE>
<TEXTAREA NAME="Variables.AttrName"
ROWS="Variables.YSIZE" COLS="Variables.XSIZE"
WRAP="Variables.WRAPSTYLE"
REPLACEALL="Variables.AttrName,Variables.XSIZE,
Variables.YSIZE,Variables.WRAPSTYLE,Variables.empty">
<!-- For most single valued attrs, the value is
contained in MyAttrVal -->
<csvar NAME="Variables.MyAttrVal"/>
</TEXTAREA>
</ELSE>
</IF>
</then>
<else>
<!-- Multiple valued attributes -->
<!-- For single value attributes we can usually use the
default RequireInfo -->
<!-- For multiple value attributes we need to append to
RequireInfo for each value -->
<if COND="Variables.RequiredAttr=true">
<then>
<setvar NAME="RequireInfo"
VALUE="Variables.RequireInfo*Counters.TCounterVariables.
AttrName*ReqTrue*Variables.AttrType!"/>
</then>
<else>
<setvar NAME="RequireInfo"
VALUE="Variables.RequireInfo*Counters.TCounterVariables
.AttrName*ReqFalse*Variables.AttrType!"/>
</else>
</if>

<!-- Display a TEXTAREA with all parameters from Attribute
Editor xml -->
<!-- The NAME of the input must be the attribute name
prepended by the TCounter counter -->
<IF COND="Variables.disableTextArea=yes">
<THEN>
<TEXTAREA DISABLED ="yes" NAME="Counters.TCounterVariables.AttrName"
ROWS="Variables.YSIZE" COLS="Variables.XSIZE"
WRAP="Variables.WRAPSTYLE"
REPLACEALL="Counters.TCounter,
Variables.AttrName,Variables.XSIZE,
Variables.YSIZE,Variables.WRAPSTYLE">
<csvar NAME="Variables.tempval"/> </TEXTAREA>
</THEN>
<ELSE>
<TEXTAREA NAME="Counters.TCounterVariables.AttrName"
ROWS="Variables.YSIZE" COLS="Variables.XSIZE"
WRAP="Variables.WRAPSTYLE"
REPLACEALL="Counters.TCounter,
Variables.AttrName,Variables.XSIZE,
Variables.YSIZE,Variables.WRAPSTYLE">
<csvar NAME="Variables.tempval"/> </TEXTAREA>
</ELSE>
</IF>
</else>
</if>
</td>
</tr>
</THEN>
</IF> <!-- MultiValueEntry -->
</FTCS>
```

18.3.2 Adding Custom Logic to Validate an Uploaded File

The following steps describe how to add custom logic to an attribute editor's code in order to validate an uploaded file.

To add custom logic to an attribute editor's code to validate an uploaded file

1. Create a JSP element with the name `ValidateFileUpload` in the path `CustomElements/fatwire/ui/util`.
2. In the JSP element, retrieve the file name and the byte array containing the uploaded file data in the request scope by calling `ics.GetVar("filename")` and `ics.GetVar("filebytes")`.
3. (Optional) Write your custom logic to ensure that the uploaded file data is valid:
   - If the file data is valid, set the variable `fileValidated` to `true` by calling `ics.SetVar("fileValidated","true")`. This ensures that the file will be uploaded to the WebCenter Sites server.
   - If the file data is invalid, set the variable `fileValidated` to `false` by calling `ics.SetVar("fileValidated","false")`. This rejects the data uploaded by the user and displays an error message.

   Note: Sample code (except validation logic) is available in the base element `fatwire/ui/util/ValidateFileUpload`. This element should not be modified and the custom code must be written in the `CustomElements` path as specified in step 1.

18.4 Editing Attribute Editors

Note the following when editing an attribute editor:

- You can change the **Name** without causing a schema change.
- You can change the **Description** without causing data loss.
- If you change code in the attribute editor:
  - You can add input options.
  - If you have existing data, you should not remove input options. If you do, some of your existing data will no longer be valid and you will have to search through the database and fix it.
  - If you change the input style, you risk a data mismatch.

19

Configuring Bundled Attribute Editors

This chapter explains how to configure instances of attribute editors that ship with WebCenter Sites.

This chapter contains the following sections:

19.1 Configuring CKEditor

CKEditor is an open source WYSIWYG text editor from CKSource which requires no client-side installation. CKEditor is bundled with WebCenter Sites. In the FirstSiteII sample site, selected asset types containing WYSIWYG-enabled text fields are configured to use CKEditor by default.

This section contains the following topics:

19.1.1 Before You Begin

If you are using CKEditor with Internet Explorer, and WebCenter Sites is configured to use the cp1252 character set, complete the following steps to have CKEditor work correctly in Internet Explorer:

1. Open the `futuretense.ini` property file and note the value of the `cs.contenttype` property.
2. Log in to Oracle WebCenter Sites Explorer.

3. Go to the element `SiteCatalog\OpenMarket\xcelerate\Actions\CKEditorRenderer` and set its `resargs1` field to:

```
cs.contenttype=[your content type]; charset=windows-1252
```

where `[your content type]` takes the value that is set in the `futuretense.ini` file.

19.1.2 Creating a CKEditor Instance and Enabling It for a Field

This procedure shows you how to create an instance of CKEditor and enable it for a flex asset attribute based on a FirstSiteII example.

In our example, you will enable a new CKEditor instance as the input method (attribute editor) for the **FSIIAbstract** field. (**FSIIAbstract** is a flex attribute of the "Content" sample asset type.)

To create and enable a CKEditor instance

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Create the CKEditor instance:
    a. In the button bar, click **New**.
    b. In the list of asset types, click **New Attribute Editor**.

       WebCenter Sites displays the New Attribute Editor form.
    c. In the **Name** field, enter a name that uniquely identifies this instance of CKEditor. For the purpose of our example, enter **CK_FSIIAbstract**.

       Clients can use generic names to create the CKEditor and use it for multiple attributes. There is no need to uniquely identify the CKEditor instance for the attribute unless there is a specific requirement for that field. For example, the width and height of the CKEditor in that field if it is different than other CKEditor fields.
    d. Paste the following code into the **XML** field:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT NAME="CKEDITOR">
<CKEDITOR WIDTH="400" HEIGHT="200"></CKEDITOR>
</PRESENTATIONOBJECT>
```

    a. Click **Save**.
5. Enable the CKEditor instance as the input method for the **FSIIAbstract** field:
    a. Find and open the **FSIIAbstract** attribute asset in the Edit form:
        a. In the button bar, click **Search**.
        b. In the list of asset types, click **Find Content Attribute**.
        c. In the search field, enter **FSIIAbstract**. Click **Search**.
        d. In the list of search results, navigate to the **FSIIAbstract** asset and click its **Edit** (pencil) icon.

           WebCenter Sites opens the asset in the Edit form.

b. Set CKEditor as the attribute editor (input method) for this attribute. In the Attribute Editor drop-down list, select **CK_FSIIAbstract**.

c. Click **Save** to save your changes.

6. Test your new CKEditor instance:

a. Switch to the Contributor interface.

b. Find and open any "Content" asset in its Edit view:

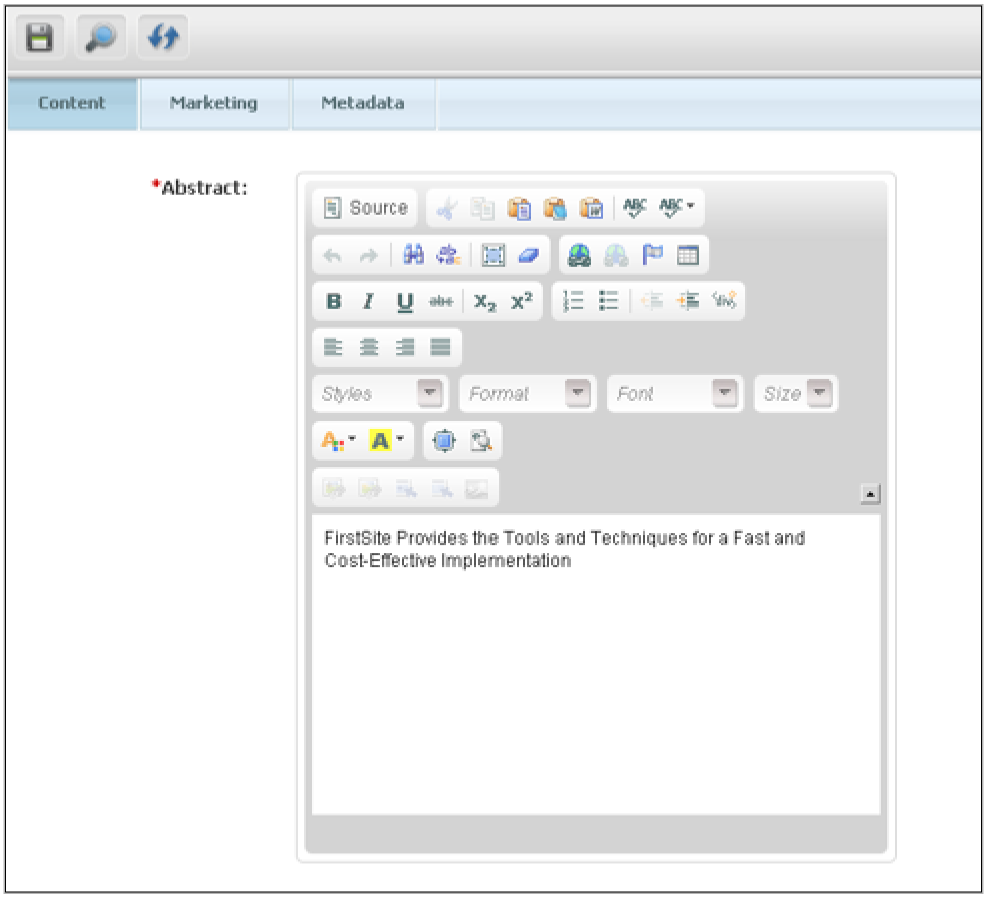
a. In the **Search** field, click the down-arrow icon to open the "Search Type" menu. In the "Search Type" menu, select **Find Content**. Then click the **magnifying glass** button.

a. A Search tab opens displaying the results of your search.

b. In the list of asset types, navigate to a "Content" asset of your choice (**FSIIAbout** in our example), right-click the asset and select **Edit** from its context menu.

A tab opens displaying the asset in its Edit view.

c. Navigate to the **Abstract** field and click in the field. It should look as follows:



FirstSite Provides the Tools and Techniques for a Fast and Cost-Effective Implementation

If CKEditor does not appear, check the attribute editor XML code and the selection you made in the Attribute Editor drop-down list you made in step 5 above.

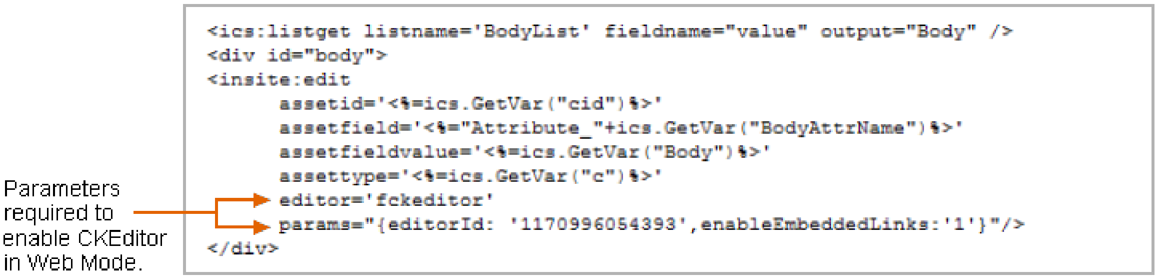
d. In the asset's toolbar, click the Inspect icon to display the asset's Inspect view.

19.1.3 Enabling CKEditor for Use in Web Mode

To enable users to edit assets using CKEditor in Web Mode, you will need to set the `editor` and `params` parameters in the `insite:edit` tag within the appropriate template. The template whose code you edit must be either for the asset whose CKEditor you wish to work with in Web Mode, or for the asset type associated with that asset.

Table 19–1 Parameters that Enable CKEditor in Web Mode

Tag	Parameter	Value	Description
insite:edit	`editor`	`ckeditor`	Specifies the name of the editor to use.
n/a	`params`	`editorId`	Specifies the ID of the CKEditor you want to use in Web Mode.
n/a	n/a	`enableEmbeddedLinks`	Enables the link icons on the CKEditor toolbar.



```
<ics:listget listname='BodyList' fieldname="value" output="Body" />
<div id="body">
<insite:edit
      assetid='<%=ics.GetVar("cid")%>'
      assetfield='<%="Attribute_"+ics.GetVar("BodyAttrName")%>'
      assetfieldvalue='<%=ics.GetVar("Body")%>'
      assettype='<%=ics.GetVar("c")%>'
      editor='fckeditor'
      params="{editorId: '1170996054393',enableEmbeddedLinks:'1'}"/>
</div>
```

Parameters required to enable CKEditor in Web Mode.

Note: If the `insite:edit` tag does not define a height and width for CKEditor, then CKEditor will use the `XSIZE` and `YSIZE` defined in its XML code. For more information see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

You can edit the `insite:edit` tag in one of the following:

- In the **Element Logic** field of the appropriate template. To work in a template's **Element Logic** field:

  a. Log in to WebCenter Sites as an administrator.

  b. Select the site in which you want to work.

  c. Select the **Admin** interface.

  d. Click **Search** (in the button bar), then select **Find Template** , then **Search**, and then *Select template name.*

  e. From the template's Edit form, select the **Element** tab and navigate to the "Element Logic field."

- Sites Explorer (for instructions on editing code with Oracle WebCenter Sites Explorer, see Section 23.8, "Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic").

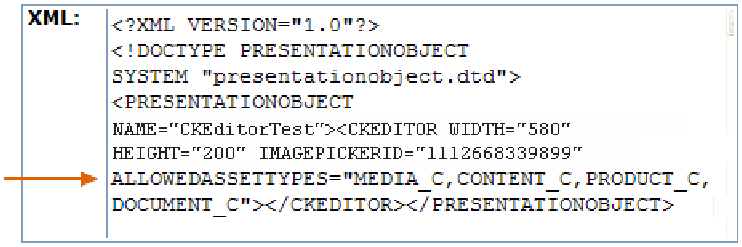
- A text editor of your choice.

19.1.4 Enabling Selected Asset Types for the CKEditor

To narrow down the user's choice of available asset types, add the ALLOWEDASSETTYPES parameter to the CKEditor XML code. As values, specify the names of the allowed asset types in a comma-separated list.

To enable asset types for the CKEditor

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Find and open the attribute editor named CKEditor in its Edit form:
   a. In the button bar, click **Search**.
   b. In the Search form select **Find Attribute Editor**.
   c. In the Search for Attribute Editors form, fill in the desired search criteria (if any) and click **Search**.
   d. In the search results list navigate to the **CKEditor** asset and click its **Edit** (pencil) icon.
5. Navigate to the XML field, add the ALLOWEDASSETTYPES parameter to the <CKEDITOR> tag. Then, specify the names of the asset types you want to enable in the value of the ALLOWEDASSETTYPES parameter.



```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT
SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT
NAME="CKEditorTest"><CKEDITOR WIDTH="580"
HEIGHT="200" IMAGEPICKERID="1112668339899"
ALLOWEDASSETTYPES="MEDIA_C,CONTENT_C,PRODUCT_C,
DOCUMENT_C"></CKEDITOR></PRESENTATIONOBJECT>
```

ALLOWEDASSETTYPES parameter is added as an additional parameter to the <CKEDITOR> tag.

The value of this parameter specifies the asset types which the user can crreate a new asset with.

6. Click **Save Changes** to save the asset.
7. Test the ALLOWEDASSETTYPES parameter:
   a. Switch to the Contributor interface.
   b. Find and open an asset with a CKEditor enabled field in Web Mode:
      a. In the **Search** field, specify the desired search criteria and then click the **magnifying glass** button.

         A "Search" tab opens displaying the results of your search.
      b. Click the name of the desired asset to open its Inspect view.
      c. If the asset opens in Form Mode, click the **Mode** switch in the asset's toolbar to switch to Web Mode.
      d. In the asset's toolbar, click the **Edit** icon.
   c. Search on one of the allowed asset types. After the allowed asset displays in the docked search results list, drag the allowed asset to CKEditor.

If you do not have any allowed assets of the type you specified with the ALLOWEDASSETTYPES parameter, select **Content** and then **New** to create a new asset. Then search on the asset and drag it from the dock to the CKEditor.

19.1.5 Setting the Approval Dependency for Included Assets

The **Include asset** and **Create and include a new asset** icons allow users to include one asset in another asset's CKEditor-enabled field. The included asset is then previewable in the field and, ultimately, embedded in the display of the main asset online.

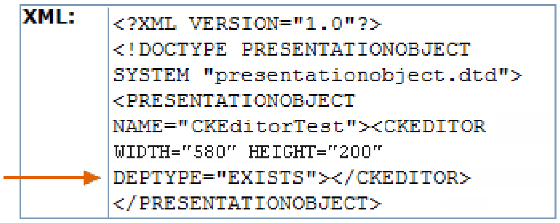
After the main and included assets have been published for the first time, the dependency between them determines how subsequent approvals and publications will work. This dependency is defined by the DEPTYPE parameter in the XML code of the CKEditor.

The DEPTYPE parameter can be set to either EXISTS or EXACT. If the DEPTYPE parameter is not set explicitly, EXACT is used by default.

- EXISTS: When the main asset is edited, approved, and re-published, the included asset does not need to be approved and re-published as long as a version of the asset already exists at the destination.
- EXACT: When the main asset is edited, approved, and re-published, the included asset, if it was edited, must be approved and re-published as well.

To define the approval dependency for included assets

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Find and open the attribute editor named CKEditor in its Edit form:
    a. In the button bar, click **Search**.
    b. In the Search form, select **Find Attribute Editor**.
    c. In the Search for Attribute Editors form, fill in the desired search criteria (if any) and click **Search**.
    d. In the search results list navigate to the **CKEditor** asset and click its **Edit** (pencil) icon.
5. Navigate to the XML field, add the DEPTYPE parameter to the <CKEDITOR> tag, and set the value of the DEPTYPE parameter to either EXISTS or EXACT.



```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT
SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT
NAME="CKEditorTest"><CKEDITOR
WIDTH="580" HEIGHT="200"
DEPTYPE="EXISTS"></CKEDITOR>
</PRESENTATIONOBJECT>
```

DEPTYPE parameter is added as an additional attribute to the <CKEditor> tag.

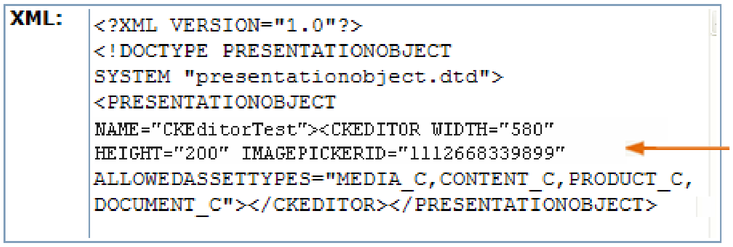
6. Click **Save Changes** to save the asset.

19.1.6 Enable "Image Picker" for the CKEditor

Users might want to invoke an "Image Picker" from a CKEditor enabled field of an asset. On the CKEditor toolbar, the user can click the **Pick an image to include** icon to invoke an "Image Picker." However, by default the icon is disabled. To enable this icon, add the `IMAGEPICKERID` parameter to the CKEditor's XML code, and set its value to the ID of the "Image Picker" you want to be invoked when a user clicks the **Pick an image to include** icon.

To enable an "Image Picker" from CKEditor

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Find and open the attribute editor named CKEditor in its Edit form:
   a. In the button bar, click **Search**.
   b. In the Search form, select **Find Attribute Editor**.
   c. In the Search for Attribute Editors form, fill in the desired search criteria (if any) and click **Search**.
   d. In the search results list navigate to the **CKEditor** asset and click its **Edit** (pencil) icon.
5. Navigate to the XML field, add the `IMAGEPICKERID` parameter to the `<CKEDITOR>` tag, set the value to the ID of the "Image Picker" you want to be invoked when the **Pick an image to include** icon is clicked.



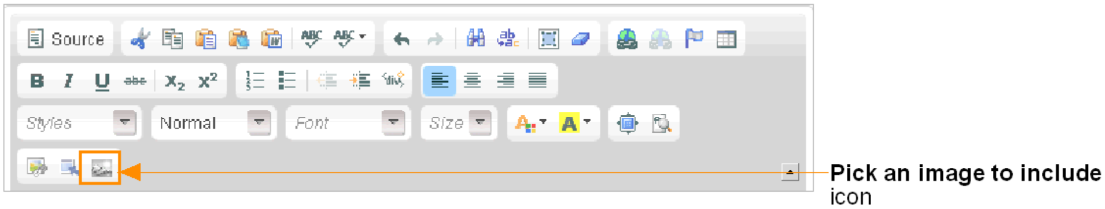
XML:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT
SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT
NAME="CKEditorTest"><CKEDITOR WIDTH="580"
HEIGHT="200" IMAGEPICKERID="1112668339899"
ALLOWEDASSETTYPES="MEDIA_C,CONTENT_C,PRODUCT_C,
DOCUMENT_C"></CKEDITOR></PRESENTATIONOBJECT>
```

IMAGEPICKERID parameter is added as an additional attribute to the <CKEditor> tag.

The value of this parameter is the ID of the Image Picker that will be invoked from the CKEditor.

6. Click **Save Changes** to save the asset.
7. Test the `IMAGEPICKERID` parameter:
   a. Switch to the Contributor interface.
   b. In Form Mode, open the Edit view of the asset you will use to test the `IMAGEPICKERID` parameter. The asset you choose must have a CKEditor enabled field.
   c. In the CKEditor toolbar, click the **Pick an Image to Include** icon.



The "Image Picker," whose ID you set as the value for the IMAGEPICKERID parameter, opens and the image assets associated with that "Image Picker" are rendered in the window.

19.1.7 Customizing the CKEditor Toolbar

You have the ability to customize the functions available in CKEditor's toolbar, as well as their arrangement. You can customize the CKEditor toolbar either system-wide or on a per-instance basis.

19.1.7.1 Customizing the CKEditor Toolbar System-Wide

1. Open the following file in a text editor:

   ```
   <cs_app>/ckeditor/config.js
   ```

   where <cs_app> refers to the directory into which the WebCenter Sites web application was deployed on your application server.

2. The toolbar can be customized for the Admin interface and the Contributor interface (for Form Mode and Web Mode):

   – locate config.toolbar_CS in <cs_app>/ckeditor/config.js for customizing the toolbar for the Admin interface.

   – locate config.toolbar_SITES in <cs_app>/ckeditor/config.js for customizing the toolbar for the Contributor interface.

   – locate config.toolbar_SITES_WEB in <cs_app>/ckeditor/config.js for customizing the toolbar for Web Mode.

3. Make your changes. For information on the toolbar definition syntax, consult the CKEditor documentation.

4. Save and close the file.

5. Redeploy the WebCenter Sites application and restart the application server for your changes to take effect.

19.1.7.2 Customizing the CKEditor Toolbar on a Per-Instance Basis

1. Create the custom toolbar definition:

   a. Open the following file in a text editor:

      ```
      <cs_app>/ckeditor/config.js
      ```

      where <cs_app> refers to the directory into which the WebCenter Sites web application was deployed on your application server.

   b. Add a new toolbar definition at the end of the file. For information on how to build a custom toolbar definition, consult the CKEditor documentation.

      ```
      config.toolbar_<toolbardef> = [
      ['Bold','Italic','Underline'],'/',
      ['Cut','Copy','Paste'],'/',
      ```

```
['FitWindow','-','Preview','Source','-','About']
];
```

The `<toolbarDef>` value is the name of your custom toolbar definition. You will use this name in substep b of step e when modifying the desired CKEditor instance to use this custom definition.

2. Modify the desired **CKEditor** instance to use your custom toolbar definition.

   a. Log in to WebCenter Sites as an administrator.

   b. Select the site in which you want to work.

   c. Select the **Admin** interface.

   d. Find the desired CKEditor instance:

      a. In the button bar, click **Search**.

      b. In the list of asset types, click **Find Attribute Editor**. In the **Search** field, enter the name of the asset holding the desired CKEditor instance and click **Search**.

      c. In the list of search results, navigate to the desired asset and click its **Edit** (pencil) icon.

   e. In the **XML** field, modify the attribute editor code as follows:

      a. Find the line highlighted in bold below:

      ```
      <?XML VERSION="1.0"?>
      <!DOCTYPE PRESENTATIONOBJECT SYSTEM
         "presentationobject.dtd">
      <PRESENTATIONOBJECT NAME="CKEditorCustomized">
      <CKEDITOR WIDTH="400" HEIGHT="200">
      </CKEDITOR></PRESENTATIONOBJECT>
      ```

      b. Add the `TOOLBAR` parameter to the <CKEDITOR> tag:

      ```
      TOOLBAR="<toolbarDef>"
      ```

      The value of the `TOOLBAR` parameter specifies the name of the custom toolbar definition you created in step 1.

      The modified line should look as follows:

      ```
      <CKEDITOR WIDTH="400" HEIGHT="200" TOOLBAR="<toolbarDef>">
      ```

   f. Click **Save Changes** to save the asset.

3. Redeploy the WebCenter Sites application and restart the application server for your changes to take effect.

For more information on configuring CKEditor, consult its documentation.

19.1.8 Configuring Spell Check Support in CKEditor

CKEditor supports integration with the following spell check software:

- ieSpell: A client-side spell checker plugin for Microsoft Internet Explorer
- Spell Pages: A server-side open-source spell checker application

For instructions on configuring CKEditor to support these spell checkers, visit the following URL:

http://docs.cksource.com/

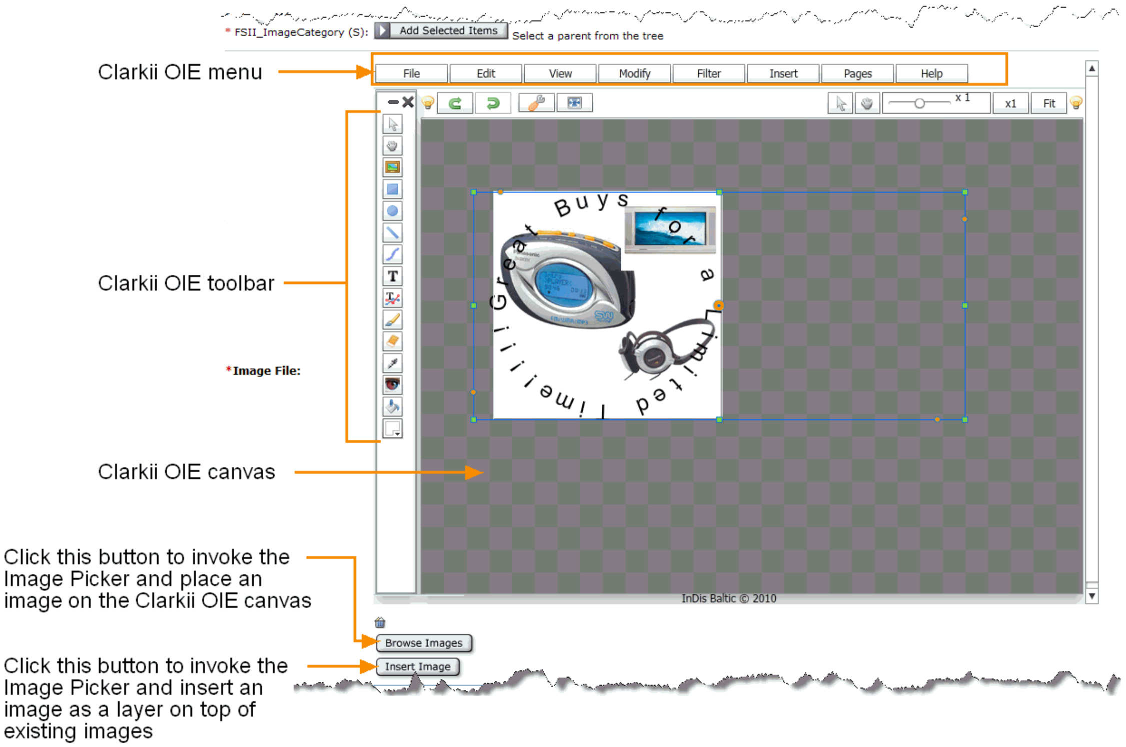
19.2 Configuring the Clarkii Online Image Editor

The Clarkii Online Image Editor (Clarkii OIE) is a third-party image editor from InDis Baltic that is supported on all browsers on which WebCenter Sites is supported, including Safari.

If you will be using the FirstSiteII sample site to create a Clarkii OIE instance, you can modify the code for the Online Image Editor (OIE) that ships bundled with WebCenter Sites by adding the tag `EDITORTYPE="clarkii"`. If you are using a site other than the FirstSiteII sample site, you can either configure separate instances of Clarkii OIE on a per-field basis for each asset type, or you can configure a single instance of Clarkii OIE to be associated with the fields of multiple asset types.

Figure 19–1 summarizes the native controls of Clarkii OIE and functions provided by WebCenter Sites for operating on images in an attribute for which Clarkii OIE is enabled.

Figure 19–1 Clarkii Online Image Editor rendered in a field of an asset's form



Clarkii OIE menu

Clarkii OIE toolbar

Clarkii OIE canvas

Click this button to invoke the Image Picker and place an image on the Clarkii OIE canvas

Click this button to invoke the Image Picker and insert an image as a layer on top of existing images

Before configuring and enabling this feature, take note of the following:

- Clarkii OIE can be enabled only for flex attributes. The instructions in this section use the "Media" flex family of the FirstSiteII sample site as an example.
- Clarkii OIE can be enabled only for attributes of type `blob`.
- Flash must be installed on the client browser in order for Clarkii OIE to be rendered in the field for which you enabled it.

Note: You can customize the functions in the Clarkii OIE toolbar and menu. Since these functions are strictly Clarkii OIE related they are not documented. For instructions about configuring Clarkii OIE specific functions, visit the following URL:

`http://www.online-image-editor-clarkii.com/`

This section contains the following topics:

- Section 19.2.1, "Creating a Clarkii OIE Instance and Enabling it for a Field"
- Section 19.2.2, "Configuring Clarkii OIE Properties"
- Section 19.2.3, "Implementing a Field Copier Filter to Classify Assets"

19.2.1 Creating a Clarkii OIE Instance and Enabling it for a Field

This procedure shows you how to create a Clarkii OIE instance and enable it for a flex attribute asset. This procedure is based on the FirstSiteII sample site.

In this example, you will enable a new Clarkii OIE instance as the attribute editor for the **FSII_ImageFile** field. **FSII_ImageFile** is an attribute of the sample "Media" asset type.

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Create a Clarkii OIE instance:
   a. In the button bar, click **New**.
   b. In the list of asset types, click **New Attribute Editor**.

      WebCenter Sites displays the New Attribute Editor form.
   c. In the **Name** field, enter a name that uniquely identifies this Clarkii OIE instance (for this example, enter **ClarkiiOIE**).
   d. Paste the following code into the **XML** field.

Note: For detailed information about each parameter, see the table in Section 19.2.2, "Configuring Clarkii OIE Properties."

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT NAME="editor">
<IMAGEEDITOR
    HEIGHT="600"
    WIDTH="800"
    EDITORTYPE="clarkii"
    FITIMAGE="false"
    ENABLEIMAGEPICKER="true"
    ASSETTYPE="Media_C"
    ATTRIBUTE="FSII_ImageFile"
    ATTRIBUTETYPE="Media_A"
    CATEGORYATTRIBUTE=""
    RESTRICTEDCATEGORYLIST=""
    OIEENABLEIMAGEPICKER="true"
```

```
        OIEASSETTYPE="Media_C"
        OIEATTRIBUTE="FSII_ImageFile"
        OIEATTRIBUTETYPE="Media_A"
        OIECATEGORYATTRIBUTE=""
        OIERESTRICTEDCATEGORYLIST="">
    </IMAGEEDITOR>
    </PRESENTATIONOBJECT>
```

e. Click **Save**.

5. Enable the Clarkii OIE instance as the attribute editor for an attribute of a given asset type. For this example, use the Media asset type's **FSII_ImageFile** attribute.

    a. Find and open the desired attribute asset in its Edit form. For this example, find and open the **FSII_ImageFile** attribute.

        a. In the button bar, click **Search**.

        b. In the list of asset types, click the asset type of the desired attribute asset. For this example, click **Find Media Attribute**.

        c. In the search field, enter the name of the attribute asset you wish to modify. For this example, enter **FSII_ImageFile**.

        d. Click **Search**.

        e. In the list of search results, navigate to the desired attribute asset (**FSII_ImageFile**) and click its **Edit** (pencil) icon.

        WebCenter Sites opens the asset in its Edit form.

    b. Set Clarkii OIE as the attribute editor for this attribute asset:

        a. In the Value Type field, make sure `blob` is selected.

        Note: Clarkii OIE requires an attribute value of type `blob`. Once an attribute asset is saved, the value selected for the **Value Type** field cannot be modified.

        b. In the Attribute Editor drop-down list, select the Clarkii OIE instance you created in step 4.

    c. Click **Save Changes**.

6. Test your new Clarkii OIE instance:

    a. Switch to the Contributor interface:

        a. If the applications bar is not already open, open it by clicking the down-arrow icon, at the right of the menu bar.

        b. In the applications bar, select the **Contributor** interface icon.

        The Contributor interface displays.

    b. Find any asset whose definition specifies the attribute you modified in step 5 as a field, and open that asset in its Edit form. For this example, select a Media asset:

        a. In the **Search** field, enter the desired search criteria.

        b. If you wish to narrow down your search to a specific asset type, click the **down-arrow** in the **Search** field to open the Search Type drop-down list.

Click the asset type that is using the field enabled with the Clarkii OIE instance. For this example, select **Find Media**.

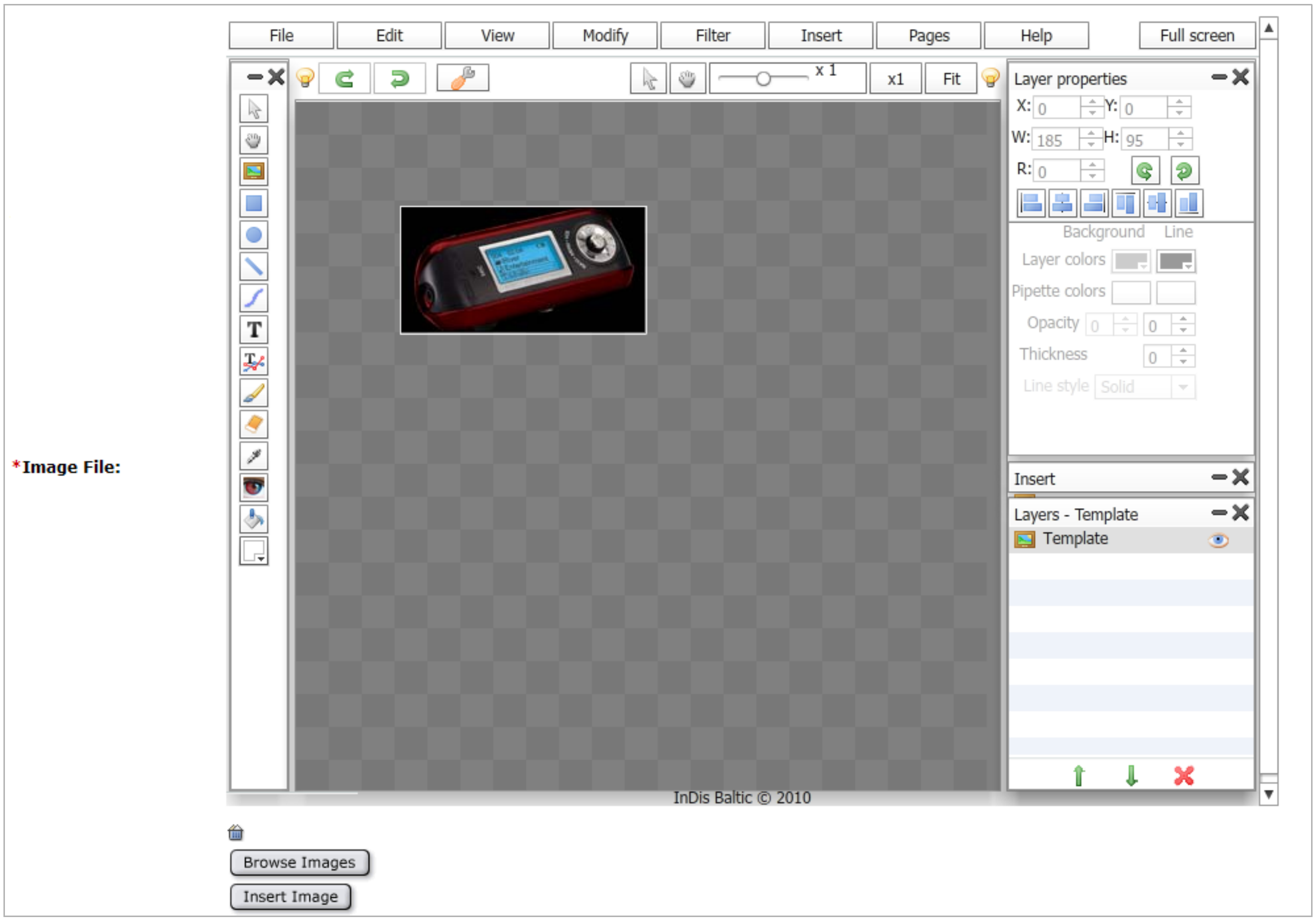
c. Click the **magnifying glass** button.

   A Search tab opens listing the results of your search.

d. In the list of search results, right-click an asset of your choice. For this example, select the **FSII AudioCo_iAC-008.jpg** Media asset and then select **Edit**.

   WebCenter Sites displays the asset in its Edit form.

c. Navigate to the field enabled with Clarkii OIE. For this example, **FSII_ ImageFile** field. It should look similar to the following:



If Clarkii OIE is not rendered in the field, do one of the following:

– Make sure Flash is installed.

– Check the XML code of the Clarkii OIE instance you created (see, substep d of step 4).

– Check the selections you made in the attribute asset for which you enabled Clarkii OIE as the attribute editor (see substep b in step 5 of this procedure).

d. Click **Cancel** to return to the asset's Inspect form.

19.2.2 Configuring Clarkii OIE Properties

The table in this section lists and defines all of the properties that can be specified in the creation of a Clarkii OIE attribute editor asset. Use this table as a reference to configure the properties for your own Clarkii OIE instance so it fits your site design.

Table 19–2 Clarkii OIE Specifications

Property	Definition
HEIGHT	Specify the height of the Clarkii OIE area as it will be displayed within the attribute field of a given asset's form. **Suggested value:** `600`
WIDTH	Specify the width of the Clarkii OIE area as it will be displayed within the attribute field of a given asset's form. **Suggested value:** `800`
EDITORTYPE	Specify the type of image editor you wish to use. To enable Clarkii OIE, the property must read: `EDITORTYPE="clarkii"`
FITIMAGE	If this property is set to `true`, when an image is edited with Clarkii OIE, that image will be resized to fit within the Clarkii OIE canvas. If this property is set to `false`, then when you edit an image with Clarkii OIE, that image is displayed on the canvas in its actual size. **Possible values:** `true|false`

Table 19–3 Browse Images Button Specifications

Property	Definition
ENABLEIMAGEPICKER	Enables the **Browse Images** button which allows users to place an image on the Clarkii OIE canvas, replacing any images that currently exist on the canvas. This button invokes the Image Picker window. **Possible values:** `true|false` **Note:** If this property is set to `false`, do not set values for the other properties related to the **Browse Images** button.
ASSETTYPE	Specify the asset type of the image assets that will be displayed in the Image Picker window when the **Browse Images** button is clicked. **Possible values:** Any asset type that has a definition containing an attribute of type `blob` intended to store images. **Example:** `"Media_C"`
ATTRIBUTE	Specify the image file attribute of the image assets that will be displayed in the Image Picker window when the **Browse Images** button is clicked. **Possible values:** Any attribute of type `blob` intended to store images. **Example:** `"FSII_ImageFile"`

Table 19–3 (Cont.) Browse Images Button Specifications

Property	Definition
ATTRIBUTETYPE	Specify the asset type of the image file attribute specified in the `ATTRIBUTE` property. Image Picker will look for attributes of only this asset type, and displays only the images with attributes of this asset type. **Possible values:** The asset type of the image file attribute specified in the ATTRIBUTE property. **Example:** `"Media_A"`
CATEGORYATTRIBUTE	Specify a `string` attribute holding a value that classifies the image assets (preferably by the names of their parent assets) to be displayed in the Image Picker window. The value of this property populates the Category drop-down list in the Image Picker with all values found for the specified attribute. For information about how a field copier filter can be used to classify assets, see Section 19.2.3, "Implementing a Field Copier Filter to Classify Assets." When the **Browse Images** button is clicked, users can use the Category drop-down list to filter the images that are displayed in the Image Picker window by selecting one of these attribute values. Only the images matching the selected attribute value are rendered in the Image Picker window. **Note:** If no value is specified for this property, all image assets of the asset type specified in the ASSETTYPE property are displayed in the Image Picker window, and the Category drop-down list is not displayed.
RESTRICTEDCATEGORYLIST	In a comma-delimited list, enter specific values of the attribute specified in the `CATEGORYATTRIBUTE` property. The values you specify will be the only values available in the Category drop-down list of the Image Picker window when the **Browse Images** button is clicked. **Possible values:** Refer to the definition for the CATEGORYATTRIBUTE property. **Note:** If no value is specified for the `CATEGORYATTRIBUTE` property, then this property is ignored.

Table 19–4 Insert Image Button Specifications

Property	Definition
OIEATTRIBUTE	Specify the image file attribute of the image assets that will be displayed in the Image Picker window when the **Insert Image** button is clicked. **Possible values:** Any attribute of type `blob` intended to store images. **Example:** `"FSII_ImageFile"`

Table 19–4 (Cont.) Insert Image Button Specifications

Property	Definition
OIEATTRIBUTETYPE	Specify the asset type of the image file attribute you specified in the OIEATTRIBUTE property. Image Picker will look for attributes of only this asset type, and displays only the images with attributes of this asset type. **Possible values:** The asset type of the image file attribute specified in the OIEATTRIBUTE property. **Example:** "Media_A"
OIECATEGORYATTRIBUTE	Specify a string attribute holding a value that classifies the image assets (preferably by the names of their parent assets) that will be displayed in the Image Picker window. The value of this property populates the Category drop-down list in the Image Picker with all values found for the specified attribute. For information about how a field copier filter can be used to classify assets, see Section 19.2.3, "Implementing a Field Copier Filter to Classify Assets." When the **Insert Image** button is clicked, users can use the Category drop-down list to filter the images that are displayed in the Image Picker window by selecting one of these attribute values. Only the images matching the selected attribute value are rendered in the Image Picker window. **Note:** If no value is specified for this property, all image assets of the asset type specified in the OIEASSETTYPE property are displayed in the Image Picker window, and the Category drop-down list is not displayed.
OIERESTRICTEDCATEGORYLIST	In a comma-delimited list, enter specific values of the attribute specified in the OIECATEGORYATTRIBUTE property. The values you specify will be the only values available in the Category drop-down list of the Image Picker window when the **Insert Image** button is clicked. **Possible values:** Refer to the definition for the OIECATEGORYATTRIBUTE property. **Note:** If no value is specified for the OIECATEGORYATTRIBUTE property, then this property is ignored.
OIEENABLEIMAGEPICKER	Enables the **Insert Image** button, which enables users to insert an image as a layer on top of existing images on the Clarkii OIE canvas. This button invokes the Image Picker window. **Possible values:** true | false **Note:** If this property is set to false, do not set values for the other properties related to the **Insert Image** button.

Table 19–4 (Cont.) Insert Image Button Specifications

Property	Definition
OIEASSETTYPE	Specify the asset type of the image assets that will be displayed in the Image Picker window when the **Insert Image** button is clicked. **Possible values:** Any asset type that has a definition containing an attribute of type `blob` intended to store images. **Example:** `"Media_C"`

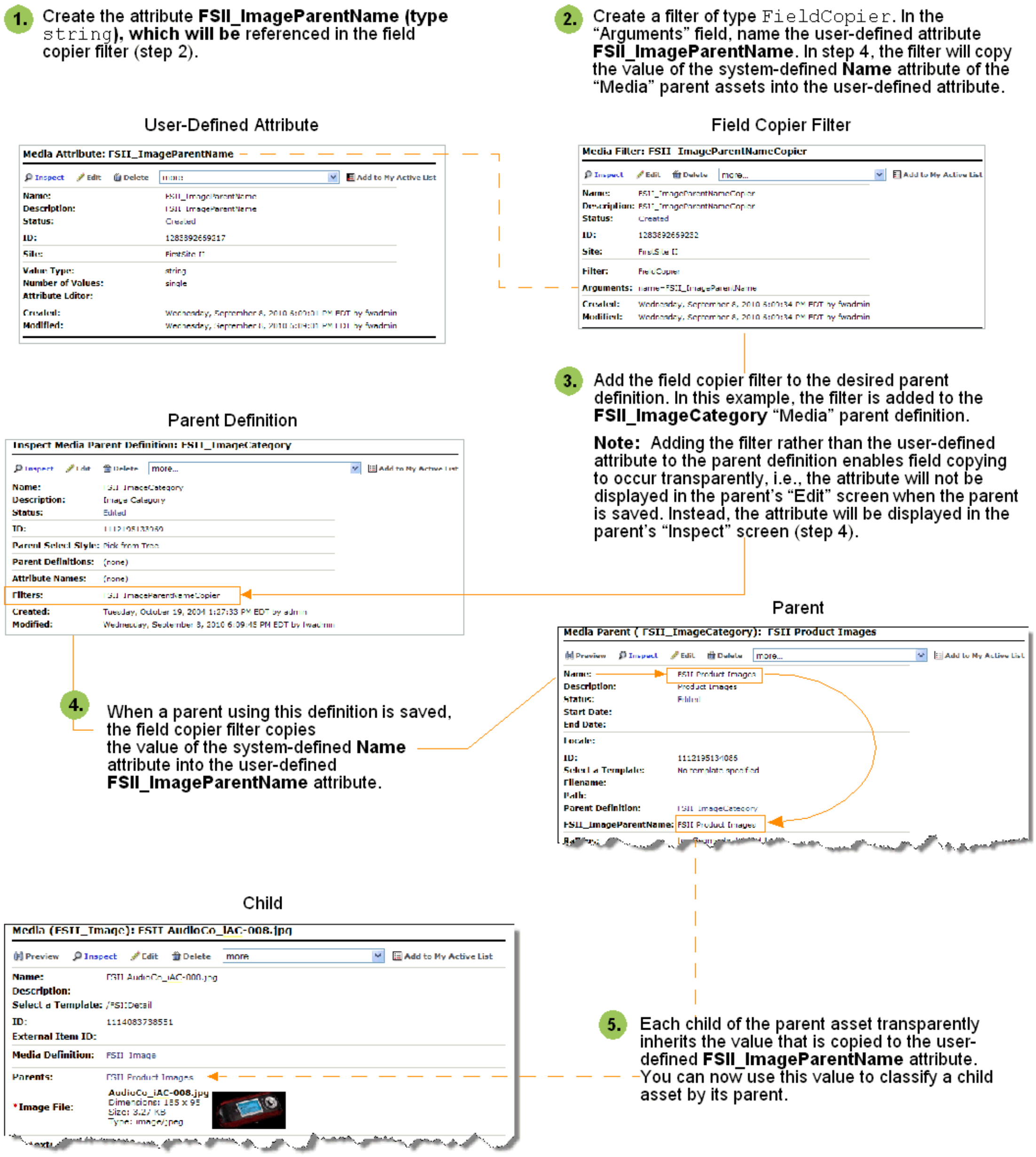
19.2.3 Implementing a Field Copier Filter to Classify Assets

If you wish to classify image assets by the names of their parent assets, you can use a field copier filter. Figure 19–2 illustrates an example of how to implement a field copier filter, using the "Media" flex family of the FirstSiteII sample site. The field copier filter in this example copies the value of the system-defined "Name" attribute of the parent assets into a user defined `string` attribute.

To ensure that the values of preexisting attributes are not overwritten, you can create a new `string` attribute (**FSII_ImageParentName**) to hold the value of the system-defined attribute to be copied by the field copier.

Note: For more information about Field Copier filters, see Section 17.1.1, "Flex Filter Classes."

Figure 19–2 Overview of using a field copier filter (using the "Media" flex family of the FirstSiteII sample site as an example)



1. Create the attribute **FSII_ImageParentName (type** `string`**), which will be** referenced in the field copier filter (step 2).

2. Create a filter of type `FieldCopier`. In the "Arguments" field, name the user-defined attribute **FSII_ImageParentName**. In step 4, the filter will copy the value of the system-defined **Name** attribute of the "Media" parent assets into the user-defined attribute.

User-Defined Attribute

Media Attribute: FSII_ImageParentName

Name: FSII_ImageParentName
Description: FSII_ImageParentName
Status: Created
ID: 1283892659217
Site: FirstSite II
Value Type: string
Number of Values: single
Attribute Editor:
Created: Wednesday, September 8, 2010 6:00:01 PM EDT by fwadmin
Modified: Wednesday, September 8, 2010 6:09:01 PM EDT by fwadmin

Field Copier Filter

Media Filter: FSII_ImageParentNameCopier

Name: FSII_ImageParentNameCopier
Description: FSII_ImageParentNameCopier
Status: Created
ID: 1283892659232
Site: FirstSite II
Filter: FieldCopier
Arguments: name=FSII_ImageParentName
Created: Wednesday, September 8, 2010 6:00:34 PM EDT by fwadmin
Modified: Wednesday, September 8, 2010 6:00:34 PM EDT by fwadmin

3. Add the field copier filter to the desired parent definition. In this example, the filter is added to the **FSII_ImageCategory** "Media" parent definition.

Note: Adding the filter rather than the user-defined attribute to the parent definition enables field copying to occur transparently, i.e., the attribute will not be displayed in the parent's "Edit" screen when the parent is saved. Instead, the attribute will be displayed in the parent's "Inspect" screen (step 4).

Parent Definition

Inspect Media Parent Definition: FSII_ImageCategory

Name: FSII_ImageCategory
Description: Image Category
Status: Edited
ID: 1112195133060
Parent Select Style: Pick from Tree
Parent Definitions: (none)
Attribute Names: (none)
Filters: FSII_ImageParentNameCopier
Created: Tuesday, October 19, 2004 1:27:33 PM EDT by admin
Modified: Wednesday, September 8, 2010 6:09:45 PM EDT by fwadmin

4. When a parent using this definition is saved, the field copier filter copies the value of the system-defined **Name** attribute into the user-defined **FSII_ImageParentName** attribute.

Parent

Media Parent (FSII_ImageCategory): FSII Product Images

Name: FSII Product Images
Description: Product Images
Status: Edited
Start Date:
End Date:
Locale:
ID: 1112195134065
Select a Template: No template specified
Filename:
Path:
Parent Definition: FSII_ImageCategory
FSII_ImageParentName: FSII Product Images

Child

Media (FSII_Image): FSII AudioCo_iAC-008.jpg

Name: FSII AudioCo_iAC-008.jpg
Description:
Select a Template: /FSIIDetail
ID: 1114083738551
External Item ID:
Media Definition: FSII_Image
Parents: FSII Product Images
*Image File: AudioCo_iAC-008.jpg
Dimensions: 155 x 95
Size: 3.27 KB
Type: image/jpeg

5. Each child of the parent asset transparently inherits the value that is copied to the user-defined **FSII_ImageParentName** attribute. You can now use this value to classify a child asset by its parent.

To use a field copier filter

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Determine the system-defined attribute whose value you wish to use as the input for the field copier filter, and the user defined `string` attribute to which the field copier will be copying the system-defined attribute's value.

Note: If you do not wish to overwrite the values of preexisting attributes, create a flex attribute of type `string` (**FSII_ImageParentName** in this example). For instructions on creating a new flex attribute, see Section 16.3.6, "Step 4: Create Flex Attributes."

5. Create a filter of type `FieldCopier` (**FSII_ImageParentNameCopier** in this example):

   a. In the button bar, click **New**.

   b. In the list of asset types, click **New Media Filter**.

   The New Media Filter form is displayed.

   c. In the Name field, enter a unique name for this filter (in this example, we use **FSII_ImageParentNameCopier**).

   d. In the Filter field, select `FieldCopier` from the drop-down list. Then click **Get Arguments**:

   – In the Name drop-down list, select the system-defined attribute whose value you wish to use as the input value for the field copier filter (**name** in this example).

   – In the Value field, type the name of the user defined attribute (**FSII_ImageParentName** in this example) into which the field copier filter will copy the value of the system-defined attribute you specified.

   e. Click **Add** to add the argument to the filter.

   f. Click **Save** to save the filter.

6. Find the desired parent (or child) definition and add the new field copier filter to it (in this example we are adding the field copier filter to the "Media" parent definition **FSII_ImageCategory**):

   a. In the button bar, click **Search**.

   b. In the list of asset types, select **Find Media Parent Definition**.

   c. In the search field, enter the name of the parent definition to which you wish to add the field copier filter (**FSII_ImageCategory**).

   d. Click **Search**.

   e. In the list of search results, navigate to the desired parent (or child) definition and click its **Edit** (pencil) icon.

   f. In the Edit form of the parent (or child) definition, navigate to the **Filters** section and select the field copier filter you created in step 5.

   g. Click **Save Changes**.

7. Find and re-save all preexisting parent (or child) assets associated with the definition to which you added the field copier filter (**FSII_ImageParentNameCopier**). This enables the filter to populate the user defined attribute (**FSII_ImageParentName**) with the value of the system-defined attribute (**Name**).

   If you added the filter to a parent definition, all children of the associated parent assets internally inherit the value that the field copier filter copied into the user defined attribute (**FSII_ImageParentName** in this example).

19.3 Configuring the Image Picker

This section lists the parameters whose values you must specify in the XML definition when configuring an instance of the Image Picker attribute editor:

Table 19–5 Image Picker Parameters

Parameter	Explanation
`ASSETTYPENAME`	Asset type of the image assets that this instance of Image Picker will display. Example: `Media_C`
`ATTRIBUTETYPENAME`	Asset type of the image file attribute within the selected image asset type. Example: `Media_A`
`ATTRIBUTENAME`	Name of the image file attribute within the selected image asset type. Example: `FSII_ImageFile`
`CATEGORYATTRIBUTENAME`	(Optional) Name of the category attribute within the selected image asset type. Example: `FSII_ImageCategory`
`RESTRICTEDCATEGORYLIST`	Accepts a comma-delimited list of values of the category attribute within the selected image asset type. The values in this list will appear in the Category drop-down list in the Image Picker window. If this parameter is omitted, Image Picker will display all assets belonging to the selected image asset type. Example: `Audio,Video,Photo`

Sample XML code for an Image Picker definition is included in Section 19.3.2, "Sample Image Picker Attribute Editor Definition Code." For instructions on creating new attribute editors (such as new instances of Image Picker) see Section 18.2, "Creating Attribute Editors." For instructions on selecting an input method (such as Image Picker) for a field in an asset form, see Section 16.3.6, "Step 4: Create Flex Attributes."

This section contains the following topics:

- Section 19.3.1, "Categorizing Image Assets for Display in Image Picker"
- Section 19.3.2, "Sample Image Picker Attribute Editor Definition Code"

19.3.1 Categorizing Image Assets for Display in Image Picker

Using the `CATEGORYATTRIBUTENAME` and `RESTRICTEDCATEGORYLIST` parameters described in the previous section, you can restrict an instance of Image Picker to display only selected categories of assets belonging to the selected image asset type. Before you do so, the following conditions must be satisfied:

1. You must add a category attribute of type string to the selected image asset type or flex definition that will store the category descriptor for each asset within the selected asset type. Image Picker will use the value of this attribute to generate a list of asset categories which it is allowed display. For instructions on creating attributes, see Section 16.3.6, "Step 4: Create Flex Attributes."
2. You or your content providers must fill in the category field for each asset of the selected image asset type, as appropriate. If an asset is not assigned a category, it will not be displayed by a category-restricted instance of Image Picker.

19.3.2 Sample Image Picker Attribute Editor Definition Code

A sample XML definition for the Image Picker attribute editor is included below for your reference. Use it to get an idea of how to configure Image Picker on your system.

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT>
<PRESENTATIONOBJECT NAME="ImagePicker">
<IMAGEPICKER>
  ASSETTYPENAME="Media_C"
  ATTRIBUTETYPENAME="Media_A"
  ATTRIBUTENAME="FSII_ImageFile"
  CATEGORYATTRIBUTENAME="FSII_ImageCategory"
  RESTRICTEDCATEGORYLIST="Audio,Video">
</IMAGEPICKER>
</PRESENTATIONOBJECT>
```

20

Importing Assets of Any Type

After you have determined your data design, created your asset types, tested them on your development system, and moved them to your management system, the next step is to import assets (content) from their current source in to the database on the management system.

For example, you could be using a wire feed service or some other source of remotely generated content, and need to import that content into the WebCenter Sites database on your management system.

To import any data into the WebCenter Sites database, you can use the XMLPost utility. This utility is based on the WebCenter Sites FormPoster Java class and it is delivered with the WebCenter Sites base product. It imports data using the HTTP POST protocol.

This chapter describes the general process of importing assets with the XMLPost utility. You use the information in this chapter for importing assets of all types. Chapter 21, "Importing Flex Assets,"provides additional information that you need to import your assets when you are using the flex asset data model.

This chapter contains the following sections:

- Section 20.1, "The XMLPost Utility"
- Section 20.2, "XMLPost Configuration Files"
- Section 20.3, "XMLPost Source Files"
- Section 20.4, "Using the XMLPost Utility"
- Section 20.5, "Customizing RemoteContentPost and PreUpdate"
- Section 20.6, "Troubleshooting XMLPost"

20.1 The XMLPost Utility

To import assets, you instruct the XMLPost utility to invoke one of the importing (posting) elements provided by WebCenter Sites, as appropriate for that asset type.

There are four components involved in this process:

- The **XMLPost** utility, which is delivered with WebCenter Sites.
- A **posting element**. WebCenter Sites delivers a posting element named `RemoteContentPost`. WebCenter Sites delivers three additional posting elements, described in Chapter 21, "Importing Flex Assets."
- A **configuration file** with an `.ini` file extension. You create a configuration file for each asset type that you plan to import. This file contains information about what

to expect in the source files (what tags XMLPost will find there), what to do with the data provided, and which importing (posting) element to use to import the data.

- **Source files**. You provide an individual source file for each asset that you want to import (well-formed XML files). Each tag in a file identifies a field for that asset type. The information contained in the tag is the data to be written to that column.

The XMLPost utility parses the configuration file to determine how to interpret the data provided for the asset type, parses the source files and creates name/value pairs for each field value, and passes those name/value pairs as ICS variables to the `RemoteContentPost` element. The `RemoteContentPost` element then creates the asset from the variables.

You can also create your own posting elements that work with the XMLPost utility. However, for importing assets, the posting elements that are provided by WebCenter Sites should meet your needs.

> **Note:** For added security, you can rename the `RemoteContentPost` page to prevent attempts to hack into the system.

What the Developer Does

This section provides a brief overview of the steps that the developer completes before invoking the XMLPost utility and what the XMLPost utility does.

When you import assets into your WebCenter Sites database, you perform four general steps:

1. You create a configuration file that identifies the type of asset that is to be imported and the tags that are used in the source files.

   This file also sets several configuration properties, including the name of the SiteCatalog entry for the posting element that you want XMLPost to use. For all assets, the name of this posting element is RemoteContentPost. For information about the posting elements for flex assets, see Chapter 21, "Importing Flex Assets."

   Note that the configuration file is specific for this asset type. You must provide a separate configuration file for each asset type.

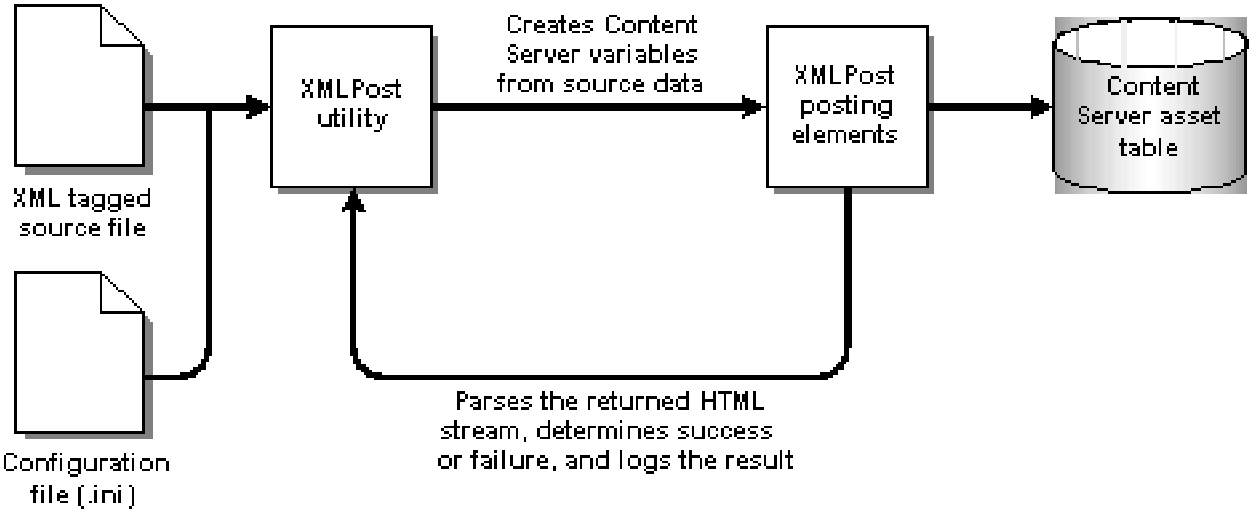
2. You create the source files for the data that you want to import. Note that you create a separate source file for each individual asset.

3. You place the source and configuration files in a directory on the management system.

4. From that directory, you invoke the XMLPost utility, identifying the source files and the configuration file to use for those source files.

What XMLPost and WebCenter Sites Do

After you invoke the XMLPost utility to import the source files, this is what happens next, as shown in the following diagram and list of steps:


XML tagged source file
Configuration file (.ini)
XMLPost utility
Creates Content Server variables from source data
XMLPost posting elements
Content Server asset table
Parses the returned HTML stream, determines success or failure, and logs the result

1. The XMLPost utility parses the configuration file.
2. XMLPost parses the source file and creates name/value pairs for each field value specified in the source file.
3. XMLPost invokes the FormPoster Java class by posting (HTTP POST) the name/value pairs as ICS variables to the page name passed in from the configuration file. When you are importing basic asset types, that pagename is:

   ```
   OpenMarket/Xcelerate/Actions/RemoteContentPost
   ```
4. WebCenter Sites locates the page in the `SiteCatalog` table and invokes the root element of the `RemoteContentPost` page, which has the same name by default (`RemoteContentPost`).
5. The `RemoteContentPost` element passes the data from the source files as variables to the `PreUpdate` element for assets of that type.
6. The `PreUpdate` element for assets of that type sets the variable values for that asset and then returns to the `RemoteContentPost` element.
7. The `RemoteContentPost` element creates the asset.
8. The web server returns a stream of HTML to XMLPost, which then parses the stream to determine whether the import operation succeeded or failed, logging the results to a text file that you specify in the configuration file.
9. If the asset type of the asset that you are importing uses a search engine, `RemoteContentPost` indexes the new element.
10. If you set a certain parameter in the configuration file, `RemoteContentPost` deletes the source files for the assets that were successfully imported.

20.2 XMLPost Configuration Files

There are three types of properties in a configuration file for XMLPost:

- Properties that provide information to XMLPost about the database and environment. These properties remain the same even if you create your own posting element.
- Properties that provide configuration values for the posting (importing) process. This chapter describes the properties that you must provide for the `RemoteContentPost` element to function correctly.

Examples of properties include the URL of the page that invokes RemoteContentPost, a user name and password that gives XMLPost write privileges to the asset type table in the database, the name of the asset type that you want to import, how to log errors, and any data values that are the same for all of the assets that you are importing.

- Properties that specify the tags that are used in the source files.

Certain information, such as which site the assets should belong to or which workflow should be assigned to the asset, can be configured either in the RemoteContentPost section of the configuration file or the source file section.

For example, if you have only one content management site or if all of the assets that you are importing belong to the same site, specify the name of the site in the configuration section so you do not have to repeat that information in each source file. If your system has more than one content management site, specify which sites an asset belongs to in the individual source files.

This section contains the following topics:

- Section 20.2.1, "Configuration Properties for XMLPost"
- Section 20.2.2, "Configuration Properties for the Posting Element"
- Section 20.2.3, "Configuration Properties for the Source Files"
- Section 20.2.4, "Sample XMLPost Configuration File"

20.2.1 Configuration Properties for XMLPost

The following table lists the properties that specify database connection information and other general configuration instructions that the XMLPost utility needs:

Table 20–1 Configuration Properties for XMLPost

Property	Description
`xmlpost.xmlfilenamefilter`	Required. The file extension for your source files. Typically set to `xml`. For example: `xmlpost.xmlfilenamefilter: .xml`
`xmlpost.proxyhost`	Optional. If a firewall separates you and the WebCenter Sites database that you want to import the assets in to, use this property to specify the host name of the proxy server. For example: `xmlpost.proxyhost:` nameOfServer
`xmlpost.proxyport`	Optional. If a firewall separates you and the WebCenter Sites database that you want to import the assets in to, use this property to specify the port number on the proxy server that XMLPost should connect to. For example: `xmlpost.proxyport: 80`

Table 20–1 (Cont.) Configuration Properties for XMLPost

Property	Description

`xmlpost.url`	Required.
	The first part of the URL for the page entry of the posting element.
	XMLPost creates the URL for the posting element by prepending the value specified for this property to the value specified for the pagename `postargname` (described below).
	The value that you set for this property should use the following convention:
	■ The name of the server that holds the WebCenter Sites database.
	■ The CGI path appropriate for the application server software installed on the server. For WebLogic and WebSphere this path is `/servlet/`.
	■ The name of the ContentServer servlet.
	For example:
	`xmlpost.url:`
	`http://`servername`/servlet/ContentServer`

`xmlpost.logfile`	Optional. The name of the file to log the results of importing (posting) each source file. Each source file is posted to the WebCenter Sites database through a post request. When the post request returns from the web server, XMLPost parses the HTML stream that the web server returned, searching for the `postsuccess` and `postfailure` parameters. XMLPost then writes the result to the file that you name identify with this parameter. For example: `xmlpost.logfile: ArticlePost.txt`
`xmlpost.success`	Optional. The string to look for in the response to determine if the post was a success. For example: `xmlpost.success: Success!`
`xmlpost.failure`	Optional. The string to look for in the response to determine if the post was a failure. For example: `xmlpost.failure: Error`

Table 20–1 (Cont.) Configuration Properties for XMLPost

Property	Description
`xmlpost.deletefile`	Optional. Whether to delete the source files after they have been successfully imported into the WebCenter Sites database. Valid settings are `y` (yes) or `n` (no). By default, the source files are not deleted. For example: `xmlpost.deletefile: y`

20.2.2 Configuration Properties for the Posting Element

The following table lists the arguments that specify information that must be posted to the `RemoteContentPost` page (and passed to the `RemoteContentPost` element). The values of these arguments are concatenated into the URL that is posted to the `RemoteContentPost` page; they can be in any order in the configuration file:

Table 20–2 Configuration Properties for the Posting Element

Property	Description
`xmlpost.numargs`	Required. There are several required variables that the configuration file passes to XMLPost as name/value pairs attached to the URL, the primary of which is the page name. Use this property (`xmlpost.numargs`) to tell XMLPost how many variables the configuration file is passing in. For example: `xmlpost.numargs: 7` Note that you can also specify your own custom variables with these name/value pairs.
`xmlpost.argname1: pagename`	Required. The pagename for the `RemoteContentPost` element. Typically the `pagename` argument is specified as `xmlpost.argname1`. For example: `xmlpost.argname1: pagenamexmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost`

`xmlpost.argname2: AssetType`	Required.
	The asset type of the assets that are defined in the source files. Typically, `AssetType` is specified as `xmlpost.argname2`.
	For example:
	`xmlpost.argname2: AssetType`
	`xmlpost.argvalue2: Collection`
	Note that the value for the `AssetType` argument must exactly match the table name of the table that holds assets of this type.

Table 20–2 (Cont.) Configuration Properties for the Posting Element

Property	Description

`xmlpost.argname3: authusername`	Required.
	The user name that you want XMLPost to use to log in to the WebCenter Sites database that you are importing the assets into. Typically, `authusername` is specified as `xmlpost.argname3`.
	For example:
	`xmlpost.argname3: authusername`
	`xmlpost.argvalue3: editor`
	The user name that you specify must have permission to write to the table that holds assets of the type that you are importing. (That is, it must have the appropriate ACLs assigned to it.)

`xmlpost.argname4: authpassword`	Required.
	The password for the user that XMLPost logs in as to the WebCenter Sites database that you are importing the assets into. Typically, `authpassword` is specified as `xmlpost.argname4`.
	For example:
	`xmlpost.argname4: authpassword`
	`xmlpost.argvalue4: xceleditor`

`xmlpost.argname5: xmlpostdebug`	Optional
	Whether or not to include debugging information with the results information that is written to the XMLPost log file identified with the `xmlpost.logfile` property.
	You can set this property to any value. For example:
	`xmlpost.argname5: xmlpostdebug`
	`xmlpost.argvalue5: on`
	Note: Be sure to include a value for the `xmlpost.logfile` property if you enable debugging.

`xmlpost.argname6: inifile`	Optional.
	The name of the `ini` file to use when connecting to the WebCenter Sites database. Typically, `inifile` is specified as `xmlpost.argname5.`
	For example:
	`xmlpost.argname6: inifile`
	`xmlpost.argvalue6: futuretense.ini`

Table 20–2 (Cont.) Configuration Properties for the Posting Element

Property	Description
`xmlpost.argname7: publication`	Optional. Athough using this property is optional, you must specify a site for each asset that you are importing. If your system uses one content management site (publication) or if all assets of this type should be enabled on the same site, use this argument to set the name of the site. For example: `xmlpost.argname7: publicationxmlpost.argvalue7: Burlington Financial` If your system uses more than one content management site, you must specify the value for site for each asset in the individual source files. See Section 20.2.3, "Configuration Properties for the Source Files" for more information.

`xmlpost.argname8:startmenu`	Optional.
	If you are using workflow and you want the same workflow assigned to all of the assets that you are importing, use this argument to set the Start Menu shortcut for the assets. (It is a Start Menu shortcut that assigns a workflow ID to a new asset.)
	For example:
	`xmlpost.argname8: startmenu`
	`xmlpost.argvalue8: New Article`
	If you have more than one workflow for assets of this type, you must specify the value for the Start Menu shortcut for each asset in the individual source files. See Section 20.2.3, "Configuration Properties for the Source Files" for more information.

20.2.3 Configuration Properties for the Source Files

The source file section in a configuration file specifies which tags are used in the source files.

A tag represents a column name in the table that holds assets of this type. The content between a pair of tags is the information that is to be written to that column. Configuration files must list a tag for each column in the asset type's primary storage table, which is why you must provide a separate configuration file for each asset type.

This section contains the following topics:

- Section 20.2.3.1, "Site Properties"
- Section 20.2.3.2, "Asset Type Properties"

20.2.3.1 Site Properties

In addition to the tags that pertain to your asset types, there are four tags that you can use with all asset types to specify certain site configuration properties, that is, which sites an asset should be associated with and which workflow it should use. This table lists the site tags:

Table 20–3 Site Properties

Site tag property	Value	Description
`postpublication`	`y` or `n` (yes or no)	Optional. Specifies that a source file will provide a site name that identifies which site the asset belongs to. For example: postpublication: y Note that a site (publication) value provided in a source file with the `publication` tag overrides the value specified for a publication argument in the XMLPost section of the configuration file.
`postprimarypubid`	`y` or `n` (yes or no)	Optional. Specifies that a source file will provide a value for pubid (a unique ID for the site) that identifies which site the asset belongs to. For example: postprimarypubid: y
`postpublist`	`y` or `n` (yes or no)	Optional. Specifies that a source file will provide a list of sites that the asset is shared with. For example: postpublist: y
`poststartmenu`	`y` or `n` (yes or no)	Optional. Specifies that a source file will provide a value for the Start Menu short cut that places the asset into a workflow process. For example: poststartmenu: y

Remember that if the site or the workflow is the same for all of the assets that you are importing, you can specify the value for site or workflow as an argument in the XMLPost section of the configuration file. That way, you do not have to duplicate the same information in all of the source files.

20.2.3.2 Asset Type Properties

To set up the tags that are specific to your asset types, you specify a tag for each column in the database table for assets of that type. However, the source files are not required to include data tagged with every tag in the configuration file. (Of course, they must include data for required fields.)

For each tag representing a field (column), you specify the name of the tag and optionally some additional processing properties for the tag. The name of the tag is the name of the field (column). For the additional properties, the convention is a word prepended to the name of the tag.

The following table describes how to specify the tags that are specific to your asset types:

Table 20–4 Asset Type Properties

Tag property	Value	Description
`post`*`tagname`*	`y` or `n` (yes or no)	Required. Specifies the name of the tag. The name should exactly match the name of the field that it represents. For example, the tag property for a name field is: `postname: y`
`trunc`*`tagname`*	*`N`* (integer)	Optional. Whether to truncate the data in the source file marked by this tag. For example: `truncname: 64` By setting this property for the tag, if XMLPost finds a string in the `<name>` tag that exceeds 64 characters, it shortens it to 64 characters and stores the truncated string in the variable.
`notrim`*`tagname`*	`y` or `n` (yes or no)	Optional. Whether to trim the white space at the beginning or end of the tag. If you do not want the white space trimmed, set this property to `y` (yes). For example: `notrimname: y` If you do not specify this property, XMLPost trims the white space for the tag by default.

Table 20–4 (Cont.) Asset Type Properties

Tag property	Value	Description
multi*tagname*	combine or separate	Required if the same tag is used more than once in a single source file. Determines how many variables to use for the data when a tag is used more than once in the source file. If you set it to combine, the data from all of the tags is stored in the same variable with commas separating each value (a comma delimited string). If you set it to separate, the data from each tag is stored in a separate variable. Those variables are identified by appending the value that you set for seed*tagname* to the variable name. For example, for a keyword field (column): ■ If you set multikeyword: combine, XMLPost stores all the values marked by a keyword tag to the same keyword variable. ■ If you set multikeyword: separate and seedkeyword: 1, XMLPost stores each value in a separate variable. The first value it finds is stored in a variable named keyword1. The second value is stored in a variable named keyword2, and so on.
seed*tagname*	*seed value*	Required when multi*tagname* is set to separate. The number to start at when XMLPost increments the suffix assigned to variable names when a tag is used more than once and you do not want the data contained in those tags written to the same variable. See the description of multi*tagname*. For example: multikeyword: separate seedkeyword: 1

Table 20–4 (Cont.) Asset Type Properties

Tag property	Value	Description

`file`*`tagname`*	`y` or `n`
	(yes or no)
		Required if the tag represents an upload field (a URL column or BLOB).
		If the tag represents a field that has a URL column, you must include this property and the source file must specify the name of the file that `RemoteContentPost` is to upload to that column.
		For example, the imagefile asset type from the Burlington Financial sample site has an upload field named `urlpicture`. A configuration file for the imagefile asset type would have the following propeties:
		`posturlpicture: y`
		`fileurlpicture: y`
		Then, in the source file for an imagefile asset, you specify the value for the `urlpicture` field like this:
		`<urlpicture>`*`relative_path_to/filename.jpg`*
		`</urlpicture>`
		Note that you must specify the location of the file with a relative path (relative to the directory in which you are running the XMLPost utility).

20.2.4 Sample XMLPost Configuration File

Here is a sample configuration file named `imagefile.ini`, used to import imagefile assets for the Burlington Financial sample site.

```
xmlpost.xmlfilenamefilter: xml
#xmlpost.xmlproxypost: Future
#xmlpost.xmlproxyport: 80
xmlpost.url: http://localhost/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: ImageFile
xmlpost.argname3: authusername
xmlpost.argvalue3: user_author
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: publication
xmlpost.argvalue6: BurlingtonFinancial
xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: ImageFilePost.txt

postpublication: y
postprimarypubid: y
postpublist: y

postcategory: y
truncategory: 4

postpath: y
truncpath: 255

postname: y
truncname: 32

posttemplate: y
trunctemplate: 32

postsubtype: y
truncsubtype: 24

postfilename: y
truncfilename: 64

poststartdate: y

postdescription: y
truncdescription: 128

postsource: y

posturlpicture: y
fileurlpicture: y

posturlthumbnail: y
fileurlthumbnail: y

postmimetype: y
postwidth: y
postheight: y
postalign: y
postalttext: y

postkeywords: y
multikeywords: combine
trunckeywords: 128

postimagedate: y
```

20.3 XMLPost Source Files

Source files must be made up of well-formed XML without the need for a document type definition (DTD) file. Actually, the configuration file functions something like a DTD file in that it defines the tags that will be processed in the source files.

The data in your source files must be tagged with tags whose names match the column names for the table that holds assets of that type. For example, a source file for a

Burlington Financial imagefile asset uses tags named `name`, `caption`, `picutureurl`, and so on.

This chapter does not describe how to automate the generation of your XML source files. How you create your source files depends on the source of your data and the tools that you have to convert your data into XML files. This chapter describes what needs to be in your source files and what XMLPost does with them.

This section contains the following topics:

- Section 20.3.1, "Sample XMLPost Source File"
- Section 20.3.2, "XMLPost and File Encoding"

20.3.1 Sample XMLPost Source File

Here is a sample source file for a Burlington Financial imagefile asset. Its tags are defined in the sample configuration file in Section 20.2.4, "Sample XMLPost Configuration File."

```
<document>
<name>High Five 25</name>
<keyword>Five</keyword>
<category>a</category>
<artist>by Ann. Artist</artist>
<alttext>Congratulations</alttext>
<align>CENTER</align>
<caption>A man extends <keyword>congratulations</keyword> with a boy.</caption>
<pictureurl>/images/eZine/highfive.jpg</pictureurl>
</document>
```

How the Data is Passed (Posted)

All of the text contained between a pair of XML tags in a source file is passed to the `RemoteContentPost` element from XMLPost as a variable that uses the `Variables.`tagname syntax convention.

For example, this line of code:

```
<name>High Five 25</name>
```

is sent to `RemoteContentPost` as `Variables.name` and the value of `name` is the string "High Five".

20.3.2 XMLPost and File Encoding

If the data in a source file does not use the WebCenter Sites system's default file encoding but the database can accommodate that character set, you can specify the alternate file encoding in the XML version statement at the beginning of the file.

For example:

```
<?xml version= "1.0" encoding="UTF-8" ?>
```

20.4 Using the XMLPost Utility

You can invoke the XMLPost utility in one of many ways:

- From the command line

- From a script or batch file
- From a program

No matter how you start XMLPost, you must provide the following pieces of information:

- The name of the configuration file to use
- The source files, which can be specified as a single file, a list of files, or a directory of files

This section contains the following topics:

- Section 20.4.1, "Before You Begin"
- Section 20.4.2, "Running XMLPost from the Command Line"
- Section 20.4.3, "Options for Identifying Source Files"
- Section 20.4.4, "Running XMLPost as a Batch Process"
- Section 20.4.5, "Running XMLPost Programmatically"

20.4.1 Before You Begin

- Before you can use the XMLPost utility, the following must be true:
  - Your asset types are created. (Otherwise, there are no database tables to import the assets into.)
  - Your content management sites are created and the appropriate asset types are enabled for each site.
  - If you are using workflow, your workflow processes are created.
  - Your Start Menu shortcuts are created and, if you are using workflow, they assign the appropriate workflow process to the appropriate asset types.
  - The templates for the asset type are created.
  - The association fields for the asset types are created. However, to use XMLPost to set the value of an association field requires custom code. See Section 20.5, "Customizing RemoteContentPost and PreUpdate" for more information.
- When invoking XMLPost, include the following command before the classpath to ensure UTF-8 encoding: **-Dfile.encoding=UTF-8**

> **Note:** Check all `.jar` files for version number, which can differ from one WebCenter Sites patch to the next.

20.4.2 Running XMLPost from the Command Line

Complete the following steps:

1. Place the configuration file and source files in a directory on a system that has WebCenter Sites installed.
2. Run the following command (on a single command line) from that directory.

   This example uses Windows syntax, for Unix-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

   ```
   java -Xmx512m -Dfile.encoding=UTF-8 -classpath
   <cs_app_dir>\WEB-INF\lib\apache-mime4j-0.5.jar;
   ```

```
<cs_app_dir>\WEB-INF\lib\commons-lang-2.4.jar;
<cs_app_dir>\WEB-INF\lib\commons-codec-1.4.jar;
<cs_app_dir>\WEB-INF\lib\commons-logging-1.1.1.jar;
<cs_app_dir>\WEB-INF\lib\cs.jar;
<cs_app_dir>\WEB-INF\lib\cs-core.jar;
<cs_app_dir>\WEB-INF\lib\httpclient-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\httpcore-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\httpmime-4.1.2.jar;<
cs_app_dir>\WEB-INF\lib\MSXML.jar;
<cs_app_dir>\WEB-INF\lib\servlet-api.jar;
<cs_app_dir>\WEB-INF\lib\spring-2.5.5.jar;
<cs_app_dir>\WEB-INF\lib\sites-security-11.1.1.8.0.jar;
<cs_app_dir>\WEB-INF\lib\esapi-2.0.1.jar;
<cs_app_dir>\WEB-INF\lib\log4j-1.2.16.jar;<j2eeSDK_dir>\j2ee.jar;
<cs_app_dir>\WEB-INF\classes
 COM.FutureTense.XML.Post.XMLPostMain-sSourcefile.xml -cConfigfile.ini
```

where:

- `-Xmx512m` sets the maximum memory to use, which is needed with larger inserts
- `<cs_install_dir>` is the directory where WebCenter Sites is installed.
- `<cs_app_dir>` is the directory on your application server where the WebCenter Sites application has been deployed.
- `<j2eeSDK_dir>` is the directory where your J2EE SDK is installed.

Note that there are several options for designating the source file. See Section 20.4.3, "Options for Identifying Source Files" for information.

Note: The `j2ee.jar` file is part of the J2EE SDK. You must install the SDK before running XMLPost.

If the source files and configuration file are not in the directory that you are working in, you must provide the path to those files in the command line. For example: `-s/products/product.xml`.

20.4.3 Options for Identifying Source Files

The source parameter that you use to identify the source files to the XMLPost utility can point to any of the following:

- A single file.
- A directory of files. All the files in that directory that have the file extension (typically `.xml`) designated by the configuration file will be posted (imported).
- A list file that provides a list of all the files that you want to import. It is similar to an `.ini` file but it has a file extension of `.lst`.

A Single File

To post the contents of one file, specify the name of that file in the command line. The following example instructs XMLPost to use a configuration file named `articlepost.ini` and one source file named `article.xml`.

This example uses Windows syntax, for Unix-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

```
java -Xmx512m -Dfile.encoding=UTF-8 -classpath<
```

```
cs_app_dir>\WEB-INF\lib\apache-mime4j-0.5.jar;
<cs_app_dir>\WEB-INF\lib\commons-lang-2.4.jar;
<cs_app_dir>\WEB-INF\lib\commons-codec-1.4.jar;
<cs_app_dir>\WEB-INF\lib\commons-logging-1.1.1.jar;
<cs_app_dir>\WEB-INF\lib\cs.jar;
<cs_app_dir>\WEB-INF\lib\cs-core.jar;
<cs_app_dir>\WEB-INF\lib\httpclient-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\httpcore-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\httpmime-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\MSXML.jar;
<cs_app_dir>\WEB-INF\lib\servlet-api.jar;
<cs_app_dir>\WEB-INF\lib\spring-2.5.5.jar;
<cs_app_dir>\WEB-INF\lib\sites-security-11.1.1.8.0.jar;
<cs_app_dir>\WEB-INF\lib\esapi-2.0.1.jar;
<cs_app_dir>\WEB-INF\lib\log4j-1.2.16.jar;<j2eeSDK_dir>\j2ee.jar;
<cs_app_dir>\WEB-INF\classes
 COM.FutureTense.XML.Post.XMLPostMain-sarticle.xml -carticlepost.ini
```

A Directory of Files

To post all the files in a directory, specify the path to that directory in the command line. The following example instructs XMLPost to import the files in the `xmlpostfiles` directory.

This example uses Windows syntax, for Unix-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

```
java -Xmx512m -Dfile.encoding=UTF-8 -classpath
<cs_app_dir>\WEB-INF\lib\apache-mime4j-0.5.jar;
<cs_app_dir>\WEB-INF\lib\commons-lang-2.4.jar;
<cs_app_dir>\WEB-INF\lib\commons-codec-1.4.jar;
<cs_app_dir>\WEB-INF\lib\commons-logging-1.1.1.jar;
<cs_app_dir>\WEB-INF\lib\cs.jar;
<cs_app_dir>\WEB-INF\lib\cs-core.jar;
<cs_app_dir>\WEB-INF\lib\httpclient-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\httpcore-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\httpmime-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\MSXML.jar;
<cs_app_dir>\WEB-INF\lib\servlet-api.jar;
<cs_app_dir>\WEB-INF\lib\spring-2.5.5.jar;
<cs_app_dir>\WEB-INF\lib\sites-security-11.1.1.8.0.jar;
<cs_app_dir>\WEB-INF\lib\esapi-2.0.1.jar;
<cs_app_dir>\WEB-INF\lib\log4j-1.2.16.jar;<j2eeSDK_dir>\j2ee.jar;
<cs_app_dir>\WEB-INF\classes
 COM.FutureTense.XML.Post.XMLPostMain-sxmlpostfiles -carticlepost.ini
```

A List File

As an alternative to specifying a directory, you can create a list file that uses the format of an `.ini` file and includes the following properties:

- `numfiles`, which specifies how many files are included in the list.
- `file`N, which specifies the path to a file and its file name. The N stands for the file's order in the list file. The first file listed is `file1`, the second is `file2`, and so on.

  The value of N for the last `file`N in the list must match the value specified by the `numfiles` property. XMLPost stops importing when it has imported as many files as it is told to expect by the numfiles property. If you have included more files than numfiles states, XMLPost does not import them.

The file extension for a list file must be .lst.

Here is an example list file, named xmlpostfiles.lst:

```
numfiles: 3
file1: c:\xmlpost\article1.xml
file2: c:\xmlpost\article2.xml
file3: c:\xmlpost\article3.xml
```

To post the files referenced in this file list, specify the name of the list file in the command line. The following example instructs XMLPost to import the files specified in the xmlpostfiles.lst file.

This example uses Windows syntax, for Unix-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

```
java -Xmx512m -Dfile.encoding=UTF-8 -classpath<
cs_app_dir>\WEB-INF\lib\apache-mime4j-0.5.jar;
<cs_app_dir>\WEB-INF\lib\commons-lang-2.4.jar;
<cs_app_dir>\WEB-INF\lib\commons-codec-1.4.jar;
<cs_app_dir>\WEB-INF\lib\commons-logging-1.1.1.jar;
<cs_app_dir>\WEB-INF\lib\cs.jar;
<cs_app_dir>\WEB-INF\lib\cs-core.jar;
<cs_app_dir>\WEB-INF\lib\httpclient-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\httpcore-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\httpmime-4.1.2.jar;
<cs_app_dir>\WEB-INF\lib\MSXML.jar;
<cs_app_dir>\WEB-INF\lib\servlet-api.jar;
<cs_app_dir>\WEB-INF\lib\spring-2.5.5.jar;
<cs_app_dir>\WEB-INF\lib\sites-security-11.1.1.8.0.jar;
<cs_app_dir>\WEB-INF\lib\esapi-2.0.1.jar;
<cs_app_dir>\WEB-INF\lib\log4j-1.2.16.jar;<j2eeSDK_dir>\j2ee.jar;
<cs_app_dir>\WEB-INF\classes
 COM.FutureTense.XML.Post.XMLPostMain-sc:\xmlpostfiles.lst -carticlepost.ini
```

20.4.4 Running XMLPost as a Batch Process

When you want to import assets of more than one type, you must identify a unique configuration file for each asset type and therefore run XMLPost individually for each asset type. You can run XMLPost either manually, or automatically from a batch file. In the batch file, include a command line statement for each asset type (the statement identifies the configuration file and the location of the source files). You can use any of the ways described in the preceding section to identify the source files.

20.4.5 Running XMLPost Programmatically

You can also invoke the XMLPost utility programmatically by creating an XMLPost object and calling the doIt method doIt(String[] args), where the input is a string array. The elements of the array are the same flags that you use when running XMLPost from the command line.

For example:

```
String args [] = {"-sSourcefile.xml","-cConfigfile.ini"};
COM.FutureTense.XML.Post.XMLPost poster = new
COM.FutureTense.XML.Post.XMLPost();
try {
  poster.doIt(args);
  } catch (Exception e) {
  e.printStackTrace();("error in XMLPost under program control");
}
```

Note that you must include the complete path to the source files and the configuration file.

20.5 Customizing RemoteContentPost and PreUpdate

If necessary, you can customize the XMLPost process by adding or modifying code in the `RemoteContentPost` element or the `PreUpdate` element for your asset types.

If you want to import information about an asset to other tables, you must modify the `PreUpdate` element for that asset type.

This section provides two customization examples:

- Customizing the `PreUpdate` element for the article asset type so that it sets headline information in the description field. There is a description column in the `Article` table but the field in the New or Edit article form is called **Headline**.
- Customizing the `PreUpdate` element for the article asset type so that it can add associations to articles.

This section contains the following topics:

- Section 20.5.1, "Setting a Field Value Programmatically"
- Section 20.5.2, "Setting an Asset Association"

20.5.1 Setting a Field Value Programmatically

The article asset type has a special condition: it has a field in the **New** and **Edit** forms called **Headline**, but the value for **Headline** is stored in the `description` column in the `Article` table. In order for headline text to be written to the correct column in the `Article` table when an article asset is imported (that is, the `description` column), the `PreUpdate` element for the article asset type was modified.

First, examine the sample configuration file named `ArticlePost.ini` that is located in the `Xcelerate/Samples/XMLPost` directory in your WebCenter Sites product kit. It has a tag specified for the **Headline** field:

```
# headline gets stored in the description field
postheadline: y
```

The following code in the `PreUpdate` element for the article asset type writes the data that `RemoteContentPost` passes in as `Variable.headline` to the correct database column:

```
<if COND="IsVariable.headline=true">
  <then>
    <ASSET.SET NAME="theCurrentAsset"
     FIELD="description"
     VALUE="Variables.headline"/>
  </then>
</if>
```

This example uses a tag called `ASSET.SET`. This tag sets data in a field for the asset that is currently in memory. It takes three parameters:

- NAME (required). The name of the asset object that is in memory. This asset object must have been previously instantiated either with the ASSET.LOAD tag or the ASSET.CREATE tag. By convention, WebCenter Sites uses the name theCurrentAsset to refer to the current asset object.
- FIELD (required). The name of the field whose value you want to set. The name of this field must exactly match the name of a column in the storage table for assets of this type.
- VALUE (required). The data to be inserted in the column.

20.5.2 Setting an Asset Association

If an asset has an association with another asset, that information is written to the AssetRelationTree table. Because the standard behavior of XMLPost is to write asset information to the primary storage table of the asset type only, you must modify the PreUpdate element for the asset type if you want to specify asset associations.

For example, the article asset type has an association field named **MainImageFile.** When a content provider creates an article asset, she selects the appropriate imagefile asset in this field.

Examine the sample configuration file named ArticlePost.ini that is located in the Xcelerate/Samples/XMLPost directory in your WebCenter Sites product kit. It has a tag specified for the **MainImageFile** association field:

```
postMainImageFile-name: y
```

The following code in the PreUpdate element for the article asset type writes the data that RemoteContentPost passes in as Variable.mainimagefile to the correct database table:

```
<if COND="IsVariable.MainImageFile-name=true">
<then>
 <ASSET.LOAD NAME="anAssociatedImage" TYPE="ImageFile"
  FIELD="name" VALUE="Variables.MainImageFile-name"/>
  <if COND="IsError.Variables.errno=false">
    <then>
      <ASSET.GET NAME="anAssociatedImage" FIELD="id" OUTPUT="imageid"/>
      <ASSET.ADDCHILD NAME="theCurrentAsset" TYPE="ImageFile"
       CHILDID="Variables.imageid" CODE="MainImageFile"/>
    </then>
  </if>
</then>
</if>
```

Note: The ASSET.ADDCHILD tag creates only the link between the two assets; it does not create the associated asset. In order for this code to work, the asset specified with the CHILDID parameter must already exist in the WebCenter Sites database.

This example uses a tag named ASSET.ADDCHILD. This tag associates a child asset with the asset that is currently held in memory. It takes five parameters:

- NAME (required). The name of the asset object that is in memory. This asset object must have been previously instantiated either with the ASSET.LOAD tag or the ASSET.CREATE tag. By convention, WebCenter Sites uses the name theCurrentAsset to refer to the current asset object.

- TYPE (required). The asset type of the child asset.
- CHILDID (required). The ID of the child asset.
- CODE (optional). The name of the association. This value is written to the ncode column in the AssetRelationTree table.
- RANK (optional). A numeric value to establish an order for the child assets. This value is written to the nrank column in the AssetRelationTree table.

For information about ASSET.GET and ASSET.LOAD, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

20.6 Troubleshooting XMLPost

This is a brief list of some possible problems that can occur when you run the XMLPost utility.

XMLPost does not run and does not create a log file message

There are two possible reasons for XMLPost to not start:

- An invalid server name has been specified in the xmlpost.URL property setting in your configuration file.
- WebCenter Sites is not running on the system you are importing to. Start it.

XMLPost fails and there is a "Missing Entity" statement in the log file

When you see this message in the log file, it means that there is invalid XML in the source file. Typically, your XML includes HTML code and that code includes special HTML characters that are not referred to by their character entity codes. For best coding practice, embed any HTML code in a <![CDATA[...]]> tag.

Error 105 is triggered when XMLPost tries to save an asset

There are several reasons why saving an asset can cause a database error.

One common reason for a 105 error is XMLPost trying to save data that is too large for the column (field). Resolving this depends on your goals. If it is acceptable for XMLPost to truncate the data that doesn't fit into the column, you can add a trunctag property to the configuration file. For example, truncbody: 2000.

Another common reason for this error code is that an asset of that type with the same name already exists. Try changing the name of the asset and importing the asset again.

Debugging the Posting Element

If you have modified the RemoteContentPost element in any way or have created your own posting element, you can use the XML Debugger utility to test it before you use it.

To use XML Debugger, replace ContentServer with DebugServer in the xmlpost.url property setting.

For example, change xmlpost.url: http://6ipjk/servlet/ContentServer

to xmlpost.url: http://6ipjk/servlet/DebugServer

For more information about the XML Debugger utility, see Chapter 8, "WebCenter Sites Tools and Utilities."

21

Importing Flex Assets

This chapter describes the posting elements for the XML-Post utility.

Chapter 20, "Importing Assets of Any Type" presents the core information about using the XMLPost utility. If you are using the flex asset data model, you have more tools for importing your assets. WebCenter Sites provides posting elements for the XMLPost utility and a bulk processing utility named BulkLoader.

This chapter contains the following sections:

- Section 21.1, "Understanding Flex Assets"
- Section 21.2, "XMLPost and the Flex Asset Model"
- Section 21.3, "Importing the Structural Asset Types in the Flex Model"
- Section 21.4, "Importing Flex Assets with XMLPost"
- Section 21.5, "Editing Flex Assets with XMLPost"
- Section 21.6, "Deleting Assets with XMLPost"

This chapter refers to the BulkLoader utility, but for in-depth information about how to run it, see Chapter 22, "Importing Flex Assets with the BulkLoader Utility."

21.1 Understanding Flex Assets

WebCenter Sites provides two methods for importing assets that use the flex data model into the WebCenter Sites database:

- XMLPost. WebCenter Sites provides three additional posting elements that work with XMLPost: `addData`, `modifyData`, and `deleteData`.
- The BulkLoader utility.

This section contains the following topics:

- Section 21.1.1, "Importing the Data Structure Flex Asset Types"
- Section 21.1.2, "Importing the Flex Assets"
- Section 21.1.3, "Importing Flex Assets: The Process"
- Section 21.1.4, "Using Custom Data Delimiters"

21.1.1 Importing the Data Structure Flex Asset Types

Before you can use either method, you must first create or import the data design or "structural" asset types into your flex families with XMLPost and the standard posting element, `RemoteContentPost,` provided by the WebCenter Sites product. That is, first you create or import the attribute editors, flex attributes, flex definitions, and flex parent definitions with the standard XMLPost posting element. If you are using the BulkLoader utility, the flex parents must also be imported with XMLPost or created.

21.1.2 Importing the Flex Assets

After you import your data structure asset types, then you can import your flex assets with one of the two import methods, depending on the situation:

- Use BulkLoader to import a large number (thousands or hundreds of thousands) of flex assets.
- Use the posting element to load a moderate number (hundreds) of flex and flex parent assets.

When to Use BulkLoader

When working within the basic asset model, it is typical to use XMLPost to import assets into the database on the management system and then publish those assets to the delivery system. This methodology changes with flex assets because the volume of data involved in a flex asset data model tends to be much greater than that in a basic asset model.

You use the BulkLoader utility during the initial setup of your WebCenter Sites system. See Chapter 22, "Importing Flex Assets with the BulkLoader Utility."

When to Use XMLPost

For regular or incremental updates after the initial setup of your WebCenter Sites system, perhaps some or all of your data originates in an ERP system, for example, you use the XMLPost utility and the `addData` posting element.

21.1.3 Importing Flex Assets: The Process

Because assets using the flex model have dependencies on each other, flex asset types must be imported in a specific sequence. And, as with basic assets, the asset types must exist, there must be sites created, and so on before you can use XMLPost to import assets.

For information about the basic prerequisites for using XMLPost that apply to all asset types (both asset models), see Section 20.4.1, "Before You Begin."

After those basic requirements are met, you must import your flex asset types into the WebCenter Sites database on the management system in the following sequence:

1. Attribute editors are optional, but if you plan to use them you must either import them or create them before you import your flex attributes. The configuration file must instruct XMLPost to call the `RemoteContentPost` element. For information, see Section 21.3.1, "Attribute Editors."
2. Flex attributes. The configuration file must instruct XMLPost to call the `RemoteContentPost` element. For information, see Section 21.3.2, "Flex Attributes."
3. Flex parent definitions. The configuration file must instruct XMLPost to call the `RemoteContentPost` element. For information, see Section 21.3.3, "Flex Definitions and Flex Parent Definitions."

   Flex definitions. The configuration file must instruct XMLPost to call the `RemoteContentPost` element. For information, see Section 21.3.3, "Flex Definitions and Flex Parent Definitions."

> **Note:** You must import the flex parent definitions in the proper order. That is, if a parent definition refers to another parent definition asset, the referenced asset must already exist in the database.
>
> It is typical to import parent definitions one hierarchical level at a time, starting with the top level definitions.

1. Flex parent assets. Do one of the following:

   – If you are going to use XMLPost to import the flex assets, you can either import the flex parents individually or you can import them as part of the flex family tree for a flex assets.

   – If you are going to use the BulkLoader utility to import the flex assets, you must first use XMLPost to import the flex parent assets. The configuration file must instruct XMLPost to call the `RemoteContentPost` element. The file cannot specify the `addData` element because you are importing the parents without the entire family tree for the flex assets.

   For information, see Section 21.3.4, "Flex Parents."

2. (Optional) If you plan to use the BulkLoader utility to import flex assets into both the management system and the delivery system, you must first approve and publish all of the structural assets (attribute editors, flex attributes, flex definitions, parent definitions, and flex parents) from the management system to the delivery system.

3. Flex assets. Do one of the following:

   – Use the BulkLoader utility. See Chapter 22, "Importing Flex Assets with the BulkLoader Utility."

   – Use XMLPost. See Section 21.4, "Importing Flex Assets with XMLPost."

You must follow the sequence outlined in the preceding steps because there are dependencies built in to the data structure of a flex asset family. Additionally, note the following dependencies:

- If you have attributes of type `asset` and a flex parent or flex asset has such an attribute, the asset that you designate as the value of that attribute field must have already been created or imported.
- An asset that you set as the value for an attribute of type `asset` must be of the correct asset type.

21.1.4 Using Custom Data Delimiters

If the data you are importing, editing, or deleting via XMLPost uses a different data delimiting schema than the WebCenter Sites default schema (see the table below for CS-default delimiter characters), you can specify custom delimiters as explained in the following table.

Table 21–1 Custom Data Delimiters

Tag and property	Description

tag:
`<_xmlnamevaldelim_>`
property:
`post_xmlnamevaldelim_`
	Optional.
	Lets you specify a custom character for delimiting name/value pairs from one another.
	To specify a custom delimiter:
	1. Set the property to `y` in the configuration file.
	2. Use the tag in your XML file to define the custom delimiter. For example, to use the "at" character as a delimiter:
	`<_xmlnamevaldelim_>`**@**`</_xmlnamevaldelim_>`
	The default delimiter is the colon (:).

tag:
`<_xmlpostequaldelim_>`
property:
`post_xmlpostequaldelim_`
	Optional.
	Lets you specify a custom character for delimiting attribute names from their values.
	To specify a custom delimiter:
	1. Set the property to `y` in the configuration file.
	2. Use the tag in your XML file to define the custom delimiter. For example, to use two equal signs as a delimiter:
	`<_xmlpostequaldelim_>`**==**`</_xmlpostequaldelim_>`
	The default delimiter is the equal sign (=).

tag:
`<_xmlpostmulvaldelim_>`
property:
`post_xmlpostmulvaldelim_`
	Optional.
	Lets you specify a custom character for delimiting the values of a multivalued attribute from one another.
	To specify a custom delimiter:
	1. Set the property to `y` in the configuration file.
	2. Use the tag in your XML file to define the custom delimiter. For example, to use a hyphen as a delimiter:
	`<_xmlpostmulvaldelim_>`**-**`</_xmlpostmulvaldelim_>`
	The default delimiter is the semicolon (;).

21.2 XMLPost and the Flex Asset Model

The XMLPost utility works the same no matter which asset model or WebCenter Sites product you are using. However, WebCenter Sites provides additional processing logic in some of its standard elements for the flex asset types to support XMLPost because flex assets store their data in more than one database table (unlike basic asset types, which have one database table).

Additionally, WebCenter Sites provides both a posting element that enables you to use XMLPost to edit flex assets (`modifyData`) and a posting element that enables you to use XMLPost to delete assets of any type (`deleteData`).

This chapter provides additional information about creating configuration and source files specifically for the asset types in a flex family (and attribute editors). Be sure to also read Chapter 20, "Importing Assets of Any Type" for basic information that pertains to all XMLPost configuration and source files.

In the flex asset model, you specify a different posting element based on the following categories of asset types:

- Structural asset types that give the flex asset type and flex parent asset type their data structure. That is, attribute editors, attributes, flex definitions, and flex parent definitions.

  Use the standard WebCenter Sites posting element `RemoteContentPost` to import the structural asset types. (You cannot use the `addData` element with assets of these types.)

- Flex and flex parent asset type (for example, the product and product parent types in the GE Lighting sample site).

  Depending on the situation, you can use either the posting element `addData` to import the flex and flex parent asset types or the posting element `RemoteContentPost`. (See Section 21.3.4, "Flex Parents" and Section 21.4, "Importing Flex Assets with XMLPost" for information about which posting element to use.)

In both cases, you create configuration files and source files (as described in Section 21.1, "Understanding Flex Assets" and supplemented in this chapter), and then invoke the XMLPost utility (as described in Section 20.4, "Using the XMLPost Utility").

> **Note:** For reference, sample XMLPost code is provided **in** the WebCenter Sites installer package, in the **/Xcelerate/Samples/XMLPost** directory. The same folder contains the readme.txt file that describes the sample files.

Internal Names vs. External Names

When you create your flex family of asset types (see Section 16.3.3, "Step 1: Create a Flex Family"), you specify both an internal and an external name for your asset types.

The internal name is used for the primary storage table in the database. The external name is used in the New, Edit, and Inspect forms, in search results list, and so on. For example, the internal name for the attribute editor asset type is AttrTypes but that name is not used in the user interface. And the internal name for the GE Lighting sample site's article asset type is `AArticles` but that name is not used in the user interface.

Because XMLPost communicates with the database, you must always use the internal name of the asset type in the configuration files and source files. For example, in a configuration file for attribute editors, you would specify the following:

```
postargname2: AssetType
postargvalue2: AttrTypes
```

21.3 Importing the Structural Asset Types in the Flex Model

All of the information about configuration and source files for basic assets that is presented in Chapter 20, "Importing Assets of Any Type" applies to the configuration and source files for the flex asset types.

Additionally, this section provides example configuration and source files for the structural flex asset types.

This section contains the following topics:

- Section 21.3.1, "Attribute Editors"
- Section 21.3.2, "Flex Attributes"

- Section 21.3.3, "Flex Definitions and Flex Parent Definitions"
- Section 21.3.4, "Flex Parents"

21.3.1 Attribute Editors

Attribute editors store their data in one table, named `AttrTypes`. AttrTypes is the internal name of the attribute editor asset type. Be sure to use this name in your configuration file for attribute editors.

The following table describes the configuration file properties and source file tags that you use with attribute editors:

Table 21–2 Attribute Editor Tag and Properties

Attribute editor tag and property	Description

tag:
`<name>`
property:
`postname`
	Required.
	Name of the attribute editor asset; this is a required value for all asset types. Attribute names are limited to 64 characters and cannot contain spaces.

tag:
`<description>`
property:
`postdescription`
	Optional.
	Description of the use or function of the attribute.

tag: `<AttrTypeText>` property: `postAttrTypeText`	Required. Either the name of the file with the attribute editor XML code, or the actual code. This tag corresponds to the XML in file field and Browse button and the XML field in the New and Edit attribute editor forms in the WebCenter Sites interface.

21.3.1.1 Sample Configuration File: Attribute Editor

This is a sample configuration file for the attribute editor asset type. It works with the sample source file immediately following this example.

```
xmlpost.xmlfilenamefilter: .xml
xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: AttrTypes
# notice that you use the internal name of the asset type

xmlpost.argname3: authusername
xmlpost.argvalue3: user_editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: startmenu
xmlpost.argvalue6: New Attribute Editor

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: attreditorpostlog.txt

xmlpost.deletefile: y

postpublication: y

postname: y
postdescription: y
postAttrTypeText: y
```

21.3.1.2 Sample Source File: Attribute Editor

The following source file is tagged for importing a check box attribute editor, or presentation object, for the GE Lighting sample catalog. It works with the preceding sample configuration file.

```
<document>
<publication>GE Lighting</publication>
<name>Editor4-CheckBoxes</name>
<description>Attribute Type Four Check Box</description>
<AttrTypeText>
  <![CDATA[
    <?XML VERSION="1.0"?>
    <!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
    <PRESENTATIONOBJECT NAME="CheckBoxTest">
      <CHECKBOXES LAYOUT="VERTICAL">
        <ITEM>Red</ITEM>
        <ITEM>Green</ITEM>
        <ITEM>Blue</ITEM>
      </CHECKBOXES>
    </PRESENTATIONOBJECT>
  ]]>
</AttrTypeText>
</document>
```

21.3.2 Flex Attributes

Flex attributes have several tables but XMLPost writes to only two of them: the main storage table (for example, the `PAttributes` table for the GE Lighting sample site) and the attribute asset type's `_Extension` table (`PAttributes_Extension`, for example).

This means that the source file section of the configuration file must specify and the source file itself must use tags that represent columns in both tables. Those source file tags and configuration file properties are as follows:

Table 21–3 Flex Attribute Tags and Properties

Flex attribute tag and property	Description

tag:
`<name>`
property:
`postname`
	Required.
	Name of the attribute; this is a required value for all asset types. Attribute names are limited to 64 characters and cannot contain spaces.

Table 21–3 (Cont.) Flex Attribute Tags and Properties

Flex attribute tag and property	Description

tag:
`<description>`
property:
`postdescription`
	Optional.
	Description of the use or function of the attribute.

tag: `<valuestyle>` property: `postvaluestyle`	Optional. Whether the attribute can hold a single value (`S`) or multiple values (`M`). If no this tag is not used, the attribute is set to hold a single value by default.

tag:
`<type>`
property:
`posttype`
	Required.
	The data type of the attribute. Valid options are `asset, date, float, int, money, string, text,` or `blob`. For definitions of these data types, see Section 11.3.3.1, "Data Types for Attributes."

tag: `<assettypename>` property: `postassettypename`	Required if `<type>` is set to `asset`. The name of the asset type that the attribute holds.

tag:
`<upload>`
property:
`postupload`
	Required if `<type>` is set to `blob`.
	The path to the directory that you want to store the attribute values in. Note that the value that you enter in this field is appended to the value set as the default storage directory (defdir) for the attribute table by the `cc.urlattrpath` property in the `gator.ini` file (which is `FutureTense/futuretense_cs/ccurl/` by default).

tag: `<attributetype>` property: `postattributetype`	Optional. The name of the attribute editor to use, if applicable.

tag:
`<enginename>`
property:
`postenginename`
	Optional.
	If you are using a search engine on your management system, the name of the search engine.

tag:
`<charsetname>`
property:
`postcharsetname`
	Optional.
	The search engine character set to use. By default, it is set to ISO 8859-1.

tag:
`<editing>`
property:
`postediting`
	Foreign attributes only.
	Whether a foreign attribute can be edited through the WebCenter Sites forms (`L`), or edited externally using a third-party tool (`R`). `L` is the default.

tag:
`<storage>`
property:
`poststorage`
	Foreign attributes only.
	Whether the values for a foreign attribute are to be stored in a `_Mungo` table in the WebCenter Sites database (`L`) or in a foreign table (`R`). `L` is the default.

Table 21–3 (Cont.) Flex Attribute Tags and Properties

Flex attribute tag and property	Description

tag:
`<externalid>`
property:
`postexternalid`
	Foreign attributes only.
	The name of the column that serves as the primary key for the table that holds this foreign attribute; that is, the column that uniquely identifies the attribute.

tag:
`<externalcolumn>`
property:
`postexternalcolumn`
	Foreign attributes only.
	The name of the column in the foreign table that holds the values for this attribute.

tag:
`<externaltable>`
property:
`postexternaltable`
	Foreign attributes only.
	The name of the foreign table that contains the columns identified by externalid and external column.

tag:
`<publication>`
property:
`postpublication`
	Optional.
	The names of all the sites that can use this attribute.

21.3.2.1 Sample Configuration File: Flex Attribute

This is sample configuration file for the product attribute asset type from the GE Lighting sample site. It works with the sample source file immediately following this example.

```
xmlpost.xmlfilenamefilter: .xml

xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: PAttributes
# Notice that this is the internal name of the asset
# type. The external name of this asset type is
# Product Attribute.

xmlpost.argname3: authusername
xmlpost.argvalue3: user_editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: startmenu
xmlpost.argvalue6: New Product Attribute

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: attributespostlog.txt

xmlpost.deletefile: y
```

```
postpublication: y
postname: y
postattributetype: y
postdescription: y
postvaluestyle: y
posttype: y
postediting: y
poststorage: y
postenginename: y
poststatus: y
postassettypename: y
postupload: y
postexternalid: y
postexternalcolumn: y
postexternaltable: y
postcharsetname: y
```

21.3.2.2 Sample Source File: Attribute

This is a sample source file for importing a product attribute named `footnotes` for the GE Lighting sample site. It works with the preceding sample configuration file.

```
<document>
  <publication>GE Lighting</publication>
  <name>footnotes</name>
  <description>Footnotes</description>
  <valuestyle>S</valuestyle>
  <type>URL</type>
  <editing>L</editing>
  <storage>L</storage>
</document>
```

> **Note:** Remember that all the dependencies and restrictions concerning the data type of a flex attribute apply whether you are creating an attribute through the WebCenter Sites interface (the **New** or **Edit** flex attribute forms) or through XMLPost. For information, read Section 16.3.6, "Step 4: Create Flex Attributes."

21.3.3 Flex Definitions and Flex Parent Definitions

The flex definition and flex parent definition asset types are very similar and you code their configuration and source files in nearly the same way. They require several of the same tags in their source files and the same properties in their configuration files. Each has one additional property/tag.

This section contains the following topics:

- Section 21.3.3.1, "Sample Configuration File: Flex Definition"
- Section 21.3.3.2, "Sample Source File: Flex Definition"

The source file tags and configuration file properties for flex definitions and flex parent definitions are listed in the following table. Note that they are case-sensitive.

Table 21–4 Flex Definition and Flex Parent Definition Tags and Properties

Flex definition and flex parent definition tag and property	Description

tag:
`<internalname>`
property:
`postinternalname`
	Required.
	The name of the asset; this is a required value for all asset types. Flex definition and flex parent definition names are limited to 64 characters and they cannot contain spaces.

tag:
`<internaldescription>`
property:
`postinternaldescription`
	Optional.
	The description of the use or function of the asset.

tag:
`<renderid>`
property:
`postrenderid`
	Optional. For flex definitions only.
	The ID of the Template asset that is to be assigned to all the flex assets that are created with this flex definition.

tag:
`<parentselectstyle>`
property:
`postparentselectstyle`
	Optional. For flex parent definitions only.
	Defines how flex parents are to be selected when a user creates a flex asset using the definition.
	This property/tag represents the Parent Select Style field in the New and Edit parent definition forms.
	When using the tag in the source file, the options are `treepick` and `selectboxes`.

The next four tags and properties perform the same function as the buttons and fields in the Product Parent Definition section on the New and Edit forms for parent definitions and flex definitions. See Section 16.3.7, "Step 5: (Optional) Create Flex Filter Assets" and Section 16.3.9, "Step 7: Create Flex Definition Assets."	n/a

tag:
`<OptionalSingleParentList>`
property:
`postOptionalSingleParentList`
	Use this tag to specify any single optional parent definition.

tag:
`<RequiredSingleParentList>`
property:
`postRequiredSingleParentList`
	Use this tag to specify any single required parent definition.

tag:
`<RequiredMultipleParentList>`
property:
`postRequiredMultipleParentList`
	Use this tag to specify more than one required parent definitions.

Table 21–4 (Cont.) Flex Definition and Flex Parent Definition Tags and Properties

Flex definition and flex parent definition tag and property	Description

tag:
`<OptionalMultipleParentList>`
property:
`postOptionalMultipleParentList`
	Use this tag to specify more than one optional parent definition.

The next three tags and properties perform the same functions as the buttons and fields in the Attributes section on the New and Edit forms for flex definitions and flex parent definitions. See Section 16.3.7, "Step 5: (Optional) Create Flex Filter Assets" and Section 16.3.9, "Step 7: Create Flex Definition Assets."	n/a

tag:
`<RequiredAttrList>`
property:
`postRequiredAttrList`
	The list of attributes that are required for the flex parents or the flex assets that use the definition.

tag:
`<OptionalAttrList>`
property:
`postOptionalAttrList`
	The list of attributes that are optional for the flex parents or the flex assets that use the definition.

tag:
`<OrderedAttrList>`
property:
`postOrderedAttrList`
	The order in which all attributes, be they required or optional, should appear in the New, Edit, Inspect, and similar forms.
	If you use this tag, it replaces the other attribute tags. The example source file in this section shows an example of how to use this tag in a source file.

A configuration file must include all the properties that could be used by any one of the assets of the type that the configuration file works with. The individual source files include only the tags that are needed to define those individual assets.

21.3.3.1 Sample Configuration File: Flex Definition

The following example is a configuration file used to import product definitions for the GE Lighting sample site. It works with the sample source file immediately following this example.

```
xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: ProductTmpls
# Notice that this is the internal name of the asset type.
# The external name of this asset type is
# Product Definition.

xmlpost.argname3: authusername
xmlpost.argvalue3: user_editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: startmenu
xmlpost.argvalue6: New Product Definition

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdefpostlog.txt
xmlpost.deletefile: y

postpublication: y
postinternalname: y
postinternaldescription: y

postparentselectstyle: y

postOptionalSingleParentList: y
postRequiredSingleParentList: y
postRequiredMultipleParentList: y
postOptionalMultipleParentList: y

postRequiredAttrList: y
postOptionalAttrList: y
postOrderedAttrList: y

postrenderid: y
```

21.3.3.2 Sample Source File: Flex Definition

The following source file, `lighting.xml`, is the source for a product definition named Lighting for the GE Lighting sample site. It works with the preceding sample configuration file.

```
<document>
<publication>GE Lighting</publication>
<internalname>Lighting</internalname>
<internaldescription>Generic Lighting Template</internaldescription>
<RequiredAttrList>sku</RequiredAttrList>
  <OptionalAttrList>
    productdesc;caseqty;bulbshape;bulbsize;basetype;
    colortemp;meanlength;lightcenterlength;reducedwattage;beamspread;
    fixturetype;ballasttype;colorrenderingindex;minstarttemp;powerfactor;
    totalharmonicdist;spreadbeam10h;spreadbeam10v;spreadbeam50h;
    spreadbeam50v;halogen;operatingposition;filamenttype;bulbimage;
    baseimage;filamentimage;footnotes;price;life;voltage;wattage
  </OptionalAttrList>
<parentselectstyle>treepick</parentselectstyle>
<OptionalMultipleParentList>SubCategory</OptionalMultipleParentList>
</document>
```

Examine the list of attributes, above. When you include multiple values in a tag, separate them from each other with a semicolon (;).

Note that while GE Lighting uses the optional/multiple parent model, there are these other possible configurations:

```
<OptionalSingleParentList>flexparentdefinition</OptionalSingleParentList>
<RequiredSingleParentList>flexparentdefinition</RequiredSingleParentList>
```

```
<RequiredMultipleParentList>flexparentdefinition</RequiredMultipleParentList>
```

Supplying a List of Ordered Attributes

If you want to use the `<OrderedAttrList>` tag because the attributes need to be displayed in a specific order, do not also include the `<RequiredAttrList>` and `<OptionalAttrList>` tags. In the string contained in the `<OrderedAttrList>` tag, specify which attributes are required and which are optional, as follows:

- For required attributes, precede the attribute name with `R` (required)
- For optional attributes, precede the attribute name with or `O` (optional)
- Be sure to list the attributes in the desired order.
- Be sure to use a semicolon (;) to separate the values.

For example:

```
<OrderedAttrList>Rsku;Oproductdesc;Ocaseqty;Obulbshape;Obulbsize;Obasetype;Ocolort
emp;Omeanlength;Olightcenterlength;Oreducedwattage;<OrderedAttrList>
```

21.3.4 Flex Parents

You can use XMLPost to import flex parent assets in two ways:

- Individually. You code a separate XMLPost source file for each flex parent and an XMLPost configuration file that identifies the asset type and the pagename for the standard `RemoteContentPost` posting element. If you plan to use the BulkLoader utility, you must first import the flex parent assets with XMLPost in this way.
- As part of the flex family tree for a flex asset. If you are using XMLPost to import your flex assets (rather than the BulkLoader), you can combine the flex parents with the flex assets and import the flex parents as a part of a flex family tree, within the context of a specific flex asset. You code a separate XMLPost source file for each flex asset and identify all the parents for that flex asset in that source file. XMLPost then creates the variables for one flex asset and multiple flex parents (if they do not yet exist) when it parses the source file.

This section describes the source and configuration file for importing them individually. For information about importing them with the flex assets, see Section 21.4, "Importing Flex Assets with XMLPost."

This section contains the following topics:

- Section 21.3.4.1, "Sample Configuration File: Individual Flex Parent"
- Section 21.3.4.2, "Sample Source File: Individual Flex Parent"

21.3.4.1 Sample Configuration File: Individual Flex Parent

The following example is a configuration file used to import product parents for the GE Lighting sample site. It works with the sample source file immediately following this example.

```
xmlpost.xmlfilenamefilter: .xml

xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: ProductGroups
# notice that you use the internal name of the asset type

xmlpost.argname3: authusername
xmlpost.argvalue3: user_editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: startmenu
xmlpost.argvalue6: New Product Parent

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdefpostlog.txt
xmlpost.deletefile: y

postpublication: y
postinternalname: y
postinternaldescription: y
postflexgrouptemplateid: y
postfgrouptemplatename: y
postParentList: y
postcat1: y
postcat2: y
```

21.3.4.2 Sample Source File: Individual Flex Parent

The following source file creates a product parent (flex parent) named Halogen for the GE Lighting sample. It works with the preceding sample configuration file.

```
<document>
<publication>GE Lighting</publication>
<internalname>Halogen</internalname>
<fgrouptemplatename>Category</fgrouptemplatename>
<cat1>Halogen</cat1>
</document>
```

Remember that when you use the `RemoteContentPost` posting element, you must provide one source file for each parent asset.

21.4 Importing Flex Assets with XMLPost

Before you can use XMLPost to import flex assets, you must have already imported the structural asset types (attributes, flex definitions, and flex parent definitions).

There are two posting elements that you can use for flex assets, `RemoteContentPost` or `addData`:

- The `addData` posting element creates parent assets for the flex asset if they do not yet exist. For example, if you are not using the BulkLoader utility, you use this posting element for the initial import of your flex assets.

  When you use the `addData` posting element, the source file must specify the entire family tree for the flex asset. You need a separate source file for each flex asset, but you can specify any number of parents for that flex asset in that source file and XMLPost creates the flex asset and its parents (if they do not yet exist).

- The `RemoteContentPost` element creates flex assets and sets values for their parents. Those parents must already exist. For example, if you plan to use

BulkLoader once, just for the initial import and will use XMLPost from then on, you might want to use this posting element.

When you use `RemoteContentPost` to import a flex asset, the source file must specify only the asset's immediate parents (which requires you to include fewer lines of code). However, if you need to create a new flex parent for the new flex asset, you must use the `addData` posting element and specify the entire family tree in the source file.

This section contains the following topics:

- Section 21.4.1, "Configuration File Properties and Source File Tags for Flex Assets"
- Section 21.4.2, "Sample Flex Asset Configuration File for addData"
- Section 21.4.3, "Configuration File Properties and Attributes of Type Blob (or URL)"
- Section 21.4.4, "Sample Flex Asset Source File for addData"
- Section 21.4.5, "Sample Flex Asset Configuration File for RemoteContentPost"
- Section 21.4.6, "Sample Flex Asset Source File for RemoteContentPost"

21.4.1 Configuration File Properties and Source File Tags for Flex Assets

As with the structural asset types, you must use the internal name of the flex and flex parent asset types in your configuration and source files.

However, unlike the structural asset types, you do not need to include an argument for the asset type in the configuration file. Source files for flex assets have a required tag that identifies the asset type so you do not have to repeat this information in the configuration file.

This section contains the following topics:

- Section 21.4.1.1, "For the addData Posting Element"
- Section 21.4.1.2, "For the RemoteContentPost Posting Element"
- Section 21.4.1.2, "For the RemoteContentPost Posting Element"

21.4.1.1 For the addData Posting Element

The following table lists the source file tags and configuration file properties for flex assets (and their flex parents) when you are using the `addData` posting element. Note that they are case-sensitive.

Table 21–5 addData Posting Element

Tag and property	Description

tag:
`<_ASSET_>`
property:
`post_ASSET_`
	Required.
	The internal name of the asset type. For example, the internal asset type names of the GE Lighting sample site flex assets are `Products`, `AArticles`, and `AImages`.

tag:
`<_TYPE_>`
property:
`post_TYPE_`
	Required.
	The name of the flex definition that this flex asset is using.

Table 21–5 (Cont.) addData Posting Element

Tag and property	Description

tag:
`<_ITEMNAME_>`
property:
`post_ITEMNAME_`
	Required.
	The name of the asset.

tag:
`<_ITEMDESCRIPTION_>`
property:
`post_ITEMDESCRIPTION_`
	Optional.
	The description of the asset.

`tag:`
`<_GROUP_parentDefinitionName>`
property:
`post_GROUP_parentDefinitionName`
	Optional.
	The flex asset's parents. The configuration file must include a tag for each possible parent definition. For example, if your flex assets could have parents that use either of two parent definitions named Division and Department, the configuration file needs two properties that define a tag for each:
	`post_Group_Department`
	`post_Group_Division`
	See Section 21.4.4.4, "Specifying the Parents of a Flex Asset" for more information about using this tag and property.

tag:
`<_GROUPDESCRIPTIONS_>`
property:
`post_GROUPDESCRIPTIONS_`
	Optional.
	If the parent that you are designating is new, you can also include the description of the parent definition.

tag:
`<displaytype>`
property:
`postdisplaytype`
	Optional.
	The name of the Template asset for the flex asset.

tag:
`<AttributeName>`
property:
`postAttributeName`
	Include a property in the configuration file for each attribute that assets of the type can have (both required and optional). The source files then need to supply a value for each required attribute and any optional ones that apply to that asset.
	For example, if there were an attribute named SKU, you would include a property called `postSKU` and in the source files, would include lines of code like this:
	`<SKU>123445</SKU>`

21.4.1.2 For the RemoteContentPost Posting Element

The following table lists the source file tags and configuration file properties for flex assets (and their flex parents) when you are using the `RemoteContentPost` posting element. Note that they are case-sensitive.

Table 21–6 RemoteContentPost Posting Element

Tag and property	Description

tag:
`<_DEFINITION_>`
property:
`post_DEFINITION_`
	Required.
	The name of the flex definition that this flex asset is using.
	(Note that `post_TYPE` will also work.)

tag:
`<_ITEMNAME_>`
property:
`post_ITEMNAME_`
	Required.
	The name of the asset.

tag:
`<_ITEMDESCRIPTION_>`
property:
`post_ITEMDESCRIPTION_`
	Optional.
	The description of the asset.

`tag:`
`<ParentList>`
property:
`post_ParentList`
	Optional.
	The flex asset's immediate parents.

tag:
`<template>`
property:
`posttemplate`
	Optional.
	The name of the Template asset for the flex asset.
	(Note that `postdisplaytype` will also work.)

tag:
`<AttributeName>`
property:
`postAttributeName`
	Include a property in the configuration file for each attribute that assets of the type can have (both required and optional). The source files then need to supply a value for each required attribute and any optional ones that apply to that asset.
	For example, if there were an attribute named SKU, you would include a property called `postSKU` and in the source files, would include lines of code like this:
	`<SKU>123445</SKU>`

21.4.2 Sample Flex Asset Configuration File for addData

This is a sample configuration file for the product asset type from the GE Lighting sample site. It invokes the `addData` posting element and works with the source file example immediately following this example file.

```
xmlpost.xmlfilenamefilter: .xml

#xmlpost.proxyhost: Future
#xmlpost.proxyport: 80

xmlpost.url: http://wally9:80/servlet/ContentServer

# notice that it uses addData
# rather than RemoteContentPost
xmlpost.numargs: 5

xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Gator/XMLPost/addData

# Notice that you do not need to provide
# the name of the asset type because that information
# is required in the source files for flex assets.

xmlpost.argname2: inifile
xmlpost.argvalue2: futuretense.ini
xmlpost.argname3: authusername
xmlpost.argvalue3: editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: xceleditor
xmlpost.argname5: startmenu
xmlpost.argvalue5: New Product

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdatalog.txt

xmlpost.postdeletefile: y

post_ASSET_: y
post_ITEMNAME_: y
post_TYPE_: y
post_GROUP_Category: y
post_GROUP_SubCategory: y
postpublication: y
postsku: y
postproductdesc: y
postcaseqty: y
postbulbshape: y
postbulbsize: y
postbasetype: y
postcolortemp: y
postmeanlength: y
postlightcenterlength: y
postreducedwattage: y
postbeamspread: y
postfixturetype: y
postballasttype: y
postcolorrenderingindex: y
postminstarttemp: y
postpowerfactor: y
posttotalharmonicdist: y
postspreadbeam10h: y
postspreadbeam10v: y
postspreadbeam50h: y
postspreadbeam50v: y
posthalogen: y
postoperatingposition: y
postfilamenttype: y
postbulbimage: y
postbaseimage: y
postfilamentimage: y
postfootnotes: y
postcat1: y
postcat2: y
postprice: y
postvoltage: y
postwattage: y
```

```
postlife: y
```

21.4.3 Configuration File Properties and Attributes of Type Blob (or URL)

If the asset type has an attribute of type `blob` (or `url`), the configuration file needs two entries for the tag that references the attribute: one to identify the attribute and one to identify the file name of either the file that holds the content for the attribute (an upload field) or the name that you want WebCenter Sites to give the file that it creates from text entered directly into a text field (a text field of type `blob` or `URL`).

Attribute of Type Blob (or URL) as an Upload Field

There are no attributes of type blob in the GE Lighting sample site. However, lets say that there is an attribute of type `blob` named `footnotes`. It is an upload field with a **Browse** button for finding the file rather than a text field that you enter text in to. Therefore it has two properties:

- `posttag`, which in this scenario is `postfootnotes: y`
- `filetag`, which in this scenario is `filefootnotes: y`

When you include a value for this attribute in a source file, you use the following convention:

```
<footnotes>FileName.txt</footnotes>
```

Note that when you are importing an asset that has this kind of field (attribute), the file that holds the text that you want to store as the attribute value for the flex asset must be located in the same directory as the source file for the asset.

Attribute of Type Blob (or URL) as a Text Field

If the fictitious `footnotes` attribute is a field that takes text directly rather than a file, the configuration file requires the following properties:

- `postfootnotes: y`
- `postfootnotes_file: y`

Then, when you include a value for the attribute in the source file, you use the following convention:

```
<footnotes>lots and lots of text</footnotes>
<footnotes_file>FileNameYouWantUsed.txt</footnotes_file>
```

21.4.4 Sample Flex Asset Source File for addData

This following source file works with the example flex asset configuration file preceding this section.

This section contains the following topics:

- Section 21.4.4.1, "Sample File"
- Section 21.4.4.2, "Handling Special Characters"
- Section 21.4.4.3, "Flex Assets and Their Parents"
- Section 21.4.4.4, "Specifying the Parents of a Flex Asset"
- Section 21.4.4.5, "Setting Attribute Values for Parents"
- Section 21.4.4.6, "Setting Multiple Values in a Flex Source File"

21.4.4.1 Sample File

This source file creates a lightbulb product named 10004 from the product definition named Lighting:

```
<document>

# the first three tags are required
<_ASSET_>Products</_ASSET_>
<_ITEMNAME_>10004</_ITEMNAME_>
<_TYPE_>Lighting</_TYPE_>

# This tag is required because the publication is
# not set in the configuration file
<publication>GE Lighting</publication>

# This tag assigns a Template asset to the product
<displaytype>Lighting Detail</displaytype>

# The rest of these tags set flex attribute values for the product
<price>5</price>
<sku>10004</sku>
<productdesc>F4T5/CW</productdesc>
<caseqty>24</caseqty>
<bulbshape>T</bulbshape>
<bulbsize>5</bulbsize>
<basetype>Miniature Bipin (G5)</basetype>
<colortemp>4100</colortemp>
<meanlength></meanlength>
<lightcenterlength></lightcenterlength>
<reducedwattage></reducedwattage>
<beamspread></beamspread>
<fixturetype></fixturetype>
<ballasttype></ballasttype>
<colorrenderingindex>60</colorrenderingindex>
<minstarttemp></minstarttemp>
<powerfactor></powerfactor>
<totalharmonicdist></totalharmonicdist>
<spreadbeam10h></spreadbeam10h>
<spreadbeam10v></spreadbeam10v>
<spreadbeam50h></spreadbeam50h>
<spreadbeam50v></spreadbeam50v>
<halogen></halogen>
<operatingposition></operatingposition>
<filamenttype></filamenttype>
<bulbimage>BLB-260.gif</bulbimage>
<baseimage>BLB-250.gif</baseimage>
<filamentimage></filamentimage>
<footnotes>
</footnotes>
<life>6000</life>
<voltage></voltage>
<wattage>4</wattage>
<cat1>Fluorescent</cat1>
<cat2>Preheat Lamps</cat2>

<!-- GROUP tags that specify the parents. Remember that you have to
specify the entire family tree for the flex asset when using the addData posting
element-->

<_GROUP_Category>Fluorescent</_GROUP_Category>
```

```
<_GROUP_SubCategory>Preheat Lamps</_GROUP_SubCategory>
</document>
```

The preceding source file set the product's parent to Preheat Lamps and the parent of Preheat Lamps to Fluorescent.

21.4.4.2 Handling Special Characters

XMLPost uses the HTTP POST protocol, which means that it sends data in an HTTP stream. Therefore, certain characters are considered to be special characters and must be encoded because they are included in URLs.

If your source file includes attribute values that contains any of the special characters listed in the following table, be sure to replace all instances of that character with its corresponding URL encoding sequence, found in Section 4.8, "Values for Special Characters."

21.4.4.3 Flex Assets and Their Parents

The `GROUP` tags specify the parents in the family tree. When XMLPost uses the `addData` posting element and parses the `GROUP` section of the source file, it does the following:

1. Determines which parent definitions are legal for an asset using this flex definition.
2. For each legal parent definition, it verifies whether the source file specifies a parent of that definition:
   - If yes, it sets the parent and if the parent does not yet exist, it creates the parent.
   - If no, it does not set the parent. However, if a parent of that definition is required, it returns an error.

21.4.4.4 Specifying the Parents of a Flex Asset

To specify the parents of a flex asset, you provide the name of the parents nested in the `<_GROUP_parentDefinitionName>` tag. For example:

```
<_GROUP_subcategory>Blacklights</_GROUP_subcategory>
```

Where `subcategory` is the name of the parent definition for the Blacklights parent (product parent).

Remember that you must specify the entire family tree for the flex asset. In the GE Lighting sample site, a product asset has a parent and a grandparent. In addition to specifying the parent for the lightbulb (Blacklight), you need to specify the grandparent. For example:

```
<_GROUP_subcategory>Blacklights</_GROUP_subcategory>

<_GROUP_category>Fluorescent</_GROUP_category>
```

21.4.4.5 Setting Attribute Values for Parents

How does XMLPost know which parent the attribute values belong to? It does not. If an attribute can belong to more than one parent, you must specify which parent it belongs to. For example, let's say that the bulbshape attribute is assigned to parents rather than products. In this case, you would include a line of code such as this:

```
<bulbshape>Halogen=T</bulbshape>
```

21.4.4.6 Setting Multiple Values in a Flex Source File

All of the tags that configure parents and the tags that specify attributes (as long as the attribute is configured to accept multiple values) can handle multiple values. Those tags are as follows:

- _GROUP_parentDefinitionName
- _GROUPDESCRIPTIONS_
- the attribute tags

When you have multiple parents from the same definition for a flex asset, you provide all of the names of the parents in the same _GROUP_parentDefinitionName tag and you use a semicolon (;) to separate the parent names.

For example:

```
<_GROUP_Cateogry>Incandescent;Halogen</_GROUP_Category>
```

When XMLPost imports this asset, it sets its parents as Incandescent and Halogen, which are both of the Category parent definition. If Incandescent and Halogen do not exist yet, XMLPost creates them.

You use a similar syntax when you want to set multiple attribute values for the multiple parents. Once again, let's say that the Category definition requires that parents of that definition have a value for the bulbshape attribute. You can set the value of the bulbshape attribute for both of the parents that were specified by the <_GROUP_Category> tag as follows:

```
<bulbshape>Incandescent=E;K:Halogen=T</bulbshape>
```

Note the following about this syntax:

- You use parentName=attributeValue pairs to set the attribute value (Halogen=T).
- You use a colon to separate the parents from each other (Incandescent=S:Halogen=T).
- You use a semicolon to separate the attribute values for a parent when that parent has more than one value for the attribute (Incandescent=E;K:Halogen=T).

And, as mentioned, you can specify descriptions for the parents that you identify in the same tag, too. For example:

```
<_GROUPDESCRIPTIONS>
Incandescent=From Detroit:Halogen=From Chicago
</_GROUPDESCRIPTIONS>
```

21.4.5 Sample Flex Asset Configuration File for RemoteContentPost

This is a sample configuration file for the product asset type from the GE Lighting sample site. It works with the source file example immediately following this example file.

```
xmlpost.xmlfilenamefilter: .xml

#xmlpost.proxyhost: Future
#xmlpost.proxyport: 80

xmlpost.url: http://wally9:80/servlet/ContentServer
xmlpost.numargs: 5
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost

# Notice that you do not need to provide
# the name of the asset type because that information
# is required in the source files for flex assets.

xmlpost.argname2: inifile
xmlpost.argvalue2: futuretense.ini
xmlpost.argname3: authusername
xmlpost.argvalue3: editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: xceleditor
xmlpost.argname5: startmenu
xmlpost.argvalue5: New Product

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdatalog.txt

xmlpost.postdeletefile: y

postpublication: y

post_ASSET_: y
post_ITEMNAME_: y
post_DEFINITION_: y
posttemplate: y

postsku: y
postproductdesc: y
postcaseqty: y
postbulbshape: y
postbulbsize: y
postbasetype: y
postcolortemp: y
postmeanlength: y
postlightcenterlength: y
postreducedwattage: y
postbeamspread: y
postfixturetype: y
postballasttype: y
postcolorrenderingindex: y
postminstarttemp: y
postpowerfactor: y
posttotalharmonicdist: y
postspreadbeam10h: y
postspreadbeam10v: y
postspreadbeam50h: y
postspreadbeam50v: y
posthalogen: y
postoperatingposition: y
postfilamenttype: y
postbulbimage: y
postbaseimage: y
postfilamentimage: y
postfootnotes: y
postcat1: y
postcat2: y
postprice: y
```

```
postvoltage: y
postwattage: y
postlife: y

postParentList: y
```

21.4.6 Sample Flex Asset Source File for RemoteContentPost

This following source file works with the example configuration file immediately preceding this section. This source file creates a lightbulb product named `10004` from the product definition named `Lighting`:

```
<document>

# the first three tags are required
<_ASSET_>Products</_ASSET_>
<_ITEMNAME_>10004</_ITEMNAME_>
<_DEFINITION_>Lighting</_DEFINITION_>

# This tag is required because the publication is
# not set in the configuration file
<publication>GE Lighting</publication>

# This tag assigns a Template asset to the product
<template>Lighting_Detail</template>

# The rest of these tags set flex attribute values for the product
<price>5</price>
<sku>10004</sku>
<productdesc>F4T5/CW</productdesc>
<caseqty>24</caseqty>
<bulbshape>T</bulbshape>
<bulbsize>5</bulbsize>
<basetype>Miniature Bipin (G5)</basetype>
<colortemp>4100</colortemp>
<colorrenderingindex>60</colorrenderingindex>
<bulbimage>BLB-260.gif</bulbimage>
<baseimage>BLB-250.gif</baseimage>
<filamentimage></filamentimage>
<life>6000</life>
<voltage></voltage>
<wattage>4</wattage>
<cat1>Fluorescent</cat1>
<cat2>Preheat Lamps</cat2>

# this tag sets the immediate parents only
<ParentList>Preheat Lamps</ParentList>

</document>
```

The preceding source file sets several attribute values for the product and sets its immediate parent to Preheat Lamps. This parent must already exist.

21.5 Editing Flex Assets with XMLPost

You can edit the following information for flex assets and flex parent assets with XMLPost:

- The value of an attribute

- The asset's parents (either the flex asset's parents or the parent's parents)

You cannot edit attribute assets, flex definition assets, or flex parent definition assets with XMLPost.

To edit the attribute value for a flex asset, the source file needs to include only the name of the asset and the attribute that you want to change.

To edit the attribute value for a flex parent, you must provide the context of a flex asset. The source file must name the flex asset and can then reference just parent and the parent attribute that you want to change. But you must specify a flex asset for XMLPost to start with so that it can work its way through the family tree.

This section contains the following topics:

- Section 21.5.1, "Configuration Files for Editing Flex Assets"
- Section 21.5.2, "Source Files for Editing Flex Assets"

21.5.1 Configuration Files for Editing Flex Assets

There are two differences in the configuration file for editing a flex asset: the pagename argument and an additional tag and property.

Pagename Argument

The `pagename` argument must be set to: `OpenMarket/Gator/XMLPost/modifyData`.

For example:

```
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Gator/XMLPost/modifyData
```

You invoke XMLPost from the command line as usual, identifying the configuration file and the source files.

Additional Tag/Property

You can use the following optional tag and property when you are editing a flex asset:

- tag: `<_REMOVE_parentDefinitionName>`
- property: `post_REMOVE_parentDefinitionName`

It removes a parent from the flex asset.

21.5.2 Source Files for Editing Flex Assets

The source file for an edited flex asset does not need to include all the information for that asset, you only need to provide the information that you want to change. Any attributes that you do not specify are not modified in any way.

This section contains the following topics:

- Section 21.5.2.1, "Changing the Value of an Attribute"
- Section 21.5.2.2, "Removing an Attribute Value"
- Section 21.5.2.3, "Editing Parent Relationships"

21.5.2.1 Changing the Value of an Attribute

To change the value of an attribute, you specify the new attribute value in the source file. When XMLPost runs the import, it writes over the old value with the value provided in the source file.

The following sample source file changes two attribute values (bulbshape and bulbsize) for the GE Lighting product named 10004 that was defined in Section 21.4.4, "Sample Flex Asset Source File for addData."

```
<document>
<!-- predefined xml tags (required) -->
<_ASSET_>Products</_ASSET_>
<_ITEMNAME_>10004</_ITEMNAME_>
<_TYPE_>Lighting</_TYPE_>

<!-- attribute xml tags -->
<bulbshape>E</bulbshape>
<bulbsize>9</bulbsize>
</document>
```

21.5.2.2 Removing an Attribute Value

To remove an attribute value and leave it blank, code a line that names the attribute and specify `_EMPTY_` as the attribute's value.

For example:

```
<bulbsize>_EMPTY_</bulbsize>
```

You can also edit attribute values for parents. Let's say that the bulbsize attribute is set at the parent level. If that were the case, the following lines of code would set two parents and provide a value for bulbsize for each:

```
<_GROUP_SubCategory>All-Weather Lamps;Appliance Lamps
</GROUP_SubCategory>
<bulbsize>All-Weather Lamps=10:Appliance Lamps=8</bulbsize>
```

Option 1

This line of code clears the bulbsize for the All-Weather Lamps parent:

```
<bulbsize>All-Weather Lamps=_EMPTY_:Appliance Lamps=8</bulbsize>
```

Option 2

Alternatively, you could just use this line of code, without repeating the value for Appliance Lamps:

```
<bulbsize>All-Weather Lamps=_EMPTY_</bulbsize>
```

21.5.2.3 Editing Parent Relationships

You can use XMLPost to make the following edits to the parent relationships for a flex asset:

- Add another parent to the existing parents.
- Change a parent from one parent to another.

The `GROUP_parentDefinitionName` tag works differently than do the attribute tags.

- When you use an attribute tag, XMLPost writes the new value over the old value.
- When you use a `GROUP_parentDefinitionName` tag, XMLPost does not overwrite an old parent with a new parent, even when the parent definition name is the

same, it adds the new parent to the list of parents that the asset has, which may not be what you want.

To add another parent to the list of existing parents, include the line of code in the source file. For example:

```
<_GROUP_SubCategory>Blacklights</_GROUP_SubCategory>
```

If you want to remove a parent, use the <_REMOVE _> tag. Note that you must be careful not to remove a required parent unless you are replacing it.

```
<_REMOVE_Processor>Appliance Lamps</_REMOVE_Processor>
```

21.6 Deleting Assets with XMLPost

You can use XMLPost to delete any asset of any type. There are two requirements:

- Your configuration file must instruct XMLPost to call the `deleteData` element.

  For example:

  ```
  xmlpost.argname1: pagename
  xmlpost.argvalue1: OpenMarket/Gator/XMLPost/deleteData
  ```
- There are two required source file tags and configuration file properties:

  `<_ASSET_>/post_ASSET_` which identifies the asset type of the asset you want to delete.

  `<_ITEMNAME>/post_ITEMNAME_` which identifies the asset you want to delete.

When XMLPost uses this posting element, it changes the value in the `Status` column for that asset to `VO` for void. (It does not physically remove it from the database).

This section contains the following topics:

- Section 21.6.1, "Configuration Files for Deleting Assets"
- Section 21.6.2, "Source Files for Deleting Assets"

21.6.1 Configuration Files for Deleting Assets

Here is an example configuration file:

```
xmlpost.xmlfilenamefilter: .xml

xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 4
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Gator/XMLPost/deleteData
xmlpost.argname2: authusername
xmlpost.argvalue2: user_editor
xmlpost.argname3: authpassword
xmlpost.argvalue3: user
xmlpost.argname4: inifile
xmlpost.argvalue4: futuretense.ini

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdefpostlog.txt
xmlpost.deletefile: y

postpublication: y
post_ASSET_: y
```

```
post_ITEMNAME_: y
```

You invoke XMLPost from the command line as usual.

21.6.2 Source Files for Deleting Assets

The source files for deleting assets are short and simple. For example:

```
<document>
<_ASSET_>Products</_ASSET_>
<_ITEMNAME_>Pentium 90</_ITEMNAME_>
<publication>my publication</publication>
</document>
```

This code instructs XMLPost to delete a product asset named `Pentium 90` (it changes the status of `Pentium 90` to `VO`, for void).

22

Importing Flex Assets with the BulkLoader Utility

This chapter describes the BulkLoader utility, which you use to import flex assets during the during the initial setup of your WebCenter Sites system.

This chapter contains the following sections:

22.1 Overview of the BulkLoader Utility

The BulkLoader utility enables you to quickly extract large amounts of flex asset data in a user-defined way from your own data sources and import that data into the WebCenter Sites database on any of your systems (development, management, testing, or delivery).

The extraction mechanism is abstracted away using a Java interface that customers can implement. BulkLoader invokes methods on this interface to extract input data from your data sources. For backward functional and data compatibility, WebCenter Sites also includes an implementation of this Java interface so that BulkLoader will still be able to extract data from an external JDBC-compliant data source.

This section contains the following topics:

22.1.1 BulkLoader Features

Features in BulkLoader include the following:

- Support for a user-defined extraction mechanism, using a Java API. Users can provide a custom implementation of this extraction interface or use the built-in support for extracting from a JDBC data source.
- Support for inserts, voids, and updates of flex asset and group data.
- Support for incremental inserts, voids and updates.
- Performance improvements for higher throughput, using concurrent multi-threaded import operations while data extraction is in progress.
- Support for chunk (slice) processing of input data.
- Support for importing asset data that belongs to multiple flex families.
- Backward functional and data compatibility. Supports importing asset data from an external JDBC source.

22.1.2 How BulkLoader Works

The BulkLoader has been redesigned for higher performance, throughput, and scalability. Instead of reading all input data and then generating output SQL files, BulkLoader reads input data in chunks. As soon as each chunk is read, it is handed over to an import thread while the main BulkLoader thread goes back to read the next chunk. The import thread uses a direct JDBC connection to the WebCenter Sites database. In this way, reading and importing are done in parallel, thereby achieving higher throughput. For scalability, users can increase the number of BulkLoader import threads if the database computer's hardware has additional CPUs and an I/O configuration that supports higher concurrency.

The BulkLoader utility requires a configuration file containing parameters that specify the number of processing threads, the name of the Java class that implements the data extraction interface, commit frequency, the starting unique ID to be used as the asset ID, and more.

The following diagrams show a client-specific implementation and the built-in ready-to-use implementation supplied by WebCenter Sites:

Figure 22–1 Client-specific implementation of BulkLoader

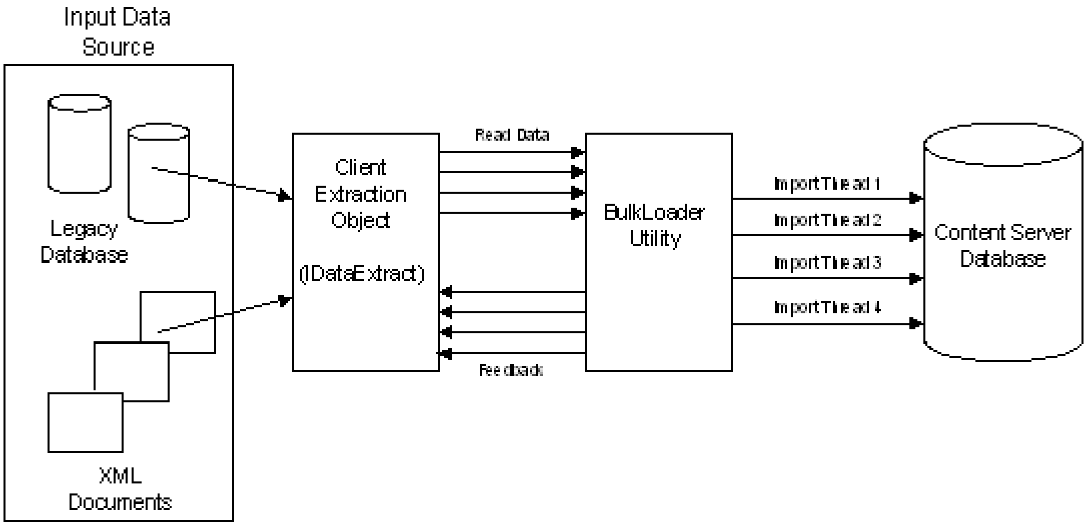
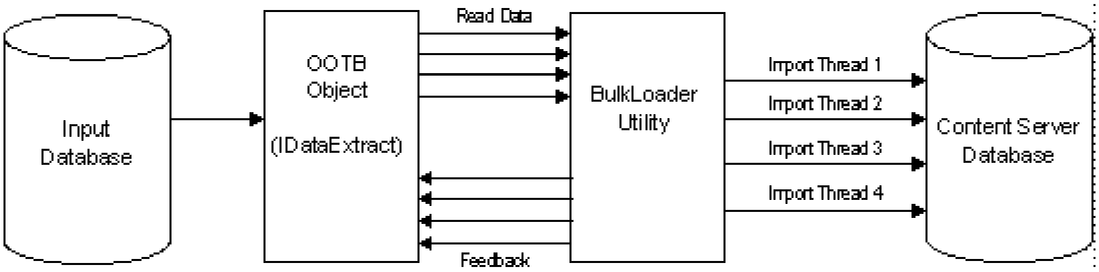


Figure 22–2 Built-in OOTB ("out-of-the-box") implementation of BulkLoader



22.1.3 Using the BulkLoader Utility

There are two ways to use the BulkLoader depending on how you supply input data to import into the WebCenter Sites database.

- If you want BulkLoader to import input data from an external JDBC data source, you provide input data in a flat table or view.
- If you want to provide your own way of supplying input data to the BulkLoader, you use a Java object that implements the extraction interface, IDataExtract.

> **Note:** For reference, sample BulkLoader code is provided on the WebCenter Sites installation medium, in the "Samples" folder. The same folder contains the readme.txt file that describes the sample files.

22.1.4 Importing Flex Assets from Flat Tables

This section describes the general procedure that you use to import flex assets with BulkLoader, followed by subsequent sections that describe each step in detail. Using this model, you can import new flex assets and parents, as well as void assets that were previously imported. This model also supports changing and deleting attribute values for existing assets.

The Basic Steps

The basic process of importing flex assets with the BulkLoader utility is as follows:

1. Use XMLPost to import the structural assets into the WebCenter Sites database on the management system. The structural flex assets are as follows: attribute editors, flex attributes, flex parent definitions, flex definitions, and flex parent assets.
2. Write a view or a stored procedure that gives you a view of the source database that you want to import into the WebCenter Sites database as a flat table. This flat table is your source table.
3. In the same source database, create a mapping table with two columns: one column that lists the names of the columns in the source file and the other column that lists the names that are used for those attributes in the WebCenter Sites database.
4. Code a configuration file that identifies the source table and the mapping table.
5. Put the configuration file on a system from which you have access to both the WebCenter Sites database on the management system, and to your source database.
6. Stop the application server on the management system.

7. Run the BulkLoader utility. BulkLoader will import the flex asset data and gives the feedback in a table named `bulk_feedback`, that has been created at the input data source.

8. Restart the application server on the management system.

9. Use the BulkApprover utility to approve all of the assets that were loaded

> **Note:** Because the BulkLoader utility is designed for speed, it does **not** check for the existence of the attributes or flex parent definitions or flex definitions. You must import all of the structural asset types before you run the BulkLoader utility.

Driver Requirements

The BulkLoader requires JDBC (or Java DataBase Connectivity) drivers, which are not provided by Oracle Corporation You must obtain JDBC drivers for both the source database and the destination database, that is, the WebCenter Sites database. If you have a source database that is ODBC-compliant, you can use a JDBC-ODBC bridge, which is included as part of the Java SDK.

Requirement for DB2

If you are using the DB2 database with your WebCenter Sites system, you must run the `usejdbc2.bat` file on the client computer before you can use BulkLoader. You only need to run the batch file once; then you can run BulkLoader as usual.

22.1.5 Step 1: Use XMLPost to Import Structural Assets

Use XMLPost and the `RemoteContentPost` posting element to import the structural assets into the WebCenter Sites database on the management system. Import assets of the following types:

- attribute editors
- flex attributes
- flex parent definitions
- flex definitions
- flex parent assets

For information about this step, see Section 21.3, "Importing the Structural Asset Types in the Flex Model."

22.1.6 Step 2: Create the Input Table (Data Source)

You must create input flat tables (data sources) for holding all new asset data and for holding update data. These are flat tables/views in which each row corresponds to a single flex asset item and each column corresponds to a flex attribute asset for the BulkLoader utility.

There is no requirement regarding the names of columns in the data source, but you must supply a separate mapping table, described in Section 22.1.7, "Step 3: Create the Mapping Table."

This section contains the following topics:

- Section 22.1.6.1, "Inserts"
- Section 22.1.6.2, "Updates"

22.1.6.1 Inserts

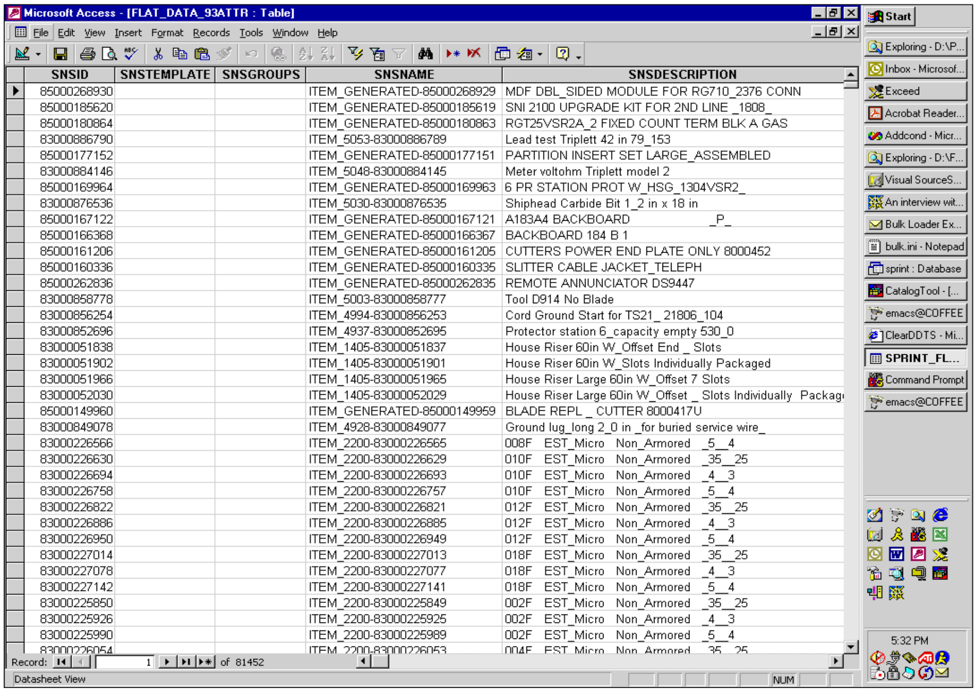
The name of the data source table is specified by the `inputTable` parameter in the configuration file.

The source table must also include the names of the following four columns, which you specify in the configuration file with the following properties:

- `inputTableTemplateColumn`: The name of the column in the source table that holds the names of the flex definitions.
- `inputTableNameColumn`: The name of the column in the source table that holds the names of the flex assets. The name of this column cannot exceed 64 characters.
- `inputTableDescriptionColumn`: The name of the column in the source table that holds the description of the flex assets.
- `inputTableGroupsColumn`: The name of the column in the source table that holds the names of the parent definitions. Each value in this column can include multiple flex parent definition names, separated by the `multivalueDelimeter` character, which is defined in the configuration file.

Note that you can optionally specify the name of the column that serves as a unique identifier for each input item, using the following parameter in the configuration file: `inputTableUniqueIdColumn`. If there is no value assigned for this parameter, BulkLoader will generate a unique identifier for each input item and store it in a mapping table (`bulkloader_ids`) in the WebCenter Sites database.

This is an example of a source table (input table):



SNSID	SNSTEMPLATE	SNSGROUPS	SNSNAME	SNSDESCRIPTION
85000268930			ITEM_GENERATED-85000268929	MDF DBL_SIDED MODULE FOR RG710_2376 CONN
85000185620			ITEM_GENERATED-85000185619	SNI 2100 UPGRADE KIT FOR 2ND LINE _1808_
85000180864			ITEM_GENERATED-85000180863	RGT25VSR2A_2 FIXED COUNT TERM BLK A GAS
83000886790			ITEM_5053-83000886789	Lead test Triplett 42 in 79_153
85000177152			ITEM_GENERATED-85000177151	PARTITION INSERT SET LARGE_ASSEMBLED
83000884146			ITEM_5048-83000884145	Meter voltohm Triplett model 2
85000169964			ITEM_GENERATED-85000169963	6 PR STATION PROT W_HSG_1304VSR2_
83000876536			ITEM_5030-83000876535	Shiphead Carbide Bit 1_2 in x 18 in
85000167122			ITEM_GENERATED-85000167121	A183A4 BACKBOARD _P_
85000166368			ITEM_GENERATED-85000166367	BACKBOARD 184 B 1
85000161206			ITEM_GENERATED-85000161205	CUTTERS POWER END PLATE ONLY 8000452
85000160336			ITEM_GENERATED-85000160335	SLITTER CABLE JACKET_TELEPH
85000262836			ITEM_GENERATED-85000262835	REMOTE ANNUNCIATOR DS9447
83000858778			ITEM_5003-83000858777	Tool D914 No Blade
83000856254			ITEM_4994-83000856253	Cord Ground Start for TS21_ 21806_104
83000852696			ITEM_4937-83000852695	Protector station 6_capacity empty 530_0
83000051838			ITEM_1405-83000051837	House Riser 60in W_Offset End _ Slots
83000051902			ITEM_1405-83000051901	House Riser 60in W_Slots Individually Packaged
83000051966			ITEM_1405-83000051965	House Riser Large 60in W_Offset 7 Slots
83000052030			ITEM_1405-83000052029	House Riser Large 60in W_Offset _ Slots Individually Packag
85000149960			ITEM_GENERATED-85000149959	BLADE REPL _ CUTTER 8000417U
83000849078			ITEM_4928-83000849077	Ground lug_long 2_0 in _for buried service wire_
83000226566			ITEM_2200-83000226565	008F EST_Micro Non_Armored _5__4
83000226630			ITEM_2200-83000226629	010F EST_Micro Non_Armored _35__25
83000226694			ITEM_2200-83000226693	010F EST_Micro Non_Armored _4__3
83000226758			ITEM_2200-83000226757	010F EST_Micro Non_Armored _5__4
83000226822			ITEM_2200-83000226821	012F EST_Micro Non_Armored _35__25
83000226886			ITEM_2200-83000226885	012F EST_Micro Non_Armored _4__3
83000226950			ITEM_2200-83000226949	012F EST_Micro Non_Armored _5__4
83000227014			ITEM_2200-83000227013	018F EST_Micro Non_Armored _35__25
83000227078			ITEM_2200-83000227077	018F EST_Micro Non_Armored _4__3
83000227142			ITEM_2200-83000227141	018F EST_Micro Non_Armored _5__4
83000225850			ITEM_2200-83000225849	002F EST_Micro Non_Armored _35__25
83000225926			ITEM_2200-83000225925	002F EST_Micro Non_Armored _4__3
83000225990			ITEM_2200-83000225989	002F EST_Micro Non_Armored _5__4
83000226054			ITEM_2200-83000226053	004F EST_Micro Non_Armored _35__25

Based on the column names in this source table, the source table properties in the corresponding configuration file would be set as follows:

```
inputTableTemplateColumn=SNSTEMPLATE
inputTableNameColumn=SNSNAME
inputTableDescriptionColumn=SNSDESCRIPTION
inputTableGroupsColumn=SNSGROUPS
```

22.1.6.2 Updates

If you want to update attribute data for existing assets, to add new parents or delete existing parents for existing assets, then you need to use the update parameter.

Use the `inputTableForUpdates` parameter in the configuration file to specify the name of the data source table. The source table must also include the names of the following three columns, which you specify in the configuration file with the following properties:

- `inputTableForUpdatesUniqueIdColumn`: The name of the column in the source table that uniquely identifies the flex asset or parent in the WebCenter Sites database.
- `inputTableForUpdatesDeleteGroupsColumn`: The name of the column in the source table that specifies a list of parents to be deleted for the current flex asset.
- `inputTableForUpdatesAddGroupsColumn`: The name of the column in the source table that specifies a list of parents to be added for the current flex asset.

BulkLoader interprets column values as follows when applying updates to the attributes:

- A null value in a specific attribute column indicates that the attribute for the current flex asset should be deleted. For example, a null value in the `deletegroups` column indicates that no parents need to be deleted. A null value in the `addgroups` column indicates that no parents need to be added.
- A non-null value indicates that the existing attribute value should be replaced with the given value. For example, a non-null value in the `deletegroups` column specifies a list of parents to be deleted. A non-null value for `addgroups` denotes the addition of new parents to a given flex asset.

22.1.7 Step 3: Create the Mapping Table

You must also create a mapping table for the BulkLoader utility, and it must have the following two columns:

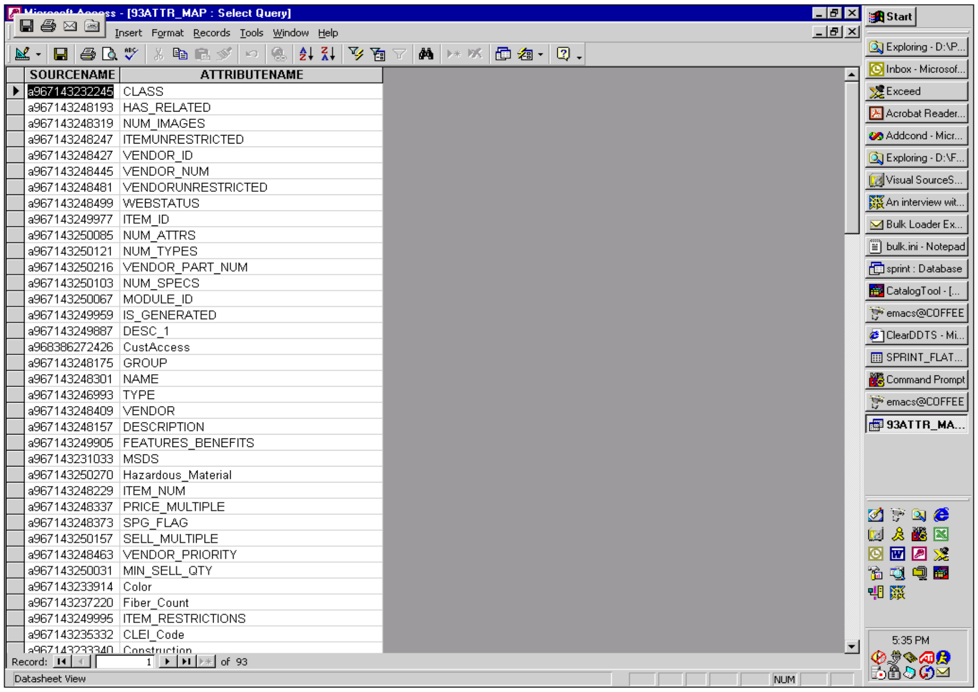
- A column that holds the names of the flex attribute columns in your flat data source
- A column that holds their corresponding names in the WebCenter Sites database

The mapping table provides a one-to-one correspondence between these two columns. For example, your source table might have a column of vendor names with an automatically generated name like `A96714328445` that maps to a product attribute asset named, simply, `VENDOR_ID`.

You include the following configuration file properties for the mapping table:

- `inputAttributeMapTable`: The name of the mapping table file
- `inputAttributeMapTableKeyCol`: The name of the column in the mapping table that lists the attribute names in the source table
- `inputAttributeMapTableValCol`: The name of the column that lists the corresponding attribute asset names in the WebCenter Sites database

The following is an example of a mapping table:


Microsoft Access - [93ATTR_MAP : Select Query]
Insert Format Records Tools Window Help
Record: 1 of 93
Datasheet View
NUM
Start
5:35 PM

SOURCENAME	ATTRIBUTENAME
a967143232245	CLASS
a967143248193	HAS_RELATED
a967143248319	NUM_IMAGES
a967143248247	ITEMUNRESTRICTED
a967143248427	VENDOR_ID
a967143248445	VENDOR_NUM
a967143248481	VENDORUNRESTRICTED
a967143248499	WEBSTATUS
a967143249977	ITEM_ID
a967143250085	NUM_ATTRS
a967143250121	NUM_TYPES
a967143250216	VENDOR_PART_NUM
a967143250103	NUM_SPECS
a967143250067	MODULE_ID
a967143249959	IS_GENERATED
a967143249887	DESC_1
a968386272426	CustAccess
a967143248175	GROUP
a967143248301	NAME
a967143246993	TYPE
a967143248409	VENDOR
a967143248157	DESCRIPTION
a967143249905	FEATURES_BENEFITS
a967143231033	MSDS
a967143250270	Hazardous_Material
a967143248229	ITEM_NUM
a967143248337	PRICE_MULTIPLE
a967143248373	SPG_FLAG
a967143250157	SELL_MULTIPLE
a967143248463	VENDOR_PRIORITY
a967143250031	MIN_SELL_QTY
a967143233914	Color
a967143237220	Fiber_Count
a967143249995	ITEM_RESTRICTIONS
a967143235332	CLEI_Code
a967143233340	Construction

Based on the column names in this source table, the source table properties in the corresponding configuration file would be set as follows:

```
inputAttributeMapTable=93ATTR_MAP
inputAttributeMapTableKeyCol=SOURCENAME
inputAttributeMapTableValCol=ATTRIBUTENAME
```

22.1.8 Step 4: Create the BulkLoader Configuration File

You configure the BulkLoader utility by creating a configuration file for it that has the properties described in this section. You can name the file anything you want.

You set the properties in the file according to the following syntax:

```
property=value
```

Note: All property names and values in the configuration file are case-sensitive.

This section contains the following topics:

- Section 22.1.8.1, "BulkLoader Configuration File Properties"
- Section 22.1.8.2, "Setting the initID Parameter"
- Section 22.1.8.3, "Example Configuration File"

22.1.8.1 BulkLoader Configuration File Properties

The following table describes properties in a BulkLoader configuration file:

Table 22–1 BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
`maxThreads`	Required	The maximum number of concurrent processing threads. This can be the number of database connections to the WebCenter Sites database server. Use as many threads as the number of CPUs on the database host. For a single CPU database host, set it to 2. Example: `4`
`dataSliceSize`	Required	Number of items retrieved in one read request; this number will also be processed by a single processing thread. Example: `2000`
`dataExtractionImplClass`	Required	User-specific Implementation class for data extraction API. Needs a constructor with (`String configFilename`) signature. The one mentioned here is a reference implementation class for backward compatibility. Data in flat tables. Default value (ready-to-use): `com.openmarket.gatorbulk.objects.DataExtractImpl`
`initId`	Required	Starting WebCenter Sites ID used the very first time BulkLoader operates; subsequently will use the value from `idSyncFile`. `Example: 800000000000` For more information, see Section 22.1.8.2, "Setting the initID Parameter."
`idSyncFile`	Required	Next available WebCenter Sites ID is saved in this file; updated during a BulkLoader session `Example: C:\FutureTense\BulkLoaderId.txt`
`idPoolSize`	Required	Each time BulkLoader needs to generate WebCenter Sites IDs, it collects this many IDs and caches in memory. A good estimate is (number of assets * average number of attributes *2). Example: `1000`

Table 22–1 (Cont.) BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
`commitFrequency`	Required	Number of flex asset groups to be part of a database transaction. Example: `100`
`outputJdbcDriver`	Required	The name of the JDBC driver class to access the WebCenter Sites database. The value here reflects the Oracle 9.0 driver. Example: `oracle.jdbc.driver.OracleDriver`
`outputJdbcURL`	Required	The JDBC URL. The following example value is a typical type2 oracle JDBC driver URL: `Jdbc:oracle:oci8:@foo`
`outputJdbcUsername`	Required	WebCenter Sites database user name
`outputJdbcPassword`	Required	WebCenter Sites database user password
`inputTable`	Required	Name of the flat, input table from which new asset data is inserted
`inputAttributeMapTable`	Required	Name of the mapping table that lists the source table columns and the corresponding attribute names.
`inputAttributeMapTableKeyCol`	Required	The name of the column in the mapping table that lists the source table column names. For example: `inputAttributeMapTableKeyCol=SOURCENAME`
`inputAttributeMapTableValCol`	Required	The name of the column in the mapping table that lists the corresponding attribute names. For example: inputAttributeMapTableValCol=ATTRIBUTENAME
`inputTableDescriptionColumn`	Required	The name of the column in the source table that contains the descriptions of the flex assets. For example: inputTableDescriptionColumn=SNSDESCRIPTION

Table 22–1 (Cont.) BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
`inputTableGroupsColumn`	Required	Name of the column in the source table that contains the names of parents. Each value can include several parents, separated by the `multivalueDelimeter` character, which is defined in the configuration file. For example: `inputTableGroupsColumn=SNS GROUP`
`inputTableNameColumn`	Required	The name of the column in the source table that contains the name of the product (or advanced article or advanced image) for each row. For example: inputTableNameColumn=SNS NAME
`inputTableTemplateColumn`	Required	The name of the column in the source table that contains the flex definitions. For example: `inputTableTemplateColumn=SNSTEMPLATE`
`createdby`	Required	The user name that you want to be entered in the `createdby` field for your flex assets. For example: `createdby=editor`
`multivalueDelimeter`	Required	The delimiter that separates multiple attribute values. The default character is the semicolon (;). For example: `multivalueDelimiter=;`
`siteName`	Required	The name of the site. All products will behave as if they were created under this site. For example: `siteName=GE Lighting`
`status`	Required	The status code for all imported flex assets. You should set this to `PL` for imported. For example: `status=PL`
`tableProducts`	Required	Name of the flex asset type as defined in the WebCenter Sites database.

Table 22–1 (Cont.) BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
`inputTableUniqueIdColumn`	Required	Name of the column in the source table that serves as a unique identifier when importing a new flex asset. This will be used for any subsequent updates and void operations. Leave this value empty, if you want the BulkLoader to generate unique identifiers for you
`targetName`	Required	Name of the publish target, as defined in WebCenter Sites
`renderTemplate`	Optional	Name of the template used for rendering flex assets (deprecated).
`inputFeedbackTable`	Required	Name of the table that BulkLoader creates and uses for recording the processing feedback for every input item that was processed. Note that this table is created in the input data source.
`inputTableForUpdates`	Optional	Needed only if an update action is specified when running the BulkLoader utility. Otherwise, this can be an empty value. This is the name of the source table that contains attributes and parents that need updates.
`inputTableForUpdatesUniqueIdColumn`	Optional	Needed only if an update action is specified when running the BulkLoader utility. This is the name of the column in the source table that specifies a unique identifier for the flex asset.
`inputTableForUpdatesDeleteGroupsColumn`	Optional	Needed only if update action is specified and you have one or more flex assets that need one or more parents to be deleted. This is the name of the column in the source table that specifies the list of parents to be deleted.
`inputTableForUpdatesAddGroupsColumn`	Optional	Needed only if an update action is specified and you have one or more flex assets that need one or more parents to be added. This is the name of the column in the source table that specifies a list of parents to be added.

Table 22–1 (Cont.) BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
`inputLimitRows`	Optional	Needed only for testing. Limits the number of input items processed for each action (insert, void, or update).
`updatedby`	Optional	The user name that you want to be entered in the `createdby` field for your flex assets. For example: `updateby=editor`
`updatedstatus`	Optional	The status code for all updated flex assets. This must be set to `ED`. For example: `updatestatus=ED`

22.1.8.2 Setting the initID Parameter

The `initID` parameter is the seed value that the BulkLoader starts at and increments from when creating a unique asset ID for each asset. You must choose a seed value number that allows the BulkLoader to create a contiguous block of ID numbers that cannot cause ID conflicts with existing (or future) asset ID numbers that are generated by WebCenter Sites.

Currently, WebCenter Sites starts at 1 trillion for the asset IDs that it creates. To be sure that you won't have conflicts, select a number low enough that when the BulkLoader utility is done, the highest ID number is under 900,000,000,000.

The BulkLoader creates one asset for each row/column value in the data source table. Each output table row requires its own unique asset ID.

Use these guidelines to determine the approximate number of asset IDs that are created by the BulkLoader utility:

- Five rows for each flex asset, plus
- Two rows per attribute for each flex asset

For example, if your data source table contains the following:

- 10,000 product assets
- 20 attributes per product (as determined by the product definition)
- 10 inherited attributes per product (as determined by the product parent definitions)

Then you need to allow for the following number of IDs:

(5 x 10,000) + (2 x 30 x 10,000) = 50,000 + 600,000 = 650,000 asset IDs

If your `initID` value is 800,000,000,000, then the BulkLoader creates ID numbers ranging from 800,000,000,000 to approximately 800,000,650,000.

22.1.8.3 Example Configuration File

The following is an example of the BulkLoader configuration file that you could use with the GE Lighting sample site.

```
# New BulkLoader configuration for backward compatibility
#
```

```
# input datasource configuration
inputJdbcDriver=sun.jdbc.odbc.JdbcOdbcDriver
inputJdbcURL=jdbc:odbc:access-db-conn
inputJdbcUsername=
inputJdbcPassword=
#
# Source tables
#
inputTable=PRD_FLAT_50000
inputAttributeMapTable=PRD_FLAT_ATTRIBUTE_MAP
inputAttributeMapTableKeyCol=SOURCENAME
inputAttributeMapTableValCol=ATTRIBUTENAME
#
# input column names
#
inputTableTemplateColumn=CCTemplate
inputTableNameColumn=CCName
inputTableDescriptionColumn=CCDescription
inputTableGroupsColumn=CCGroups
#
# WebCenter Sites database
#
#  This database is always used for looking up Attributes, #  Product Types and
Product Group Types. #  Data is imported into this database.
#
outputJdbcDriver=oracle.jdbc.driver.OracleDriver
outputJdbcURL=jdbc:oracle:oci8:@foo
outputJdbcUsername=csuser
outputJdbcPassword=csuser
#
# Data-specific settings
#
siteName=GE Lighting
targetName=Mirror Publish to burst37
initId=800000000000
createdby=user_designer
status=PL
renderTemplate=CLighting Detail
MAX_ATTRIBUTES=100
multivalueDelimiter=;
commitFrequency=50
#
# The following denotes the flex asset type that we are importing.
tableProducts=Products
#
# Additional information needed for BulkLoader
maxThreads=2
# dataSliceSize 0 means read all input data in one slice.
dataSliceSize=500
dataExtractionImplClass=com.openmarket.gatorbulk.objects.DataExtractImpl
idSyncFile=C:\\FutureTense50\\bulk_uniqueid.dat
idPoolSize=50000
# For inserts
inputTableUniqueIdColumn=
inputFeedbackTable=bulk_feedback
# For updates
inputTableForUpdates=prod_flat_2_upd
inputTableForUpdatesUniqueIdColumn=input_id
inputTableForUpdatesDeleteGroupsColumn=CCGroups
inputTableForUpdatesAddGroupsColumn=
inputLimitRows=1000
############################################################
```

22.1.9 Step 5: Run the BulkLoader Utility

Before you begin, be sure that you have the appropriate JDBC drivers for **both** your **source** database and your **target** WebCenter Sites database.

Complete the following steps:

1. Put the configuration file on a system from which you have access to both the WebCenter Sites database on the management system, and to your source database.
2. Stop the application server on the management system.
3. Enter the following command, all on a single line, with paths that are appropriate for your installation:

   For UNIX

   ```
   java -ms16m -mx256m -cp <path to gatorbulk.jar>/gatorbulk.jar:<path to
   commons-logging-1.1.1.jar>/commons-logging-1.1.1.jar:<path to
   cs-core.jar>/cs-core.jar:<path to cs.jar>/cs.jar:<path to
   commons-codec-1.4.jar>/commons-codec-1.4.jar:<path to
   httpclient-4.1.2.jar>/httpclient-4.1.2.jar:<path to
   httpcore-4.1.2.jar>/httpcore-4.1.2.jar:<path to
   httpmime-4.1.2.jar>/httpmime-4.1.2.jar:<path to
   jtds-1.2.2.jar>/jtds-1.2.2.jar:<path to
   commons-lang-2.4.jar>/commons-lang-2.4.jar
   com.openmarket.gatorbulk.objects.BulkLoader config=bulkloader.ini
   action=<insert|void|update> validate=<yes|no>
   ```

   For Windows

   ```
   java -ms16m -mx256m -cp <path to gatorbulk.jar>\gatorbulk.jar;<path to
   commons-logging-1.1.1.jar>\commons-logging-1.1.1.jar;<path to
   cs-core.jar>\cs-core.jar;<path to cs.jar>\cs.jar;<path to
   commons-codec-1.4.jar>\commons-codec-1.4.jar;<path to
   httpclient-4.1.2.jar>\httpclient-4.1.2.jar;<path to
   httpcore-4.1.2.jar>\httpcore-4.1.2.jar;<path to
   httpmime-4.1.2.jar>\httpmime-4.1.2.jar;<path to
   jtds-1.2.2.jar>\jtds-1.2.2.jar;<path to
   commons-lang-2.4.jar>\commons-lang-2.4.jar
   com.openmarket.gatorbulk.objects.BulkLoader config=bulkloader.ini
   action=<insert|void|update> validate=<yes|no>
   ```

   Note that the action parameter specifies what BulkLoader needs to do: `insert`, `void`, or `update`. Setting the validate parameter to `yes` makes BulkLoader do extra validations during updates and voids. You may also need to increase the memory for the JVM, depending on the size of your input data.

4. Examine the screen output to be sure that the BulkLoader utility was able to connect to the appropriate database.

22.1.10 Step 6: Review Feedback Information

After the BulkLoader utility completes an operation, review the feedback information in the `bulk_feedback` table that is located in your input data source. That table contains information about all the input items that BulkLoader processed.

After reviewing that information, take any corrective actions that might be necessary. If you modify any of your input data, you should run BulkLoader again to verify that the errors were corrected.

22.1.11 Step 7: Approve and Publish the Assets to the Delivery System

Use the BulkApprover utility to approve the assets that you just loaded. For instructions on how to use BulkApprover, see Section 22.3.2, "Using BulkApprover."

22.2 Importing Flex Assets Using a Custom Extraction Mechanism

Sometimes users need alternative mechanisms to provide input asset data to BulkLoader. In such cases, the data may have to be gathered from multiple types of sources, such as XML documents, files, and legacy databases. To accomplish that, users can implement their own mechanism to provide data to BulkLoader, using the Java interface `com.openmarket.bulkloader.interfaces.IDataExtract`, which is provided with WebCenter Sites.

A user can implement a Java object supporting `IDataExtract` and specify the Java object in the BulkLoader configuration file. BulkLoader will then invoke methods on this interface to initialize a read request, to repetitively read chunks of input data and then signal the end of the read request. This interface also has a method that provides import feedback from the BulkLoader utility, which can be used by the input provider to know the status of import and know any errors that may occur during import.

There are three Java interfaces that can help users with custom implementations of `IDataExtract`:

- `IDataExtract`: Required for any custom extraction.
- `IPopulateDataSlice`: Provides data to the BulkLoader utility. A container object supporting this interface is created by BulkLoader and passed into the client.
- `IFeedback`: Provides the status of each input item that has been processed by the BulkLoader. A feedback object that is created and populated by BulkLoader import thread is passed into the client.

These interfaces are described in the following sections.

> **Note:** When you implement a custom extraction method, you use the same previously described procedures to run BulkLoader.

This section contains the following topics:

- Section 22.2.1, "IDataExtract Interface"
- Section 22.2.2, "IPopulateDataSlice"
- Section 22.2.3, "IFeedback Interface"

22.2.1 IDataExtract Interface

This interface is **required** for any custom extraction.

The following is sample code that implements this interface.

```
com.openmarket.gatorbulk.interfaces.IDataExtract

package com.openmarket.gatorbulk.interfaces;
import java.util.Iterator;
```

```
/**
 * To be implemented by input data provider.
 * Interface for extracting data from an input source
 * for BulkLoader.
 * BulkLoader loads an object supporting this interface and invokes
 * the GetNextInputDataSet() method on this interface repeatedly to
 * fetch data in batches.
 */

public interface IDataExtract {

  public  final int HAS_DATA    = 100;
  public  final int NO_DATA = 101;

  public  final int SUCCESS = 0;
  public  final int ERROR = -1;

  public  final int INSERT_ASSETS = 1000;
  public  final int VOID_ASSETS = 1010;
  public  final int UPDATE_ASSETS = 1020;
  public  final int NONE_ASSETS = 1030;

  /**
   * Begin requesting input data; tells the client to
   * start the database query, get a cursor, etc.
   * @param requestType
   *    IDataExtract.INSERT_ASSETS,
   *    IDataExtract.VOID_ASSETS,
   *    IDataExtract.UPDATE_ASSETS
   * @param sliceOrNot  true/false
   * true - if data will be requested in batches
   * false - data will be requested all in one attempt
   * @param sliceSize  >0 number of rows to be
   * retrieved in one data set
   * @return none
   * @exception java.lang.Exception
   */

  public void InitRequestInputData(int requestType,
  boolean sliceOrNot, int sliceSize) throws Exception ;

  /**
   * Get a set/slice of input data records.
   * @param dataSlice object to be populated using the
   * methods from IPopulateDataSlice
   * @return IDataExtract.HAS_DATA when dataSlice has some data,
   *    IDataExtract.NO_DATA when there is no data,
   *    IDataExtract.ERROR when there is an error
   * @exception java.lang.Exception
   */

  public int  GetNextInputDataSet(IPopulateDataSlice dataSlice)
  throws Exception;

  /**
   * Signal the end of extracting data for given request type
   * @param requestType
   *    IDataExtract.INSERT_ASSETS,
   *    IDataExtract.VOID_ASSETS,
   *    IDataExtract.UPDATE_ASSETS
   * @return none
   * @exception java.lang.Exception
   */

  public void EndRequestInputData(int requestType)
    throws Exception;

  /**
   * Update the client as to what happened to input data
   * processing. Note that this method would be called by multiple
   * threads, with each thread passing its own IFeedback
   * handle. The implementor of this method should write
   * thread-safe code.
   * @param requestType
   *    IDataExtract.InsertAsset,
   *    IDataExtract.VoidAsset,
   *    IDataExtract.UpdateAsset
   * @param processingStatus - An object containing processing
   * status for all items in one dataset. The implementor of this
   * interface should invoke the IFeedback interface
   * methods on processingStatus to get status for individual
   * rows. This method will be invoked by multiple BulkLoader
   * threads, so make sure this method is implemented in a
   * thread-safe way.
   * @return none
   * @exception java.lang.Exception
   */

  void UpdateStatus(int requestType, IFeedback  processingStatus)
    throws Exception;
}
```

Implementation Notes for IDataExtract

The Java object implementing `IDataExtract` needs to have a constructor with a string parameter. BulkLoader will pass the name of its configuration file to the constructor when instantiating this object.

The method `UpdateStatus(..)` is invoked by multiple BulkLoader threads, so the implementation of this method should be thread-safe.

The following table lists and describes the configuration parameters for the BulkLoader utility when using custom data extraction method:

Table 22–2 Configuration Parameters for BulkLoader

Property Name	Required/ Optional	Comments
`maxThreads`	Required	The maximum number of concurrent processing threads. This can be the number of database connections to the WebCenter Sites database server. Use as many threads as the number of CPUs on the database host. For a single CPU database host, set it to 2. Example: `4`

Table 22–2 (Cont.) Configuration Parameters for BulkLoader

Property Name	Required/ Optional	Comments
`dataSliceSize`	Required	Number of items retrieved in one read request; this number will also be processed by a single processing thread. Example: `2000`
`dataExtractionImplClass`	Required	User-specific Implementation class for data extraction API. Needs a constructor with (String configFilename) signature. The one mentioned here is a reference implementation class for backward compatibility. Data in flat tables. Default value (ready-to-use): `com.openmarket.gatorbulk.objects.DataExtractImpl`
`initId`	Required	Starting WebCenter Sites ID used the very first time BulkLoader operates; subsequently will use the value from `idSyncFile`. `Example: 800000000000`
`idSyncFile`	Required	Next available WebCenter Sites ID is saved in this file; updated during a BulkLoader session `Example: C:\FutureTense\BulkLoaderId.txt`
`idPoolSize`	Required	Each time BulkLoader needs to generate WebCenter Sites IDs, it collects this many IDs and caches in memory. A good estimate is (number of assets * average number of attributes *2). Example: `1000`
`commitFrequency`	Required	Required. Specifies when "`COMMIT`" statements will be inserted into the generated SQL file. A value of 0 means that "`COMMIT`" statements will be inserted every 50 lines (the default); any positive integer specifies the number of lines between each "`COMMIT`" statement. For example: `commitFrequency=5` (A `COMMIT` statement will be inserted for every 5 lines of SQL code.)

Table 22–2 (Cont.) Configuration Parameters for BulkLoader

Property Name	Required/ Optional	Comments
`outputJdbcDriver`	Required	The name of the JDBC driver class to access the WebCenter Sites database. The value here reflects the Oracle 9.0 driver. Example: `oracle.jdbc.driver.OracleDriver`
`outputJdbcURL`	Required	The JDBC URL. The following example value is a typical type2 oracle JDBC driver URL: `Jdbc:oracle:oci8:@foo`
`outputJdbcUsername`	Required	WebCenter Sites database user name
`outputJdbcPassword`	Required	WebCenter Sites database user password

22.2.2 IPopulateDataSlice

The following is sample code that implements this interface:

```
com.openmarket.gatorbulk.interfaces.IPopulateDataSlice

package com.openmarket.gatorbulk.interfaces;

import java.sql.Timestamp;

/**
 * To be implemented by Oracle Corporation
 * Interface to populate a dataSlice by the client.
 * BulkLoader creates an object implementing this interface and then
 * hands it over to the client, which uses this interface's methods
 * to populate that object with input data records.
 */

public interface IPopulateDataSlice {

/**
 * Creates a new input data object to hold all the data for a
 * flex asset and makes it the current object. This method is
 * invoked repetitively to populate this object with flex asset
 * input data. Each invocation is to be followed by Set..()
 * methods and AddAttribute..() methods to supply data for one
 * flex asset.
 */

public void AddNewRow();

/**
 * Specify a unique identifier for flex asset input data
 * @param id  user-specific unique identifier
 * @exception java.lang. Exception thrown if any unique-id
 * validation is enabled.
 */

public voidSetAssetUniqueId(String id);
```

```
/**
 * Specify the name of the site with which the current flex
 * asset is created or to be created under.
 * @param sitename  name of the site
 */

public void SetSiteName(String sitename);

/**
 * Set the asset type for the flex asset.
 * @param flexAssetType asset type as defined in WebCenter Sites system
 */

public void SetFlexAssetType(String flexAssetType);

/**
 * Specify the name of the parent for the current flex asset.
 * Use this method repeatedly to add a list of parent names.
 * @param groupName  name of a parent that the current asset
 * inherits some of its attributes from.
 */

public void AddParentGroup(String groupName);

/**
 * Specify the name of the parent to be deleted for the current
 * flex asset.
 * Use this method repeatedly to add a list of parent names.
 * @param groupName - name of a parent that the current asset
 * inherited some of its attributes from.
 */

public void AddParentGroupForDelete(String groupName);

/**
 * Specify definition asset name for the current flex asset.
 * @param definitionAssetName name of the flex definition asset
 */

public void SetDefinitionAssetName(String definitionAssetName);

/**
 * Specify name of the flex asset.
 * @param name - name of the flex asset.Should be unique in
 * a flex asset family
 */

public voidSetAssetName(String name);

/**
 * Specify description for the flex asset
 * @param description  description
 */

public void SetAssetDescripiton(String description);

/**
 * Specify WebCenter Sites username with which this flex asset is being
 * processed
 * @param username WebCenter Sites username
 */

public void SetCreatedByUserName(String userName);

/**
 * Set WebCenter Sites status code for this asset
 * @param status
 */

public void SetAssetStatus(String status);

/**
 * Set template name
 * @param template  WebCenter Sites template name
 */

public void SetRenderTemplateName(String template);

/**
 * Specify startMenu for workflow participation
 * @param startMenuName  start menu name for this flex asset
  */

public void SetStartMenuName(String startMenuName);

/**
 * WebCenter Sites
 * Specify publish approval target name
 * @param targetName  approval target name
 */

public void SetApprovalTargetName(String targetName);

/**
 * Add a name/value pair to specify a WebCenter Sites attribute
 * of type 'text' for the current input object.
 * Call this method more than once, if this is a
 * multi-valued attribute.
 * @param attrName  attribute name as defined in the WebCenter Sites
 * database for the flex asset being processed
 * @param value java.lang.String
 */

public void AddAttributeValueString(String attrName, String value);

/**
 * Add a name/value pair to specify a WebCenter Sites attribute
 * of type 'date' for the current input object.
 * Call this method more than once, if this is a
 * multi-valued attribute.
 * @param attrName  attribute name as defined in the WebCenter Sites
 * database for the flex asset being processed
 * @param value  java.sql.Timestamp
 */

public void AddAttributeValueDate(String attrName, Timestamp value);

/**
 * Add a name/value pair to specify an attribute for the current
 * input object.
 * Call this method more than once, if this is a multi-valued *attribute
 * @param attrName  attribute name as defined in WebCenter Sites database
 * for the flex asset being processed
 * @param value  java.lang.Double
 */

public void AddAttributeValueDouble(String attrName, Double value);

/**
 * Add a name/value pair to specify a WebCenter Sites attribute
 * of type 'money' for the current input object
 * Call this method more than once if this is a
 * multi-valued attribute
 * @param attrName  attribute name as defined in WebCenter Sites database
 * for the flex asset being processed
 * @param value  java.lang.Float
 */

public void AddAttributeValueFloat(String attrName, Float value);

/**
 * Add a name/value pair to specify a WebCenter Sites attribute
 * of type 'int' for the current input object.
 * Call this method more than once, if this is a
 * multi-valued attribute.
 * @param attrName  attribute name as defined in WebCenter Sites
 * database for the flex asset being processed
 * @param value  java.lang.Integer
 */

public void AddAttributeValueInteger(String attrName,
  Integer value);

/**
 * Add a name/value pair to specify any WebCenter Sites attribute for the
 * current input object.
 * Use the datatype-specific methods above instead of this
 * method, as this one is for
 * supporting any other new types in future.
 * Call this method more than once, if this is a
 * multi-valued attribute
 * @param attrName  attribute name as defined in the WebCenter Sites
 * database for the flex asset being processed.
 * @param value  java.lang.Object
 */

public void AddAttributeValueObject(String attrName,
  Object value);
}
```

22.2.3 IFeedback Interface

The following is sample code that implements this interface:

```
com.openmarket.gatorbulk.interfaces.IFeedback

package com.openmarket.gatorbulk.interfaces;
```

```
import java.util.Iterator;

/**
 * To be implemented by Oracle Corporation
 * Interface for the BulkLoader client to get the status of
 * processing request to insert/void/update flex assets.
 */

public interface IFeedback {
  public final int ERROR=-1;
  public final int SUCCESS=0;
  public final int NOT_PROCESSED=1;

/**
 * Get a list of keys from input data slice that has
 * been processed
 * @return java.util.Iterator
 */

public Iterator GetInputDataKeyValList();

/**
 * Get WebCenter Sites asset ID for given input identifier
 * @param inputDataKeyVal key value of the unique identifier
 * in the input data record
 * @return Get the associated asset ID from the WebCenter Sites system.
 * null if missing.
 */

public String GetWebCenter SitesAssetId(String inputDataKeyVal);

/**
 * Get the processing status for the input data record
 * identified by a key
 * @param inputDataKeyVal key value of the unique identifier
 * column in the input data record
 * @return ERROR - processed but failed, SUCCESS - processed
 * successfully, NOT_PROCESSED - unknown item or not part of
 * the processing dataset.
 */

public int GetStatus(String inputDataKeyVal);

/**
 * Get the associated error message for a given key,
 * unique identifier in input data
 * @param inputDataKeyVal  unique identifier for input data
 * @return error message, if GetStatus() returned ERROR
 * or NOT_PROCESSED
 */

public String GetErrorDescription(String inputDataKeyVal);
}
```

22.3 Approving Flex Assets with the BulkApprover Utility

BulkApprover is a utility that quickly and easily approves large numbers of flex assets that you have loaded into the system via BulkLoader.

BulkApprover can perform the following tasks:

- Notify the approval system of all updates and deletions that were made during a previous BulkLoader session.
- Approve all newly loaded flex assets for one or more publishing targets.
- Mark all newly loaded flex assets as "published" for a given mirror destination, without actually publishing the assets. For example, to bypass a long mirror publishing session, you can instead copy selected assets from the content management database to a mirror destination on the delivery system and have BulkApprover mark the assets as "published" to the mirror destination.

> **Note:** Only users with the `xceladmin` role can run BulkApprover.

This section contains the following topics:

- Section 22.3.1, "Configuring BulkApprover"
- Section 22.3.2, "Using BulkApprover"

22.3.1 Configuring BulkApprover

Before running BulkApprover for the first time, you must create a configuration file for the utility. You can create a separate `BulkApprover.ini` file on a system that can access WebCenter Sites's database, or you can append the BulkApprover configuration information to one of BulkLoader's `.ini` files.

The following table lists required and optional configuration parameters:

Table 22–3 BulkApprover Configuration Parameters

Parameter	Description
`bulkApprovalURL` (Required)	The URL on the host server that has the data imported with BulkLoader. The correct value is as follows: `http://<myServer>/cs/ContentServer?pagename=OpenMarket/Xcelerate/Actions/BulkApproval` where `<myServer>` is the name of the host server.
`adminUserName` (Required)	The WebCenter Sites username of a user with the `xceladmin` role.
`adminUserPassword` (Required)	The password for `adminUserName`.
`approvalTargetList` (Required)	A list of destinations that the assets are to be approved for. For destination names, see the **Publish** option on the **Admin** tab, or the name column of the `pubtarget` table. Separate each destination with the delimiter that you specify in the `multiValueDelimiter` parameter. The syntax is: name1<multiValueDelimiter>name2<multiValueDelimiter>name3

`multiValueDelimiter`
`(Required)`
	A delimiter that you select. Use this delimiter to separate the approval targets that you specify in the `appovalTargetList` parameter.

Table 22–3 (Cont.) BulkApprover Configuration Parameters

Parameter	Description

`assetIdSqlFilter`
(Optional)
	A statement that can be appended to a SQL `WHERE` clause in order to filter asset IDs.
	For example:
	`asset_id%20=0`
	or
	`asset_id%20!=0`

`debug` (Optional)	Turn BulkApprover debugging on and off. A value of `true` turns debugging on. Leave this parameter blank for no debugging. Debug messages are written to the file specified in the `output_file` parameter of the command line.
`assetschunksize` (Optional)	Specifies the number of assets that are approved in a single transaction. For example, setting this property to `20` means that assets will be approved in groups of `20`. Setting this property helps prevent session timeouts. Default value: `25`
`outputJdbcDriver` (Required)	The name of the JDBC driver class to access the WebCenter Sites database. Example: `oracle.jdbc.driver.OracleDriver`
`outputJdbcURL` (Required)	The JDBC URL. The following example value is a typical type 2 oracle JDBC driver URL: `Jdbc:oracle:oci8:@foo`
`outputJdbcUsername` (Required)	WebCenter Sites database user name.
`outputJdbcPassword` (Required)	WebCenter Sites database user password.

Sample BulkApprover.ini File

The following sample shows the proper syntax of the BulkApprover configuration parameters:

```
bulkApprovalURL=http://MyServer/cs/ContentServer?pagename=OpenMarket/Xcelerate/Act
ions/BulkApproval
adminUserName=admin
adminUserPassword=xceladmin
approvalTargetList=Dynamic;;;;;;testdest
multiValueDelimiter=;;;;;;
assetIdSqlFilter=
assetsChunkSize=3
debug=true
outputJdbcDriver=oracle.jdbc.driver.OracleDriver
outputJdbcURL=jdbc:oracle:thin:@19zln:1521:MyServer
#outputJdbcUsername=izod10
outputJdbcUsername=ftuser3
outputJdbcPassword=ftuser3
```

22.3.2 Using BulkApprover

After you have configured the BulkApprover utility, you can use it to approve assets that were imported into the database via the BulkLoader utility.

BulkApprover runs from the command line. Parameters are described in the following table and included in the example below:

Table 22–4 BulkApprover Parameters

Command-Line Parameter	Description
`config`	The name of the file that stores your BulkApprover configuration information; for example, `BulkApprover.ini`.
`action`	The action or actions that you want BulkApprover to perform. You must set this parameter to `notify` or `approve`. Add other actions, as necessary. To have BulkApprover perform multiple actions, name the actions in a comma-separated list. **Valid values**: ■ `notify`: Notifies the approval system about all updates and voids processed during a previous BulkLoader session. ■ `approve`: Instructs BulkApprover to approve all of the assets that it processes for the given publishing destination(s). ■ `mark_publish`: Marks all of the assets that it processes as "published" to a given mirror destination, without actually publishing the assets. Specify the publishing targets in the `approvalTargetList` parameter in the BulkApprover configuration file (see Section 22.3.1, "Configuring BulkApprover."). If you do not want the assets marked as published, do not include this parameter.
`output_file`	The name of the log file that contains all output from the server; for example, `bulkapprover.txt`.

To run BulkApprover, set paths as shown in the following example.

This example uses Windows syntax, for Unix-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

```
java -ms16m -mx256m -cp <path to gatorbulk.jar>\gatorbulk.jar;<path to
commons-logging-1.1.1.jar>\commons-logging-1.1.1.jar;<path to
cs-core.jar>\cs-core.jar;<path to cs.jar>\cs.jar;<path to
commons-codec-1.4.jar>\commons-codec-1.4.jar;<path to
httpclient-4.1.2.jar>\httpclient-4.1.2.jar;<path to
httpcore-4.1.2.jar>\httpcore-4.1.2.jar;<path to
httpmime-4.1.2.jar>\httpmime-4.1.2.jar:<path to
wem-sso-api-1.2.jar>\wem-sso-api-1.2.jar;<path to
apache-mime4j-0.5.jar>\apache-mime4j-0.5.jar;<path to
jtds-1.2.2.jar>\jtds-1.2.2.jar;<path to servlet-api.jar>\servlet-api.jar;<path to
commons-lang-2.4.jar>\commons-lang-2.4.jar
com.openmarket.gatorbulk.objects.BulkApprover config=bulkapprover.ini
action=<notify|approve|mark_publish> output_file=<out.txt>
```

23

Creating Template, CSElement, and SiteEntry Assets

The CSElement, Template, and SiteEntry asset types provide the pagelets and elements that build your online sites. They are asset representations of page names and elements, the components that WebCenter Sites uses to generate pages.

When you create a CSElement asset, you code an element. When you create a SiteEntry asset, you name a page. When you create a template, you do both: you code an element and you name a page.

This chapter describes these three asset types and provides information about how to create them. Additional information about coding templates and CSElements is included in Chapter 28, "Coding Elements for Templates and CSElements."

This chapter contains the following sections:

- Section 23.1, "Understanding Template, CSElement, and SiteEntry Assets"
- Section 23.2, "Pages, Pagelets, and Elements"
- Section 23.3, "CSElement, Template, and SiteEntry Assets"
- Section 23.4, "Creating Template Assets"
- Section 23.5, "Creating CSElement Assets"
- Section 23.6, "Creating SiteEntry Assets"
- Section 23.7, "Managing Template, CSElement, and SiteEntry Assets"
- Section 23.8, "Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic"

23.1 Understanding Template, CSElement, and SiteEntry Assets

This chapter contains procedures for creating Template, CSElement, and SiteEntry assets.

- Template assets are classified as typed or typeless depending on whether they apply to a single asset type or no asset type.
- If you are using SiteLauncher (to replicate sites or share Template and CSElement assets), WebCenter Sites requires element logic to indirectly refer to assets, asset types, attribute names, and template names. To this end, the WebCenter Sites interface introduces the **Map** screen (for example, Section 23.4.2.5, "Step 5: Configure the Map"); the API introduces the `render:lookup` tag.

Using the **Map** screen, you assign an alias to each value. You can then hardcode the aliases in the element logic and use the `render:lookup` tag to retrieve the actual values from the aliases at runtime.

- The **Cache Rules** field has been simplified to reduce errors. Template developers can choose cached, uncached, or advanced. Selecting **Advanced** allows developers to set caching rules individually for WebCenter Sites and Satellite Server.
- A new tag, `calltemplate`, was introduced to invoke templates in a way that simplifies the template writing process.
- The **PageCriteria** field has been renamed to **Cache Criteria**. It accepts the following reserved parameters: c, cid, context, p, rendermode, site, sitepfx, ft_ss, and custom-defined parameters.

  Cache criteria values are stored in the pagecriteria column of the SiteCatalog table (in previous versions they were stored in the resargs columns of the SiteCatalog table).

  The **Cache Criteria** field is also used to hold variables that enable the Extra Parameters section in the CKEditor and make them available to users, in the **Include asset link** and **Add asset link** dialog boxes. The Extra Parameters section provides a way of passing custom parameters (such as image dimensions) to the template. See substep 3 of step 2 in Section 23.4.2.4, "Step 4: Configure SiteEntry" about extra parameters. For more information, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.
- Forms for creating Template and CSElement assets have been subdivided by tabs; fields are organized by function on the tabs.

23.2 Pages, Pagelets, and Elements

In the WebCenter Sites context, an online page is the composition of several components into a viewable, final output. Creating that output is called **rendering**. (Making either that output or the content that is to be rendered available to the visitors on your public site is called publishing.)

WebCenter Sites renders pages by executing the code associated with page names. The name of a page is passed to WebCenter Sites from a browser and WebCenter Sites invokes the code associated with that page name. The code is actually a named file, a separate chunk of code called an element.

The code in your elements identifies and then loads assets to display in those pages or pagelets, and passes other page names and element names to WebCenter Sites. When WebCenter Sites invokes an element, all of the code in the element is executed. If there are calls to other elements, those elements are invoked in turn. Then the results, the images, articles, linksets, and so on, including any HTML tags, are rendered into HTML code (or some other output format if your system is configured to do so).

Template, CSElement, and SiteEntry assets represent elements and pagelets as follows:

- A CSElement asset is an element.
- A SiteEntry asset is the name of a page or a pagelet.
- A Template asset is both an element and a page or pagelet that renders an asset.

This section contains the following topics:

- Section 23.2.1, "Elements, Pagelets, and Caching"
- Section 23.2.2, "Calling Pages and Elements"

- Section 23.2.3, "Page vs. Pagelet"

23.2.1 Elements, Pagelets, and Caching

Pages and pagelets are cacheable. They have cache criteria set for them that determines whether they are cached and, if so, for how long.

Elements do not have cache criteria. When your code calls an element directly by name, without going through a page name, the output is displayed in the page that called the element's name and that output is cached as a part of that page.

If you want to cache the output from an element separately from the output of the page that called it, you must provide a page name for it and call it by its page name. The code in a Template asset has a page name by default. To provide a page name for a CSElement asset, you create a SiteEntry asset and select the CSElement asset for it.

23.2.2 Calling Pages and Elements

To see a WebCenter Sites page, you provide a URL that includes the name of the page. A WebCenter Sites URL looks like this:

For WebLogic and WebSphere:

```
http://host:port/servlet_context_path/ContentServer?pagename=name_of_page
```

where:

- `host` is the name of the server that is hosting the WebCenter Sites system,
- `port` is the port number of the web server,
- `servlet_context_path` is the path that the application server gives to the WebCenter Sites web application, and
- `name_of_page` is the page name.

This syntax passes the name of a page to the ContentServer servlet, which then renders the page.

For example, to see the home page of the Burlington Financial sample site, you enter:

```
http://127.0.0.1:7001/servlet/ContentServer?pagename=
BurlingtonFinancial/Page/Home
```

When you code your elements, you use tags that programmatically call the pagelets and elements that you want to display in your site. These tags pass the names of pages and elements to the ContentServer servlet just as a URL entered in a browser passes a page name to the ContentServer servlet.

To call a page name, use the `render:satellitepage (RENDER.SATELLITEPAGE)` tag. For example:

```
<render:satellitepage pagename="BurlingtonFinancial/Page/Home" />
```

To call an element directly by name, use the `render:callelement (RENDER.CALLELEMENT)` tag. For example:

```
<render:callelement elementname="BurlingtonFinancial/Common/TextOnlyLink" />
```

To call a template by name, use the `render:calltemplate` tag. For example:

```
<render:calltemplate
        site='<%=ics.GetVar("site")%>'
        slotname="Head"
        tid='<%=ics.GetVar("tid")%>'
        c='<%=ics.GetVar("c")%>'
        cid='<%=ics.GetVar("cid")%>'
        tname='<%=ics.GetVar("HeadVar")%>'>
    <render:argument name="p" value='<%=ics.GetVar("p")%>' />
</render:calltemplate>
```

Note: When you use Sites Explorer to examine `SiteCatalog` and `ElementCatalog` entries, they are presented as folders and subfolders that visually organize the pages and pagelets.

However, these entries are simply rows in a database table (there is no actual hierarchy). Therefore your code must always call a page entry or an element entry by its entire name. You cannot use a relative path.

How does your code call template, CSElement, and SiteEntry assets? As follows:

- Because a SiteEntry is a pagelet, you use the `render:satellitepage` tag to call SiteEntry assets from within your element code.
- Because a CSElement is an element, you use the `render:callelement` tag to call CSElement assets from within your element code.
- Because a template is both an element and a page name, you can use either of the above, although typically the render:calltemplate tag is designed to be used for templates. It encapsulates the functionality of `render:satellitepage` and `render:calleelement` as well as other features, such as parameter validation.

23.2.3 Page vs. Pagelet

For the sake of clarity, the following table lists the various terms that include the word page and defines them in the context of their usage in the documentation for WebCenter Sites, the WebCenter Sites modules, and the products:

Term	**Definition**
pagelet	The results of an HTTP request displayed in a browser as one piece of a rendered page. It has an associated element file. A pagelet can be cached in the WebCenter Sites and Satellite Server page caches.
page	The results of an HTTP request displayed in a browser window. A page is created by compiling several parts of pages (pagelets) into one final, displayed or rendered page. It has an associated element file. A page can be cached in the WebCenter Sites and Satellite Server page caches.
page name	The complete name of a page or pagelet. For example: BurlingtonFinancial/Article/Full.
page asset	Page assets do **not** represent page names. They represent logical containers for content. These containers can be arranged into a tree structure for navigation of site content. You create page assets and then place them in position in the Site Plan tree which is visible on the **Site** tab in the tree in the left pane of the WebCenter Sites interface. You associate other content and site design assets with them and then you publish them.

23.3 CSElement, Template, and SiteEntry Assets

As mentioned, CSElement assets are elements, SiteEntry assets are page names, and Template assets include both.

Because page names and elements are assets, you can manage your code and page names in the same way you manage your content: you can use workflow, revision tracking, approval, and preview, as well as the Mirror to Server publishing method to move your code and page names to the management and delivery systems.

Note: **Revision tracking.** Never use the revision tracking feature in the Sites Explorer tool to enable revision tracking directly on the SiteCatalog or ElementCatalog tables.

Mirror to Server. If templates or CSElements refer to elements that are not associated with a template or CSElement asset, these elements are not automatically mirrored to the publishing destination. You must move them manually with the CatalogMover utility. For this reason, we do not recommend using elements that are not wrapped by CSElements.

When you create a CSElement or Template asset, you create an element. You can code the element by using the asset form (**Template** or **CSElement**, depending on which type of asset you are creating).

Note: Elements for Template assets and CSElements can be coded in Sites Explorer. However, the procedure is not recommended for reasons dealing mostly with compositional dependencies and updates to the cache. Developers who prefer to use Sites Explorer must follow the steps in Section 23.8, "Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic" in order to ensure the validity of the Template or CSElement assets.

This section contains the following topics:

- Section 23.3.1, "Template Assets"
- Section 23.3.2, "CSElement Assets"
- Section 23.3.3, "SiteEntry Assets"
- Section 23.3.4, "What About Non-Asset Elements?"

23.3.1 Template Assets

Templates render other assets into pages and pagelets. This in turn creates the look and feel of your online site. You create a standard set of templates for each asset type, except CSElement and SiteEntry assets, so that all assets of the same type are formatted in the same way.

This process allows content providers to preview their content by selecting formatting code for the content, but not requiring them to code themselves or allowing them to change your standard, approved code.

When you save a Template asset, WebCenter Sites does the following:

- Creates a row in the `Template` table for the asset.

- Creates an element entry in the `ElementCatalog` table. The name of the entry uses the following convention:

  ```
  AssetTypeName/TemplateName
  ```

  where:

  – *`AssetTypeName`* is the asset type formatted by the Template asset and element.

  – *`TemplateName`* is the name of the template.

- Creates a page entry in the SiteCatalog table. The name of the page entry uses the following convention:

  ```
  SiteName/AssetTypeName/TemplateName
  ```

  where:

  – *`SiteName`* is the name of the site that the template belongs to, which is the site that you were working in when you created the template. WebCenter Sites obtains this name from the Publication table. (In previous versions of the product, sites were called publications.)

  – *`AssetTypeName`* is the asset type formatted by the Template asset and element

  – *`TemplateName`* is the name of the template.

  > **Note:** Do not change the name of the page entry that WebCenter Sites creates.

- Creates new rows in other tables that support the operation of the Template asset. The tables start with the name: `Template_`

- Creates a new row in the AssetPublication table to associate your template with your site.

23.3.2 CSElement Assets

You use CSElement assets for the following kinds of things:

- Code that is not for rendering an asset and that you want to reuse in more than one place and/or call from more than one type of template. For example, you have six templates that use the same top banner so you create a CSElement asset for the code in the banner and call that element from each template. This way, if you decide to change the way the banner works, you only have to change it in one place.

- Recommendations for Engage. If you create a dynamic list recommendation, you must create a CSElement asset to build the dynamic list. For more information, see Chapter 39, "Recommendation Assets" These assets do not render content, but exist for logic processing.

When you save a CSElement, WebCenter Sites does the following:

- Creates a row in the `CSElement` table for the asset.

- If you have coded the element in the CSElement form, creates an element entry in the ElementCatalog table. The name of the entry is the name that you entered into the **ElementCatalog Entry Name** field in the form.

- Creates a new row in the `AssetPublication` table to associate your template with your site.

23.3.3 SiteEntry Assets

You use SiteEntry assets for the following kinds of things:

- If you are using the CS-Designer tool, you use SiteEntry assets to represent code snippets. In that interface, when you drag and drop a code snippet into a page, you are dropping in a WebCenter Sites call to a page entry through a render:satellitepage tag.
- When the code in a CSElement asset is rendered, the code is displayed in the page that called it, and is cached as part of that page (if that page is cached, that is). If you want the output from a CSElement to be cached as a separate pagelet and have its own cache criteria set for it (timeout value, page criteria values, and so on), your code must invoke that element through a page name. In such a case, you create a SiteEntry asset to accompany your CSElement asset.

When you create and save a SiteEntry asset, you associate a CSElement asset with it. The element in that CSElement asset becomes the root element for the SiteEntry's page entry.

When you save a SiteEntry asset, WebCenter Sites does the following:

- Creates a row in the `SiteEntry` table for the asset.
- Creates a page entry in the `SiteCatalog` table. The root element of the page entry is the element from the CSElement asset that you specified.
- Tracks an approval dependency between the SiteEntry asset and the CSElement asset. Both the SiteEntry asset and its CSElement asset must be approved before the SiteEntry asset can be published.

Note: Compositional dependencies are also tracked. The SiteEntry defines the page criteria and the default arguments that contain the dependency information. The CSElement records the id of the SiteEntry and CSElement assets into the rendering engine using `render:logdep` tags that are added to the CSElement code stub.

23.3.4 What About Non-Asset Elements?

If you code customizations for the WebCenter Sites interface on the management system, you create elements that are not assets because you do not want them to be published to your delivery system.

For example, when you create workflow elements that implement actions or conditions, you do not create them as CSElement assets. Rather, you use the Sites Explorer tool to manually create an entry in the ElementCatalog table.

Remember that if you create workflow or other custom elements on your delivery system, you must use the CatalogMover utility to copy those elements to the ElementCatalog on your management system.

Note: You can write code to invoke the mirror engine to mirror your elements. The topic is advanced and beyond the scope of this guide. For assistance, contact Oracle Support at `www.oracle.com/support` or visit `www.oracle.com/accessibility` if you are hearing impaired.

23.4 Creating Template Assets

Templates render assets. When you create a Template asset, you create it as either

- a typed template, for rendering assets of a specific type, or
- a typeless template, which applies to assets of any type. A typeless template is generally used to specify the layout of a page in which assets can then be rendered by the typed templates.

> **Note:** The only field that makes a template typed or typeless is the **For Asset Type** field. The purpose of distinguishing templates as typed or typeless is to help developers manage the construction of pages and easily keep track of which templates are responsible for page layout and which for asset rendering.

Before creating a Template asset, complete Section 23.4.1, "Prerequisites" to determine how you will set template properties (such as the template name) and how you will code the template's element logic. You will then complete the following steps, using the WebCenter Sites interface:

- Step 1: Open the 'Template' Form
- Step 2: Name and Describe the Template Asset
- Step 3: Configure the Template's Element (To specify its usage, file type, and logic.)
- Step 4: Configure SiteEntry (To specify page and pagelet caching parameters.)
- Step 5: Configure the Map (If you wish to support template sharing and site replication.)
- Step 6: (Optional) Create a Thumbnail (To graphically represent the template in its **Inspect** form.)
- Step 7: Inspect the Template

Information that you enter into the **Template** form will be written to database tables when the template is saved.

> **Note:** Do not create Template assets directly in the database tables. Doing so will require you to write to several tables and can result in incorrect tracking of dependencies. Instead, use the **Template** form and the procedures in this section to create Template assets. For help with coding the template's element logic (in typed templates), see Chapter 28, "Coding Elements for Templates and CSElements."

This section contains the following topics:

- Section 23.4.1, "Prerequisites"
- Section 23.4.2, "Procedures for Creating Template Assets"

23.4.1 Prerequisites

Before you begin creating a Template asset, determine the following:

- *`TemplateName`* (a name for your Template asset; the value of the **Name** field in the **Name** screen, To name and describe the Template asset).

- Whether the Template asset is to be typed or typeless.
- Whether the Template asset will be shared and whether the site you are working in will be replicated. These considerations determine how you will code the template's element logic.
- Whether to code the Template's element logic in Sites Explorer instead of the **Template** form. Coding in Sites Explorer, although practiced, is not recommended for the reasons outlined in Section 23.8, "Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic."

23.4.1.1 Naming a Template Asset

- Once a Template asset is saved, its name cannot be changed.
- WebCenter Sites appends the template name to SiteName (also to AssetTypeName for typed templates). The template name should make sense in relation to SiteName and AssetTypeName. WebCenter Sites's naming conventions must not be overridden; names created by WebCenter Sites must not be changed. Table 23–1 lists the conventions that use TemplateName.

  `AssetTypeName` is used only for typed templates and represents the value of the template's **For Asset Type** field in the **Name** screen (see To name and describe the Template asset). *`SiteName`* is the name of the site to which the template belongs (the site that you are working in as you are creating the template). WebCenter Sites obtains the *`SiteName`* from the Publication table. (In previous versions of the product, sites were called publications.)

Table 23–1 Naming Conventions Using `TemplateName`

Template	Naming Conventions	Description
Typed	*`AssetTypeName/TemplateName`*	Name of the root element for a typed template. This value is written to the **Rootelement** field in step 4. **This value must not be changed.** If the default value is changed, some tags that expect the default value, such as the `render:calltemplate` tag with the `style` attribute set to `element`, will fail. **Note:** The naming convention requires root element names to be unique. You must not have multiple Template assets pointing to the same root element. You can, however, have two SiteEntry assets point to the same element (for example, if you wish to specify different default arguments, or different cache criteria depending on the calling scenario).
n/a	*`AssetTypeName/TemplateName`*`.xml_or_jsp_or_html`	Path to the element file of a typed template. This value is written to the **ElementStorage Path/Filename** when the file type is selected in step 3.

Table 23–1 (Cont.) Naming Conventions Using `TemplateName`

Template	Naming Conventions	Description
n/a	*SiteName/AssetTypeName/TemplateName*	Name of the page that will be rendered if the template is typed. This value is written to the **SiteCatalog Pagename** field, in Section 23.4.2.4, "Step 4: Configure SiteEntry."
Typeless	*/TemplateName*	Name of the root element for a typeless template. This value is written to the **Rootelement** field in step 4. **This value must not be changed.** If the default value is changed, some tags that expect the default value, such as the `render:calltemplate` tag with the `style` attribute set to `element`, will fail. **Note:** The AssetTypeName is omitted, as the template applies to any asset type. The forward slash is kept to identify the template as typeless. See also the note in the first row of this table.
n/a	`Typeless/`*TemplateName*`.xml_or_jsp_or_html`	Path to the element file of a typeless template. This value is written to the **ElementStorage Path/Filename** when the file type is selected in step 3.
n/a	*SiteName/TemplateName*	Name of the page that will be rendered if the template is typeless. This value is written to the **SiteCatalog Pagename** field, in Section 23.4.2.4, "Step 4: Configure SiteEntry." **Note:** The *AssetTypeName/* is omitted, as the template applies to any asset type.
Typed	*AssetTypeName/TemplateName*	Name of the root element for a typed template. This value is written to the **Rootelement** field in step 4. **This value must not be changed.** If the default value is changed, some tags that expect the default value, such as the `render:calltemplate` tag with the `style` attribute set to `element`, will fail. **Note:** The naming convention requires root element names to be unique. You must not have multiple Template assets pointing to the same root element. You can, however, have two SiteEntry assets point to the same element (for example, if you wish to specify different default arguments, or different cache criteria depending on the calling scenario).

Table 23–1 (Cont.) Naming Conventions Using **`TemplateName`**

Template	Naming Conventions	Description
n/a	`AssetTypeName/TemplateName.xml_or_jsp_or_html`	Path to the element file of a typed template. This value is written to the **ElementStorage Path/Filename** when the file type is selected in step 3.

23.4.1.2 Designating a Template as Typed or Typeless

Before creating a Template asset, determine whether it is to be typed or typeless. Once the template is saved, its status as typed or typeless cannot be changed.

23.4.1.3 Template Sharing and Site Replication

Before creating a template, decide how the template and the site you are working in will be used. Your decision determines how you will code the template's element logic (in Section 23.4.2.3, "Step 3: Configure the Template's Element").

If you wish to share your Template asset or make the current site replicable**,** ensure that the template's element logic does not directly refer to assets, asset types, attribute names, or template names. Instead, you must refer to them indirectly. Use the **Map** screen (Section 23.4.2.5, "Step 5: Configure the Map") to assign an alias (key) to each value, then hard code the aliases in your template. Use the `render:lookup` tag to retrieve the actual values from the aliases at runtime.

During its execution, the `render:lookup` tag refers to the map to look up the keys and returns the asset-specific information for use in the element logic. This dynamic lookup allows the Template asset (but not the element logic alone) to refer directly to asset data while enabling safe replication and template sharing.

For example, assume a template is named `FSIILayout`, and the site containing this template has a site prefix of `FSII`. If the site is replicated such that the new site's prefix is `New`, and the `FSIILayout` template is copied, then the copy of the template is named `NewLayout`. Referring to the NewLayout template by its hard-coded name (`FSIILayout`) would result in a failure when the template is executed. Instead, the template name is looked up:

```
<%-- Look up the name of the layout template --%>
<render:lookup
        site='<%=ics.GetVar("site")%>'
        varname="LayoutVar"
        key="Layout"
        tid='<%=ics.GetVar("tid")%>'/>

<%-- Look up the name of the wrapper page's site entry.
     Note we want the asset name only, so we must specify
     the match filter. --%>
<render:lookup
        site='<%=ics.GetVar("site")%>'
        varname="WrapperVar"
        key="Wrapper"
        tid='<%=ics.GetVar("tid")%>'
        match=":x"/>
```

To code the element logic, you must have a clear understanding of its design and the map it will refer to. You will need to determine:

- Which keys to create and which name to assign to each key.

- The type of asset information to be looked up:
  - Template Name
  - Asset Type
  - Asset (Type:Name)
  - Asset (Type:ID)
- A value for each key.
- The site to which the map applies.

Additional information about usage of the `render:lookup` tag is given in the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

23.4.2 Procedures for Creating Template Assets

This section shows you how to create a Template asset, using the WebCenter Sites interface.

> **Note:** Before starting the procedures in this section, read Section 23.4.1, "Prerequisites" for information about creating Template assets.

23.4.2.1 Step 1: Open the 'Template' Form

1. Open the Admin interface.
2. In the button bar, click **New**.
3. In the list of asset types, select **New Template**.

> **Note:** For the **New Template** option to be displayed, the Template asset type must be enabled for your site and a start menu item must be created for it.

4. The **Template** form displays. Continue with Section 23.4.2.2, "Step 2: Name and Describe the Template Asset."

> **Note:** If you see a **Choose Assignees** screen instead of the **Template** form, it means that the Template asset you will be creating is associated with a workflow. Select a name (or names) from the Users column and click **Set Assignees**. Continue with Section 23.4.2.2, "Step 2: Name and Describe the Template Asset."

23.4.2.2 Step 2: Name and Describe the Template Asset

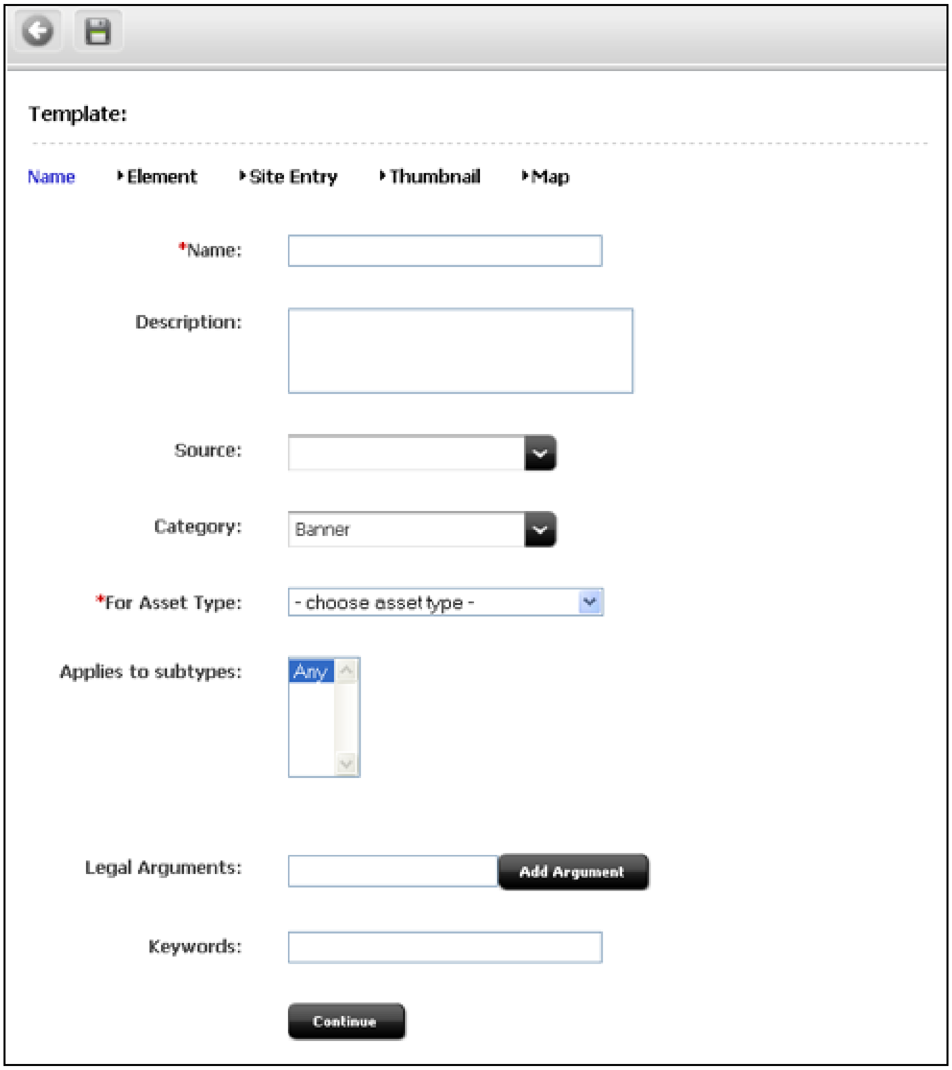
The **Name** screen is used to identify the template as typed or typeless, assign the template to a category, specify arguments that may be passed to the template, and name keywords by which the template can be located in search routines.

When the Template asset is saved, field values that you specify in the **Name** screen (with the exception of legal arguments), are written to the Template table, as indicated in the procedures below.

Note: At any time in the process of creating a template, you can save the template. WebCenter Sites will display the template's **Inspect** form. To return to the **Template** form, click the **Edit** link.

To name and describe the Template asset

In the **Name** screen, fill in the fields as explained in the steps below:



1. (Required). In the **Name** field, type a descriptive template name that is unique for the template and for the type of asset(s) that the template renders. It is best to choose a name that reflects the function or purpose of the template.

   Valid entries:

   – Up to 64 alphanumeric characters (the first character must be a letter)

   – Underscores (_)

   – Hyphens (-)

   – Spaces (these will be converted to underscores when used in the SiteCatalog pagename for the template)

> **Note:** Make sure you have chosen a name for your Template asset using the guidelines in the section Section 23.4.1, "Prerequisites."

2. In the **Description** field, type a brief description of the template. You can use up to 128 characters.

3. In the **Source** field, select an option from the drop-down list if your template is derived from a source that you wish to note.

4. In the **Category** field, select an option from the drop-down list if you wish to place the Template asset into a category.

5. (Required). In the **For Asset Type** field, identify your template as typed or typeless:

   – If you are creating a typeless template (for example to dispatch to typed templates), select **Can apply to various asset types** and skip to step 7.

   – If you are creating a typed template (which renders assets of a certain type), select an asset type. For example, if you are creating a template to render article assets, select **Article** from the drop-down list.

6. (Required for typed templates). In the **Applies to Subtypes** field, select the appropriate subtypes from the menu.

> **Note:** A typed template should be used only for specific subtypes of the asset type that you selected in the preceding field (**For Asset Type**).

7. In the **Legal Arguments** field:

   a. Enter an argument that may be passed to the template and click **Add Argument**.

   b. In the fields that are displayed:

      – Specify whether the argument is optional or required.

      – Provide a description of the argument (to help you know the purpose of the argument you are creating).

      – Specify legal values (including descriptions) for the argument.

   (You can specify as many arguments and legal values as you require by clicking the **Add Arguments** and **Add Legal Value** buttons.)

8. In the **Keywords** field, enter keywords that you and others can use as search criteria in the **Advanced Search** form when you search for this template in the future. For information about searching for assets, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

9. Click **Continue** to open the next screen, **Element**.

Note: If you chose to save the Template asset, you will notice that WebCenter Sites adds two fields:

- The **Status** field, which is pre-populated with the editorial status of the Template asset (created, edited, and so on). This field identifies the latest operation that was performed on the Template asset, regardless of whether the Template asset is associated with a workflow.
- The **ID** field, which is pre-populated with a unique number that WebCenter Sites generates and assigns to the Template asset as its ID. (The **ID** field corresponds to the tid variable.)

23.4.2.3 Step 3: Configure the Template's Element

The **Element** screen is used to create the template's element, define the element file type (XML, JSP, or HTML), provide the element logic, and name the element. For example:

- The **Create Template Element** field offers a choice of XML, JSP, or HTML file types for the element logic, and is used to seed the **Element Logic** field with standard stub code (which you need to include in any element that you create).

  When you use the **Create Template Element** field to create, for example, a .jsp file, WebCenter Sites adds JSP `taglib` statements and the `RENDER.LOGDEP` tag to the **Element Logic** field by default so that WebCenter Sites can log the compositional dependency between this Template asset and pages that are rendered from this element. For other file types, WebCenter Sites adds code specific to the file type. You will add your own code to the **Element Logic** field.

  For information about dependencies, see Section 28.1, "About Dependencies." For help with coding the element logic, see Chapter 28, "Coding Elements for Templates and CSElements."
- The **Element Storage Path/Filename** field names the file that holds the element logic and specifies the path to the file.

When the Template asset is saved, field values in the **Element** screen are written to a row (representing the element) in the ElementCatalog table, as indicated in the procedures below.

Note: Selecting an Existing Element

In the steps that follow, we assume you are creating a new element for the Template asset. If, however, you are migrating assets from an earlier WebCenter Sites release and wish to reuse an existing element, you need to identify the element correctly so that WebCenter Sites can find it and associate it with this Template asset.

To select an existing element

1. (Optional). In the **ElementCatalog Description** field, type a description of the element.
2. In the **Element Storage Path/Filename** field, enter a value according to the naming convention in Table 23–1.
3. In the **Element parameters** field, specify the variables or arguments that can be passed to the element. For more information, see step 7 in the "To configure a new element" section.
4. Save and re-open the Template asset.

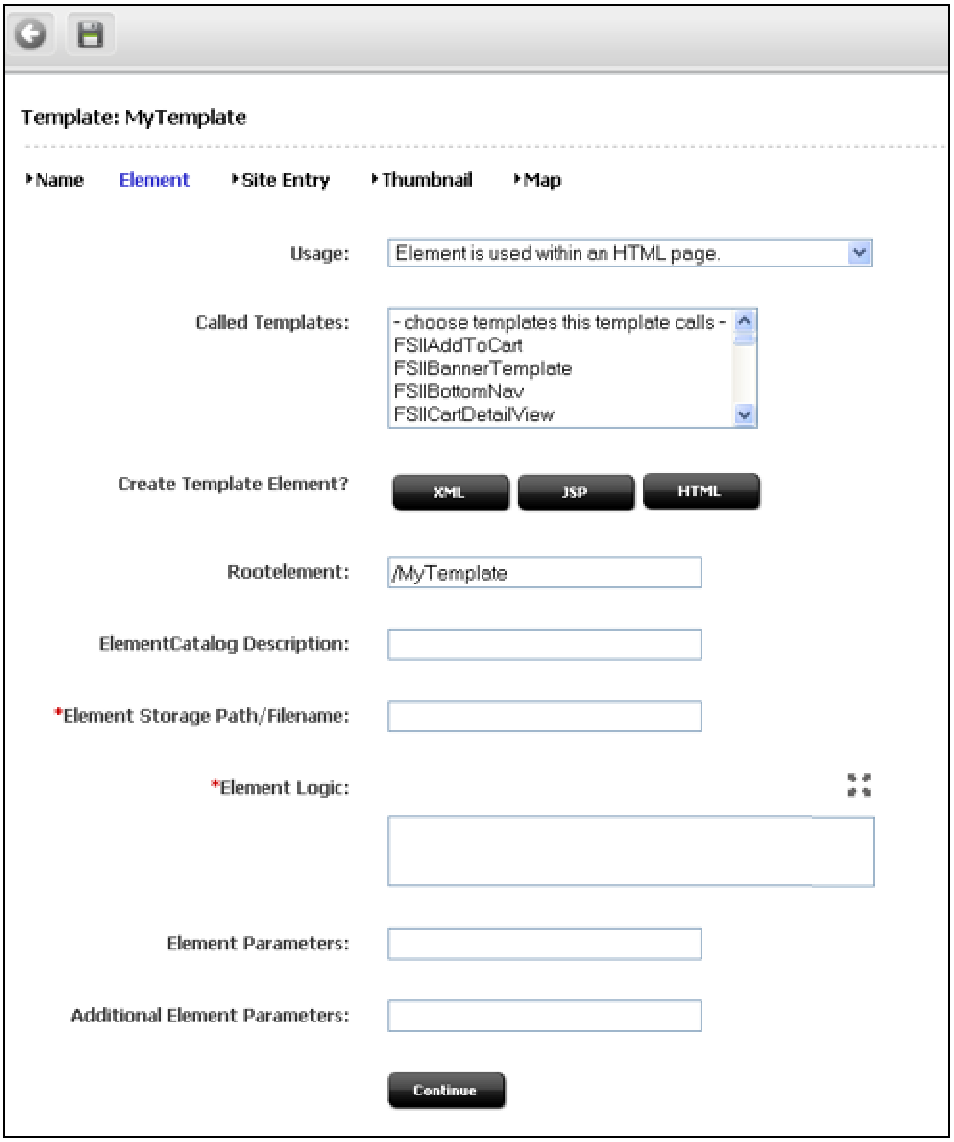
WebCenter Sites checks for the existence of the named element:

If the element has been correctly named, WebCenter Sites recognizes the element and displays its code in the **Element Logic** field.

If the named element does not exist (or is incorrectly named), WebCenter Sites does nothing. When you inspect or edit the Template asset, WebCenter Sites displays a message stating that there is no root element in the form. As soon as you code the element and give it the correct name, WebCenter Sites detects it and associates it with the template.

To configure a new element

In the **Element** screen, fill in the fields as explained in this section.



Template: MyTemplate

Usage: Element is used within an HTML page.

Called Templates: - choose templates this template calls -
FSIIAddToCart
FSIIBannerTemplate
FSIIBottomNav
FSIICartDetailView

Rootelement: /MyTemplate

ElementCatalog Description:

*Element Storage Path/Filename:

*Element Logic:

Element Parameters:

Additional Element Parameters:

1. In the **Usage** field, specify the intended usage of this template, using the table below as a guideline.

Usage Option	Description
Usage unspecified	Specifies a template that generates HTML. It is unknown whether the template is a Body template (see row 2 of this table) or a url template (see row 3 of this table).
Element is used within an HTML page	Specifies a template that is used inside the `<BODY>...</BODY>` tag of an HTML page. This option characterizes the template as a Body template.
Element is used as a layout	Specifies a layout template that generates a complete HTML page, and is used to render assets in Web Mode.

Usage Option	Description
Element defines a whole HTML page and can be called externally	Specifies a template that generates a complete HTML page and can be used in a url. This option characterizes the template as a url template.
Element is streamed as raw data	Specifies a template that generates raw binary data of an unknown type that is not HTML.

2. In the **Called Templates** field, select the template(s) that this template will call (if they exist).

3. In the **Create Template Element** field, do one of the following:

   – To create an .xml file, click **XML**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\Template\modelXML.xml` element and can be modified to use custom default logic.

   – To create a .jsp file, click **JSP**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\Template\modelJSP.xml` element and can be modified to use custom default logic.

   – To create an .html file, click **HTML**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\Template\modelHTML.xml` element and can be modified to use custom default logic.

   WebCenter Sites populates the following fields:

   – **Element Logic** field with a header and other auto-generated code.

     For example, if you clicked the **JSP** button, WebCenter Sites enters a tag library directive for each of the WebCenter Sites JSP tag libraries. WebCenter Sites also sets a RENDER.LOGDEP (render:logdep) tag to mark a compositional dependency between the Template asset and any page or pagelet rendered with the template.

   – **Element Storage Path/Filename** field. **Do not change the value of this field.**

     This field displays the element file name, preceded by the path to the element file. The naming convention is given in Table 23–1.

     When you save the Template asset, the value in the **Element Storage Path/Filename** field is written to the url column of the `ElementCatalog` table, for the row that represents the element.

4. The **Rootelement** field is pre-populated with the value given in Table 23–1. **Do not change the value of this field.** If the default value is changed, some tags that expect the default value, such as the `render:calltemplate` tag with the `style` attribute set to `element`, will fail.

5. (Optional). In the **ElementCatalog Description** field, type a description of the element. When you save the Template asset, information in this field is written to the description column for the element entry in the ElementCatalog table.

6. (Required). In the **Element Logic** field, code your element. Be sure to enter all of your code between the two `cs:ftcs` tags.

   If you are using JSP, remove the comments from the `taglib` directives that describe the tag libraries you are using.

   For help with this step, see Chapter 28, "Coding Elements for Templates and CSElements."

Note: Ensuring Template Sharing or a Replicable Site
If you wish to share your Template asset or make the current site replicable, make sure that the template's element logic does not directly refer to assets, asset types, attribute names, or template names. Instead, use the render:lookup tag and prescribed keys as explained in Section 23.4.1.3, "Template Sharing and Site Replication." In Section 23.4.2.5, "Step 5: Configure the Map," you will map the same keys to the asset information that must be accessed for use in the element logic.

Calling a Template
Templates should *always* be called by the `render:calltemplate` tag, and never the `render:callelement` tag or `render:satellitepage` tag.

7. (Optional). The **Element Parameters** field and **Additional Element Parameters** field are used to enter variables or arguments that can be passed to the element, if the site design requires them.

   – The **Element Parameters** field corresponds to the resdetails1 column in the ElementCatalog. When you save the template, WebCenter Sites writes the template ID (`tid`) to this field (i.e., to the `resdetails1` column).

   – The **Additional Element Parameters** field corresponds to the resdetails2 column in the `ElementCatalog`. WebCenter Sites leaves this field blank.

   If your site design requires you to use variables in addition to tid in your template element, enter the variables into one of the fields above. Enter them as name=value pairs with multiple arguments separated by the ampersand (&) character. For example:

   ```
   MyArgument=value1&YourArgument=value2
   ```
   Each field supports up to 255 characters.

   For more information about using variables, see Chapter 4, "Programming with Oracle WebCenter Sites."

8. Click **Continue** to open the next screen, **SiteEntry**.

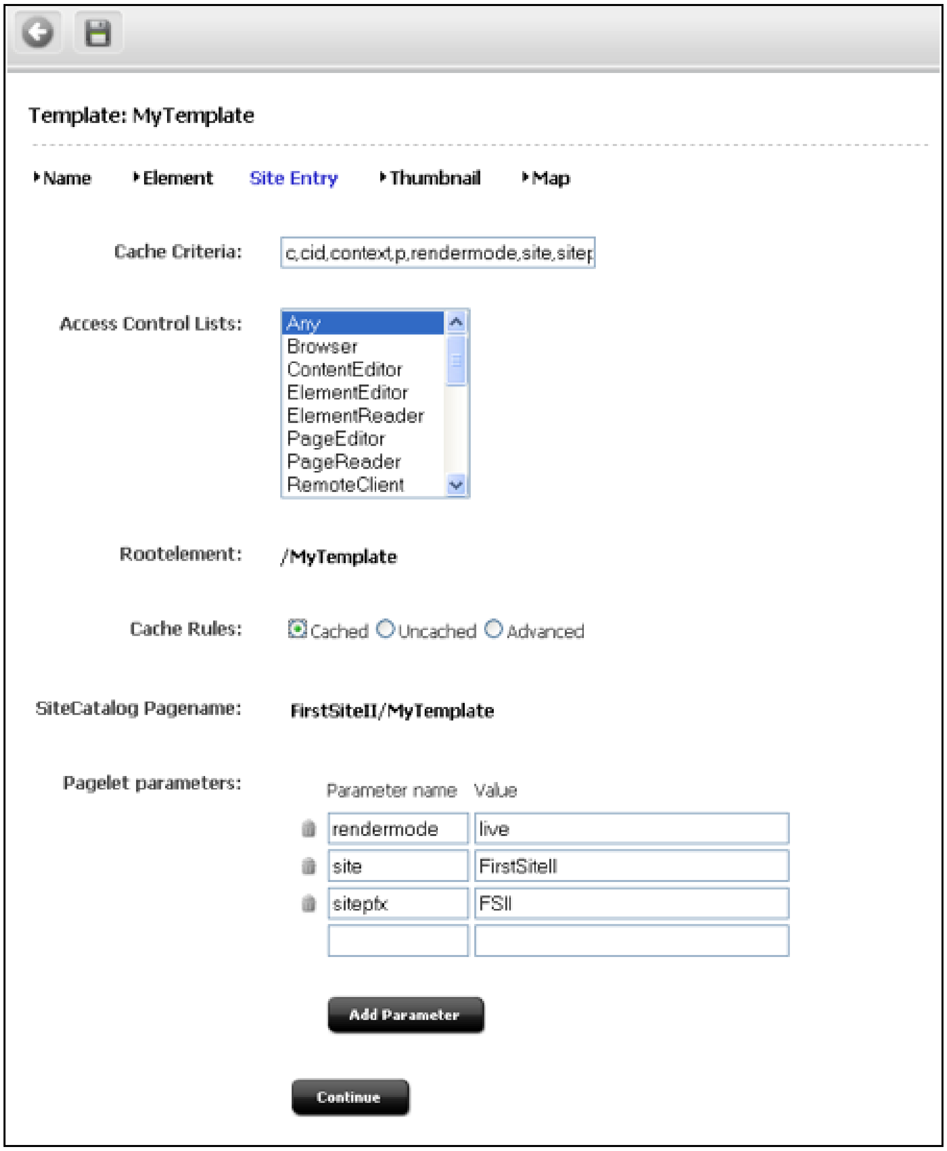
23.4.2.4 Step 4: Configure SiteEntry

The **SiteEntry** screen is used to specify caching and pagelet parameters for the page to be rendered by this Template asset.

When the Template asset is saved, field values that you specify in the **SiteEntry** screen are written to the SiteCatalog table, as indicated in the procedures below.

To configure the SiteEntry

1. In the **SiteEntry** screen, fill in the fields as explained in this section.



Template: MyTemplate

Cache Criteria: c,cid,context,p,rendermode,site,sitep

Access Control Lists: Any, Browser, ContentEditor, ElementEditor, ElementReader, PageEditor, PageReader, RemoteClient

Rootelement: /MyTemplate

Cache Rules: ● Cached ○ Uncached ○ Advanced

SiteCatalog Pagename: FirstSiteII/MyTemplate

Pagelet parameters:

Parameter name	Value
rendermode	live
site	FirstSiteII
sitepfx	FSII

2. In the **Cache Criteria** field:

   a. WebCenter Sites names the following **reserved** variables as Cache Criteria:

   ```
   c,cid,context,p,rendermode,site,sitepfx,ft_ss
   ```

   Note: The reserved Cache Criteria variables should not be removed. For information about the reserved variables, see Chapter 4, "Programming with Oracle WebCenter Sites."

   ---

   b. If you need to include your own variables as Cache Criteria (for example, **`foo`**), add them to the existing list. For example:

   ```
   c,cid,context,foo,p,rendermode,site,sitepfx,ft_ss
   ```

> **Note:** The **Cache Criteria** field names the variables which, in conjunction with SiteCatalog Pagename, define a pagelet as being unique. The variables are used to identify cached pages, which means that the variables are used in the page's cache key.
>
> Only those variables that are specified as Cache Criteria are used by the caching system to create the cache key for cached pages. Therefore, if your site design requires you to use page-level variables in addition to the reserved variables, be sure to designate them as Cache Criteria variables, as shown in this step.

3. If you need to enable the Extra Parameters section in CKEditor, complete these steps:

   a. Move the "Extra Parameters" that were added to the Cache Criteria of the Template to the "Legal Arguments" section of the Template.

   b. When moving the parameters to the Legal Arguments section, it is no longer necessary to prefix the parameters with `fp:`, just use the parameter name.

      For example, instead of using `fp:imageHeight`, just use `imageHeight`.

   c. These parameters are then available in the included template, and can be retrieved in standard ways, such as using `<ics:getvar name="imageHeight"/>`.

   The Extra Parameters section provides a way of passing custom parameters, such as image dimensions, to the template. These extra parameters will be available in the **Include asset link** and **Add asset link** dialog boxes. The parameter names (`imageHeight` and `imageWidth` in our example) will be displayed in CKEditor's Extra Parameters section, as options in the Name menu. The Value field enables the user to specify a value for the chosen parameter. For more information about extra parameters, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.

   When the Template asset is saved, the Cache Criteria variables are written to the `pagecriteria` column in the `SiteCatalog` table.

4. (Optional). The **Access Control Lists** field corresponds to the acl column in the `SiteCatalog` table. If you want to allow only certain visitors to request this page, select the ACLs that the visitors must have in order to see the page. For more information about ACLs, see Chapter 31, "User Management on the Delivery System."

5. The **Rootelement** field is pre-populated with a value that is shown in Table 23–1.

6. The **Cache Rules** field corresponds to the cscacheinfo and sscacheinfo columns in the `SiteCatalog` table. Do one of the following:

   – Select **Cached** if the pagelet to be rendered by this template's element must be cached. The pagelet is set to be cached forever. The cache will be flushed by CacheManager's active cache management logic. This option sets both WebCenter Sites and Satellite Server caching conditions.

   – Select **Uncached** if you wish to turn off caching for the pagelet to be rendered by this template's element. This option sets both WebCenter Sites and Satellite Server caching conditions.

   – Select **Advanced** if you wish to set caching rules individually for WebCenter Sites and Satellite Server. Selecting **Advanced** displays two additional fields: one for WebCenter Sites caching and one for Satellite Server caching.

Note: CacheManager is designed to manage the lifecycle for cached pages on both WebCenter Sites and Satellite Server. It is designed to operate with pages that are set to be cached forever. If the cache expires on WebCenter Sites before it expires on Satellite Server, CacheManager will fail to flush the cache properly and invalid pages may be served from cache. Only advanced users should configure these settings manually.

For more information about page caching settings, see Chapter 5, "Page Design and Caching."

For more information about page caching settings, see Chapter 5, "Page Design and Caching."

1. **SiteCatalog Pagename** field. **Do not change the value of this field.** This field is pre-populated with the name of the page entry. The page naming convention is given in Table 23–1.
2. In the **Pagelet parameters** section, you can enter pagelet parameters (name-value pairs), which will be passed into the template each time it is executed. (The **Pagelet parameters** section supports a total of 510 characters.)

Note: The **Pagelet parameters** section is pre-populated with the following default pagelet parameters (reserved variables that were named in step 2, including their values:

```
site, sitepfx, rendermode
```

The default parameter values will be overwritten if they are explicitly specified when the template is called.

- If you are specifying a pagelet parameter in this step, make sure to list its name as a Cache Criteria variable (see step 2).
- If you named your own Cache Criteria variables (in step 2), the variables are listed in the **Page parameters** section. If you do not specify values for these parameters, WebCenter Sites ignores the parameters.

When the Template asset is saved, the name-value pairs that are specified as **Pagelet parameters** are written to either the resargs1 or resargs2 column of the SiteCatalog table. The column to which they are written is not important and is managed automatically. (Each column supports up to 255 characters.)

3. Click **Continue** to open the next screen, **Thumbnails**.
4. Click **Continue** to open the next screen, **Map**.

Note: You will return to the **Thumbnail** screen after you have completed creating the Template asset and saved the Template asset.

23.4.2.5 Step 5: Configure the Map

The purpose of mapping is to enable site replication and the sharing of Template assets, as explained in Section 23.4.1.3, "Template Sharing and Site Replication."

Note: Skip this section if you are designing a non-replicable site.

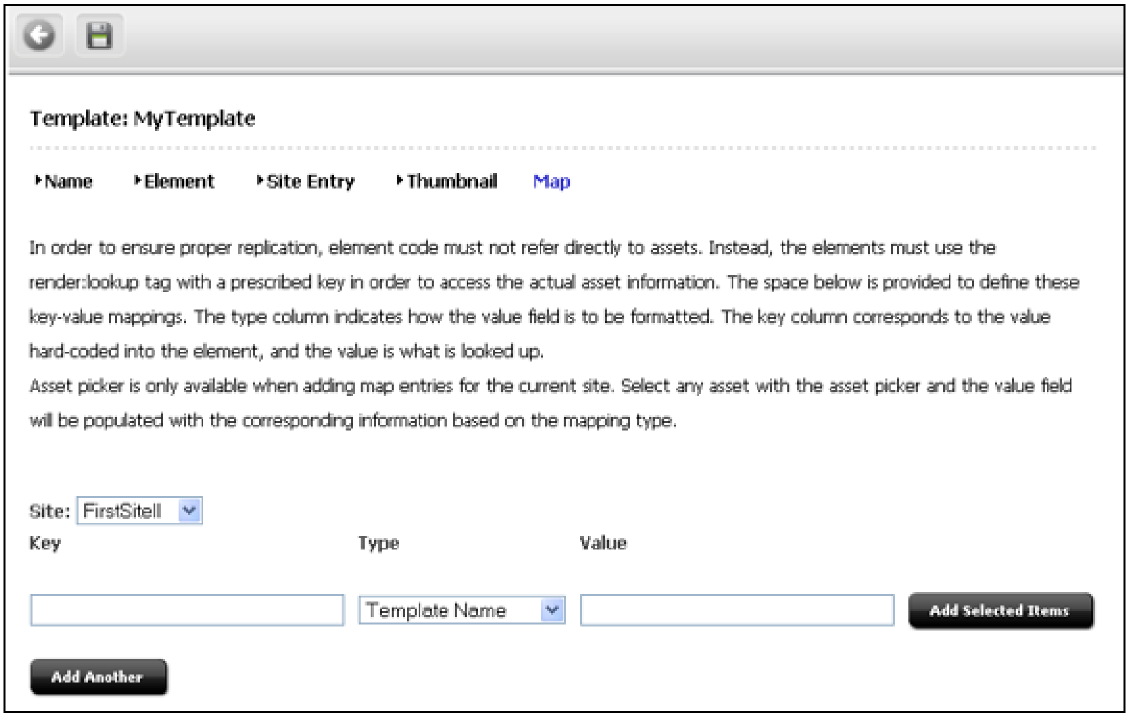
Using the **Map** form, you will:

- Map each key in the `render:lookup` tags of the template's element logic to a value that will be used by the element logic.
- Map each key's value to the asset information that must be used in the element logic: asset, asset type, attribute name, or template name.

When the Template asset is saved, the map is written to the Template_Map table.

To configure a map

In the **Map** form, fill in the fields as explained in this section.



Template: MyTemplate

In order to ensure proper replication, element code must not refer directly to assets. Instead, the elements must use the render:lookup tag with a prescribed key in order to access the actual asset information. The space below is provided to define these key-value mappings. The type column indicates how the value field is to be formatted. The key column corresponds to the value hard-coded into the element, and the value is what is looked up.

Asset picker is only available when adding map entries for the current site. Select any asset with the asset picker and the value field will be populated with the corresponding information based on the mapping type.

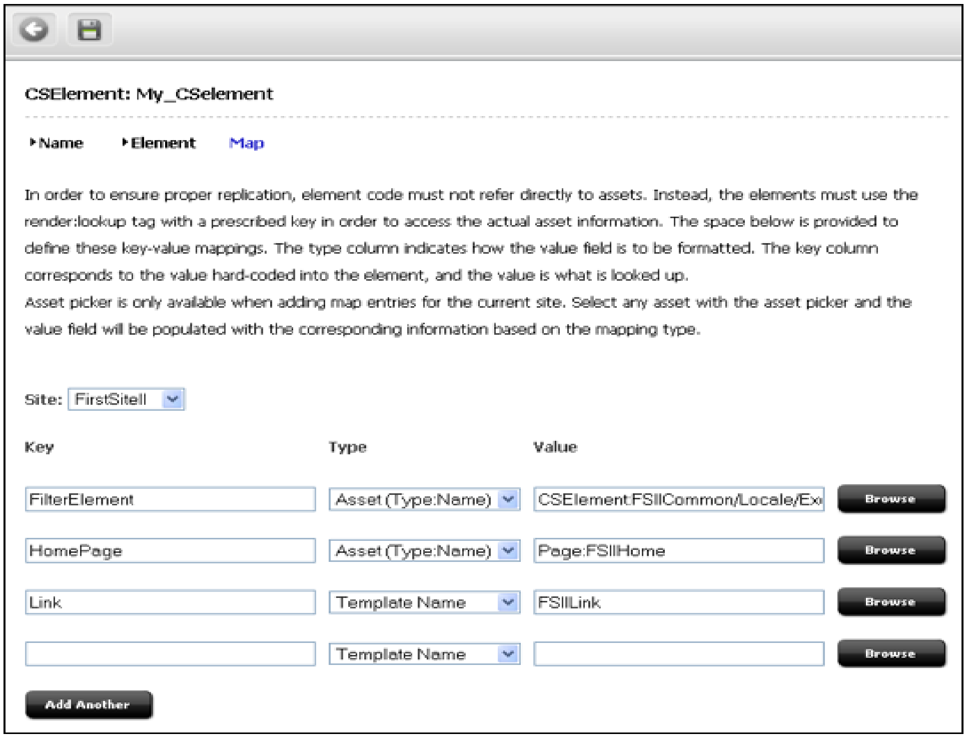
Site: FirstSiteII

Key	Type	Value
	Template Name	

1. The **Key** field represents a value that the element logic will look up. Enter the key that is named in a `render:lookup` tag of the element logic.
2. The **Type** field identifies the type of asset information to be accessed. Select one of the following options:
   - **Template Name**: Maps a template name to the key value (which you will specify in the **Value** field, in the next step). The information that will be accessed is a template name that matches the value that you will specify in the next step. For an example, see Figure 23–1.
   - **Asset Type**: Maps an asset type to the key value. The information that will be accessed is an asset type, equal to the value that you will specify in the next step.
   - **Asset (Type:Name)**: Maps an attribute type:name to the key value. The information that will be accessed is an asset whose type and name match the value that you will specify in the next step.

– **Asset (Type:ID)**: Maps an attribute type:ID to the key value. The information that will be accessed is an asset whose type and name match the value that you will specify in the next step.

Figure 23–1 Template Asset: Sample Map



CSElement: My_CSelement

▸Name ▸Element Map

In order to ensure proper replication, element code must not refer directly to assets. Instead, the elements must use the render:lookup tag with a prescribed key in order to access the actual asset information. The space below is provided to define these key-value mappings. The type column indicates how the value field is to be formatted. The key column corresponds to the value hard-coded into the element, and the value is what is looked up.
Asset picker is only available when adding map entries for the current site. Select any asset with the asset picker and the value field will be populated with the corresponding information based on the mapping type.

Site: FirstSiteII

Key	Type	Value	
FilterElement	Asset (Type:Name)	CSElement:FSIICommon/Locale/Ex	Browse
HomePage	Asset (Type:Name)	Page:FSIIHome	Browse
Link	Template Name	FSIILink	Browse
	Template Name		Browse

Add Another

3. In the **Value** field, enter a value for the key. This value will be looked up by the element logic when the Template is executed.

4. In the **siteid** field, select the name of the site to which the mapping applies.

5. To add a key, click **Add Another** and repeat the steps in this section.

6. When you have completed creating your template, save the template (click **Save Changes**).

   WebCenter Sites displays the template's **Inspect** form.

7. If you wish to create a thumbnail for your template, continue with Section 23.4.2.6, "Step 6: (Optional) Create a Thumbnail." Otherwise, skip to Section 23.4.2.7, "Step 7: Inspect the Template."

23.4.2.6 Step 6: (Optional) Create a Thumbnail

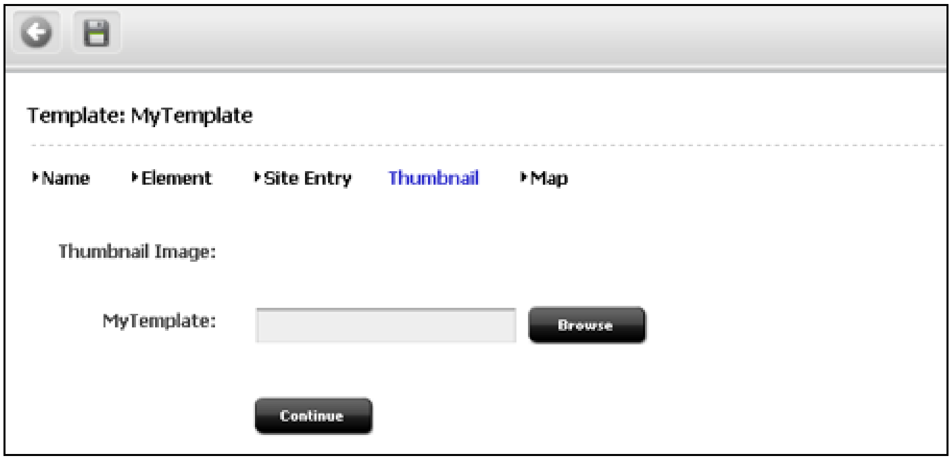
A thumbnail graphically assists template users in determining how your Template asset lays out pages or renders content. The thumbnail that you create will be displayed in the Template's **Inspect** form.

When the Template asset is saved, the name of the thumbnail file is written to the `urlthumbnail` column of the `Template_Thumb` table.

To create a thumbnail

1. Preview your Template asset. For instructions, see Section 23.7.5.2, "CSElement and SiteEntry Assets and Preview."

2. Capture the preview as an image file and save it to a file system.
3. Open the **Template** form and click **Thumbnail** at the top of the screen.
4. In the **Thumbnail Image** field, enter (or browse for) the path to the image file that you created in step 2.



5. To display the thumbnail in the **Inspect** form:
   a. Save the template (click the **Save icon**).

      WebCenter Sites uploads the image file to the WebCenter Sites database and displays template's **Inspect** form.
   b. In the **Inspect** form, scroll down to the **Thumbnail Image** section. If the displayed image is too large or too small, resize the image in its source file and repeat steps 4 and 5.
6. To operate in the image in the **Thumbnail** screen:
   a. Scroll to the top of the **Inspect** form, and click the **Edit** link.
   b. At the top of the **Template** form, click **Thumbnail**.
7. To copy, send, and perform other operations on the thumbnail, right-click on the thumbnail and select an option.
8. If you wish to delete the thumbnail, select **Delete thumbnail image?** and click **Save Changes**.

   WebCenter Sites displays the template's **Inspect** form.

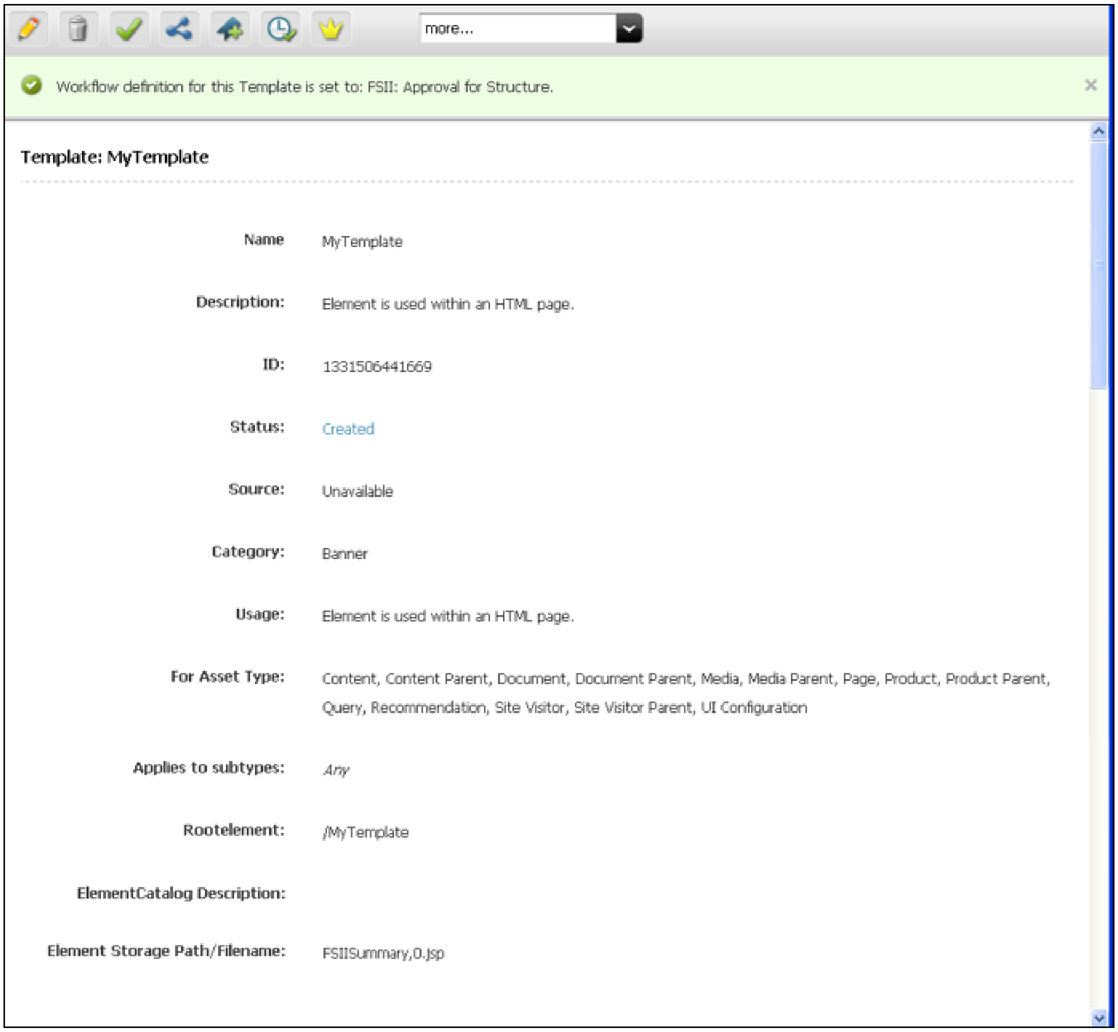
23.4.2.7 Step 7: Inspect the Template

When you have finished creating the Template asset and clicked **Save**, WebCenter Sites does the following:

- Writes to the database tables:
  – Creates a template entry in the Template table.
  – Creates an element entry in the `ElementCatalog` table, using the *`AssetTypeName/TemplateName`* naming convention. If the element was coded in the template form (rather than Sites Explorer), WebCenter Sites also creates the element file.

  - Determines the name of the site that the template belongs to and creates a page entry in the `SiteCatalog` table using the *`SiteName/AssetTypeName/TemplateName`* naming convention.
  - Sets the name of the root element of the new `SiteCatalog` page entry to the name of the `ElementCatalog` entry.
  - Creates a thumbnail entry in the `Template_Thumb` table.
  - Creates a map entry in the `Template_Map` table.
- Displays the **Inspect** form (Figure 23–5), which provides the following kinds of information:
  - Information in the **Name** screen (standard summary information, such as asset name, description, status, source, and ID, for assets of all types).
  - Information in the **Element** screen (root element, element logic, path to the element file, and `tid`).
  - Information in the **SiteEntry** screen (SiteCatalog pagename, pagelet parameters, cache criteria, and the ACLs of users who are authorized to view the page).
  - Information in the **Thumbnail** screen (a thumbnail image, if one was chosen).
  - Information in the **Map** screen (a map of key-value-asset information, if the site was designed to be replicable, or the template is sharable).
  - If you have shared the Template asset, the **Inspect** form also lists all of the additional page entries in the `SiteCatalog` for this Template asset (there is a page entry for each site that the template is shared with).

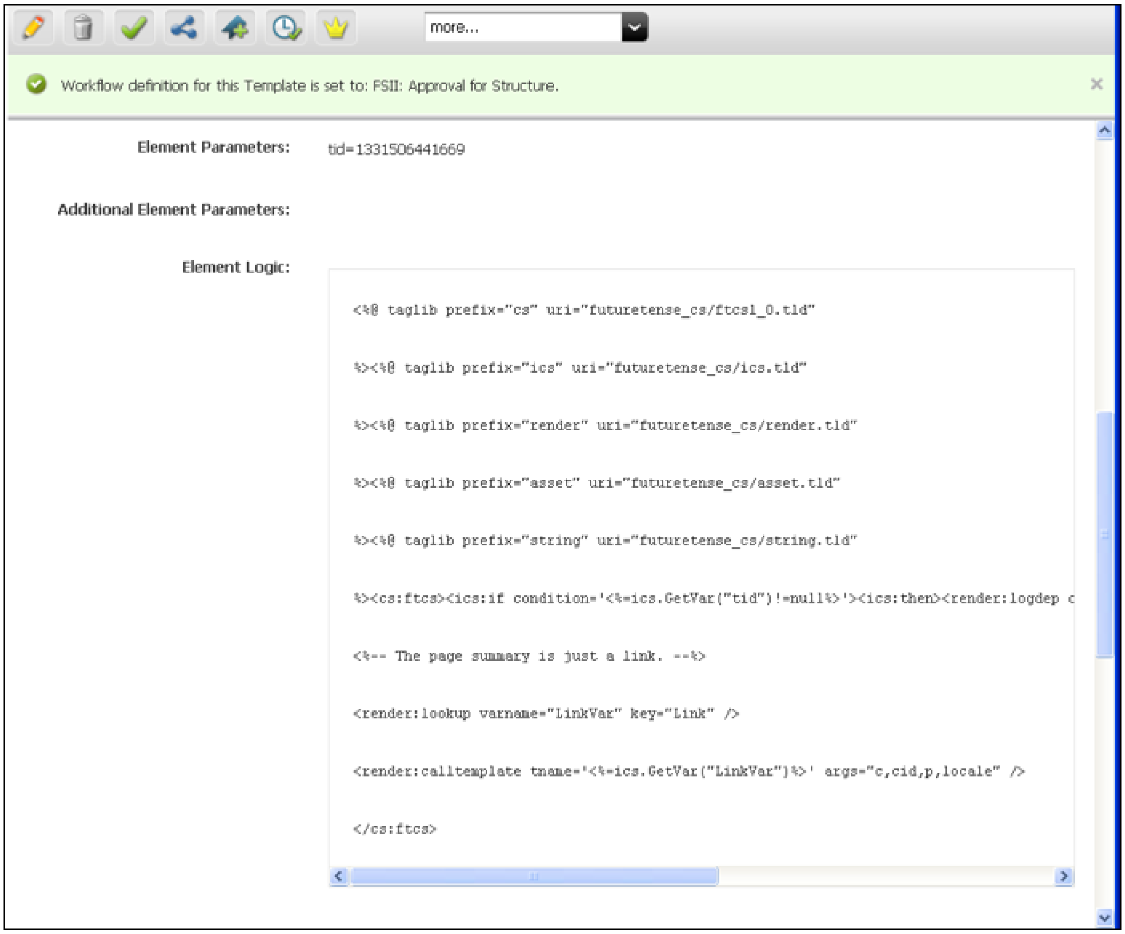
Figure 23–2 Template Asset: Sample Inspect Form



Workflow definition for this Template is set to: FSII: Approval for Structure.

Template: MyTemplate

Name	MyTemplate
Description:	Element is used within an HTML page.
ID:	1331506441669
Status:	Created
Source:	Unavailable
Category:	Banner
Usage:	Element is used within an HTML page.
For Asset Type:	Content, Content Parent, Document, Document Parent, Media, Media Parent, Page, Product, Product Parent, Query, Recommendation, Site Visitor, Site Visitor Parent, UI Configuration
Applies to subtypes:	*Any*
Rootelement:	/MyTemplate
ElementCatalog Description:	
Element Storage Path/Filename:	FSIISummary,0.jsp



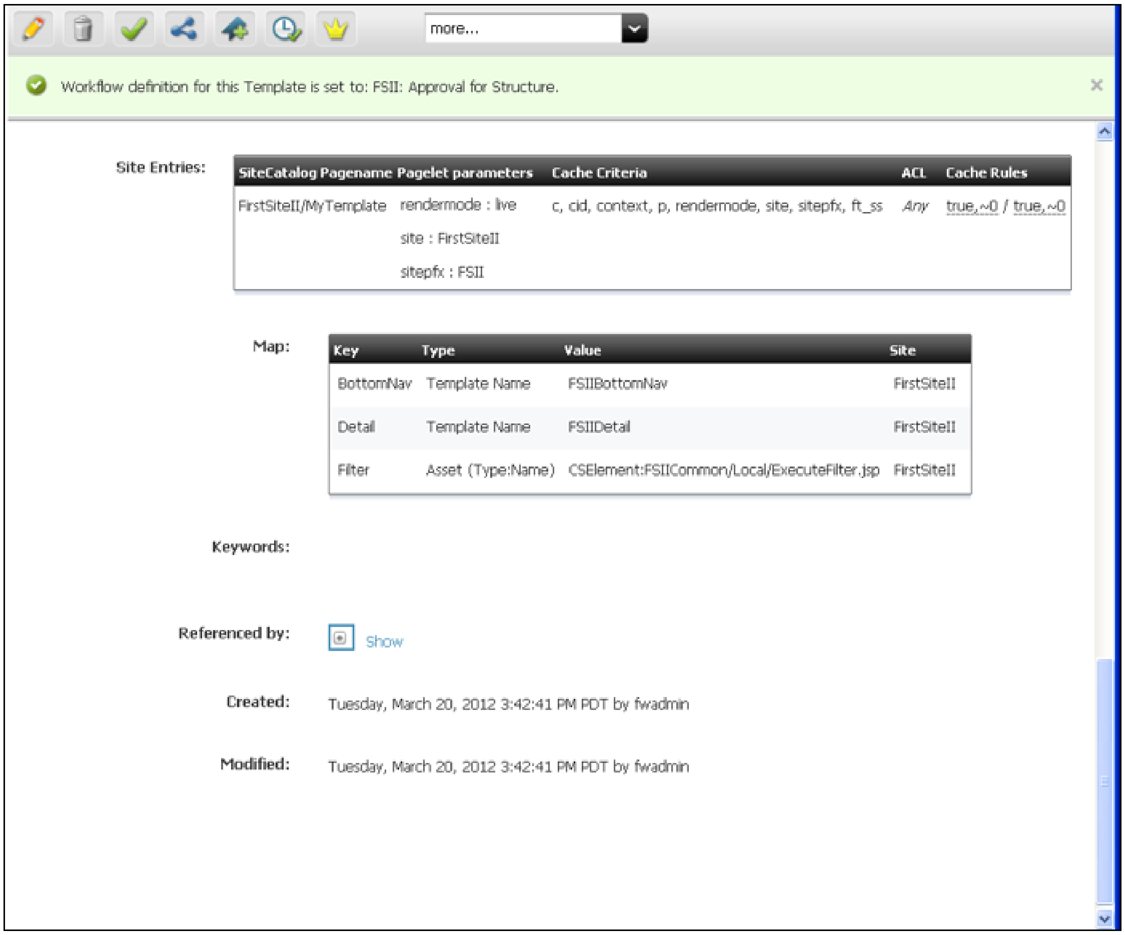
Workflow definition for this Template is set to: FSII: Approval for Structure.

Element Parameters: tid=1331506441669

Additional Element Parameters:

Element Logic:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"

%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"

%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"

%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"

%><%@ taglib prefix="string" uri="futuretense_cs/string.tld"

%><cs:ftcs><ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep c

<%-- The page summary is just a link. --%>

<render:lookup varname="LinkVar" key="Link" />

<render:calltemplate tname='<%=ics.GetVar("LinkVar")%>' args="c,cid,p,locale" />

</cs:ftcs>
```



Workflow definition for this Template is set to: FSII: Approval for Structure.

Site Entries:

SiteCatalog Pagename	Pagelet parameters	Cache Criteria	ACL	Cache Rules
FirstSiteII/MyTemplate	rendermode : live site : FirstSiteII sitepfx : FSII	c, cid, context, p, rendermode, site, sitepfx, ft_ss	*Any*	true,~0 / true,~0

Map:

Key	Type	Value	Site
BottomNav	Template Name	FSIIBottomNav	FirstSiteII
Detail	Template Name	FSIIDetail	FirstSiteII
Filter	Asset (Type:Name)	CSElement:FSIICommon/Local/ExecuteFilter.jsp	FirstSiteII

Keywords:

Referenced by: Show

Created: Tuesday, March 20, 2012 3:42:41 PM PDT by fwadmin

Modified: Tuesday, March 20, 2012 3:42:41 PM PDT by fwadmin

23.5 Creating CSElement Assets

When you create a CSElement asset, you do three things: you create an asset, you code an element for the asset, and you configure a key-value-asset information map (similar to the map for a Template asset).

To create a CSElement asset, you must first complete Section 23.5.1, "Prerequisites" to determine how you will set CSElement properties that cannot (or must not) be changed once the CSElement is saved, and how you will code the CSElement's element logic. You will then complete the following steps, using the WebCenter Sites interface:

- Step 1: Open the 'CSElement' Form
- Step 2: Name and Describe the CSElement Asset
- Step 3: Configure the Element (To specify its file type and logic.)
- Step 4: Configure the Map (If you wish to support CSElement sharing and site replication.)
- Step 5: Save and Inspect the CSElement
- Step 6: Add the CSElement to Bookmarks (If you plan to use the CSElement as a root element for a Site Entry asset. See Section 23.6, "Creating SiteEntry Assets.")

Information that you enter into the **CSElement** form will be written to database tables when the CSElement asset is saved, as indicated in the procedures below.

> **Note:** Do not create CSElement assets directly in the database tables. Doing so will require you to write to several tables and can result in incorrect tracking of dependencies. Instead, use the **CSElement** form and the procedures in this section to create CSElement assets. For help with coding the CSElement logic, see Chapter 28, "Coding Elements for Templates and CSElements."

This section contains the following topics:

- Section 23.5.1, "Prerequisites"
- Section 23.5.2, "Procedures for Creating CSElement Assets"

23.5.1 Prerequisites

Before you begin creating a CSElement asset, you must determine several things:

- A name for your CSElement asset.
- Whether your CSElement will be sharable and the site replicable. These considerations determine how you will code the CSElement's element logic.
- Whether you plan to code the CSElement's element logic in Sites Explorer instead of the **CSElement** form. This approach is not recommended for the reasons outlined in Section 23.8, "Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic."

23.5.1.1 Naming the CSElement

- Once the CSElement asset is saved, its name cannot be changed.
- The CSElement logic file takes the name of the CSElement (followed by the file extension:

  `CSElementName.xml_or_jsp_or_html`
  The name of the CSElement logic file must not be changed.

23.5.1.2 CSElement Sharing and Site Replication

Before creating a CSElement, decide whether the CSElement must be shared or the site you are working in must be replicable. If so, the CSElement logic will be coded in the same way. If sharing and replication are not required, you will skip key-value mapping (see Section 23.5.2.4, "Step 4: Configure the Map").

For information about coding element logic to support CSElement sharing and site replication, see Section 23.4.1.3, "Template Sharing and Site Replication." The information applies without exception to CSElement assets.

23.5.2 Procedures for Creating CSElement Assets

This section shows you how to create a CSElement asset, using the WebCenter Sites interface.

> **Note:** Before starting the procedures in this section, read Section 23.5.1, "Prerequisites" for information about creating CSElement assets.

23.5.2.1 Step 1: Open the 'CSElement' Form

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. In the button bar, click **New**.
5. In the list of asset types, select **New CSElement**.

> **Note:** For the **New CSElement** option to be displayed, the CSElement asset type must be enabled for your site and a start menu item must be created for it.

6. The **CSElement** form displays. Continue with Section 23.5.2.2, "Step 2: Name and Describe the CSElement Asset."

> **Note:** If you see a **Choose Assignees** screen instead of the **CSElement** form, it means that the CSElement you will be creating is associated with a workflow. Select a name (or names) from the Users column and click **Set Assignees**. Continue with Section 23.5.2.2, "Step 2: Name and Describe the CSElement Asset."

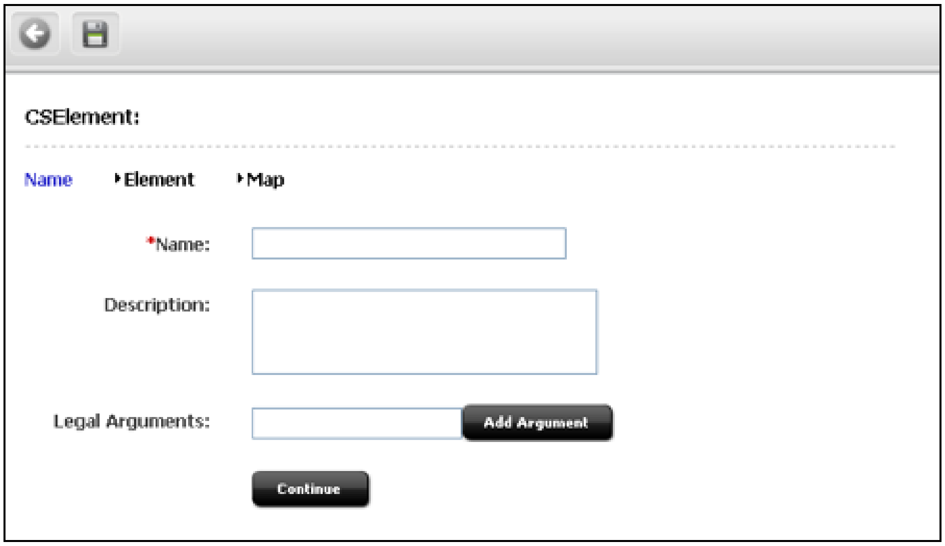
23.5.2.2 Step 2: Name and Describe the CSElement Asset

The **Name** screen is used to define metadata about the CSElement. From this metadata, a developer will be able to identify what the CSElement does and the arguments it uses to perform its function.

> **Note:** At any time in the process of creating a CSElement, you can save the CSElement. WebCenter Sites will display the CSElement's **Inspect** form. To return to the **CSElement** form, click the **Edit** link.

To name and describe the CSElement

1. In the **Name** screen, fill in the fields as explained in this section.



2. (Required). In the **Name** field, type a unique, descriptive name for the CSElement asset. It's best to use a name that describes what the CSElement does.

   Valid entries:

   – Up to 64 alphanumeric characters (the first character must be a letter)

   – Underscores (_)

   – Hyphens (-)

   – Spaces (these will be converted to underscores when used in the SiteCatalog pagename for the template)

   > **Note:** Make sure you have chosen a name for your CSElement asset using the guidelines in the section Section 23.6.1, "Prerequisites."

1. In the **Description** field, type a brief description of the CSElement asset. You can enter up to 128 characters.

2. In the **Legal Arguments** field:

   a. Enter an argument that may be passed to the CSElement and click **Add Argument.**

   b. In the fields that are displayed:

      – Specify whether the argument is optional or required.

      – Provide a description of the argument (to help you know the purpose of the argument you are creating).

      – Specify legal values (including descriptions) for the argument.

   (You can specify as many arguments and legal values as you require by clicking the **Add Arguments** and **Add Legal Value** buttons.)

3. Click **Continue** to open the next screen, **Element**.

23.5.2.3 Step 3: Configure the Element

The **Element** screen (step 1) is used to create the CSElement's element, define the element file type (XML, JSP, or HTML), provide the element logic, and name the element. For example:

- The **Create Element** field offers a choice of XML, JSP, or HTML file types for the element logic, and is used to seed the **Element Logic** field with standard stub code (which you need to include in any element that you create).

- When you use the **Create Element** field to create, for example, a .jsp file, WebCenter Sites adds JSP taglib statements and the render.logdep tag to the **Element Logic** field by default so that the compositional dependency between this CSElement asset and pages that are rendered from this element is logged. For other file types, WebCenter Sites adds code specific to the file type. You will add your own code to the **Element Logic** field.

  For information about dependencies, see Section 28.1, "About Dependencies." For help with coding the element logic, see Chapter 28, "Coding Elements for Templates and CSElements."

- The **Element Storage Path/Filename** field names the file that holds the element logic and specifies the path to the file.

When the CSElement is saved, field values in the **Element** screen are written to a row (representing the element) in the `ElementCatalog` table, as indicated in the procedures below.

Note: Selecting an Existing Element

In the steps that follow, we assume you are creating a new element for the CSElement asset. If, however, you are migrating assets from an earlier WebCenter Sites release and wish to reuse an existing element, you need to identify the element correctly so that WebCenter Sites can find it and associate it with the CSElement asset.

To select an existing element

- (Optional). In the ElementCatalog Description field, type a description of the element.
- In the **Element Storage Path/Filename** field, enter a value according to the convention in Section 23.5.1.1, "Naming the CSElement."
- If your site design requires it, enter the appropriate arguments in the element parameter fields. For instructions, see step 1.
- Save and re-open the CSElement asset.

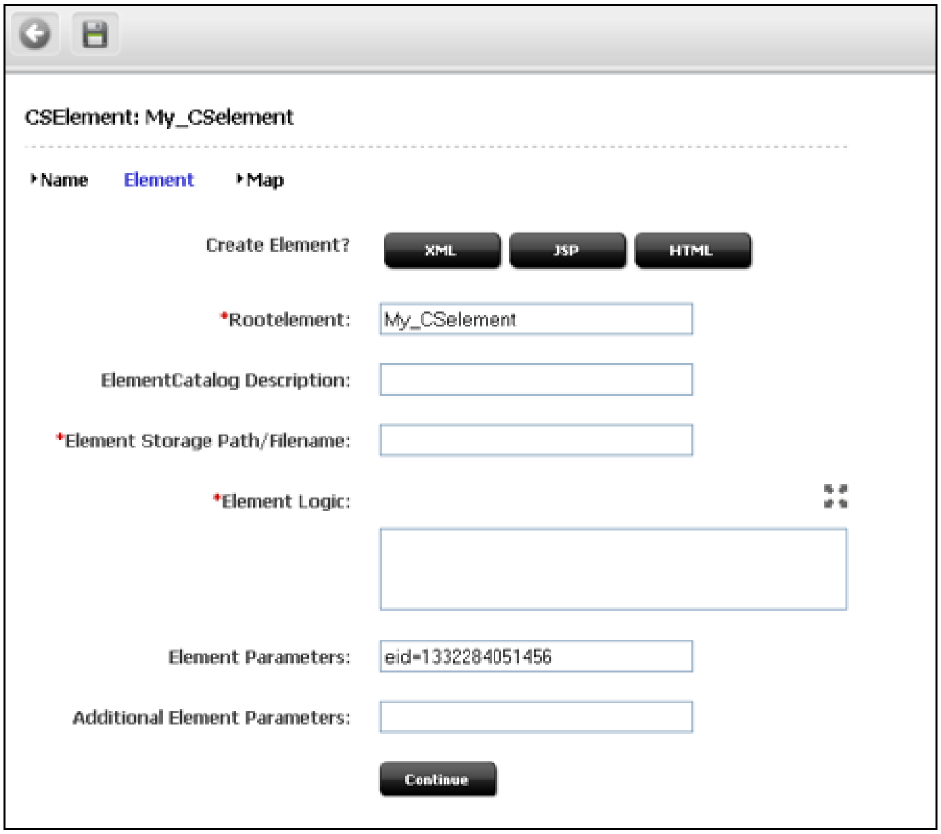
WebCenter Sites checks for the presence of an element with the correct name:

If the element has been correctly named, WebCenter Sites recognizes the element and displays its code in the **Element Logic** field.

If the named element does not exist (or is incorrectly named), WebCenter Sites does nothing. When you inspect or edit the CSElement asset, WebCenter Sites displays a message stating that there is no root element in the form. As soon as you code the element and give it the correct name, WebCenter Sites detects it and associates it with the CSElement asset.

To configure a new element

1. In the **Element** screen, fill in the fields as explained in this section.



CSElement: My_CSelement

Create Element?

*Rootelement: My_CSelement

ElementCatalog Description:

*Element Storage Path/Filename:

*Element Logic:

Element Parameters: eid=1332284051456

Additional Element Parameters:

2. In the **Create Element** field, do one of the following:

   – To create an .xml file, click **XML**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\CSElement\modelXML.xml` element and can be modified to use custom default logic.

   – To create a .jsp file, click **JSP**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\CSElement\modelJSP.xml` element and can be modified to use custom default logic.

   – To create an .html file, click **HTML**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\CSElement\modelHTML.xml` element and can be modified to use custom default logic.

   WebCenter Sites populates the following fields:

   – **Element Storage Path/Filename field. Do not change the value of this field.**

     This field displays the element file name preceded by the path to the element file. By default, the file takes the name of the CSElement asset (entered in step 2) followed by the file extension:

     `CSElementName.xml_or_jsp_or_html`

     When you save the CSElement asset, the value in this field is written to the url column of the ElementCatalog table, for the row that represents the element.

   – **Element Logic** field with a header and other information.

     For example, if you clicked the **JSP** button, WebCenter Sites sets a tag library directive for some common WebCenter Sites JSP tag libraries (`asset, siteplan, render`). WebCenter Sites also sets the beginning and ending `cs:ftcs` tags, and a `RENDER.LOGDEP (render:logdep)` tag to mark a compositional dependency between the CSElement asset and any page or pagelet rendered by the element.

3. The **Rootelement** field is pre-populated with the name of the element file *(CSElementName.xml_or_jsp_or_html)*. **Do not change the value of this field.**

   The root element is listed by this name in the `ElementCatalog` table. When you create code that calls this element (`RENDER.CALLELEMENT`), this is the name you should use. It uses the name of the CSElement asset by default.

4. (Optional). In the **ElementCatalogDescription** field, type a description of the element.

   When you save the CSElement asset, information in this field is written to the `description` column for the element entry in the ElementCatalog table.

5. (Required). In the **Element Logic** field, code your element. Be sure to enter all of your code before the ending `cs:ftcs` tag.

   If you are using JSP, remove the comments from the taglib directives that describe the tag families you are using.

   For help with this step, see Chapter 28, "Coding Elements for Templates and CSElements."

   Note: Ensuring Template Sharing or a Replicable Site
   If you wish to share your CSElement or make the current site replicable, make sure that the CSElement's element logic does not directly refer to assets, asset types, attribute names, or template names. Instead, use the `render:lookup` tag and prescribed keys as explained in Section 23.4.1.3, "Template Sharing and Site Replication." In Section 23.5.2.4, "Step 4: Configure the Map," you will map the keys to the asset information that must be accessed for use in the element logic.

   Calling a Template
   Templates should *always* be called by the `render:calltemplate` tag, and never the `render:callelement` tag or `render:satellitepage` tag.

6. (Optional). The **Element Parameters** field and **Additional Element Parameters** field are used to enter variables or arguments that can be passed to the element, if the site design requires them.

   – **Element parameters** field. WebCenter Sites populates this field with the CSElement ID (`eid`), generated by WebCenter Sites as a unique identifier of the CSElement asset. **Do not change or delete this value**.

     This field corresponds to the resdetails1 column of the ElementCatalog table. When you save the CSElement, WebCenter Sites writes the CSElement ID to the `resdetails1` column, in the row that represents the CSElement.

   – **Additional element parameters** field. WebCenter Sites leaves this field blank.

     This field corresponds to the `resdetails2` column of the `ElementCatalog`.

   If your site design requires you to use variables in addition to eid, enter the variables into one of the fields above. Enter them as *`name=value`* pairs with multiple arguments separated by the ampersand (&) character. For example:

   ```
   MyArgument=value1&YourArgument=value2
   ```
   Each field supports up to 255 characters.

For more information about WebCenter Sites variables, including scope and precedence, see Chapter 4, "Programming with Oracle WebCenter Sites."

7. Click **Continue** to open the next screen, **Map**.

23.5.2.4 Step 4: Configure the Map

The purpose of mapping is to enable site replication and sharing of CSElement assets. The concepts behind mapping are identical to those for Template assets. They are explained in Section 23.4.1.3, "Template Sharing and Site Replication."

> **Note:** Skip this section if you are designing a non-replicable site or a CSElement asset that will not be shared.

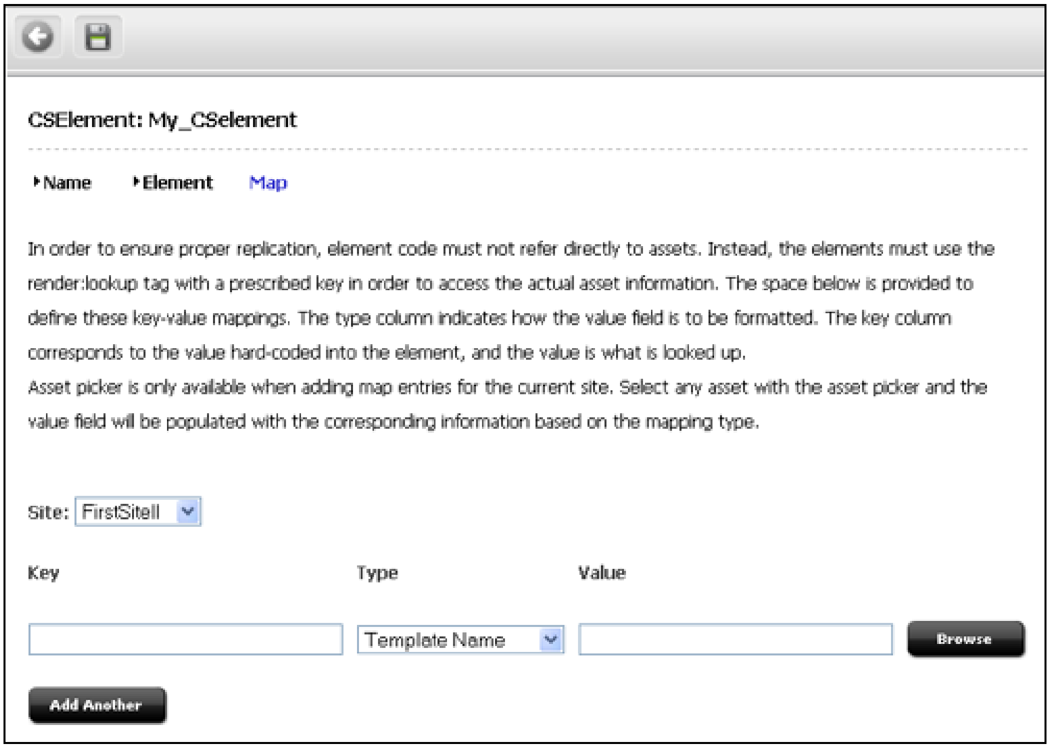
Using the **Map** screen, you will:

- Map each key in the `render:lookup` tag of the element logic to the value that must be used in the element logic.
- Map each key's value to the asset information that must be used in the element logic: asset, asset type, attribute name, or template name.

When the CSElement asset is saved, the map is written to the `CSElement_Map` table.

To configure a map

1. In the **Map** form, fill in the fields as explained in this section.



CSElement: My_CSelement

In order to ensure proper replication, element code must not refer directly to assets. Instead, the elements must use the render:lookup tag with a prescribed key in order to access the actual asset information. The space below is provided to define these key-value mappings. The type column indicates how the value field is to be formatted. The key column corresponds to the value hard-coded into the element, and the value is what is looked up.
Asset picker is only available when adding map entries for the current site. Select any asset with the asset picker and the value field will be populated with the corresponding information based on the mapping type.

Site: FirstSiteII

Key

Type: Template Name

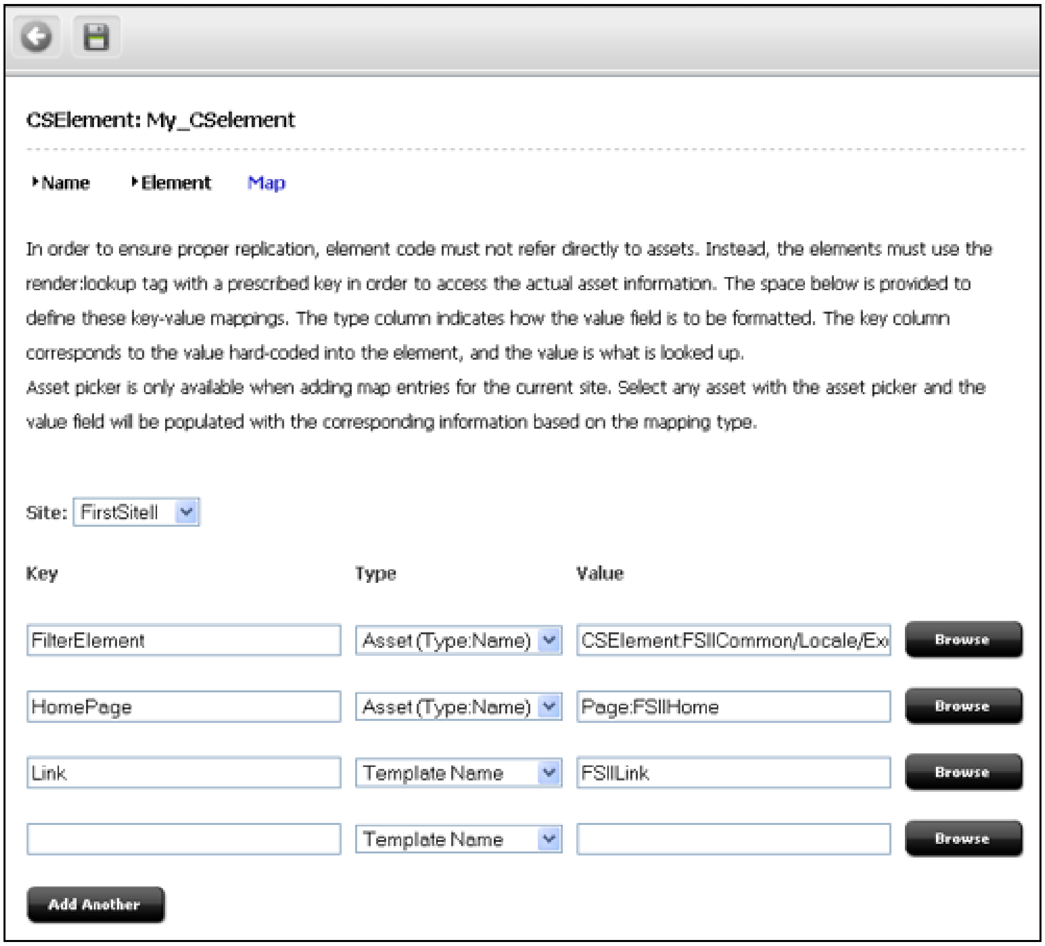
Value

2. The **Key** field represents the value that the element logic will look up. In this field, enter the key that is named in a `render:lookup` tag of the element logic.

3. The **Type** field identifies the type of asset information to be accessed. Select one of the following options:

- **Template Name**: Maps a template name to the key value (which you will specify in the **Value** field, in the next step). The information that will be accessed is a template name that matches the value you will specify in the next step. (For an example, see Figure 23–1.)
- **Asset Type**: Maps an asset type to the key value. The information that will be accessed is an asset type, equal to the value that you will specify in the next step.
- **Asset (Type:Name)**: Maps an attribute type:name to the key value. The information that will be accessed is an asset whose type and name match the value that you will specify in the next step.
- **Asset (Type:ID)**: Maps an attribute type:ID to the key value. The information that will be accessed is an asset whose type and name match the value that you will specify in the next step.

Figure 23–3 CSElement Asset: Sample Map



CSElement: My_CSelement

In order to ensure proper replication, element code must not refer directly to assets. Instead, the elements must use the render:lookup tag with a prescribed key in order to access the actual asset information. The space below is provided to define these key-value mappings. The type column indicates how the value field is to be formatted. The key column corresponds to the value hard-coded into the element, and the value is what is looked up.
Asset picker is only available when adding map entries for the current site. Select any asset with the asset picker and the value field will be populated with the corresponding information based on the mapping type.

Site: FirstSiteII

Key	Type	Value	
FilterElement	Asset (Type:Name)	CSElement:FSIICommon/Locale/Ex	Browse
HomePage	Asset (Type:Name)	Page:FSIIHome	Browse
Link	Template Name	FSIILink	Browse
	Template Name		Browse

4. In the **Value** field, enter a value for the key. This value will be looked up by the element logic when the CSElement asset is invoked.
5. In the **siteid** field, select the name of the site to which the mapping applies.
6. To add a key, click **Add Another** and repeat the steps in this section.

23.5.2.5 Step 5: Save and Inspect the CSElement

When you have finished creating the CSElement asset, click Save.

WebCenter Sites does the following:

- Writes to the database tables:
  - Creates a row in the CSElement table for the CSElement asset, where it enters the CSElement name and description that you specified in the previous steps.
  - Creates an element entry in the ElementCatalog table using values specified in the **Element** screen:
    a. The value of the Rootelement field is used to position the element file in the appropriate folder.
    b. The value of the Element Storage Path/Filename field is written to the url column.
    c. The value of the eid variable is set to the ID of the CSElement asset in the resdetails1 column.
- Flushes the pagecache of any pagelets that call this element.
- Displays the **Inspect** form (Figure 23–5), which provides the following kinds of information:
  - Information in the **Name** screen (standard summary information, such as asset name, description, status, and ID, for assets of all types).
  - Information in the **Element** screen (root element, element logic, path to the element file, and the element's eid).
  - Information in the **Map** screen (a map of key-value-asset information, if the site was designed to be replicable, or the template is sharable).
  - **Preview with Arguments** button, enabling you to preview the page(s) rendered by the SiteEntry asset.

23.5.2.6 Step 6: Add the CSElement to Bookmarks

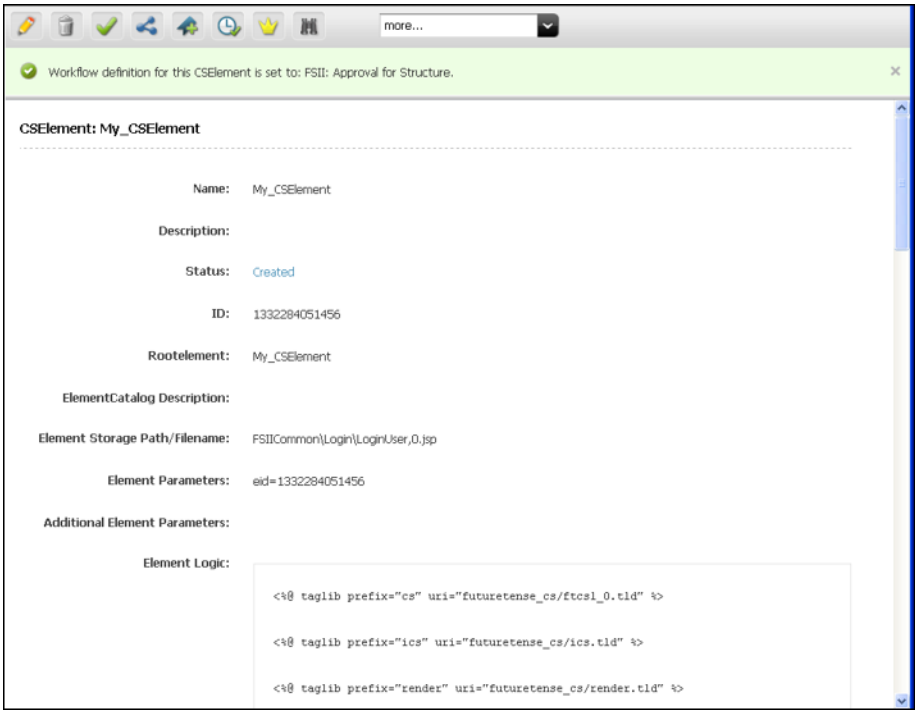
> **Note:** Complete the steps in this section if you are planning to use your CSElement to create a SiteEntry asset. This step makes the CSElement available for selection in WebCenter Sites's tree by adding it to your Bookmarks.
>
> If you are not planning to create the SiteEntry asset in this session, you might want to add the CSElement to your Bookmarks so that you can easily find it later.

To add the CSElement to Bookmarks

1. Run a search on the CSElement asset you created.
2. In the results list, select the check box in the right-hand column to add the CSElement to **Bookmarks**.

   This CSElement is listed in WebCenter Sites's tree, in **Bookmarks** tab, where it is a selectable option for SiteEntry assets.
3. Create the SiteEntry asset. For instructions, see Section 23.6, "Creating SiteEntry Assets."

Figure 23–4 CSElement Asset: Sample Inspect Form



Workflow definition for this CSElement is set to: FSII: Approval for Structure.

CSElement: My_CSElement

Name:	My_CSElement
Description:	
Status:	Created
ID:	1332284051456
Rootelement:	My_CSElement
ElementCatalog Description:	
Element Storage Path/Filename:	FSIICommon\Login\LoginUser,0.jsp
Element Parameters:	eid=1332284051456
Additional Element Parameters:	
Element Logic:	

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>

<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>

<%@ taglib prefix="render" uri="futuretense_cs/render.tld" %>
```

23.6 Creating SiteEntry Assets

When you create a SiteEntry asset, you are creating both an asset and a page entry in the SiteCatalog table. The fields in the first part of the **SiteEntry** form define the page entry as an asset. The rest of the fields provide information about the page entry as a WebCenter Sites page, information that is written to the SiteCatalog table.

To create a SiteEntry asset, you must first complete Section 23.5.1, "Prerequisites" to create its root element and determine how you will set SiteEntry properties (such as SiteEntry name). You will then complete the following steps, using the WebCenter Sites interface:

- Section 23.6.2.1, "Step 1: Open the 'SiteEntry' Form"
- Section 23.6.2.2, "Step 2: Create the SiteEntry Asset"
- Section 23.6.2.3, "Step 3. Save and Inspect the SiteEntry Asset"

Information that you enter into the **SiteEntry** form will be written to database tables when the CSElement asset is saved, as indicated in the procedures below.

> **Note:** Do not create SiteEntry assets directly in the database tables. Doing so will require you to write to several tables and can result in incorrect tracking of dependencies. Instead, use the **SiteEntry** form and the procedures in this section to create SiteEntry assets.

This section contains the following topics:

- Section 23.6.1, "Prerequisites"
- Section 23.6.2, "Procedures for Creating SiteEntry Assets"

23.6.1 Prerequisites

Before you begin creating a SiteEntry asset, complete the following steps:

- Create a root element for the page entry:

  a. Create a CSElement asset. For instructions, see Section 23.5, "Creating CSElement Assets."

     (A root element is required for any page entry. The root element is the element of the CSElement, which you will select for the SiteEntry asset.)

  b. Make the CSElement available. For instructions, see Section 23.5.2.6, "Step 6: Add the CSElement to Bookmarks."

     (To specify a CSElement for your SiteEntry asset, you will select the CSElement from the **Bookmarks** tab in WebCenter Sites's tree, or the **History** tab if you created the CSElement in the current session).

- Determine a name for your SiteEntry asset. The same name will be assigned to the page. Neither the SiteEntry name nor the page name can be changed once the SiteEntry asset is saved.

23.6.2 Procedures for Creating SiteEntry Assets

This section shows you how to create a SiteEntry asset, using the WebCenter Sites interface.

> **Note:** Before starting the procedures in this section, read Section 23.6.1, "Prerequisites" for information about creating SiteEntry assets.

23.6.2.1 Step 1: Open the 'SiteEntry' Form

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. In the button bar, click **New**.
5. In the list of asset types, select **New SiteEntry**.

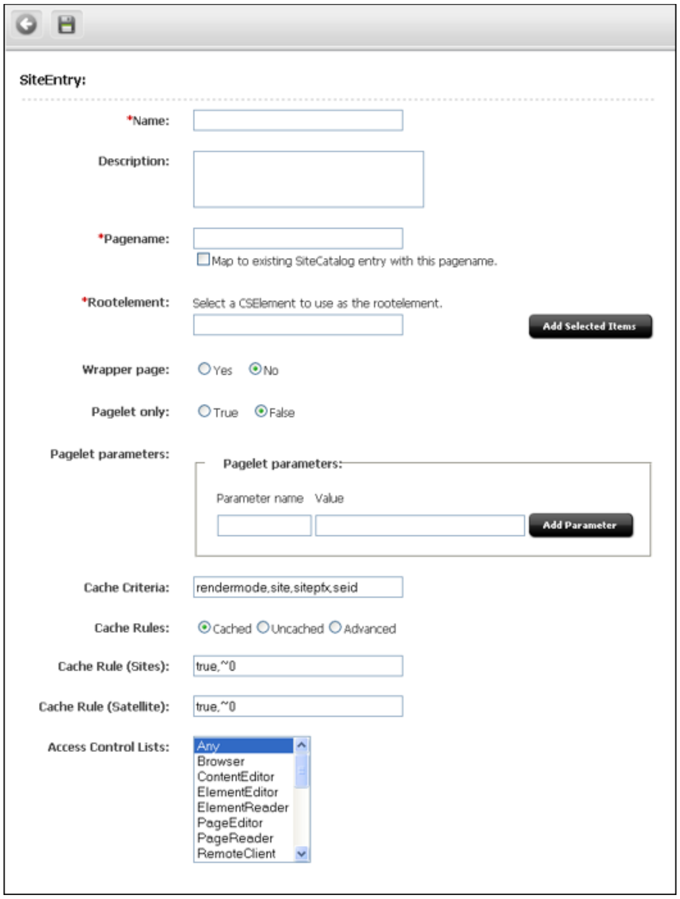
> **Note:** For the **New SiteEntry** option to be displayed, the SiteEntry asset type must be enabled for your site and there must be a start menu item created for it.

6. The **SiteEntry** form displays. Continue with Section 23.5.2.2, "Step 2: Name and Describe the CSElement Asset."

> **Note:** If you see a **Choose Assignees** screen instead of the **SiteEntry** form, it means that the SiteEntry you will be creating is associated with a workflow. Select a name (or names) from the Users column and click **Set Assignees**. Continue with Section 23.5.2.2, "Step 2: Name and Describe the CSElement Asset."

23.6.2.2 Step 2: Create the SiteEntry Asset

1. In the **SiteEntry** form, fill in the fields as explained in this section.



SiteEntry:

*Name:

Description:

*Pagename:

□ Map to existing SiteCatalog entry with this pagename.

*Rootelement: Select a CSElement to use as the rootelement.

Wrapper page: ○ Yes ● No

Pagelet only: ○ True ● False

Pagelet parameters:

Pagelet parameters:

Parameter name Value

Cache Criteria: rendermode,site,sitepfx,seid

Cache Rules: ● Cached ○ Uncached ○ Advanced

Cache Rule (Sites): true,~0

Cache Rule (Satellite): true,~0

Access Control Lists: Any, Browser, ContentEditor, ElementEditor, ElementReader, PageEditor, PageReader, RemoteClient

2. (Required). In the **Name** field, type a descriptive name for the SiteEntry asset. It's best to use a name that describes the purpose of the page.

   Valid entries:

   – Up to 64 alphanumeric characters (the first character must be a letter)

   – Underscores (_)

   – Hyphens (-)

   – Spaces (these will be converted to underscores when used in the SiteCatalog pagename for the template).

3. In the **Description** field, type a brief description of the SiteEntry asset. You can enter up to 128 characters.

4. (Required). Click in the **Pagename** field to automatically populate it with the name of the page entry and the path to the page entry (for example: `FSIICommon/SideNav/ProductView`). **Do not change the value of this field.**

   > **Note:** The value in this field is the name of the page entry (which will be stored in the `SiteCatalog` table when the SiteEntry is saved). When you create code that calls this SiteEntry asset `(RENDER.SATELLITEPAGE)`, this is the name you should use.

5. If you wish to use the row of a pre-existing page entry in the SiteCatalog table, select the **Map to existing SiteCatalog entry** check box.
6. (Required). In the **Rootelement** field, select the appropriate CSElement asset from the tree and click **Add Selected Items**.

> **Note:** Only one CSElement can be added.

7. In the **Wrapper page** field, select one of the options to specify whether the asset you are creating is a wrapper page. Selecting the **NO** option displays the **Pagelet only** field.
8. If the **Pagelet only** field is displayed, select one of the options to specify whether the asset you are creating is a pagelet.
9. In the **Pagelet parameters** section, you can enter pagelet parameters (name-value pairs), which will be passed into the page or pagelet each time it is executed. (The **Pagelet parameters** section supports a total of 510 characters).

> **Note:** The **Pagelet parameters** section is pre-populated with the following default parameters (reserved variables that are named by default in the Cache Criteria field (next step), including their values: `site, seid, sitepfx, rendermode`
>
> The default values will be overwritten if they are explicitly specified when the page or pagelet is called.
>
> - If you are specifying a pagelet parameter in this step, make sure to list its name as a Cache Criteria variable in the next step.

When the SiteEntry asset is saved, the name-value pairs that are specified as **Pagelet parameters** are written to either the resargs1 or resargs2 column of the `SiteCatalog` table. The column to which they are written is not important and is managed automatically. (Each column supports up to 255 characters.)

10. In the **Cache Criteria** field:
    a. WebCenter Sites names the following **reserved** variables as Cache Criteria:

```
rendermode,seid,site,sitepfx,ft_ss
```

> **Note:** The reserved Cache Criteria variables should not be removed. For information about the reserved variables, see Chapter 4, "Programming with Oracle WebCenter Sites."

11. If you need to include your own variables as Cache Criteria (for example, foo), add them to the existing list. For example

```
foo,rendermode,seid,site,sitepfx,ft_ss
```

> **Note:** The **Cache Criteria** field names the variables which, in conjunction with Pagename, define a pagelet as being unique. The variables are used to identify cached pages, which means that the variables are used in the page's cache key.
>
> Only those variables that are specified as Cache Criteria are used by the caching system to create the cache key for cached pages. Therefore, if your site design requires you to use page-level variables in addition to the reserved variables, be sure to designate them as Cache Criteria variables, as shown in this step.

When the SiteEntry asset is saved, Cache Criteria variables and their values are written to the `pagecriteria` column in the SiteCatalog table.

12. The **Cache Rules** field corresponds to the `cscacheinfo` and `sscacheinfo` columns in the `SiteCatalog` table. Do one of the following:
    - Select **Cached** if the pagelet to be rendered by this SiteEntry's CSElement must be cached. The pagelet is set to be cached forever. The cache will be flushed by CacheManager's active cache management logic. This option sets both WebCenter Sites and Satellite Server caching conditions.
    - Select **Uncached** if you wish to turn off caching for the pagelet to be rendered by this SiteEntry's CSElement. This option sets both WebCenter Sites and Satellite Server caching conditions.
    - Select **Advanced** if you wish to set caching rules individually for WebCenter Sites and Satellite Server. Selecting **Advanced** displays two additional fields: one for WebCenter Sites caching and one for Satellite Server caching.

> **Note:** CacheManager is designed to manage the lifecycle for cached pages on both WebCenter Sites and Satellite Server. It is designed to operate with pages that are set to be cached forever. If the cache expires on WebCenter Sites before it expires on Satellite Server, CacheManager will fail to flush the cache properly and invalid pages may be served from cache. Only advanced users should configure these settings manually.
>
> For more information about page caching settings, see Chapter 5, "Page Design and Caching."

13. The **Access Control Lists** field corresponds to the `acl` column in the `SiteCatalog` table. If you want to allow only certain visitors to request this page, enter the ACLs that visitors must have in order to see the page. For more information about ACLs, see Chapter 31, "User Management on the Delivery System."

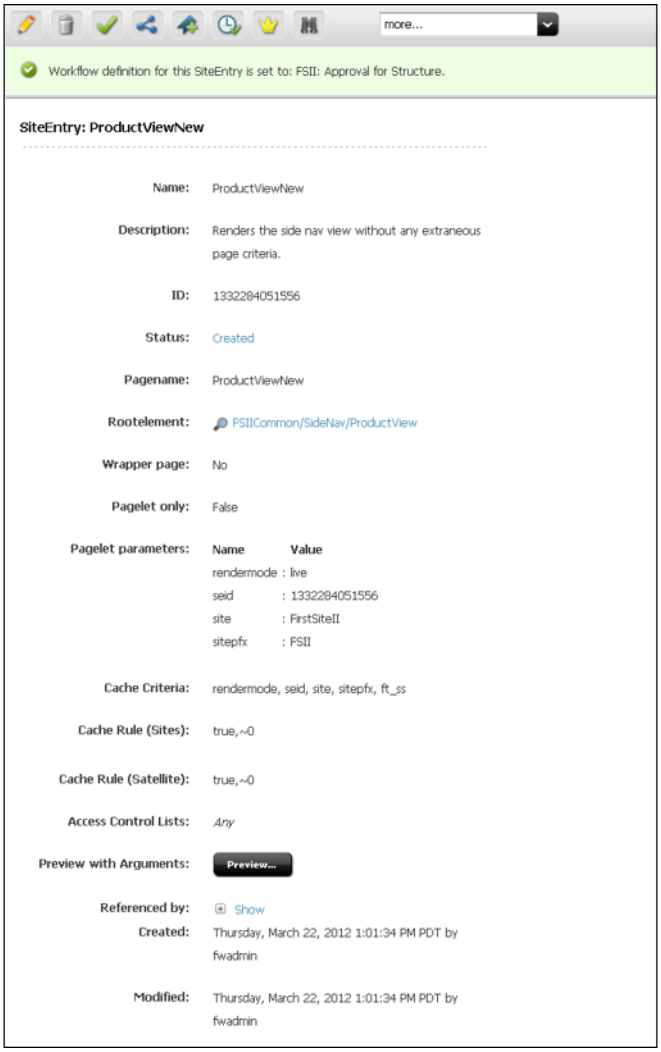
23.6.2.3 Step 3. Save and Inspect the SiteEntry Asset

When you have finished creating the SiteEntry asset, click **Save**. WebCenter Sites does the following:

- Writes to the database tables:
  - Creates a row in the `SiteCatalog` table for the SiteEntry asset, where it enters the values that you specified in the previous steps.
- Displays the **Inspect** form, which provides the following information:

- Standard summary information (asset name, description, status, ID) and the page entry criteria you specified in the previous steps.
- **Preview with Arguments** button, enabling you to preview the page(s) rendered by the SiteEntry asset.

Figure 23–5 Inspect the Asset



Workflow definition for this SiteEntry is set to: FSII: Approval for Structure.

SiteEntry: ProductViewNew

Name: ProductViewNew

Description: Renders the side nav view without any extraneous page criteria.

ID: 1332284051556

Status: Created

Pagename: ProductViewNew

Rootelement: FSIICommon/SideNav/ProductView

Wrapper page: No

Pagelet only: False

Pagelet parameters:

Name		Value
rendermode	:	live
seid	:	1332284051556
site	:	FirstSiteII
sitepfx	:	FSII

Cache Criteria: rendermode, seid, site, sitepfx, ft_ss

Cache Rule (Sites): true,~0

Cache Rule (Satellite): true,~0

Access Control Lists: *Any*

Preview with Arguments:

Referenced by: ⊞ Show

Created: Thursday, March 22, 2012 1:01:34 PM PDT by fwadmin

Modified: Thursday, March 22, 2012 1:01:34 PM PDT by fwadmin

23.7 Managing Template, CSElement, and SiteEntry Assets

This section presents additional procedures for working with template, CSElement, and SiteEntry assets and contains the following topics:

- Section 23.7.1, "Designating Default Approval Templates (Static Publishing Only)"
- Section 23.7.2, "Editing Template, CSElement, and SiteEntry Assets"
- Section 23.7.3, "Sharing Template, CSElement, and SiteEntry Assets"
- Section 23.7.4, "Deleting Template, CSElement, and SiteEntry Assets"
- Section 23.7.5, "Previewing Template, CSElement, and SiteEntry Assets"

23.7.1 Designating Default Approval Templates (Static Publishing Only)

When assets are approved for a publishing destination that uses the Export to Disk publishing method, the approval system examines the template assigned to the asset to determine its dependencies.

If you design your online site to render assets with more than one template (a text-only version and a summary version and a full version for the same type of asset, for example), you should create a template that contains a representative set of approval dependencies for all of the templates, and then specify that template as the Default Approval Template for the asset type.

For more information about approval templates, see Section 28.1.1.3, "Approval Templates for Export to Disk." For an example of a template that could be used as a default approval template, see the Burlington Financial template for article assets named Full.

To designate that a template is the default approval template

1. On the **Admin** tab, select **Publishing**, then **Destinations** and then **Static**.
2. Under the name of a static destination, select **Set Default Templates**.
3. In the Default Templates form, click **Edit**.
4. In the edit form, select a default template for each asset type. If you are using the Subtype feature for any of your asset types, you can designate a default approval template for each subtype of that asset type. For information about subtypes, see Section 15.2.10, "Step 8: (Optional) Configure Subtypes."
5. When you have finished, click **Save**.

23.7.2 Editing Template, CSElement, and SiteEntry Assets

Creating a template, CSElement, and SiteEntry assets also creates entries in the SiteCatalog and/or ElementCatalog tables. The names of those entries are based on the asset's name, and for Template assets, the asset type, and the site the template belongs to. Because these naming dependencies exist, the following restrictions apply when you edit templates, CSElements, or SiteEntry assets:

- You cannot rename a template, CSElement, or SiteEntry asset after it has been saved.
- For templates, you cannot change the asset type selected in the **Asset Type** field after the Template asset has been saved.
- For templates and CSElements, you cannot change the name of the root element.
- For SiteEntries, you cannot change the name of the page entry.

> **Caution:** If you have manually created (using Sites Explorer) one or more site entries that point to a template, and then edit the template through the Admin interface, the manually created site entries are automatically deleted.

For the basic procedure on editing assets, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide.*

23.7.3 Sharing Template, CSElement, and SiteEntry Assets

When you share a CSElement, template, or SiteEntry asset, WebCenter Sites creates a row in the AssetPublication table for each site that you share the asset with.

Additionally, for **Template assets only**, WebCenter Sites does the following:

- Creates a new SiteCatalog page entry for each site that you share the asset with. It uses the name of the site in the name of the page entry. All of the new page entries point to the same root element, the template element.

> **Note:** Do not change the root elements of these page entries. All page entries for a shared template must point to the same root element.

- Lists all the other page entries for the shared template that share this root element in the **Inspect** form.

For the basic procedure on sharing assets, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

> **Note:** For templates and CSElements to be sharable, their element logic must not be hard-coded with asset type names, attribute names, template names, or IDs. Instead, use the render:lookup tag and hard-code the keys for which you have created a map that the render:lookup tag can refer to in order to look up asset information for use in the element logic.

23.7.4 Deleting Template, CSElement, and SiteEntry Assets

WebCenter Sites does not allow you to delete an asset if there is another asset using it. However, it does not check to see whether a template or CSElement is referenced by the code in other template or CSElement elements.

Before you delete a template or SiteEntry asset, be sure to remove any page calls to that asset's page entry from your elements. Before you delete a CSElement asset, be sure to remove any element calls to that asset's root element from your other elements.

When you delete an asset, WebCenter Sites does the following:

- Changes the value of the asset's name column in the Template, CSElement, or SiteEntry table (depending on the asset type) to its object ID.
- Changes the value of the asset's `status` column in the Template table to VO, for void.
- For templates, deletes all the SiteCatalog table entries (if the template is shared, there are as many page entries as there are sites that the template is shared with) and the ElementCatalog table entry for the template.
- For CSElements, deletes the `ElementCatalog` table entry for the asset.

For the basic procedure on deleting assets, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

23.7.5 Previewing Template, CSElement, and SiteEntry Assets

Because template, CSElement, and SiteEntry assets provide logic and code for formatting other assets, you preview assets of these types differently from the way you preview your content assets.

23.7.5.1 Templates and Preview

You preview a template by previewing an asset and selecting the template that you want to use to render the asset. WebCenter Sites invokes the code in the template and renders a page with the asset as the content.

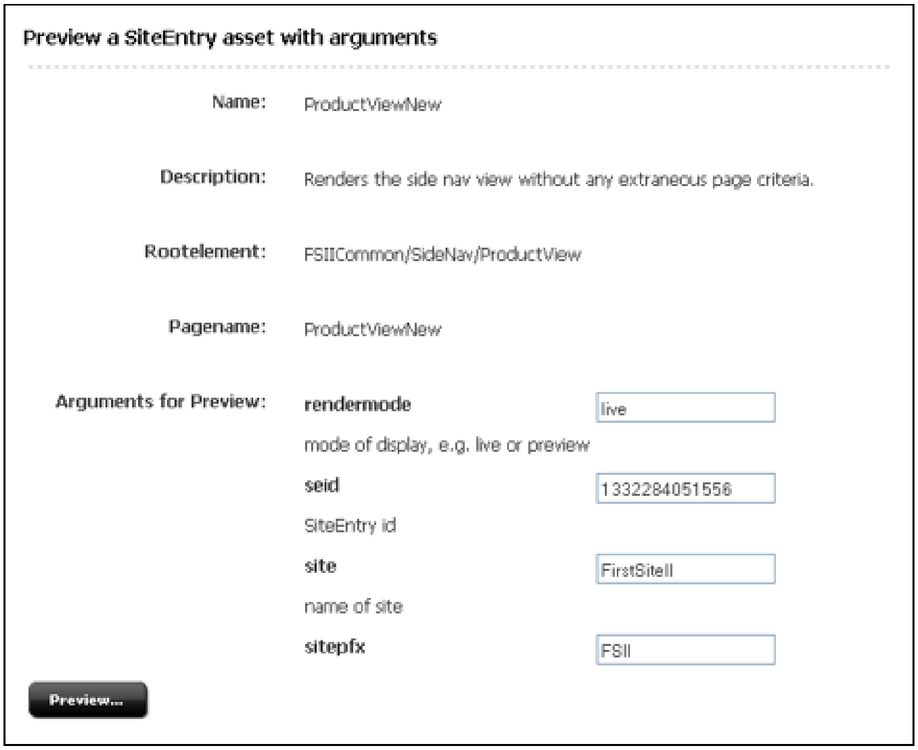
23.7.5.2 CSElement and SiteEntry Assets and Preview

You preview CSElement and SiteEntry assets directly. If the element that will be called has self-contained context, a banner that does not expect variables or arguments. For example, you can simply click the **Preview** icon. But when the results of the rendered element depend on values that are passed to it, you must manually set those values in the CSElement or SiteEntry form in order to preview that asset.

For example, the Burlington Financial CSElement asset named `BurlingtonFinancial/Query/ShowHotTopics` expects a value for the p variable. If it doesn't receive one, the value of p defaults to the object ID of the Home page asset. If you want to preview this CSElement for a page asset other than the Home page, you must pass in the ID of that page asset as the value of the p variable with the argument fields in the New or Edit form for that CSElement asset.

To specify argument values for previewing CSElement or SiteEntry assets

1. Find the asset and inspect it (click the icon with the letter i).
2. Scroll to the bottom of the **Inspect** form. Next to **Preview with Arguments**, click **Preview**. The following form appears for a CSElement asset:



Preview a SiteEntry asset with arguments

Name: ProductViewNew

Description: Renders the side nav view without any extraneous page criteria.

Rootelement: FSIICommon/SideNav/ProductView

Pagename: ProductViewNew

Arguments for Preview:

rendermode mode of display, e.g. live or preview	live
seid SiteEntry id	1332284051556
site name of site	FirstSiteII
sitepfx	FSII

3. Enter values for the arguments. You can also select values by double-clicking in the fields and selecting from the drop-down list.
4. Click **Preview**.
5. Click the links that are displayed to preview the pages that are rendered by this SiteEntry asset.

23.8 Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic

> **Note:** It is not recommended that you create or edit element logic directly from Sites Explorer. However, should you prefer to do so, you will need to ensure the validity of your Template and CSElement assets. Take note of the information in this section and follow the instructions that are included.

When a Template (CSElement) asset is created and saved in the WebCenter Sites interface (**Template** or **CSElement** form), several important steps are taken that are not taken when you use Sites Explorer:

- The interface seeds your element with stub code that sets compositional dependencies and, if you are using JSP, drops in the appropriate tag library directives for you. Compositional dependencies are described in the section Section 28.1, "About Dependencies."
- When you save the Template (CSElement) in the WebCenter Sites interface:
  - The approval system receives information that the asset was changed and can therefore change its approval status.
  - Most importantly, the CacheManager servlet can update the cache (that is, flush pages and pagelets from the WebCenter Sites and Satellite Server caches).

If you choose to work with the CSElement asset in Sites Explorer, be sure that you do not alter the value of the eid variable or accidentally delete it.

A practical reason for using the WebCenter Sites interface is to avoid switching between WebCenter Sites and Sites Explorer, especially if you are mapping asset information (to support template sharing and site replication). Mapping is supported only in the WebCenter Sites interface (in the **Map** screen of the **Template** form and in the **Map** screen of the **CSElement** form).

This section contains the following topics:

- Section 23.8.1, "Creating Templates and CSElements"
- Section 23.8.2, "Editing Templates and CSElements"

23.8.1 Creating Templates and CSElements

If you prefer to use Sites Explorer to code your element logic, follow the steps below:

1. Start creating your Template (or CSElement) asset using the **Template** form (or **CSElement** form). Start with Section 23.4.2.1, "Step 1: Open the 'Template' Form" (or Section 23.5.2.1, "Step 1: Open the 'CSElement' Form") and continue sequentially.
2. In Section 23.4.2.3, "Step 3: Configure the Template's Element" (or Section 23.5.2.3, "Step 3: Configure the Element"), select your element type (JSP, XML, or HTML). Do not change the element logic that is auto-generated for you. Also, be sure that you do not alter the value of the tid variable or accidentally delete it.
3. Continue through the form you have chosen until you finish. If you know which keys and asset values you must map, add them to the **Map** form (in

Section 23.4.2.5, "Step 5: Configure the Map"). The same step applies to the CSElement asset).

4. Save the asset.
5. Open Sites Explorer and edit your element. Save your changes.
6. The final step is to re-save your asset in the **Template** form (or CSElement form). You do not have to change any data in the form, but you must re-save it. This will ensure that no functionality is bypassed.

23.8.2 Editing Templates and CSElements

Any time that you edit an element's logic in the Sites Explorer tool, open and save the template (or CSElement) in the WebCenter Sites interface so that (1) the approval system knows the asset was changed and can change its approval status, and (2) the CacheManager servlet can update the cache.

24

Creating Templates and Wrappers

This chapter presents information on how templates and wrappers are typically used when implementing websites.

This chapter contains the following sections:

- Section 24.1, "Working with Templates"
- Section 24.2, "Working with Wrappers"

24.1 Working with Templates

In this section, we examine the difference between the three types of templates and their practical use in actual implementations.

- Section 24.1.1, "Layout Templates"
- Section 24.1.2, "Pagelet Templates"
- Section 24.1.3, "Page Templates"

24.1.1 Layout Templates

A layout template is a template asset where the **Usage** field is set to **Element is used as a layout**. A layout template can be typed or typeless (see Section 23.4, "Creating Template Assets" for details on typed versus typeless asset).

A layout template has three main characteristics:

- A layout template can be invoked from a browser
- A layout template can be assigned to an asset
- A layout template typically renders an entire web page

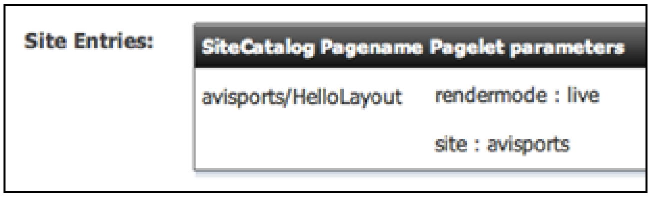
This section provides details for each of these characteristics and presents use-case scenarios that we will use across multiple sections to demonstrate various concepts. This section also presents Use Case 1: Building a Layout Template for Article Assets.

24.1.1.1 A layout template can be invoked from a browser

Given a layout template's pagename, a web page is obtained by calling the Satellite servlet and passing the pagename along with the asset type (c) and asset id (cid) of the content to render:

```
http://localhost:8080/cs/Satellite?pagename=
  <template_pagename>&c=Article&cid=1234567
```

The pagename corresponding to any given template can be found by inspecting a template asset and looking at the **SiteCatalog Pagename** column in the **Site Entries** field. In the screen capture below, the pagename is avisports/HelloLayout.

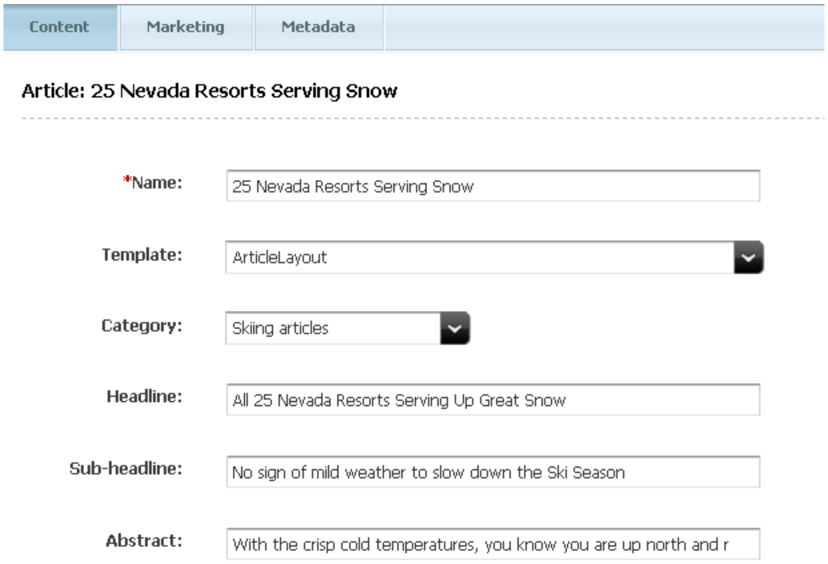


Site Entries:

SiteCatalog Pagename	Pagelet parameters
avisports/HelloLayout	rendermode : live site : avisports

24.1.1.2 A layout template can be assigned to an asset

Every content asset has template metadata which is viewable by selecting the **Content** tab. This field stores a template name, the possible values include all layout templates applicable to the asset type and subtype.

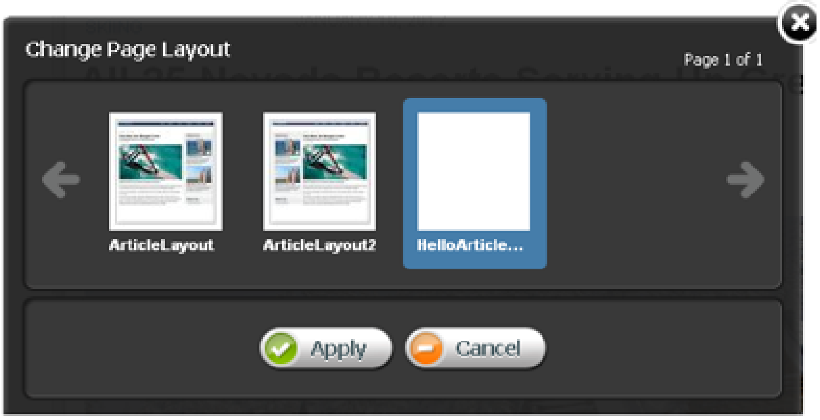


Article: 25 Nevada Resorts Serving Snow

*Name:	25 Nevada Resorts Serving Snow
Template:	ArticleLayout
Category:	Skiing articles
Headline:	All 25 Nevada Resorts Serving Up Great Snow
Sub-headline:	No sign of mild weather to slow down the Ski Season
Abstract:	With the crisp cold temperatures, you know you are up north and r

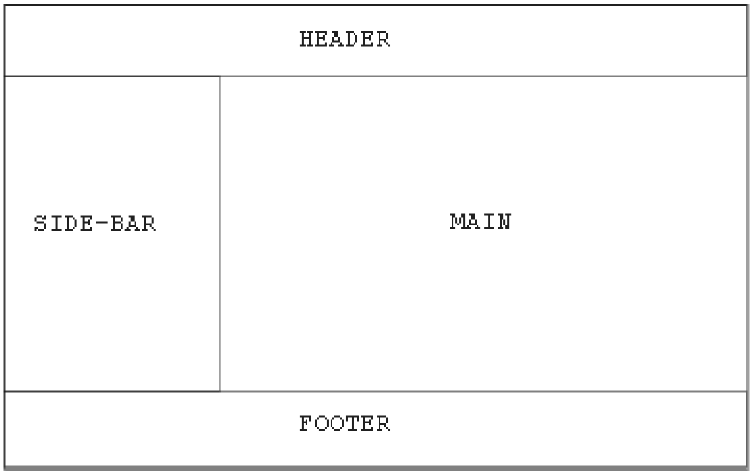
How does WebCenter Sites use the template field? Basically, when inspecting or editing an asset in Web Mode of the Contributor interface, WebCenter Sites uses the assigned layout template as the default template to render the asset (this is also the case when simply previewing an asset). In practice, this means that only layout templates can be used to work with assets in Web Mode. Note that previewing does not require a layout template. See Section 23.7.5, "Previewing Template, CSElement, and SiteEntry Assets" for more information.

When working in Web Mode, the default layout template can also be assigned by using the Change Layout functionality (either from the toolbar or menu bar), and selecting a layout template visually, using the template picker. In either case, the value of the asset's template field is modified.



24.1.1.3 A layout template typically renders an entire web page

A standard web page is built with HTML code using <DIV> elements to define divisions within the document, typically a header, footer, side bar, and main area as shown in the diagram below.



The HTML structure used by the avisports sample site is used in the example below of a standard web page with a header, footer, side bar, and main area.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body class="inner">
  <div id="main">
    <div id="header">
      <!--contains the top menu bar -->
    </div>
    <div id="container">
      <div class="content">
        <!--contains the main area -->
      </div>
      <div class="side-bar">
        <!--contains the side nav bar -->
      </div>
    </div>
    <div id="footer">
```

```
      <!--contains the footer -->
    </div>
  </div>
</body>
</html>
```

An actual JSP page would include tag library directives, an opening and closing `<cs:ftcs>` tag, and the `<render:logdep>` tag, used by the cache manager. See Chapter 4, "Programming with Oracle WebCenter Sites" for more information.

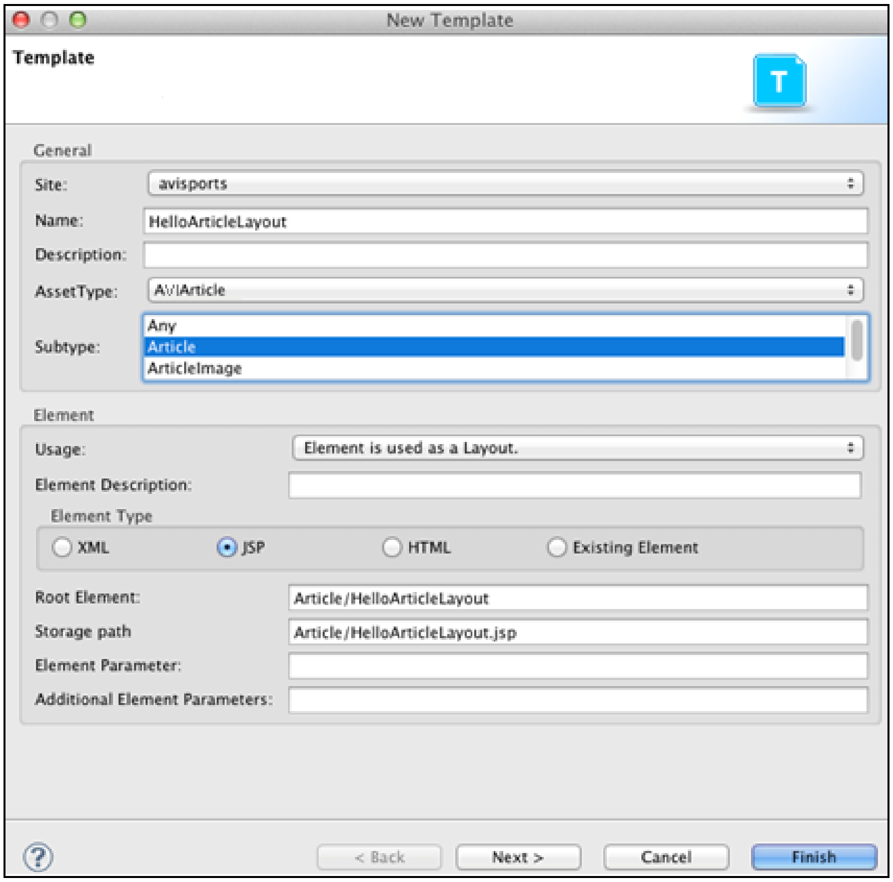
```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld" %>

<cs:ftcs>
<!DOCTYPE html>

<%
// The render:logdep tag is mandatory. It allows the cache manager
// to automatically flush any page/pagelet generated using this
// template ("tid" is automatically populated with the Template
// asset id)
%>

<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template" />

<html>
<head>
</head>
<body class="inner">
  <div id="main">
    <div id="header">
      <!--contains the top menu bar -->
    </div>
    <div id="container">
      <div class="content">
        <!--contains the main area -->
      </div>
      <div class="side-bar">
        <!--contains the side nav bar -->
      </div>
    </div>
    <div id="footer">
      <!--contains the footer -->
    </div>
  </div>
</body>
</html>
</cs:ftcs>
```

The document must be enclosed within `<cs:ftcs>` tags. This tag creates the WebCenter Sites context, alerting WebCenter Sites that code contained within the opening and closing `<cs:ftcs>` tags will contain WebCenter Sites tags. WebCenter Sites is unaware of any code which falls outside of these tags. See Section 4.5, "WebCenter Sites Tags" for more information on the `<cs:ftcs>` and `<render:logdep>` tags.

24.1.1.4 Use Case 1: Building a Layout Template for Article Assets

In this section we define a layout template for article assets.

1. Using Eclipse with the WebCenter Sites Developer Tools plug-in, create a new template in the avisports sample site with these characteristics:
   - **Site**: avisports
   - **Name**: HelloArticleLayout
   - **Asset Type**: AVIArticle
   - **Subtype**: Article
   - **Element Usage**: Element is used as a Layout.
   - **Element Type**: JSP
   - **Root Element**: AVIArticle/HelloArticleLayout
   - **Storage Path**: AVIArticle/HelloArticleLayout.jsp



General

Site: avisports

Name: HelloArticleLayout

Description:

AssetType: AVIArticle

Subtype: Any / **Article** / ArticleImage

Element

Usage: Element is used as a Layout.

Element Description:

Element Type: ○ XML ● JSP ○ HTML ○ Existing Element

Root Element: Article/HelloArticleLayout

Storage path Article/HelloArticleLayout.jsp

Element Parameter:

Additional Element Parameters:

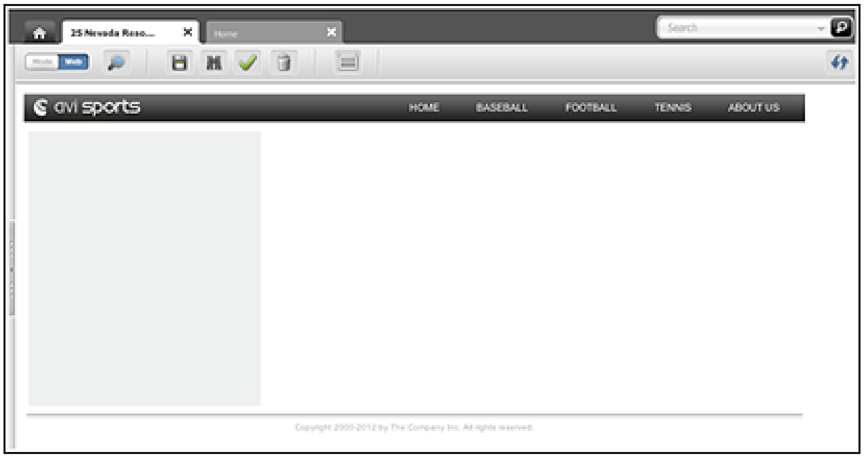
2. Use the following JSP code.

   Note that we are reusing some of the components already written for avisports in order to render the standard avisports header and footer, and head section, which allows us to import the avisports stylesheets:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld" %>

<cs:ftcs>
<!DOCTYPE html>

<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template" />
```

```
<html>
<head>
<!--inserts the standard avisports head -->
<render:calltemplate
  tname="/Head"
  args="c,cid"
  style="element" />
</head>
<body class="inner">
  <div id="main">
    <div id="header">
      <!--inserts the avisports navbar -->
      <render:satellitepage
        pagename="avisports/navbar" />
    </div>
    <div id="container" style="height: 350px">
      <div class="content">
        <!--contains the main area -->
      </div>
      <div class="side-bar" style="height: 300px">
        <!--contains the side nav bar -->
      </div>
    </div>
    <div id="footer">
      <!--inserts the avisports footer -->
      <render:callelement
        elementname="avisports/footer" />
    </div>
  </div>
</body>
</html>
</cs:ftcs>
```

1. Open any article asset in a new tab.
2. Assign the HelloArticleLayout template to the asset. You should get a web page like the following:



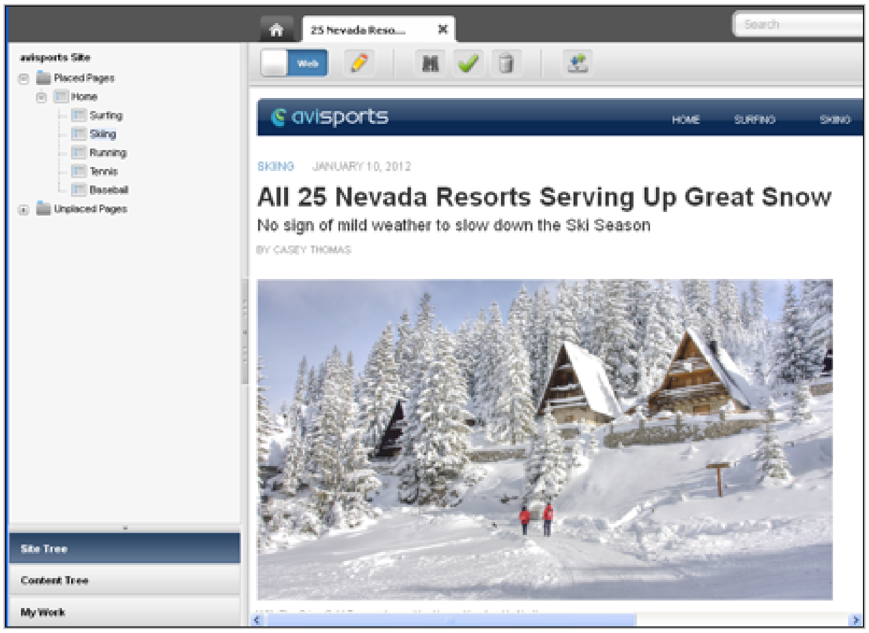
Note that we added some temporary style to the container and side-bar <DIV> elements so they are visible. We will remove them when those contain actual content.

1. Add code to the layout template so it renders actual content. This code renders the headline, post date, related image and body fields. See Chapter 11, "Data Design: The Asset Models" for information on attribute values.

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="dateformat" uri="futuretense_cs/dateformat.tld"%>

<cs:ftcs>
<!DOCTYPE html>

<render:logdep
  cid='<%=ics.GetVar("tid")%>'
  c="Template"/>

<%
// load article content
%>
<assetset:setasset
  name="article"
  type='<%=ics.GetVar("c") %>'
  id='<%=ics.GetVar("cid") %>' />

<%
// fetch the headline, relatedImage, and postDate attributes
// from the database
%>
<assetset:getmultiplevalues
  name="article"
  prefix="article">
    <assetset:sortlistentry
      attributename="headline"
      attributetypename="ContentAttribute" />
    <assetset:sortlistentry
      attributename="relatedImage"
      attributetypename="ContentAttribute" />
    <assetset:sortlistentry
      attributename="postDate"
      attributetypename="ContentAttribute" />
</assetset:getmultiplevalues>

<%
// fetch the body attribute
// body has to be fetched separately, since it is a 'text'
// attribute, and the getmultiplevalues tag does not support
// 'text' attributes
%>
<assetset:getattributevalues
  name="article"
  listvarname="bodyList"
  attribute="body"
  typename="ContentAttribute" />

<%
// read the related AVIImage asset id
%>
<ics:listget
  listname="article:relatedImage"
  fieldname="value"
  output="imageId" />

<%
```

```
// read the date value and format it
%>
<ics:listget
  listname="article:postDate"
  fieldname="value"
  output="postDate" />

<dateformat:create
  name="df"
  datestyle="long" />

<dateformat:getdate
  name="df"
  varname="formattedDate"
  valuetype="jdbcdate"
  value='<%=ics.GetVar("postDate") %>' />

<html>
<head>
<!--inserts the standard avisports head -->

<render:calltemplate
  tname="/Head"
  args="c,cid"
  style="element" />
</head>

<body class="inner">
  <div id="main">
    <div id="header">
      <render:satellitepage
        pagename="avisports/navbar" />
    </div>
    <div id="container">
      <div class="content">
        <div class="top-section section-title">
          <h1>
            <ics:listget
              listname="article:headline"
              fieldname="value" />
          </h1>
          <span class="date">
            <ics:getvar name="formattedDate" />
          </span>
        </div>
        <div class="article post">
          <render:getbloburl
            outstr="imageURL"
            c="AVIImage"
            cid='<%=ics.GetVar("imageId")%>'
            field="largeThumbnail" />
          <img class="photo left"
            src='<%=ics.GetVar("imageURL")%>' />
          <render:stream list="bodyList" column="value" />
        </div>
      </div>
      <div class="side-bar" style="height: 300px">
        <!--contains the side nav bar -->
      </div>
    </div>
  <div id="footer">
   <render:callelement elementname="avisports/footer" />
  </div>
 </div>
</body>
</html>
</cs:ftcs>
```

2. Viewing our asset in Web Mode of the Contributor interface, using HelloArticleLayout, should render a web page with the article detail as shown below. Note that the side bar is intentionally empty.



24.1.2 Pagelet Templates

A pagelet template is a template asset for which the Usage field is set to Element is used within an HTML page.

A pagelet template has the following characteristics:

- A pagelet template cannot be invoked directly from a browser
- A pagelet template cannot be assigned to an asset
- A pagelet template renders a page fragment

In the following section, we detail each characteristic and continue enhancing our use-case example. This section also presents Use Case 2: Using Pagelet Templates.

24.1.2.1 A pagelet template cannot be invoked directly from a browser

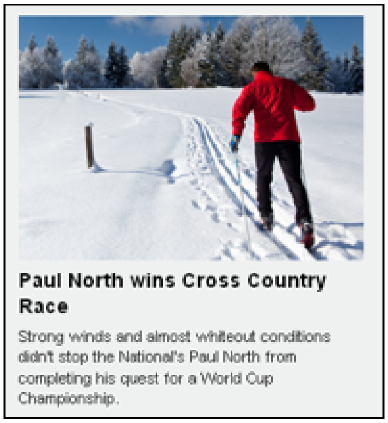
Although a pagelet template has a pagename, attempting to access a pagelet template directly from a browser using a WCS URL will return a 403 HTTP error code (forbidden).

24.1.2.2 A pagelet template cannot be assigned to an asset

Only layout templates are available in the asset's template field. This means that, these assets cannot be directly previewed using a pagelet template. However, it is possible to set up preview templates, that would give editorial users a simple way to preview pagelet templates (see Section 24.1.3, "Page Templates" for more information).

24.1.2.3 A pagelet template renders a page fragment

A page template renders a web page fragment, not an entire web page. Ideally, a pagelet template represents a reusable page fragment. For instance, a pagelet template could render an article summary block such as the following:



Paul North wins Cross Country Race

Strong winds and almost whiteout conditions didn't stop the National's Paul North from completing his quest for a World Cup Championship.

Layout templates can then be used to assemble multiple page fragments in order to produce a complete web page. In order to maximize reusability, pagelet templates should provide neutral fragments from a look and feel point of view, with CSS stylesheet rules effectively controlling the visual result (based on where a given fragment is used, it would render differently, only by applying a distinct set of stylesheet rules).

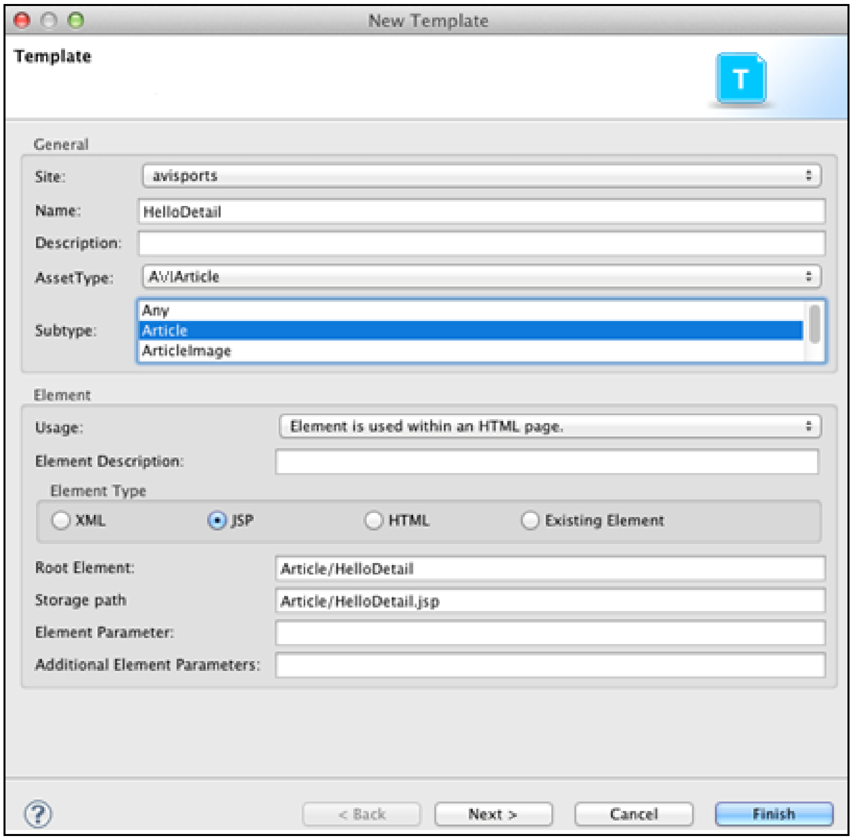
24.1.2.4 Use Case 2: Using Pagelet Templates

See Section 24.1.1.4, "Use Case 1: Building a Layout Template for Article Assets" for previous steps.

If the code rendering the article detail is meant to be reused in multiple context, it makes sense to extract the code from the layout template, and turn it into a pagelet template.

Let's create the corresponding template asset with the following characteristics:

1. Using Eclipse with the WebCenter Sites Developer Tools plug-in, create the corresponding template in the avisports sample site with these characteristics:
   - **Site**: avisports
   - **Name**: HelloDetail
   - **Asset Type**: AVIArticle
   - **Subtype**: Article
   - **Element Usage**: Element is used within an HTML page.
   - **Element Type**: JSP
   - **Root Element**: AVIArticle/HelloDetail
   - **Storage Path**: AVIArticle/HelloDetail.jsp



New Template

Template

General

Site: avisports

Name: HelloDetail

Description:

AssetType: AVIArticle

Subtype: Any, Article (selected), ArticleImage

Element

Usage: Element is used within an HTML page.

Element Description:

Element Type: ○ XML ● JSP ○ HTML ○ Existing Element

Root Element: Article/HelloDetail

Storage path: Article/HelloDetail.jsp

Element Parameter:

Additional Element Parameters:

2. Let's extract the code responsible for looking up and rendering the article field values inside the `<div class="content">` element and turn it into a separate JSP:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="dateformat" uri="futuretense_cs/dateformat.tld"%>

<cs:ftcs>

<render:logdep
  cid='<%=ics.GetVar("tid")%>'
  c="Template" />

<assetset:setasset
  name="article"
  type='<%=ics.GetVar("c") %>'
  id='<%=ics.GetVar("cid") %>' />

<assetset:getmultiplevalues
  name="article" prefix="article">
    <assetset:sortlistentry
      attributename="headline"
      attributetypename="ContentAttribute" />
    <assetset:sortlistentry
      attributename="relatedImage"
      attributetypename="ContentAttribute" />
    <assetset:sortlistentry
      attributename="postDate"
      attributetypename="ContentAttribute" />
```

```
</assetset:getmultiplevalues>

<assetset:getattributevalues
  name="article"
  listvarname="bodyList"
  attribute="body"
  typename="ContentAttribute" />
<ics:listget
  listname="article:relatedImage"
  fieldname="value"
  output="imageId" />
<ics:listget
  listname="article:postDate"
  fieldname="value"
  output="postDate" />
<dateformat:create
  name="df"
  datestyle="long" />
<dateformat:getdate
  name="df"
  varname="formattedDate"
  valuetype="jdbcdate"
  value='<%=ics.GetVar("postDate") %>' />

    <div class="top-section section-title">
      <h1>
        <ics:listget
          listname="article:headline"
          fieldname="value" />
      </h1>
      <span class="date">
        <ics:getvar name="formattedDate" />
      </span>
    </div>

    <div class="article post">
      <render:getbloburl
        outstr="imageURL"
        c="AVIImage"
        cid='<%=ics.GetVar("imageId") %>'
        field="largeThumbnail" />
      <img class="photo left"
        src='<ics:getvar name="imageURL" />' />
      <render:stream list="bodyList" column="value" />
    </div>
</cs:ftcs>
```

3. Our layout template can then be modified by simply relying on HelloDetail, invoked using the `<render:calltemplate>` tag:

```
<html>
<head>
  <render:calltemplate
    tname="/Head"
    args="c,cid"
    style="element" />
  </head>
<body class="inner">
  <div id="main">
    <div id="header">
      <render:satellitepage
```

```
            pagename="avisports/navbar" />
        </div>
        <div id="container">
          <div class="content">
            <render:calltemplate
              tname="HelloDetail"
              args="c,cid" />
          </div>
          <div class="side-bar" style="height: 300px">
            <!-contains the side nav bar -->
          </div>
        </div>
        <div id="footer">
          <render:callelement
            elementname="avisports/footer" />
        </div>
      </div>
    </body>
    </html>
```

24.1.3 Page Templates

Page templates are Template assets for which the **Usage** field is set to **Element defines a whole HTML page and can be called externally**.

A page template has the following characteristics:

- A page template can be invoked from a browser
- A page template cannot be assigned to an asset
- A page template can be used for previewing

In this section we discuss each of these characteristics and then demonstrate a practical application of page templates.

A page template can be invoked from a browser

Like layout templates, a page template can be used to render a web page in a browser by invoking its pagename through the Satellite servlet.

A page template cannot be assigned to an asset

Only layout templates are assignable. Practically speaking, this means that editorial users cannot work in Web Mode of the Contributor interface with a page template, they have to use a layout template.

A page template can be used for previewing

Previewing an asset is different from using Web Mode in the Contributor interface, in the sense that it only allows editorial users to view an asset, there are no editing capabilities involved. When previewing an asset, WebCenter Sites will use, by default, the layout template set in the asset's template field.

When bringing up the Change Preview Template dialog, the template picker shows the following as valid options:

- All layout templates applicable to the current asset
- All page templates applicable to the current asset

Unlike the Change Layout template picker, selecting a different preview template does not assign this template to the asset's template field. It simply renders the current asset with the selected preview template.

24.2 Working with Wrappers

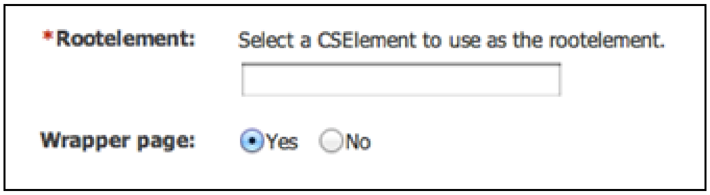
A wrapper is usually not rendering any markup. Instead, a wrapper is usually uncached, and contains business logic meant to be executed before rendering the actual layout template. This is useful when performing such actions as accessing some session data in order to implement security checks, determining which locale should be set, disassembling a friendly URL, etc.

This section presents the following information on wrappers:

- Creating a Wrapper Page
- Wrappers and Previewing

Creating a Wrapper Page

A wrapper is a normal SiteEntry asset, for which the **Wrapper page** flag is set to **Yes**. This specifies that the asset you are creating is a wrapper page. Selecting the **No** flag displays the Pagelet only field. See Section 23.6, "Creating SiteEntry Assets" for more information.



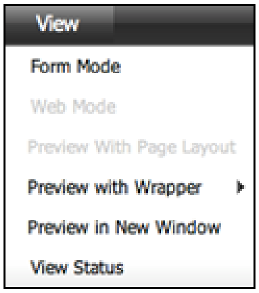
*Rootelement: Select a CSElement to use as the rootelement.

Wrapper page: ● Yes ○ No

Wrappers and Previewing

Depending on the implementation, it might be necessary to execute a wrapper before rendering an asset with a layout template. When previewing an asset, or when working in Web Mode of the Contributor interface, WebCenter Sites will systematically run a wrapper if there is at least one wrapper enabled on the current site.

If a default preview wrapper has been configured for the current editorial site, WebCenter Sites will use this wrapper. Otherwise, it will use the first wrapper available in the list. If several wrappers are available, a different wrapper can be modified by selecting **View** and then **Preview with Wrapper**.



In preview mode, wrappers are especially useful in situations where a layout template requires extra arguments to properly render a web page (for example, a locale argument might be expected) since by default, WebCenter Sites generate a minimal preview URL mainly setting the pagename, with the asset type (c) and asset id (cid) parameters with, respectively, the template pagename, and the type and identifier of the asset to render.

In this case, a preview wrapper can be defined, setting extra arguments as required, and then proceeding by rendering the layout template. Note that the pagename to call is made available in the childpagename variable.

```
<cs:ftcs>

<%
// establish appropriate values for required template arguments
%>
<ics:setvar name="foo" value="bar" />

<render:satellitepage
  pagename='<%=ics.GetVar("childpagename")%>'
  args="c,cid,foo" />

</cs:ftcs>
...
```

25 Coding Templates for In-Context and Presentation Editing

This chapter presents information on coding templates for in-context editing and presentation editing. For information on new tags used in this chapter, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

This chapter contains the following sections:

- Section 25.1, "Coding Templates for In-Context Content Editing"
- Section 25.2, "Coding Templates for Presentation Editing"
- Section 25.3, "Enabling Content Creation for Web Mode"

25.1 Coding Templates for In-Context Content Editing

Note: Any mention of in-context refers to the Web Mode of the WebCenter Sites Contributor interface.

In this section, we examine how templates need to be instrumented to allow editorial users to edit content or create content in the context of their website, instead of using the standard content forms.

This section modifies the HelloDetail template introduced in Chapter 24, "Creating Templates and Wrappers." See these previous topics that build on the HelloDetail template:

- Section 24.1.1.4, "Use Case 1: Building a Layout Template for Article Assets"
- Section 24.1.2.4, "Use Case 2: Using Pagelet Templates"

Note: **Doctype and Internet Explorer**: The in-context editorial UI may not be fully functional if IE renders a page in quirks mode. In order to ensure that local settings cannot affect the UI, it is best to ensure that pages get rendered with an appropriate doctype.

See the DOCTYPE element description in the HTML/XHTML Reference of the Microsoft Library for details: http://msdn.microsoft.com/

This section contains the following topics:

- Section 25.1.1, "Attribute Data Types"

- Section 25.1.2, "String Fields"
- Section 25.1.3, "Text Fields"
- Section 25.1.4, "Date Fields"
- Section 25.1.5, "Binary Fields"
- Section 25.1.6, "Asset Fields"
- Section 25.1.7, "Number Fields"

25.1.1 Attribute Data Types

Asset types are defined by one or more attributes, which can be of the following types:

- **string**: A short string (normally 255 characters max)
- **text**: Aa long string. Typically mapped to a CLOB database type (maximum size depends on the underlying database).
- **date**: A date field
- **binary**: A BLOB attribute. Typically meant to store binary files such as image files, PDFs, etc.
- **asset**: A reference to another asset
- **number**: An integer, float, or money data type

25.1.2 String Fields

In this section, we examine how to make string fields editable. These fields are restricted in length (usually 255 characters) and are best suited to hold content metadata such as article headlines, author, etc.

This section modifies the HelloDetail template introduced in Chapter 24, "Creating Templates and Wrappers" in order to make the article headline field editable.

In the previous examples, we used the `ics:listget` tag to print the value of the headline field (which is stored in the value column of the `article:headline` list).

```
<h1>
<ics:listget
  listname="article:headline"
  fieldname="value" />
</h1>
```

1. To make this field editable in-context, add the following taglib directive:

   ```
   <@ taglib prefix="insite" uri="futuretense_cs/insite.tld" %>
   ```

   and replace the previous code with the code snippet below:

   ```
   <h1>
   <insite:edit
     field="headline"
     list="article:headline" column="value"
     assetid='<%=ics.GetVar("cid")%>'
     assettype='<%=ics.GetVar("c")%>' />
   </h1>
   ```

   where:

   - The field parameter designates which field of the asset is edited.

- The `list` and `column` parameters designate where the current field value is stored.
- The `assetid` and `assettype` parameters designate the edited asset

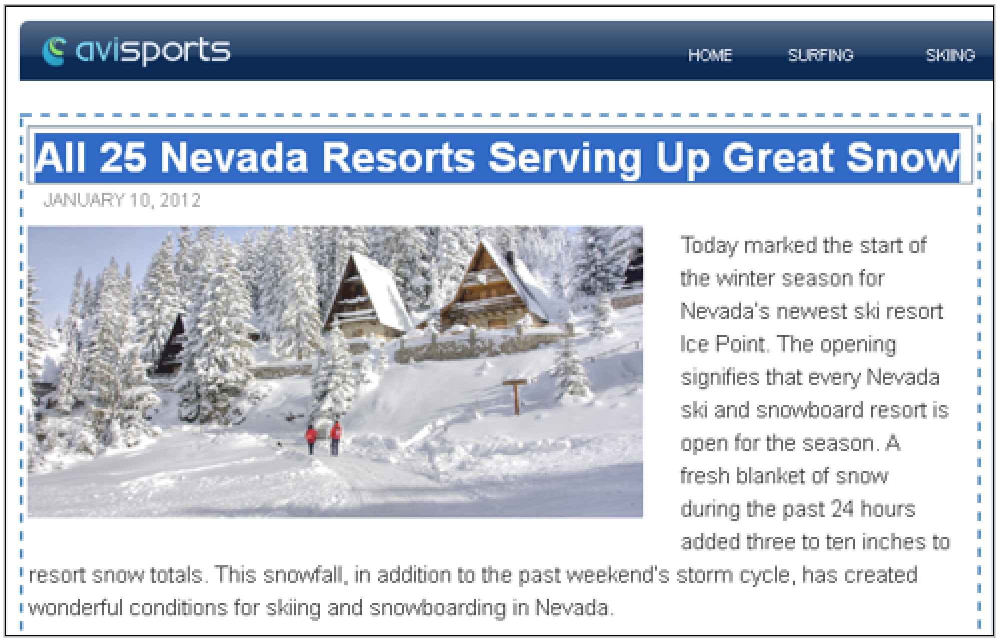
The syntax can be simplified as follows:

```
<h1>
<insite:edit
  field="headline"
  list="article:headline"
  column="value" />
</h1>
```

Because the edited asset is designated by the asset type (c) and asset id (cid) variables, WebCenter Sites performs the retrieval of these values directly from the ICS scope.

Examining any asset in Web Mode of the Contributor interface, using HelloArticleLayout will not show any noticeable differences from the previous display. In fact, when in preview or inspect view, the `insite:edit` tag behaves exactly like the `ics:listget` tag we just replaced.

2. Selecting **Edit View** makes all editable areas active. In our case, the headline field as shown in the following screen capture:

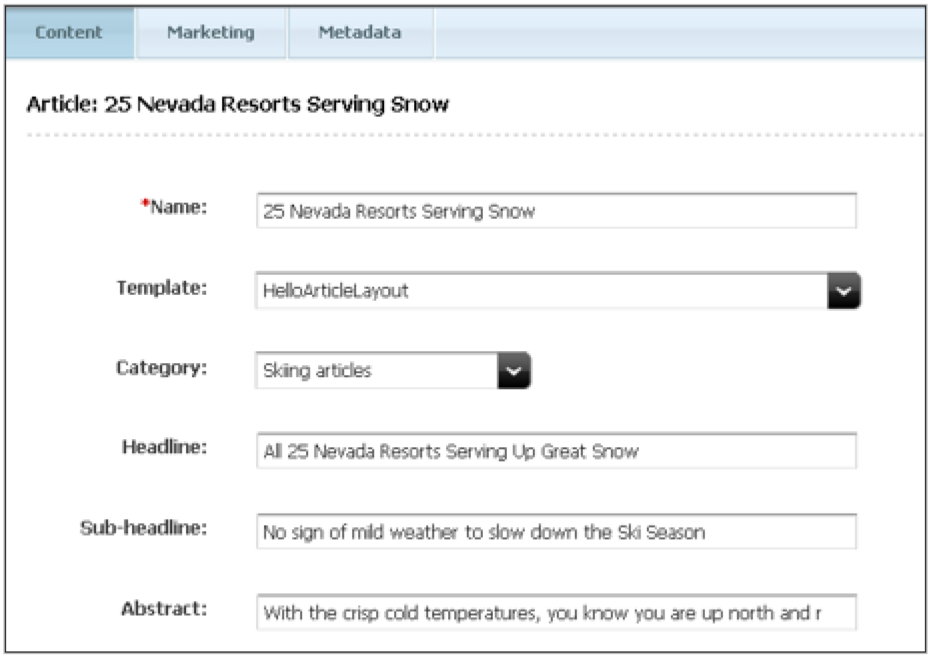


All 25 Nevada Resorts Serving Up Great Snow

JANUARY 10, 2012

Today marked the start of the winter season for Nevada's newest ski resort Ice Point. The opening signifies that every Nevada ski and snowboard resort is open for the season. A fresh blanket of snow during the past 24 hours added three to ten inches to resort snow totals. This snowfall, in addition to the past weekend's storm cycle, has created wonderful conditions for skiing and snowboarding in Nevada.

3. Editorial users are now able to edit the headline field in the context of the article detail web page and click **Save** to make this change permanent. This action is equivalent to going to the article asset form, editing the headline field of the article, and then clicking **Save**:



Article: 25 Nevada Resorts Serving Snow

*Name: 25 Nevada Resorts Serving Snow

Template: HelloArticleLayout

Category: Skiing articles

Headline: All 25 Nevada Resorts Serving Up Great Snow

Sub-headline: No sign of mild weather to slow down the Ski Season

Abstract: With the crisp cold temperatures, you know you are up north and r

25.1.2.1 Variants of the <insite:edit/> Tag

In the previous example, the value of the edited field was stored in a list. This is how the assetset tags work; they query the database and return attribute values in a list object whether attributes are single-valued or multivalued.

In some cases, the value might be made available in a variable. The following variant of insite:edit can then be used:

```
<h1>
<insite:edit
  variable="headlineVar"
  field="headline" />
</h1>
```

In a case where the field value is not available in a variable or a list, it is possible to specify the value directly using the value attribute in the insite:edit tag.

```
<%String headline = getHeadlineValueFromSomewhere(); %>
<insite:edit
  value="<%=headline%>"
  field="headline" />
```

The insite:edit tag also supports the property and ssvariable attributes, in case the field value is available in, respectively, a WebCenter Sites property file or a session variable. These variants are rarely used.

25.1.3 Text Fields

In this section, we examine how to make text fields editable. This section builds on the HelloDetail template introduced in the Chapter 24, "Creating Templates and Wrappers" chapter.

Similar to string fields, we can replace the render:stream tag in the HelloDetail template with the insite:edit tag. See Section 25.1.2, "String Fields" for an example of working with string fields in the HelloDetail template:

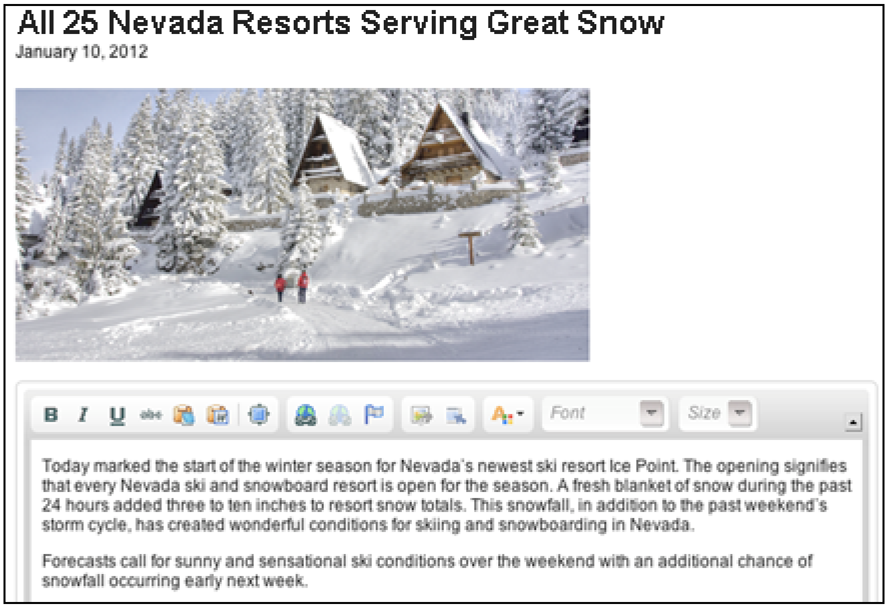
Previously, we used the `render:stream` tag:

```
<render:stream
  list="bodyList"
  column="value" />
```

To make this field editable, change the `render:stream` tag: to an `insite:edit` tag. In this example, we also define body as the field and ckeditor as the editor type.

```
<insite:edit
  list="bodyList"
  column="value"
  field="body"
  editor="ckeditor" />
```

Clicking on the body text displays the CKEditor widget as shown in this screen capture:



All 25 Nevada Resorts Serving Great Snow

January 10, 2012

Today marked the start of the winter season for Nevada's newest ski resort Ice Point. The opening signifies that every Nevada ski and snowboard resort is open for the season. A fresh blanket of snow during the past 24 hours added three to ten inches to resort snow totals. This snowfall, in addition to the past weekend's storm cycle, has created wonderful conditions for skiing and snowboarding in Nevada.

Forecasts call for sunny and sensational ski conditions over the weekend with an additional chance of snowfall occurring early next week.

Note that additional arguments can be passed to CKEditor as follows:

```
<insite:edit
  list="bodyList"
  column="value"
  field="body"
  editor="ckeditor"
  params="{width: '500px', height: '350px',
           toolbar: 'MyToolbar'}" />
```

See Section 19.1, "Configuring CKEditor" for more information.

25.1.4 Date Fields

In this section, we examine how to make date fields editable. This section builds on the HelloDetail template introduced in Chapter 24, "Creating Templates and Wrappers."

Date fields are made editable in-context using the same `insite:edit` tag used for string or text fields. However, when dates need to be formatted, a few extra steps are required. The value of date fields when initially retrieved from the database is in JDBC format, which is, `2012-01-01 00:00:00.0`. This format is unsuitable for rendering on a website where dates are generally rendered as a readable string, such as `January 1, 2012`.

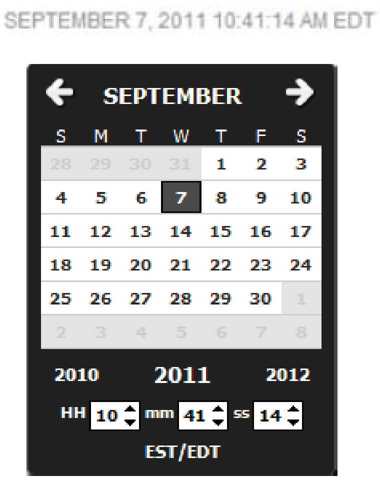
In addition, the date is interpreted in the server's time zone, which we can display in a different time zone. We first get the time zone ID by using the `java.util` TimeZone API. We then format the date by using the "long" date format (which is a predefined format).

Note: For more about predefined date formats see:

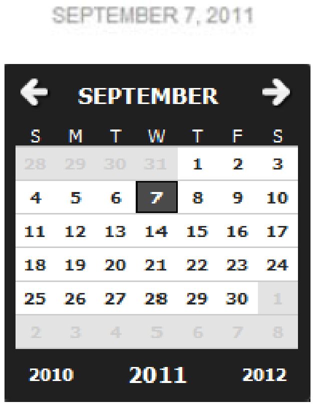
`http://docs.oracle.com/javase/tutorial/i18n/format/dateFormat.html`

If you choose to format the date, you can use one of the date formatting APIs, described in the next section. Depending on your decision, the resulting date and calendar widget will look like one of the following:

With timestamp and time zone ID



Without timestamp and time zone ID



Date Formatting APIs

The HelloDetail template uses the `formattedDate` variable to display the date. See Section 24.1.1.4, "Use Case 1: Building a Layout Template for Article Assets" for details.

```
<span class="date">
  <ics:getvar name="formattedDate" />
</span>
```

The `formattedDate` variable must be set by using one of the date formatting APIs: `fmt:formatDate` or the Sites `dateformat` API.

The `formattedDate` can be rendered with a timestamp if you add the time style parameter. It can also be rendered with a specific time zone if we add a parameter for time zone. If the time zone parameter is omitted, the date is interpreted and rendered in the server's time zone. Following is a description of the APIs:

- The `fmt:formatDate` (which supports EL expression) is one such API, which takes the date in `java.util.Date()` in String format and takes the `timeZone` parameter as shown below:

  ```
  <fmt:formatDate value="${asset.postDate}" dateStyle="long"
    type="both" timeStyle="long" var="formattedDate"
    timeZone="US/Eastern" />
  ```

- The Sites `dateformat` API has an additional parameter named `timezoneid` and may be used in place of `fmt:formatDate`:

  ```
  <dateformat:create name="dateFormat" datestyle="long"
    timestyle="long" timezoneid="US/Eastern" />
  ```

  If timestamp is needed, we use `dateformat:getdatetime`:

  ```
  <dateformat:getdatetime name="dateFormat" value='<%=postDate%>'
    valuetype="jdbc" varname="formattedDate"/>
  ```

  If timestamp is not needed, we use `dateformat:getdate`:

  ```
  <dateformat:getdate name="dateFormat" value='<%=postDate%>'
    valuetype="jdbc" varname="formattedDate"/>
  ```

  The Sites `dateformat` API takes `millis` or `jdbc` in the `valuetype` argument.

Enabling Date Fields for Editing in Web Mode

The formatted date is now ready to be used in the `insite:edit` tag. We follow the steps below to enable date fields for editing in Web Mode.

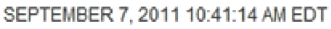
1. Because the `insite:edit` tag passes the formatted value, an appropriate `formatLength` parameter is required in the `params` argument. In our example, `formatLength` is set to set to `long`, as shown below. In addition, if we choose to display the timestamps and time zone in date fields, we would set two additional parameters; `timePicker:true` and `timeZoneID:'US/Eastern'`, also shown below:

   ```
   <span class="date">
     <insite:edit
     field="postDate"
     value="formattedDate"
     params="{constraints:{formatLength: 'long'},
       timePicker:true, timeZoneID:'US/Eastern'}" />
   </span>
   ```

   > **Note:** The date widget in edit mode will function correctly only if we pass a properly formatted date.
   >
   > The `datestyle` and `timestyle` arguments in `dateformat:create` or `fmt:formatdate` APIs must be consistent with the `formatLength` params argument in the `insite:edit` tag. In our example, the datestyle and timestyle arguments in the date formatting API must all be `long`.
   >
   > If the `formattedDate` is constructed using the time zone parameter, then the same time zone ID must be used in the `insite:edit` tag.

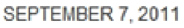
2. The date will now be rendered according to how we specified its format:

   - With timestamp and `timeZoneID:'US/Eastern'`



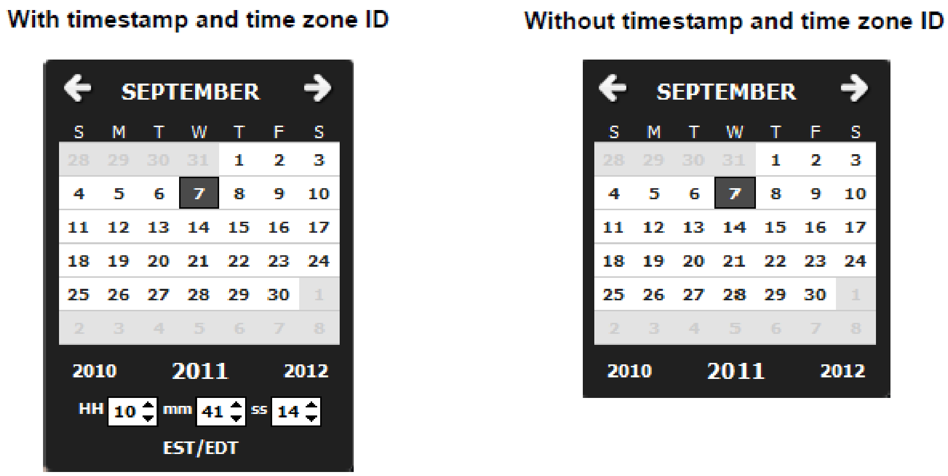
SEPTEMBER 7, 2011 10:41:14 AM EDT

   - Without timestamp and `timeZoneID`



SEPTEMBER 7, 2011

3. The date field is now editable. Clicking on the date field while viewing an article in the Contributor interface in Web Mode / Edit View displays one of the following calendar popup widgets:



Note that the date format is passed to the `insite:edit` tag using the `params` attribute. The `params` attribute is used to pass extra configuration settings to the Dojo widgets, formatted as a JSON string. In this case:

```
{constraints: {formatLength: 'long'}, timePicker : true,
  timeZoneID:'US/Eastern' }
```

For details on available widget settings, refer to the Dojo documentation at `http://dojotoolkit.org/`.

25.1.5 Binary Fields

Binary fields are typically database BLOB or CLOB fields. Binary fields typically store images, or downloadable documents. By default, the `insite:edit` tag will make binary fields editable through a file upload component.

> **Note:** This section also applies to WebCenter Sites URL columns. That is, URL columns storing a reference to a file which is stored on the file system

In our HelloDetail template, we are rendering the **largeThumbnail** field of the related AVIImage asset. In order to make this field editable in-context, we first need to retrieve the value of this field, which contains a BLOB ID (not the actual BLOB value):

```
<assetset:setasset
  name="image"
  type="AVIImage"
  id='<%=ics.GetVar("imageId")%>' />
<assetset:getattributevalues
  name="image"
  attribute="largeThumbnail"
  typename="ContentAttribute"
  listvarname="largeThumbnail" />
```

The <img> tag is then inserted between an opening and closing insite:edit tag.

```
<insite:edit
  field="largeThumbnail"
  assetid='<%=ics.GetVar("imageId")%>'
  assettype="AVIImage"
  list="largeThumbnail"
  column="value" >

    <img class="photo left"
      src='<%=ics.GetVar("imageURL")%>' />
</insite:edit>
```

Note that passing the BLOB ID to the insite:edit tag is primarily used to determine whether the binary field is empty or not. If empty, the insite:edit tag shows a default empty value indicator.

The <img> tag should be the only HTML tad inside the insite:edit tag. The <img> tag is the only tag permitted inside the inside:edit tag. The insite:edit tag can also handle <a> tags (normally used when building hyperlinks to downloadable documents). Other types of binaries may be handled but might require specific customizing.

When hovering over the image, content contributors are now shown a tooltip, giving access to the upload component, and allowed to clear the largeThumbnail field.

Note: The HelloDetail template allows contributors to edit two distinct assets on the same page (the "headline", "date", "body", "relatedImage" fields of the AVIArticle asset, and the **largeThumbnail** field of the related AVIImage asset). Generally, it is possible to render multiple assets on the same page, and make them editable simultaneously.

25.1.6 Asset Fields

Asset fields store references to other assets. In this section, we examine how to make asset fields editable.

This section builds on the HelloArticleLayout template created in Section 24.1.1.4, "Use Case 1: Building a Layout Template for Article Assets" of Chapter 24, "Creating Templates and Wrappers."

In this section we will enhance the HelloArticleLayout template and populate the side bar by rendering the article assets related to our article through the relatedStories field.

Note: Although "relatedStories" is a multivalued field, we treat it, in this section, as a single-valued field. That is, we will consider that the field holds only one related article content asset. Code samples shown in this section are therefore applicable to any single-valued field. See Section 25.1.8, "Multivalued Fields" for more information on handling multivalued fields.

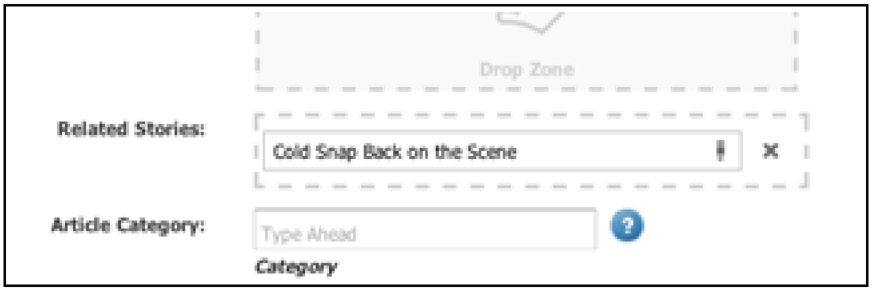
1. Create a pagelet template called HelloSideBar, applicable to the AVIArticle asset type and Article subtype, which renders the associated article asset using the avisports Summary/SideBar article template using the following code:

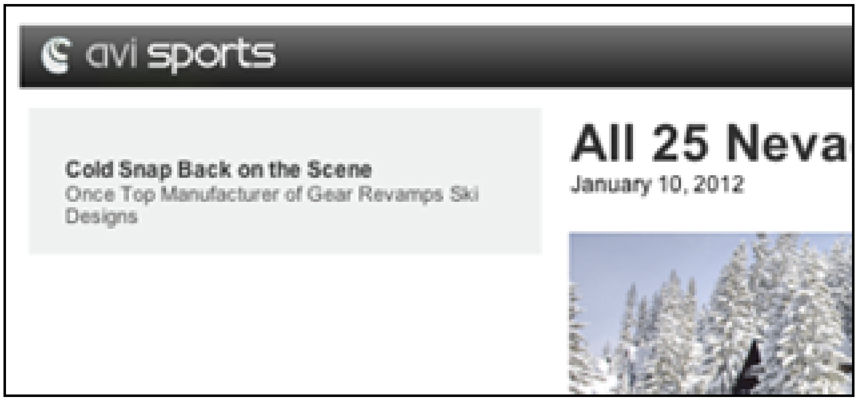
```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>

<cs:ftcs>

<render:logdep
  c="Template"
  cid='<%=ics.GetVar("tid")%>' />

<assetset:setasset
  name="article"
  type="AVIArticle"
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="article"
  attribute="relatedStories"
  listvarname="relatedStories"
  typename="ContentAttribute" />

<%-- we are getting the first item in the list --%>
<ics:listget
  listname="relatedStories"
  fieldname="value"
  output="articleId" />

<render:calltemplate
  tname="Summary/SideBar"
  c="AVIArticle"
  cid='<%=ics.GetVar("articleId")%>' />

</cs:ftcs>
```

2. Modify the HelloArticleLayout template to invoke the HelloSideBar pagelet template inside the "side-bar" div element.

```
<div class="side-bar">
    <render:calltemplate tname="HelloSideBar" args="c,cid" />
</div>
```

Assuming that relatedStories contain one asset reference (Cold Snap Back on the Scene in our example) the related article now gets rendered in the side bar using the Summary/SideBar template, as dictated by the template code.





Cold Snap Back on the Scene

Once Top Manufacturer of Gear Revamps Ski Designs

All 25 Neva

January 10, 2012

Previously, the `render:calltemplate` tag was used in the HelloSideBar pagelet:

```
<render:calltemplate
  tname="Summary/SideBar"
  c="Article"
  cid='<%=ics.GetVar("articleId")>' />
```

3. The relatedStories asset field can be made editable in-context, by turning the area occupied by the asset into a drop target. That is, an area which will accept assets dragged from other parts of the UI (such as the search result pane or the content tree). To do this, add the following taglib directive:

```
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld" %>
```

and replace the `render:calltemplate` tag in the HelloSideBar pagelet with the `insite:calltemplate` tag:

```
<insite:calltemplate
  tname="Summary/SideBar"
  c='AVIArticle'
  cid='<%=ics.GetVar("articleId")%>'
  field="relatedStories"
  assetid='<%=ics.GetVar("cid")%>'
  assettype='<%=ics.GetVar("c")%>' />
...
```

where:

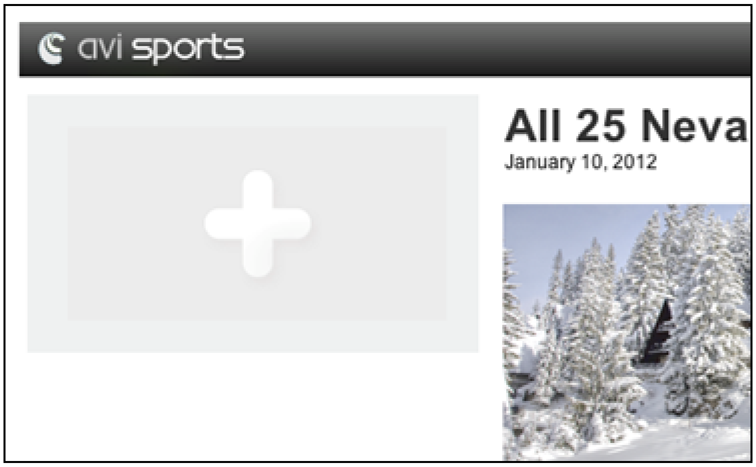
- The `tname` (template), `c` (asset type) and `cid` (asset id) variables behave exactly as in the `render:calltemplate` tag and have the same meaning.
- The `assetid` and `assetype` variables designate the asset being edited (in our example, an article asset).
- The `field` variable designates which field of the asset is being edited (in our example, "relatedStories").

The syntax can be simplified as follows:

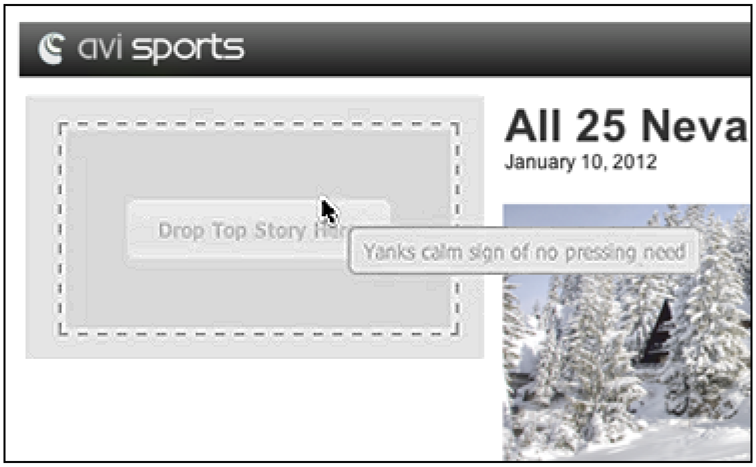
```
<insite:calltemplate
  tname="Summary/SideBar"
  c='AVIArticle'
  cid='<%=ics.GetVar("articleId")%>'
  field="relatedStories" />
```

This simplified structure can be used because the edited asset is the asset designated by the `c` and `cid` variables. Thus, WebCenter Sites retrieves the variable values by looking them up directly in the ICS context.

4. If the field is initially empty (in which case the `articleId` variable is null), viewing the asset in Web Mode of the Contributor interface shows an empty content-editable slot (provides a droppable zone for the user) as shown in this screen capture below:



5. It is now possible to drag and drop an article asset to the content-editable slot, by selecting an asset in the content tree, or from the docked search panel.



25.1.6.1 Editing an Association

When using an asset association instead of a flex attribute of data type `asset`, the `field` attribute needs to be specified as follows:

```
Association-named:<associationName>
```

For example, for an association called "topStory" the code would be:

```
<insite:calltemplate
  field="Association-named:topStory"
  ...
/>
```

25.1.6.2 Editing a Parent Asset

It is also possible to edit a flex asset's parent asset. The syntax for the `field` attribute is as follows:

```
Group_<parentDefinitionName>
```

For example, in the case of avisports, article assets have a "Category" parent definition:

```
<insite:calltemplate
  field="Group_Category"
  ...
/>
```

25.1.7 Number Fields

When dealing with number attributes (i.e. integer, double and money), raw values are retrieved from the database, and are typically formatted according to the current locale.

For example, assuming that `price` is a money attribute, a JSP reading the attribute value and rendering it as a formatted string could be written as follows:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<cs:ftcs>

<assetset:setasset
  name="theAsset"
  type='<%=ics.GetVar("c")%>'
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="theAsset"
  attribute="price"
  listvarname="pricelist"
  typename="ContentAttribute" />

<ics:listget
  listname="pricelist"
  fieldname="value"
  output="price" />

<fmt:formatNumber
  type="currency"
  value='<%=ics.GetVar("price")%>'
  var="formattedValue"
  currencySymbol="&eur;" />

The price is: ${formattedValue}
</cs:ftcs>
```

Assuming that the raw value is 123456, it would be rendered as 123,456 (en_US locale). The value is made editable using the `insite:edit` tag as follows:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<cs:ftcs>

<assetset:setasset
  name="theAsset"
  type='<%=ics.GetVar("c")%>'
  id=''<%=ics.GetVar("cid")%> />
```

```
<assetset:getattributevalues
  name="theAsset"
  attribute="price"
  listvarname="pricelist"
  typename="ContentAttribute" />

<ics:listget
  listname="pricelist"
  fieldname="value"
  output="price" />

<fmt:formatNumber
  type="currency"
  value='<%=ics.GetVar("price")%>'
  var="formattedValue"
  currencySymbol="&eur;" />

The price is: <insite:edit field="price"
                value="${formattedValue}"
                params="{currency: 'EUR'}" />
</cs:ftcs>
```

Note that the editing widget is passed the formatted value (containing the currency symbol). For this reason, the currency ISO code is specified using the `params` field.

For more configuration options, refer to the Dojo documentation at http://dojotoolkit.org/.

25.1.8 Multivalued Fields

All previous examples were intended for single-valued fields. When dealing with multivalued fields, beyond editing the existing values, editors need to access more features such as:

- adding a new value
- removing an existing value
- reordering existing values

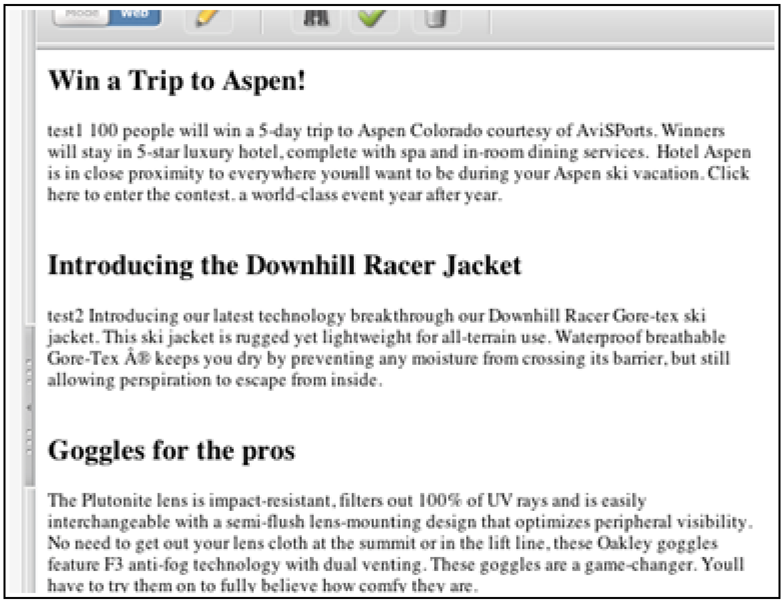
For this reason, multivalued fields need to be treated specifically. This section will discuss working with multivalued text fields and multivalued asset fields.

25.1.8.1 Example 1: Editing Multivalued Text Fields

1. Page assets of the AVIHome subtype have a multivalued text field (see attribute value "teaserText" in the `assetset:getattributevalues` tag below). Create a test layout template, rendering only the value of this field:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>

<cs:ftcs>

<render:logdep
  cid='<%=ics.GetVar("tid")%>'
  c="Template" />

<assetset:setasset
```

```
  name="page"
  type="Page"
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="page"
  attribute="teaserText"
  listvarname="teaserList"
  typename="PageAttribute" />

<div style="width: 500px; font-size: small">
  <ics:listloop listname="teaserList">
    <render:stream list="teaserList" column="value" />
  </ics:listloop>
</div>
</cs:ftcs>
```



Win a Trip to Aspen!

test1 100 people will win a 5-day trip to Aspen Colorado courtesy of AviSPorts. Winners will stay in 5-star luxury hotel, complete with spa and in-room dining services. Hotel Aspen is in close proximity to everywhere youall want to be during your Aspen ski vacation. Click here to enter the contest. a world-class event year after year.

Introducing the Downhill Racer Jacket

test2 Introducing our latest technology breakthrough our Downhill Racer Gore-tex ski jacket. This ski jacket is rugged yet lightweight for all-terrain use. Waterproof breathable Gore-Tex Â® keeps you dry by preventing any moisture from crossing its barrier, but still allowing perspiration to escape from inside.

Goggles for the pros

The Plutonite lens is impact-resistant, filters out 100% of UV rays and is easily interchangeable with a semi-flush lens-mounting design that optimizes peripheral visibility. No need to get out your lens cloth at the summit or in the lift line, these Oakley goggles feature F3 anti-fog technology with dual venting. These goggles are a game-changer. Youll have to try them on to fully believe how comfy they are.

2. Modify this template to make the multivalued `teaserText` field editable.

   Add the `insite` taglib directive:

   ```
   <%@ taglib prefix="insite" uri="futuretense_cs/insite.tld" %>
   ```

   and replace the original code:

   ```
   <ics:listloop listname="teaserList">
     <render:stream list="teaserList" column="value" />
   </ics:listloop>
   ```

   with the following code:

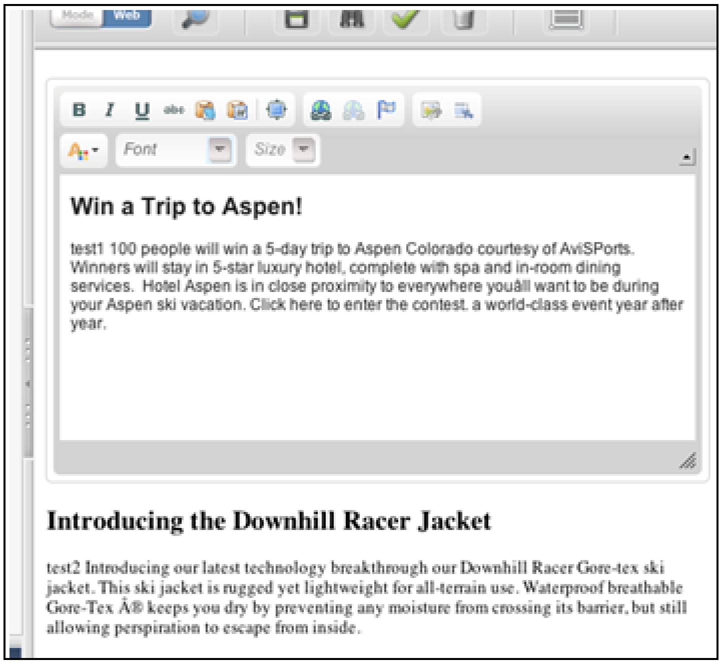
   ```
   <ics:listloop listname="teaserList">
     <ics:listget
       listname="teaserList"
       fieldname="#curRow"
       output="currentRowNb" />
     <insite:edit
       assetid='<%=ics.GetVar("cid")%>'
       assettype='<%=ics.GetVar("c")%>'
       field="teaserText"
   ```

```
    list="teaserList"
    column="value"
    index='<%=ics.GetVar("currentRowNb")%>'
    editor="ckeditor" />
</ics:listloop>
```

The syntax for the `insite:edit` tag can be simplified as follows:

```
<ics:listloop listname="teaserList">
  <ics:listget
    listname="teaserList"
    fieldname="#curRow"
    output="currentRowNb" />
  <insite:edit
    field="teaserText"
    list="teaserList"
    column="value"
    index='<%=ics.GetVar("currentRowNb")%>'
    editor="ckeditor" />
</ics:listloop>
```

3. In the Contributor interface, change to **Web Mode / Edit View**. We are now able to edit each of the existing values using CKEditor as shown in the screen capture below.



Win a Trip to Aspen!

test1 100 people will win a 5-day trip to Aspen Colorado courtesy of AviSPorts. Winners will stay in 5-star luxury hotel, complete with spa and in-room dining services. Hotel Aspen is in close proximity to everywhere youâll want to be during your Aspen ski vacation. Click here to enter the contest. a world-class event year after year.

Introducing the Downhill Racer Jacket

test2 Introducing our latest technology breakthrough our Downhill Racer Gore-tex ski jacket. This ski jacket is rugged yet lightweight for all-terrain use. Waterproof breathable Gore-Tex Â® keeps you dry by preventing any moisture from crossing its barrier, but still allowing perspiration to escape from inside.

The only noticeable difference with a single-valued field, is that we only added an `index` attribute which notifies WebCenter Sites the index of the value being edited.

At this point we are only able to edit existing values. See the next section on how to add, remove, or reorder existing values.

25.1.8.2 Example 2: Multivalued Text Fields

This section builds on Section 25.1.8.1, "Example 1: Editing Multivalued Text Fields" of this chapter and includes adding, removing, and reordering values.

Modify the previous code sample as follows:

```
<insite:list
  field="teaserText"
  editor="ckeditor"
  assetid='<%=ics.GetVar("cid")%>'
  assettype='<%=ics.GetVar("c")%>'>

    <ics:listloop listname="teaserList">
      <insite:edit list="teaserList" column="value" />
    </ics:listloop>
</insite:list>
```

The `insite:edit` tag is now nested inside an `insite:list` tag. The nested `insite:edit` tags are do not specify the `field`, `assetid`, `assettype` or `editor` attributes since they are already specified at the parent tag level, and each nested `insite:edit` tag is, by default, inheriting those values when not locally specified. This is also applicable to the `params` attribute of the `insite:edit` tag.

The `index` attribute is no longer required. In this case, the `insite:list` tag is keeping track of the current index. The tag assumes that the order in which the `insite:edit` tag appear matches the order of the multivalued field, that is, the first `insite:edit` tag edits value #1, etc. If that is not the case, the index has to be specified. See Section 25.1.8.3, "Specifying a Different Ordering" for details.

Like the `insite:edit` tag, the syntax for the `insite:list` tag can be simplified if the asset being edited is the asset designated by the `c` (asset type) and `cid` (asset id) variables; in which case the `assetid` and `assettype` attributes can be omitted.

```
<insite:list field="teaserText" editor="ckeditor" >
  <ics:listloop listname="teaserList">
    <insite:edit list="teaserList" column="value" />
  </ics:listloop>
</insite:list>
```

1. Now that we added the `insite:list` tag, a toolbar is displayed whenever the user hovers over the area showing the field values:

2. When clicking on that area, a popup gets rendered, allowing to add, edit, remove or reorder field values.

This section builds on Section 25.1.8.1, "Example 1: Editing Multivalued Text Fields" and Section 25.1.8.2, "Example 2: Multivalued Text Fields."

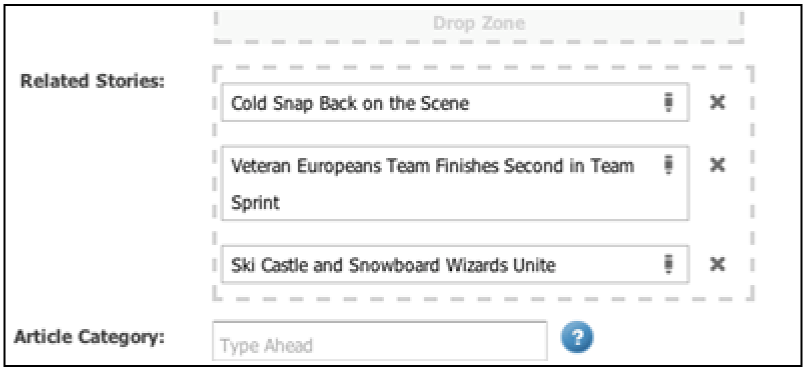
In the case of asset reference fields, we will use the `insite:slotlist` tag instead of the `insite:list` tag. In this case, rather than nested `insite:edit` tags, we will have nested `insite:calltemplate` tags.

In this example, instead of rendering a single related article, we will go through the whole list, and render each of them.

Modify the previous code sample as follows (for the time being, this code omits any in-context editing capabilities:):

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>

<cs:ftcs>

<assetset:setasset
  name="article"
  type="AVIArticle"
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="article"
  attribute="relatedStories"
  listvarname="relatedStories"
  typename="ContentAttribute" />

<ics:listloop listname="relatedStories">
  <ics:listget
   listname="relatedStories"
   fieldname="value"
   output="articleId" />
  <render:calltemplate
    tname="Summary/SideBar"
    c="AVIArticle"
    cid='<%=ics.GetVar("articleId")%>' />
</ics:listloop>
</cs:ftcs>
```

The screen capture below shows the field with three values:



Related Stories:

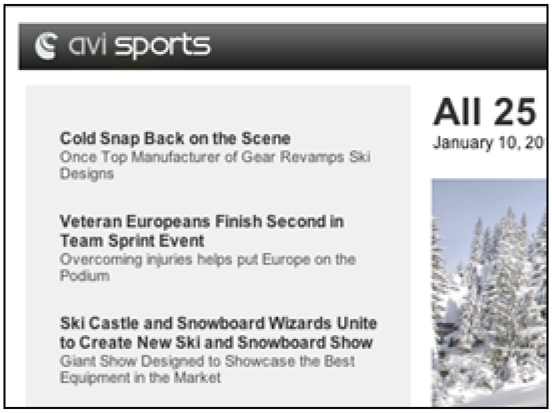
Cold Snap Back on the Scene

Veteran Europeans Team Finishes Second in Team Sprint

Ski Castle and Snowboard Wizards Unite

Article Category: Type Ahead

The SideBar will now show those three articles:



Cold Snap Back on the Scene
Once Top Manufacturer of Gear Revamps Ski Designs

Veteran Europeans Finish Second in Team Sprint Event
Overcoming injuries helps put Europe on the Podium

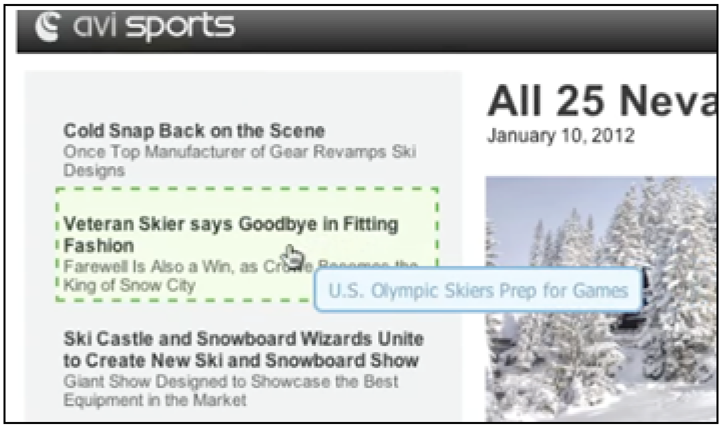
Ski Castle and Snowboard Wizards Unite to Create New Ski and Snowboard Show
Giant Show Designed to Showcase the Best Equipment in the Market

If you want to make the article list editable, we need to make the following changes:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/ assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"%>
```

```
<cs:ftcs>
<render:logdep c="Template" cid='<%=ics.GetVar("tid")%>' />

<assetset:setasset
  name="article"
  type="AVIArticle"
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="article"
  attribute="relatedStories"
  listvarname="relatedStories"
  typename="ContentAttribute" />

<insite:slotlist field="relatedStories">
  <ics:listloop listname="relatedStories">
    <ics:listget
      listname="relatedStories"
      fieldname="value"
      output="articleId" />
    <insite:calltemplate
      tname="Summary/SideBar"
      c="AVIArticle"
      cid='<%=ics.GetVar("articleId")%>' />
  </ics:listloop>
</insite:slotlist>
</cs:ftcs>
```

Every item of the list becomes a content-editable slot (droppable zone):



Cold Snap Back on the Scene
Once Top Manufacturer of Gear Revamps Ski Designs

Veteran Skier says Goodbye in Fitting Fashion
Farewell Is Also a Win, as Cr
King of Snow City

U.S. Olympic Skiers Prep for Games

Ski Castle and Snowboard Wizards Unite to Create New Ski and Snowboard Show
Giant Show Designed to Showcase the Best Equipment in the Market

All 25 Neva
January 10, 2012

In addition, since we used the `insite:slotlist` tag in our example, when hovering over the relatedStories area, a toolbar is shown, similar to the previous example.

25.1.8.3 Specifying a Different Ordering

In the previous example, the articles are rendered in a sequence, the first nested `insite:calltemplate` tag renders article #1, the second `insite:calltemplate` tag renders article #2, and so on:

```
<div class="top-stories">
  <div>article #1</div>
  <div>article #2</div>
  <div>article #3</div>
  <div>...</div>
</ul>
```

Your structure may be different depending on the structure of the HTML markup used in your site. For example, you may want articles to be displayed in a column layout such as:

```
article #1  article #2
article #3  article #4
etc.
```

Using the underlying HTML markup similar to the following:

```
<div class="left-column">
  <div>article #1</div>
  <div>article #3</div>
  ...
</div>
<div class="right-column">
  <div>article #2</div>
  <div>article #4</div>
  ...
</div>
```

In this example, articles are rendered in an order which doesn't match the ordering of the field (#1, #3, #5, then #2, #4, #6, etc.). If this is the case, WebCenter Sites needs to be aware of the ordering.

To do that, you need to explicitly indicate the index of the list item being edited, by using the `index` attribute as shown in Section 25.1.8, "Multivalued Fields."

25.1.8.4 Editing Mode and Caching

Caching is not disabled when working in **Web Mode / Edit View**. Therefore, templates which are configured to be cached, will still be cached when rendered in Edit View.

When working with assets, WebCenter Sites will automatically handle cache flushing providing that the correct dependencies are logged. For example, modifying the definition of a particular attribute (e.g., changing the allowed types of an asset reference field), will automatically flush all pagelets which rendering depends on this particular attribute.

There are however some instances where the page cache will need to be manually flushed:

- Modifying the definition of an association field.
- Removing a role from WebCenter Sites, if this role is directly referenced from a `insite:calltemplate` tag, through the `roles` attribute.
- Removing asset types or subtypes from WebCenter Sites, if this asset type or subtype is referenced from an `insite:calltemplate` tag through the `clegal` attribute.

25.2 Coding Templates for Presentation Editing

In this section, we'll look into the other aspect of the in-context UI and explain how templates can be coded to allow non-technical users to control presentation of content. This section presents information on context and on how to make a presentation change local (i.e. visible only on a given web page), or global (i.e. changing the presentation on one page can propagate the same change to multiple pages on the site).

Controlling presentation includes being able to select which page layout to select to render an entire web page, being able to select which content layout to select to render an asset in a given portion of a web page, and being able to select arguments sent to a pagelet template

This section builds on the HelloDetail template introduced in Chapter 24, "Creating Templates and Wrappers" and the examples in Section 25.1, "Coding Templates for In-Context Content Editing."

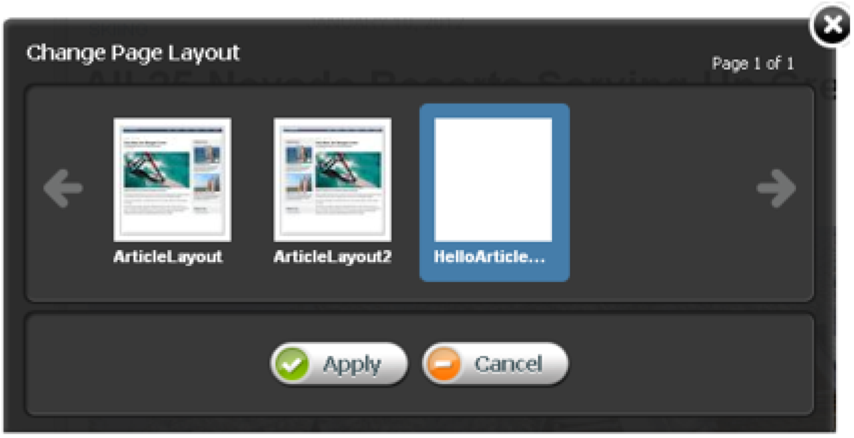
This section contains the following topics:

- Section 25.2.1, "Selecting a Different Layout for the Entire Web Page"
- Section 25.2.2, "Selecting a Different Layout for a Page Fragment"
- Section 25.2.3, "Editing Presentation and Content Simultaneously"
- Section 25.2.4, "Understanding the Context System Variable"
- Section 25.2.5, "Using Slots with CSElement and SiteEntry Assets"
- Section 25.2.6, "Constraining Asset Types"
- Section 25.2.7, "Preventing CSS and JavaScript Conflicts"

25.2.1 Selecting a Different Layout for the Entire Web Page

Editorial users can assign a layout template to assets by either:

- directly modifying the value of the `template` field in Form Mode of the Contributor interface.
- or selecting the Change Layout functionality from either the toolbar or menu bar.



Selecting a layout template for an asset will make this template the default choice when working with an asset in Web Mode of the Contributor interface, or when previewing the asset.

In order to enable content contributors to control the page layout used to render a given asset on a live site, the value of the template field should be looked up and used to calculate asset hyperlinks, as in the following example:

```
<asset:list
  type='<%=ics.GetVar("c")%>'
  list="asset"
  field1="id"
  value1='<%=ics.GetVar("cid")%>' />
```

```
<ics:listget
  listname="asset"
  fieldname="template"
  output="template" />

<render:gettemplateurl
  outstr="pageURL"
  tname='<%=ics.GetVar("template")%>'
  args="c,cid" />
```

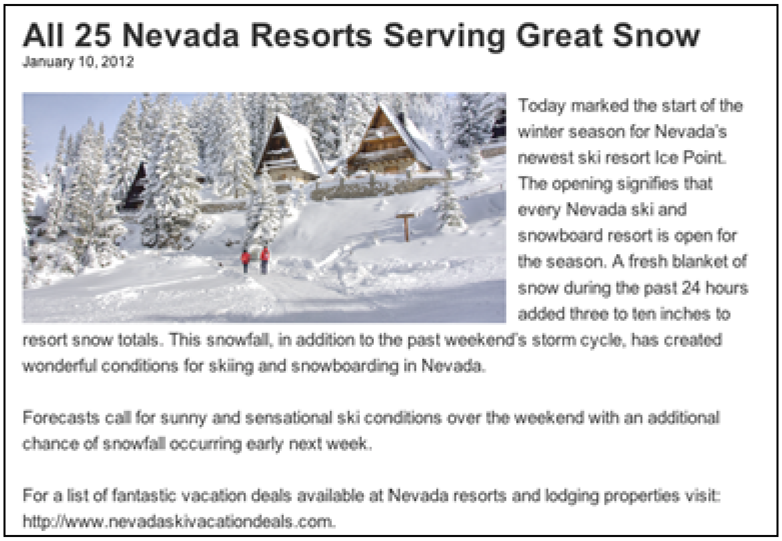
25.2.2 Selecting a Different Layout for a Page Fragment

This section builds on the HelloArticleLayout template used in Section 25.1, "Coding Templates for In-Context Content Editing."

Previously, in the HelloArticleLayout layout template, the main area of the article page was rendered using the HelloDetail template as shown in this code snippet:

```
<div id="container">
  <div class="content">
    <render:calltemplate tname="HelloDetail" args="c,cid" />
  </div>
  <div class="side-bar">
    ...
  </div>
</div>
```

which renders as (see screen capture below):



All 25 Nevada Resorts Serving Great Snow

January 10, 2012

Today marked the start of the winter season for Nevada's newest ski resort Ice Point. The opening signifies that every Nevada ski and snowboard resort is open for the season. A fresh blanket of snow during the past 24 hours added three to ten inches to resort snow totals. This snowfall, in addition to the past weekend's storm cycle, has created wonderful conditions for skiing and snowboarding in Nevada.

Forecasts call for sunny and sensational ski conditions over the weekend with an additional chance of snowfall occurring early next week.

For a list of fantastic vacation deals available at Nevada resorts and lodging properties visit: http://www.nevadaskivacationdeals.com.

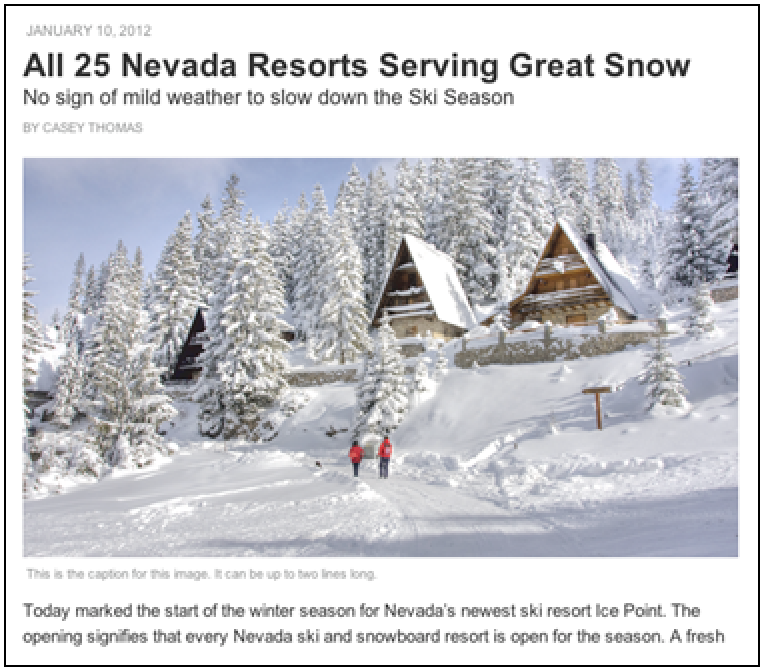
Note: The syntax used in the code sample:

```
<render:calltemplate tname="HelloDetail" args="c,cid" />
```

is a shortcut for (and is strictly equivalent to):

```
<render:calltemplate
  tname="HelloDetail"
  c='<%=ics.GetVar("c")%>'
  cid='<%=ics.GetVar("cid")%>' />
```

The avisports sample site also provides a Detail template for article assets. The Detail pagelet template is functionally equivalent to the HelloDetail template, in that it provides a detailed view of the article, and is meant to be rendered in the main area of the page, as shown in the screen capture below:



JANUARY 10, 2012

All 25 Nevada Resorts Serving Great Snow

No sign of mild weather to slow down the Ski Season

BY CASEY THOMAS

This is the caption for this image. It can be up to two lines long.

Today marked the start of the winter season for Nevada's newest ski resort Ice Point. The opening signifies that every Nevada ski and snowboard resort is open for the season. A fresh

25.2.2.1 Defining a Slot for Presentation Editing

If you want to allow non-technical users to choose which presentation (i.e. which pagelet template) to use in order to render a particular article page, you must define a slot.

The previous code in the HelloArticleLayout layout template rendered the main area of the article page using the HelloDetail template:

```
<div id="container">
  <div class="content">
    <render:calltemplate
      tname="HelloDetail"
      args="c,cid" />
  </div>
  <div class=side-bar>
    ...
  </div>
</div>
...
```

To define a slot, add the `insite` taglib directory.

```
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld" %>
```

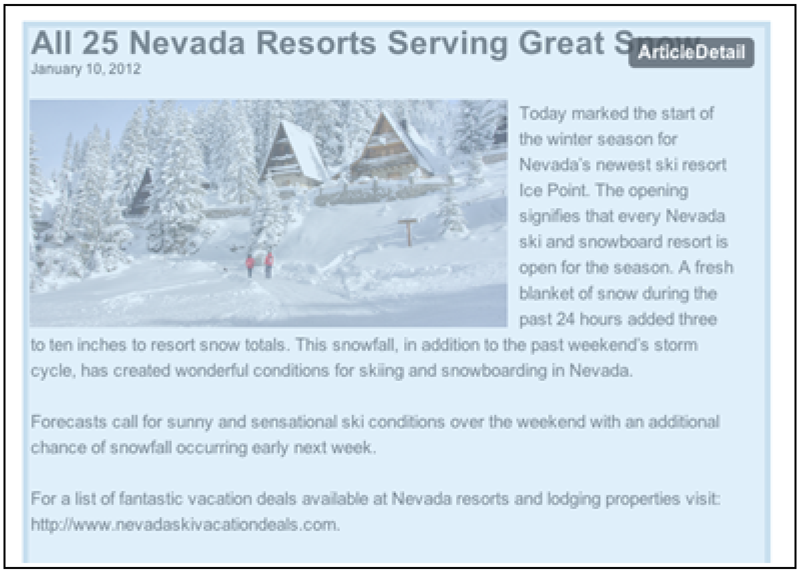
and replace the previous `render:calltemplate` tag with an `insite:calltemplate` tag and add the `slotname` and `variant` sttributes:

```
<div id="container">
  <div class="content">
    <insite:calltemplate
      slotname="HelloSlot"

      tname="HelloDetail"
      variant="HelloDetail|Detail"
      args="c,cid" />
  </div>
  <div class=side-bar>
    ...
  </div>
</div>
...
```

By adding the `slotname` attribute we are defining a slot, in our example, called `HelloSlot`. The `slotname` attribute value can be any string, but it must be unique across all templates of a given site. See Section 25.2.4, "Understanding the Context System Variable" for more information.

If we run this template again, there is initially no noticeable changes. Our article is still rendered using the HelloDetail template, as directed by the `tname` attribute. The changes become apparent by switching to **Edit View** and hovering over the main div section.

- the page fragment inside the main div is now marked with a blue overlay
- the slot name is indicated in the top right corner



All 25 Nevada Resorts Serving Great Snow

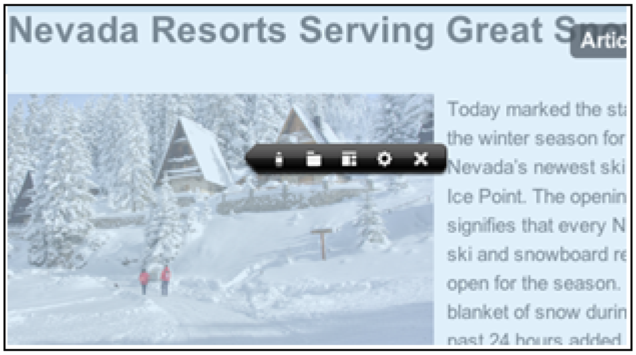
January 10, 2012

Today marked the start of the winter season for Nevada's newest ski resort Ice Point. The opening signifies that every Nevada ski and snowboard resort is open for the season. A fresh blanket of snow during the past 24 hours added three to ten inches to resort snow totals. This snowfall, in addition to the past weekend's storm cycle, has created wonderful conditions for skiing and snowboarding in Nevada.

Forecasts call for sunny and sensational ski conditions over the weekend with an additional chance of snowfall occurring early next week.

For a list of fantastic vacation deals available at Nevada resorts and lodging properties visit: http://www.nevadaskivacationdeals.com.

Clicking the blue overlay also brings up a toolbar (as shown in the screen capture below):



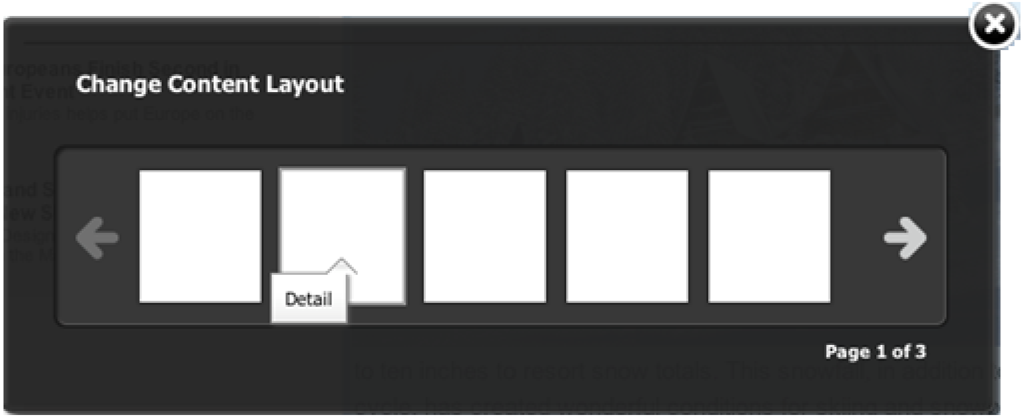
Nevada Resorts Serving Great S

Today marked the st
the winter season for
Nevada's newest ski
Ice Point. The openin
signifies that every N
ski and snowboard re
open for the season.
blanket of snow durin
past 24 hours added

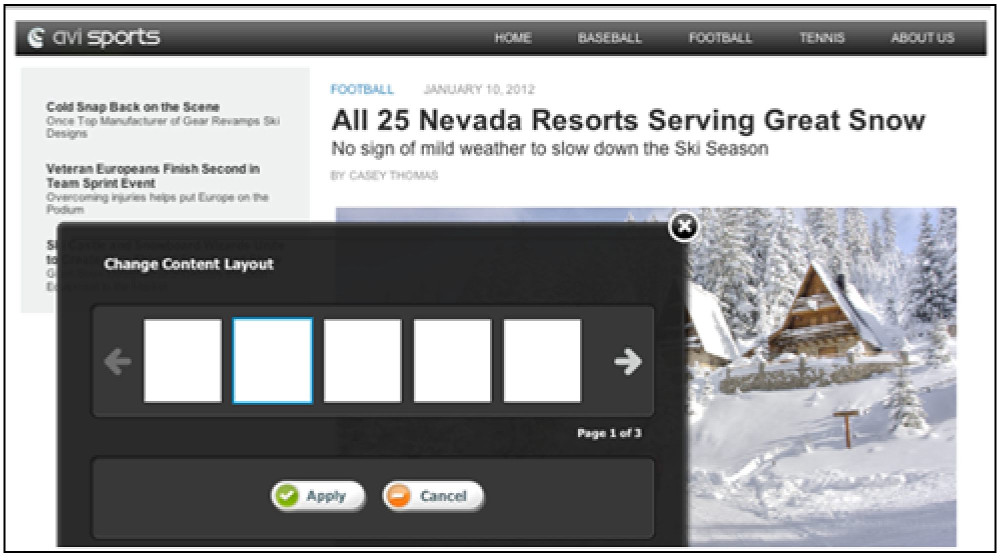
By selecting the **Change Content Layout** option, it now becomes possible to select a different content layout to render the asset in the main area of the page:

Change Content Layout icon:

Change Content Layout dialog:



The template picker shows the HelloDetail and Detail pagelet templates, specified by the `variant` attribute. Selecting the Detail template and clicking **Apply** renders the web page as shown in this screen capture:



Cold Snap Back on the Scene
Once Top Manufacturer of Gear Revamps Ski Designs

Veteran Europeans Finish Second in Team Sprint Event
Overcoming injuries helps put Europe on the Podium

FOOTBALL JANUARY 10, 2012

All 25 Nevada Resorts Serving Great Snow
No sign of mild weather to slow down the Ski Season
BY CASEY THOMAS

The slot defined in our example above is only meant for presentation editing and is not a content-editable slot (not a droppable zone).

The reason for this behavior is:

- The `insite:calltemplate` tag does not define any `field` attribute. That is, the content of the slot is not the value of an asset reference field.
- The tag is explicitly providing values for c (asset type) and cid (asset id). In this particular case, we want the article asset specified by the incoming c and cid request parameters to be rendered in this location.

In addition, if the `variant` attribute had been omitted, the slot layout would not have been editable, since the `insite:calltemplate` tag specifies an explicit value for the `tname` attribute (which acts as a default template for all assets dropped in this slot).

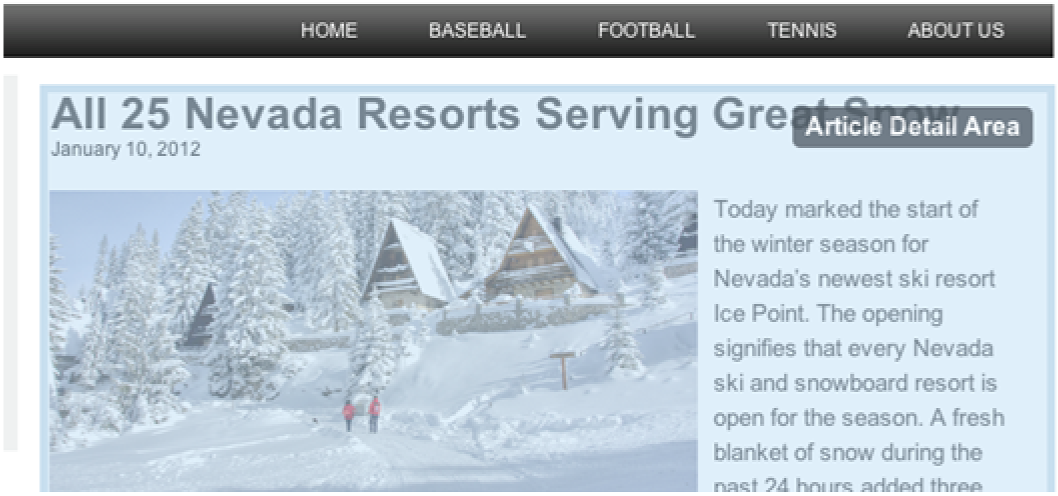
Note: The `variant` attribute can contain any regular expression. For example; `variant="Detail.*"` would restrict available templates to any pagelet template whose name starts with Detail.

25.2.2.2 Adjusting the Slot Title

Rather than showing the value of `slotname` in the blue overlay, which is typically a technical string meaningful to developers only, we can replace the value with any string. To do this, add a `title` attribute to the `insite:calltemplate` tag as shown in this code:

```
<insite:calltemplate
  slotname="HelloSlot"
  tname="HelloDetail"
  args="c,cid"
  variant="HelloDetail|Detail"
  title="Article Detail Area" />
```

Defining the `title` attribute overrides the default slot title:


HOME
BASEBALL
FOOTBALL
TENNIS
ABOUT US
All 25 Nevada Resorts Serving Great Snow
Article Detail Area
January 10, 2012
Today marked the start of the winter season for Nevada's newest ski resort Ice Point. The opening signifies that every Nevada ski and snowboard resort is open for the season. A fresh blanket of snow during the past 24 hours added three

25.2.2.3 Controlling Template Arguments

This section provides an example of controlling template arguments. We modify the HelloDetail template to accept an extra argument called `image-align` which will be used to align the article image left or right.

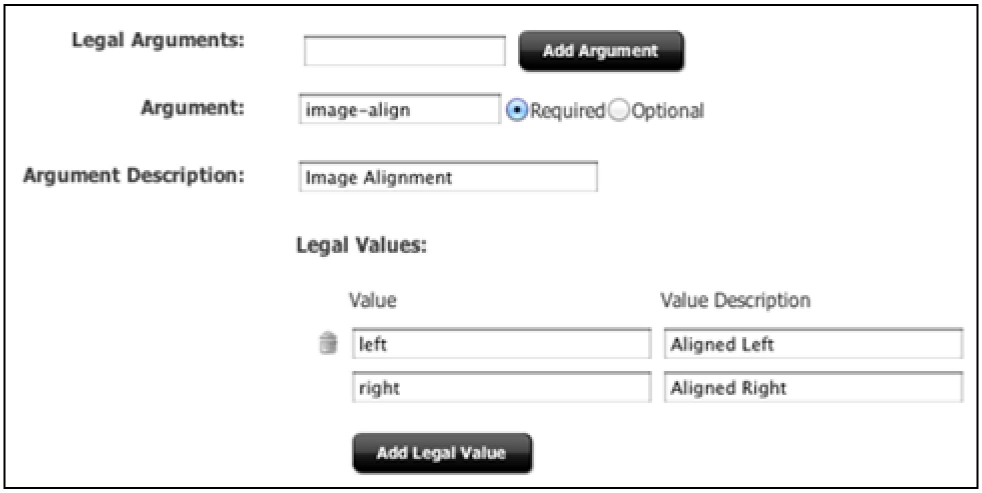
The process is as follows:

- The "image-align" argument has to be registered as a legal argument for the HelloDetail template. If this step is skipped, contributors will not be able to set its value from the editorial UI.
- In order to make sure that caching works properly, the new argument has to be declared as a cache criteria.
- Finally, the template code is modified in order to use the newly defined argument.

Note: On Using Eclipse with the WebCenter Sites Developer Tools plug-in: When editing a Template asset from the Admin interface, if this Template is also opened in WSDT at the same time, you must remember to synchronize your changes to the WSDT workspace. Template metadata stored in the WSDT workspace will otherwise override the values entered from the web interface.

---

1. Declare "image-align" as a legal argument of our HelloDetail template asset.
   a. From the Admin interface, edit the HelloDetail template asset.
   b. For Legal Arguments, enter image-align and click **Add Argument**.
   c. Select **Required**.
   d. For Argument Description enter Image Alignment.
   e. For LegalValues, add the following descriptions:
      – For Value: left enter Value Description: Aligned Left.
      – For Value: right enter Value Description: Aligned Right.
   f. Click **Save**.



Legal Arguments:

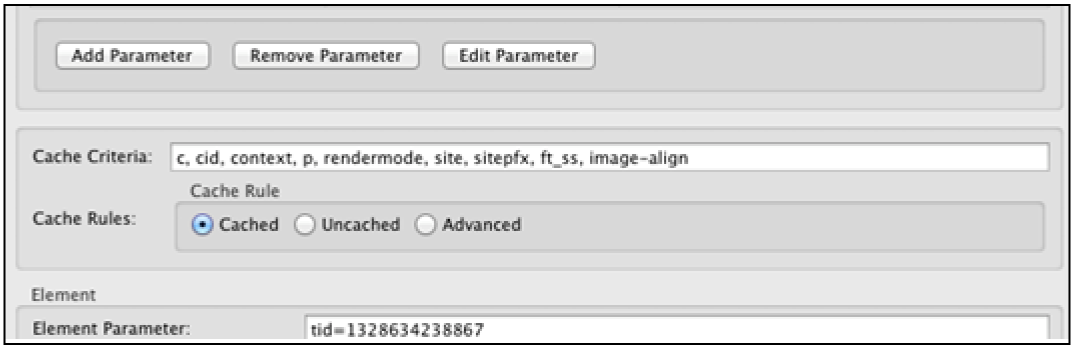
Argument: image-align ● Required ○ Optional

Argument Description: Image Alignment

Legal Values:

Value	Value Description
left	Aligned Left
right	Aligned Right

2. Remember to sync the change made in WebCenter Sites with the WSDT workspace.
3. Use the WebCenter Sites Developer Tools plug-in to add `image-align` to the set of cache criteria.
   a. Right-click the HelloDetail Template in the Sites workspace.
   b. Select **Properties**.
   c. In the Cache Criteria field, append `image-align` to the end of the list. See Chapter 23, "Creating Template, CSElement, and SiteEntry Assets" for more information on Cache Criteria values.
   d. Click **Submit**.



Cache Criteria: c, cid, context, p, rendermode, site, sitepfx, ft_ss, image-align

Cache Rules: Cache Rule ● Cached ○ Uncached ○ Advanced

Element

Element Parameter: tid=1328634238867

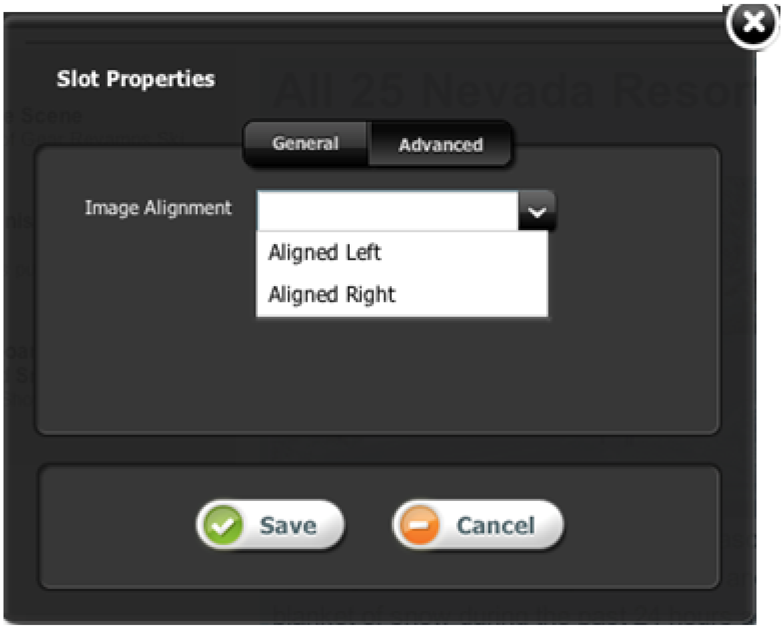
4. Optionally, a default value could be defined by using the **Additional element parameters** field and specifying the following value, for instance: "image-align=right".

5. Modify the HelloDetail pagelet template code

```
<insite:edit
  field="largeThumbnail"
  assettype="AVIImage"
  assetid='<%=ics.GetVar("imageId")%>' >

    <img class='photo <ics:getvar name="image-align"/>'
      src='<ics:getvar name="imageURL" />' />
</insite:edit>
...
```

Note that, in this particular case, the value of the parameter is used to set a different CSS class.

When going to the slot properties panel, assuming HelloDetail is the currently selected layout, the Advanced tab now shows the following options:



25.2.3 Editing Presentation and Content Simultaneously

The `insite:calltemplate` tag allows editorial users to edit associated content and edit the layout. This section explains the difference between a content-editable slot and a presentation-editable slot and how to combine the functionality of both to allow editorial users to edit both the associated content and the template used to render the content.

This section contains the following topics:

- Section 25.2.3.1, "Understanding Content-Editable Slots and Presentation-Editable Slots"
- Section 25.2.3.2, "Combining Content-Editable Slots and Presentation-Editable Slots"

25.2.3.1 Understanding Content-Editable Slots and Presentation-Editable Slots

Content-editable slots allow users to edit associated content by providing a droppable zone for the user. Presentation-editable slots allow users to select a different template to render the content.

To create a content-editable slot (creates a droppable zone for the user) the `insite:calltemplate` tag is used with the following defined parameters:

- `assetid`: The edited asset ID.
- `assettype`: The edited asset type.
- `field`: The edited field.
- `cid`: The ID of the asset to be rendered by the called template.
- `c`: The asset type to be rendered by the called template.
- `tname`: The pagelet template used to render the associated asset.

This code defines a content-editable slot that creates a droppable zone for the user:

```
<insite:calltemplate
  assetid=" "
  assettype=" "
  field=" "
  cid=" "
  c=" "
  tname=" "
/>
```

To create a presentation-editable slot (allows users to select a different template to render content) the `insite:calltemplate` tag is used with the following defined parameters:

- `slotname`: This attribute defines an identifier for the slot that is being filled with the called template. It should be reasonably easy to understand and should be unique across all templates.
- `cid`: The id of the asset to be rendered by the called template.
- `c`: The asset type to be rendered by the called template.
- `tname`: The default pagelet template to be called.

This code defines a presentation-editable slot that allows users to select a different template to render the content:

```
<insite:calltemplate
  slotname=" "
  cid=" "
  c=" "
  tname=" "
/>
```

25.2.3.2 Combining Content-Editable Slots and Presentation-Editable Slots

In this section, we present how to combine the functionality of a content-editable slot and a presentation-editable slot to allow editorial users to edit both the associated content and the template used to render the content.

To combine the functionality of both content-editable slots and presentation-editable slots, the `insite:calltemplate` tag is used along with all the attributes required for both a content-editable slot and a presentation-editable slot.

- These attributes are required for a content-editable slot (creates a droppable zone for the user):

  `field, assetid, assettype`
- This attribute is required to define a presentation-editable slot (allows users to select a different template to render the content):

  `slotname`

This code combines the attributes for a content-editable slot and a presentation-editable slot:

```
<insite:calltemplate
  slotname=" "
  assetid=" "
  assettype=" "
  field=" "
  cid=" "
  c=" "
  tname=" "
/>
```

25.2.3.2.1 Example: Combining a Content-Editable Slot and a Presentation-Editable Slot This section builds on the HelloSideBar template created in Section 25.1, "Coding Templates for In-Context Content Editing."

Previously, the HelloSideBar template was coded as shown:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"%>

<cs:ftcs>

<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
<assetset:setasset name="article"
  type='<%=ics.GetVar("c") %>' id='<%=ics.GetVar("cid") %>' />
<assetset:getattributevalues name="article"
  listvarname="relatedStories" attribute="relatedStories"
  typename="ContentAttribute" />

<insite:slotlist field="relatedStories">
  <ics:listloop listname="relatedStories">
    <ics:listget listname="relatedStories" fieldname="value"
      output="articleId" />

    <insite:calltemplate
      tname="Summary/SideBar"
      c="Article"
      cid='<%=ics.GetVar("articleId") %>' />
  </ics:listloop>
</insite:slotlist>
</cs:ftcs>
...
```

In this template, the related articles are made editable with content-editable slots (droppable zones for the user). That is, they are rendered using the Summary/SideBar template, without any possibility for editorial users to select a different template. However, they cannot change how the related article should be rendered. For this to happen, the HelloSideBar template needs to be modified as follows:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"%>

<cs:ftcs>

<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
<assetset:setasset name="article"
  type='<%=ics.GetVar("c") %>' id='<%=ics.GetVar("cid") %>' />
<assetset:getattributevalues name="article"
  listvarname="relatedStories" attribute="relatedStories"
  typename="ContentAttribute" />

  <insite:slotlist
    slotname="RelatedStoriesSlot"
    field="relatedStories">

    <ics:listloop listname="relatedStories">
      <ics:listget listname="relatedStories" fieldname="value"
        output="articleId" />

      <insite:calltemplate
        tname="Summary/SideBar"
        c="Article"
        cid='<%=ics.GetVar("articleId") %>'
        variant="Summary.*" />

    </ics:listloop>
  </insite:slotlist>
</cs:ftcs>
```

This is the same code used previously, except that we have specified a `slotname` attribute, and a `variant` attribute (in this case, the list of available templates is restricted to all pagelet templates starting with Summary).

Because `slotname` was included inside the `insite:slotlist` tag, it is not required to add it for the inner `insite:calltemplate` tag. The value is automatically inherited, as is the value of `tname` and `field`.

For example:

```
<insite:slotlist slotname=" " field=" ">
  <insite:calltemplate tname=" " c=" " cid=" " />
</insite:slotlist>
```

The Summary/SideBar now behaves as a default template for the related articles. By right-clicking any related article and selecting the Change Content Layout feature, it now becomes possible to select alternate templates.

25.2.4 Understanding the Context System Variable

In WebCenter Sites templates, `context` is a system variable maintained internally. Its value is retrieved by using the `ics:getvar` JSP tag or the `ics.GetVar()` method.

For example:

```
<ics:getvar name=context/>
ics.GetVar(context)
```

The value of context is determined by default. It is initially set to an empty string. Then, for every template called using `render:calltemplate` or `insite:calltemplate`, the value of `context` changes in the called template following this logic:

```
if parent_context is empty
  context = <c>:<cid>:<tname>
otherwise
  context = <parent_context>;<c>:<cid>:<tname>
```

This section contains the following topics:

- Section 25.2.4.1, "Defining the Scope of the Slot"
- Section 25.2.4.2, "Using the Context Variable in Action"
- Section 25.2.4.3, "Initializing the Context Value"
- Section 25.2.4.4, "Context Override"
- Section 25.2.4.5, "Context and Caching"

25.2.4.1 Defining the Scope of the Slot

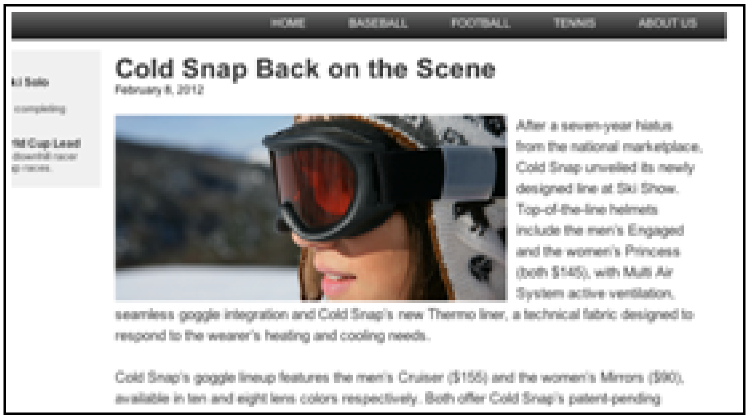
When defining a slot, it is possible for template developers to decide the scope of the slot. Typically, whether any presentation change made in this slot should be local or global.

- **local**: visible only on the currently edited web page
- **global**: spanning across multiple web pages in a site (possibly, ALL web pages in a site)

This is done by manipulating the value of the `context` variable and will be explained in the following sections.

25.2.4.2 Using the Context Variable in Action

1. Assign the HelloArticleLayout template to two avisports articles (for example,. All 25 Nevada resorts serving great snow and Cold snap back on the scene).
2. If we observe both of these articles in Web Mode of the Contributor interface, we see that the main slot is rendered with the default template HelloDetail as this is the template specified as the default template in the JSP code.

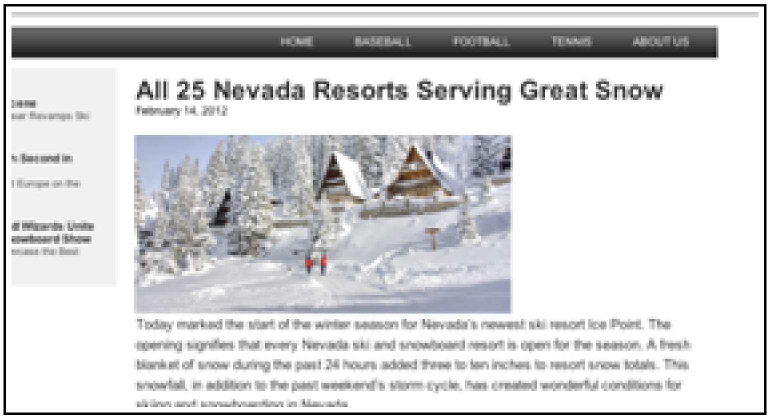


Cold Snap Back on the Scene

February 8, 2012

After a seven-year hiatus from the national marketplace, Cold Snap unveiled its newly designed line at Ski Show. Top-of-the-line helmets include the men's Engaged and the women's Princess (both $145), with Multi Air System active ventilation, seamless goggle integration and Cold Snap's new Thermo liner, a technical fabric designed to respond to the wearer's heating and cooling needs.

Cold Snap's goggle lineup features the men's Cruiser ($155) and the women's Mirrors ($90), available in ten and eight lens colors respectively. Both offer Cold Snap's patent-pending

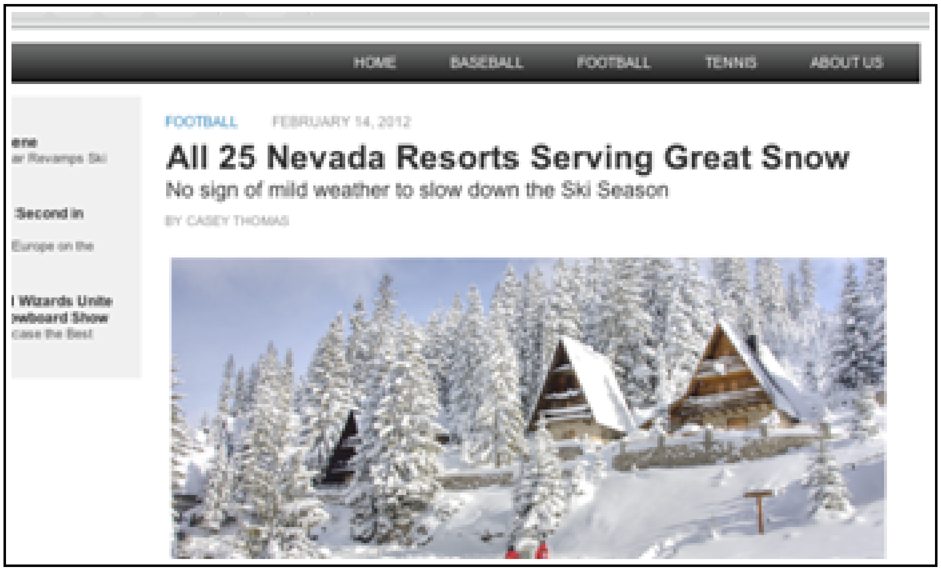


All 25 Nevada Resorts Serving Great Snow

February 14, 2012

Today marked the start of the winter season for Nevada's newest ski resort Ice Point. The opening signifies that every Nevada ski and snowboard resort is open for the season. A fresh blanket of snow during the past 24 hours added three to ten inches to resort snow totals. This snowfall, in addition to the past weekend's storm cycle, has created wonderful conditions for skiing and snowboarding in Nevada.

3. Assign the Detail template to All 25 Nevada resorts serving great snow article using the Change Content Layout option. The page should now look like this screen capture:



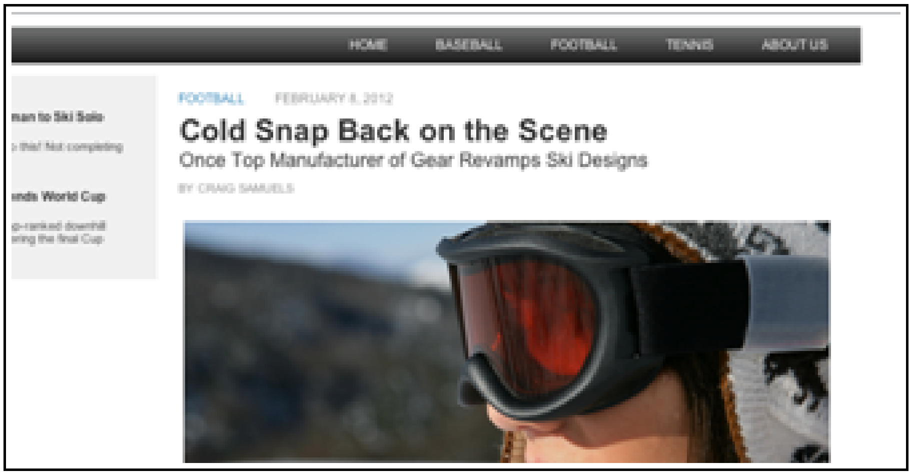
FOOTBALL FEBRUARY 14, 2012

All 25 Nevada Resorts Serving Great Snow

No sign of mild weather to slow down the Ski Season

BY CASEY THOMAS

4. Refresh the web page with the Cold snap back on the scene article. The presentation of the main slot has also been modified on this web page as it is also using the Detail template.



FOOTBALL FEBRUARY 8, 2012

Cold Snap Back on the Scene

Once Top Manufacturer of Gear Revamps Ski Designs

BY CRAIG SAMUELS

In order to understand what happened, we first need to know that WebCenter Sites stores presentation changes by recording the newly selected template against:

- the slot name
- the current site name
- and context

Consequently, when we modified the slot content layout from HelloDetail to Detail, the following presentation data was recorded:

- **site**: avisports
- **slotname**: ArticleDetail
- **context**: (empty)
- **tname**: Detail

Context is empty since, when the HelloArticleLayout template is executed, context is initially empty, and is never modified when the slot gets rendered. Thus, any web page of avisports rendered using the HelloArticleLayout template will match the recorded presentation data above. Consequently, the Detail template is now used for all article pages.

25.2.4.3 Initializing the Context Value

The behavior observed in the previous section may be the intended behavior, but in some cases, editorial users will need to be able to make local presentation changes, that is, changes visible only on the current web page being edited.

To do this, the `context` variable has to be set to a value which will uniquely identify a given web page. In our case, it is enough to initialize context with, for example, the template name, and the identifier and type of the rendered asset:

```
<ics:setvar
  name="context"
  value='<%=ics.GetVar("c")
  + ":" + ics.GetVar("cid")
  + ":HelloArticleLayout"%>' />
```

Other parameters can be added to initialize the context, depending on the intended result. We can add the line above to the HelloArticleLayout template and verify that presentation changes are *local* to each article page.

25.2.4.4 Context Override

Both the render:`calltemplate tag` and the `insite:calltemplate` tag have an optional `context` attribute, which can be used to override the current context.

25.2.4.5 Context and Caching

Context is only useful when presentation editing capabilities are enabled on your site.

If that is not the case, context can be removed from every template's cache criteria (avoiding the creation of unnecessary duplicates in the page cache).

25.2.5 Using Slots with CSElement and SiteEntry Assets

In our previous examples, slots were used to hold content. In this section, we look at using slots to hold functionalities such as a navigation bar, a login box, a code snippet showing the last ten published articles in a site, etc.These types of functionalities can be made available as a CSElement or SiteEntry asset.

By allowing CSElement or SiteEntry assets to be dropped in slots, non-technical users are given the ability to modify the behavior of a particular web page without having to modify code.

This section contains the following topics:

- Section 25.2.5.1, "Defining a Slot Containing a CSElement Asset"
- Section 25.2.5.2, "Using CSElement or SiteEntry Assets"
- Section 25.2.5.3, "Using Legal Arguments"
- Section 25.2.5.4, "Using Nested Slots"

25.2.5.1 Defining a Slot Containing a CSElement Asset

A slot meant to contain a CSElement asset is typically defined as:

```
<insite:calltemplate
  slotname="Navbar"
  clegal="CSElement"
/>
```

Or can be defined in this manner if the slot is meant to show the same content across all pages of the site:

```
<insite:calltemplate
  slotname="Navbar"
  clegal="CSElement"
  context="Global"
/>
```

There is no need to specify a `tname` attribute in this case, since the CSElement and SiteEntry assets are directly referring to the JSP element in charge of rendering them.

In order to see SiteEntry or CSElement in the list of allowed asset types for the slot, you need to make sure that both asset types have their **Can Be Child Asset** flag set to **True** (which means that this asset type can be the child asset type in an association field for another asset type.). See Section 15.2.6, "Step 4: Configure the Asset Type" for details.

> **Note:** In order to see SiteEntry or CSElement in the list of allowed asset types for the slot, you need to make sure that both asset types have their **Can Be Child Asset** flag set to **True** (which means that this asset type can be the child asset type in an association field for another asset type.). See Chapter 15, "Designing Basic Asset Types" for details.

25.2.5.2 Using CSElement or SiteEntry Assets

A SiteEntry asset does not hold any code, but simply points at a CSElement asset. The only difference between one case and the other is that, when using a SiteEntry, the element is invoked through the cache engine. Consequently, the same result can be achieved by dropping a SiteEntry asset or dropping its related CSElement asset.

The decision to use SiteEntry or CSElement is therefore implementation-dependent. Exposing a functionality as a SiteEntry only will ensure that caching is used to render this particular code snippet.

25.2.5.3 Using Legal Arguments

Like templates, it is possible to define legal arguments for CSElement assets. Once a CSElement has been dropped in a slot, the legal arguments are accessible from the slot properties panel.

The same applies when dropping a SiteEntry asset except that the legal arguments shown in the slot properties panel are the legal arguments of the related CSElement asset (SiteEntry assets do not have their own legal arguments).

25.2.5.4 Using Nested Slots

When dropping an asset in a content-editable slot, the template rendering this asset can potentially define other slots. In order to avoid too many controls and slots being rendered in a given area, the behavior of nested slots is automatically degraded to a simple droppable area (without a toolbar and overlay). It is generally recommended to avoid nested slots, in order to keep the Contributor interface simple and usable for users.

25.2.6 Constraining Asset Types

By default, WebCenter Sites allows the following asset types to be dropped into a slot:

- if `field` is defined (content-editable slot): any legal asset types given by the field definition
- if `field` is not defined (presentation-editable slot): any asset type for which the **Can Be Child Asset** flag is set to **True**. See Section 15.2, "Creating Basic Asset Types" for details.

It is also possible to further restrict allowable asset types, using the `clegal` attribute:

```
<insite:calltemplate
  slotname="Main"
  clegal="Article,Product"
  ...
/>
```

The following syntax allows to additionally restrict by asset subtypes:

```
<insite:calltemplate
  slotname="Main"
  clegal="type1:subtype1,type2:subtype2"
  ...
```

```
/>
```

Note: Using `"type:*"` (with asterisk as wildcard) is also valid, and behaves as the `"type"` value.

25.2.7 Preventing CSS and JavaScript Conflicts

The in-context UI is injecting styles and JavaScript into web pages, when rendered in the Contributor interface in **Web Mode / Edit View** (and only in this case).

This may possibly result in:

- **CSS conflicts**: for instance, slots are improperly displayed due to a site CSS rule applying to them)
- **JavaScript conflicts**: the web page already has JavaScript which conflicts with the JavaScript injected in the page in the editing view. As a result, either the editorial UI or the page itself do not work properly.

CSS conflicts are typically solved by adding extra CSS rules, which are loaded only when the template is rendered inside the editorial UI, in editing mode. This is done by using the `insite:ifedit` tag, which allows to execute JSP code only when the JSP template is ran in editing mode:

```
<insite:ifedit>
<%-- This stylesheet import will only occur in editing mode.
  -- It will not have any impact on the actual rendering of the
  -- live site.
  --%>
  <link rel="stylesheet"
   type="text/css"
   href="/css/editorial.css" />
</insite:ifedit>
```

Extra CSS classes can be added to slots by using the `cssstyle` attribute of the `insite:calltemplate` tag, to specify specific CSS rules for slots.

CSS conflicts typically when rendering slots around elements which are floated: in this case, the blue or green overlay which is marking the slot area is likely to not have the proper dimension.

JavaScript conflicts are normally solved by degrading the behavior of the web page in editing mode only, in order to let the scripts injected by the editorial UI function properly. This may mean disabling some of the page functionality in **Web Mode / Edit View** of the Contributor interface.

25.3 Enabling Content Creation for Web Mode

Contributors can be allowed to create content from the in-context UI. This is done by providing: a start menu item of type `insite` and at least one layout template applicable to the asset type to create.

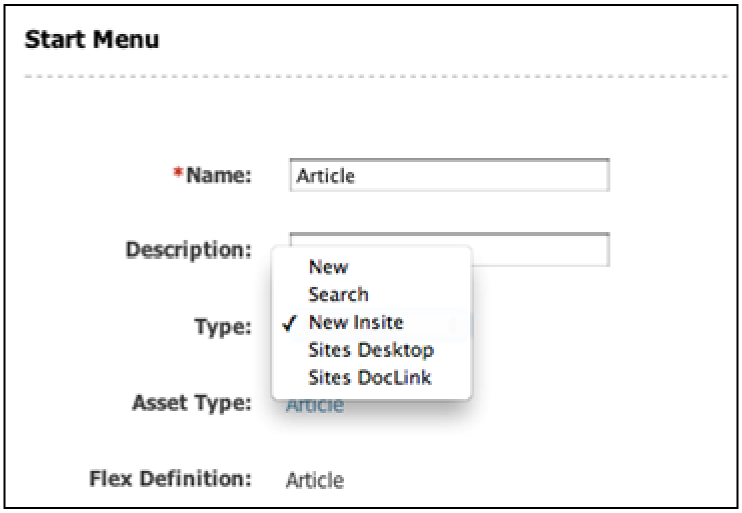
This section contains the following topics:

- Section 25.3.1, "Defining a Start Menu for In-content Creation"
- Section 25.3.2, "Providing Layout Templates for In-Context Creation"
- Section 25.3.3, "Providing Empty Value Indicators"

- Section 25.3.4, "Providing Editing-Specific Presentation Logic"

25.3.1 Defining a Start Menu for In-content Creation

In order to enable a particular asset type for in-context creation, a start menu of type insite has to be provided. That is, the **New Insite** option has to be selected in the **Type** drop-down menu:



Start Menu

*Name: Article

Description:

Type: ✓ New Insite

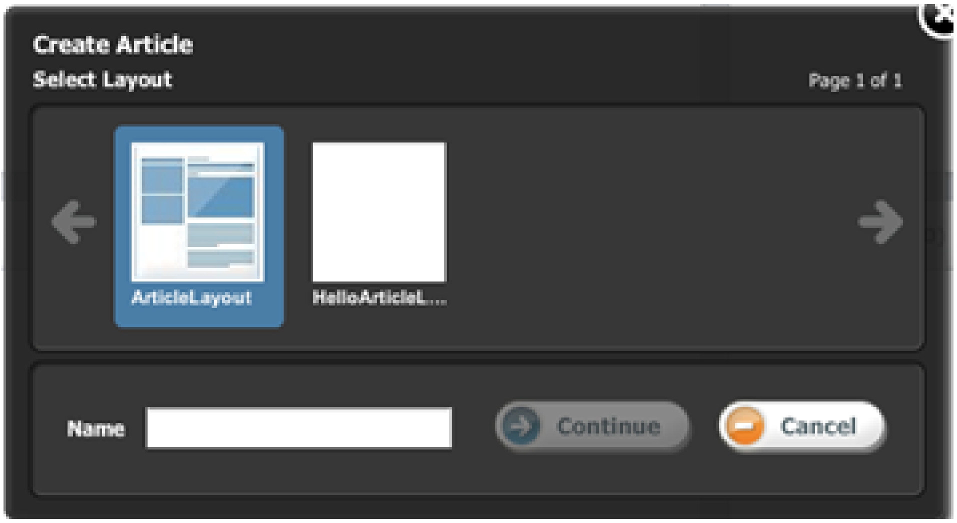
Asset Type: Article

Flex Definition: Article

In addition, if the asset being created has one or more required fields, default values have to be provided in the start menu, using the **Default Values** menu.

If required values are missing, the user will be directed to the asset form.

What happens when a user selects a **New Insite** start menu? The editorial user is first prompted to select a layout template and a name for the newly created asset:



Create Article

Select Layout

Name

Once **Continue** is clicked, assuming no workflow is configured, a new asset is created in the background, and then rendered using the selected layout template.

25.3.2 Providing Layout Templates for In-Context Creation

The same layout templates used for editing can be used for in-context creation. However, extra care must be taken in order to make sure the template will render properly with an empty asset.

This typically requires:

- style adjustments, so the various elements on the page are rendered in the appropriate positions, although no values are rendered
- meaningful help text (no value indicators), displayed to users when an editable field is empty
- additional presentation logic, only executed when the template is ran in editing mode, on the editorial platform.

This section contains the following topics:

- Section 25.3.2.1, "Adjusting Stylesheets"
- Section 25.3.2.2, "Adjusting Stylesheets for Slots"

25.3.2.1 Adjusting Stylesheets

When the stylesheets have to be specifically adjusted for creating or editing content, the corresponding import statements can be enclosed in an `insite:ifedit` tag:

```
// import "delivery" stylesheets
<link type="text/css" rel="stylesheet" href="somecss.css" />
...

// then import "editing only" stylesheets. using the insite:ifedit
// tag ensures that only those stylesheets will be imported
// when rendering the template in create/edit mode.

<insite:ifedit>
  <link type="text/css" rel="stylesheet" href="edit.css"  />
  ...
</insite:ifedit>
```

25.3.2.2 Adjusting Stylesheets for Slots

When adjusting styles for the rendering of slots, the `cssstyle` attribute of the `insite:calltemplate` tag can be used to specify additional class names which can then be used in CSS rules.

```
<insite:calltemplate
  slotname="mySlot"
  ...
  cssstyle="myClassName"
  ...
/>
```

For example, if we wanted to force `mySlot` to have a 50px height when it is empty, we could provide the following CSS rule:

```
.myClassName .emptyIndicator {height: 50px !important;}
```

CSS rules can be grouped inside a specific stylesheet, and then imported conditionally, only when the template is executed in the context of **Web Mode / Edit View** of the Contributor interface, using the `insite:ifedit` JSP tag.

25.3.3 Providing Empty Value Indicators

For slots, use the `emptytext` attribute of the `insite:calltemplate` tag:

```
<insite:calltemplate
  slotname="mySlot"
  emptytext="Drag an Article here"
  ...
/>
```

For other editing fields, use the `noValueIndicator` parameter of the `insite:edit` tag:

```
<insite:edit
  field="headline
  params="{noValueIndicator: 'Enter Headline Here'}"
  ...
/>
```

25.3.4 Providing Editing-Specific Presentation Logic

In some cases, the presentation logic will be slightly different in delivery mode, compared to create/edit mode.

For example, in create/edit mode, when rendering an in-context editable list of articles, we might want to always display 5 extra empty slots. In that case, the logic has to account for both delivery and edit mode. It could be written as follows:

```
<%
// assuming that "relatedArticles" contains a list of
// related articles
%>
<insite:slotlist field="someAssetField">
  <ics:listloop listname="relatedArticles">
    <ics:listget
      listname="relatedArticles"
      fieldname="value"
      output="articleid" />
    <div class="post">
      <insite:calltemplate
        tname="Summary"
        c="Article"
        cid='<%=ics.GetVar("articleid")%>' />
    </div>
  </ics:listloop>
</insite:slotlist>

<%
// in this example, we add five extra empty slots
%>
<insite:ifedit>
  <%
    // in order to not disrupt rendering in delivery, this code is
    // added inside the insite:ifedit tag
  %>
  <c:forEach begin="0" end="4">
    <div class="post">
      <%
      // no c, cid specified, this renders an empty slot
      %>
      <insite:calltemplate tname="Summary" />
    </div>
  </c:forEach>
</insite:ifedit>
```

Obviously, the code could be adjusted to accommodate for any particular logic. For instance, we might want to display a maximum of 5 slots, empty or not. In that case, the last part of the code snippet could be written as follows:

```
<%
// get how many articles are in the list
%>
<ics:listget
  listname="relatedArticles"
  fieldname="#numRows"
  output="nbArticles" />

<c:forEach
  begin='<%=Integer.valueOf(ics.GetVar("nbArticles"))%>'
  end='4'>
  <div class="post">
    <insite:calltemplate tname='Summary" />
  </div>
</c:forEach>
```

26

Creating Collection Assets, Query Assets, and Page Assets

The core asset types delivered with WebCenter Sites provide basic site design logic. This chapter describes how to create the page, query, and collection assets that implement the functionality of your online site. It also includes a section for the stylesheet asset type, a sample asset type delivered with the Burlington Financial sample site.

The preceding chapter describes how to create Template assets. Because you assign Template assets to your other assets, it is typical to create your templates before you create your site design asset types.

The procedures for working with assets of any type are very similar and are described thoroughly in the *Oracle Fusion Middleware WebCenter Sites User's Guide*. This chapter presents procedures that are unique for the collection, query, stylesheet, and page asset types.

This chapter contains the following sections:

- Section 26.1, "Collection Assets"
- Section 26.2, "Query Assets"
- Section 26.3, "Page Assets"

26.1 Collection Assets

A collection asset stores an ordered list of assets of one type. You **create** (or design) a collection asset by naming it and selecting query assets for it. By default, you can select up to three query assets. If your site design requires more queries for collection assets, you can create additional associations for the additional queries. For information about creating associations, see Section 15.2.11, "Step 9: (Optional) Configure Association Fields."

A collection uses a query asset to obtain a list of possible assets for the collection. You **build** (or populate) a collection by running its queries, selecting assets from the results of the queries, and then ranking and ordering the assets that you selected. This ranked, ordered list is the collection.

Using collections is one way to keep the content displayed on rendered pages current and up-to-date. The Burlington Financial sample site uses several collections. For example, you can select a collection in the Top Stories field for a Burlington Financial page asset. A publisher or content provider can then change the content identified by that association by doing one of the following:

- Selecting a different collection from the tree

- Building the assigned collection and selecting different assets in it

This section contains the following topics:

- Section 26.1.1, "Before You Begin"
- Section 26.1.2, "Creating Collection Assets"
- Section 26.1.3, "Sharing Collection Assets"

26.1.1 Before You Begin

Before you **create** collection assets, note the following:

- A collection must have at least one query, so be sure that you create the queries before you try to create your collections.
- Because you assign templates to collections, you should also create the Template assets before you create your collection assets.

Before you **build** the collection, you should determine how the Template asset assigned to it is coded. For example, if you select 100 assets for a collection but the template is coded to display only five of them, the following occurs:

- The rendered page that displays those assets displays only the first five.
- The page takes longer to render than necessary because WebCenter Sites has to sort through all 100 assets even though it displays only the first five.

For more information about building a collection, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

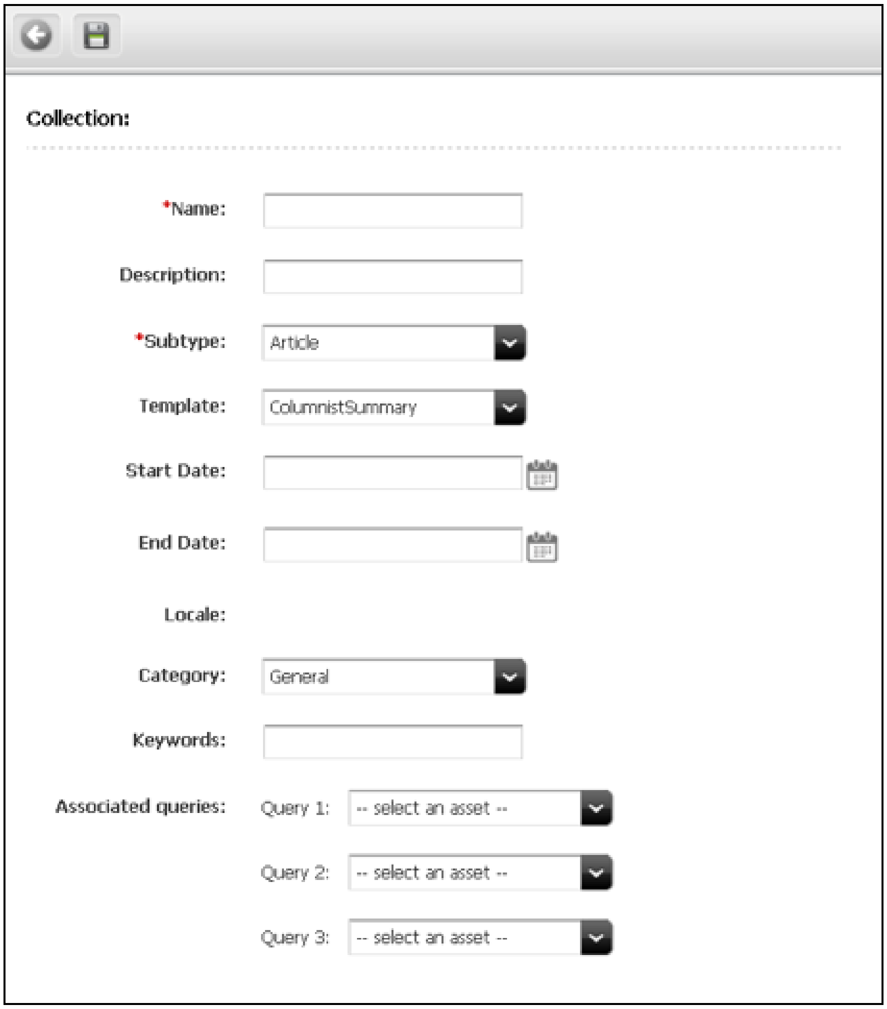
26.1.2 Creating Collection Assets

To create a collection asset

> **Note:** To use this procedure, you must have Collection asset types enabled for the site you are working in. Step 6 indicates whether they are enabled.

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Make sure that you have completed the steps Section 26.1.1, "Before You Begin."
5. Click **New** on the button bar.
6. Select **New Collection** from the list of asset types. (If Collection asset types are not enabled, the option is not displayed.)

   The Collection form appears:



Collection:

*Name:

Description:

*Subtype: Article

Template: ColumnistSummary

Start Date:

End Date:

Locale:

Category: General

Keywords:

Associated queries: Query 1: -- select an asset --

Query 2: -- select an asset --

Query 3: -- select an asset --

7. (Required) In the **Name** field, type a descriptive name for the page. You can enter up to 64 alphanumeric characters, but the first character must be a letter. Underscores (_) and hyphens (-) are acceptable, but tab and space characters are not.

8. In the **Description** field, type a brief description of the collection. You can enter up to 128 characters.

9. (Required) In the **Subtype** field, select the type of asset this collection will hold.

> **Note:** Collection subtypes are controlled by Query. When a query is set up for a certain asset type, that asset type becomes a value of the **Subtype** field. The **SubType** field thus lists every asset type for which a query was created. For information about creating Query assets, see Section 26.2.5, "Creating Query Assets."

10. In the **Select a Template** field, select a Template asset from the drop-down list.

11. In the **Category** field, select a category from the drop-down list. (If you do not select a category, the first item on the list is selected by default.)

12. In the **Keywords** field, enter keywords that you and others can use as search criteria in the Advanced Search form when you search for this collection in the future.
13. In the **Associated queries** section, select up to three queries. All of the queries that you select for this collection must return assets of the same type.
14. Click **Save**.

26.1.3 Sharing Collection Assets

Before you share a collection asset, consider the following:

- Building a collection in one site builds it in all of the sites that it is shared with. You cannot build a collection to include different assets for different sites.
- The query assets used in the shared collection must be coded to return only assets that are shared to all the sites that the collection is shared with.
- As with any shared asset, be sure that the template assigned to the collection is also shared to the other site.

26.2 Query Assets

A query asset stores a database query that retrieves a list of other assets from the database. However, if the query is to be used for a collection, it can return assets of one type only.

This section contains the following topics:

- Section 26.2.1, "Query Assets and Other Assets"
- Section 26.2.2, "How the Query Is Stored"
- Section 26.2.3, "Commonly Used Fields for Queries"
- Section 26.2.4, "Before You Begin"
- Section 26.2.5, "Creating Query Assets"
- Section 26.2.6, "Sharing Query Assets"
- Section 26.2.7, "Previewing and Approving Query Assets"

26.2.1 Query Assets and Other Assets

WebCenter Sites uses queries differently in collection assets than it does for other assets:

- When you build (or populate) a collection, you run one or more query assets and then select and order the assets that you want from the resulting list. The collection is a **static** list of assets selected from the query resultsets.
- You can select queries for a page asset either through informal relationships or through named associations. You can select queries for other asset types (article, for example) through named associations.

  When the asset is rendered, it does not invoke the query directly. Either the template element that formats the asset or a template element that formats the query is coded to invoke a standard WebCenter Sites element called:

```
OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery
```

This element runs the query asset when the asset it is associated with is rendered, which means the resultset is **dynamic**.

26.2.2 How the Query Is Stored

A query asset can store its database query in one of two ways:

- **Directly**. You can write the query directly into the **SQL query** field of the Query form. You can either use standard SQL for the query, or, if your WebCenter Sites systems use an external search engine, you can use an appropriate search engine query.
- **Indirectly**. You can write the query in an element and then store the location of that element in the query asset by identifying it in the **Element name** field in the Query form. An element for a query is like any other element: you can use XML, JSP, JavaScript, HTML, and so on.

Most of the Burlington Financial queries store the query directly; that is, the SQL query is written directly into the **SQL query** field in the Query form. For example, the following code is from the News Wire Feed Query:

```
SELECT DISTINCT Article.id, Article.name, Article.updateddate,
Article.subheadline, Article.abstract, Article.description,
Category.description AS category, StatusCode.description AS
statusdesc FROM Article, Category, AssetPublication, StatusCode
WHERE Article.status!='VO' AND Article.category=Category.category
AND Article.status=StatusCode.statuscode AND Category.assettype='Article'
AND Article.source='WireFeed' AND Article.category='n' AND Article.id =
AssetPublication.assetid AND AssetPublication.pubid = 968251170475
ORDER BY Article.updateddate DESC
```

26.2.3 Commonly Used Fields for Queries

There are several WebCenter Sites fields, four of which are used in the preceding News Wire Feed query example, that you are likely to use in your queries:

- status
- updateddate
- source
- category
- startdate and enddate

The rest of this section defines the fields in this list.

status

All assets have a **status**. When an asset is created, WebCenter Sites adds a row to the table that holds assets of that type and sets its status to PL, which means created.

The following table lists and defines the status codes that WebCenter Sites uses:

Status Code	Definition
PL	created
ED	edited
RF	received (from XMLPost, for example)

Status Code	Definition
UP	upgraded from Xcelerate 2.2
VO	deleted (void)

These codes are listed in the `StatusCode` table in the database.

When an asset is deleted, WebCenter Sites changes its status to VO and renames the string in its Name field to its object ID.

Write your queries to exclude assets whose status is VO. For example: `WHERE Article.status!='VO'`

updateddate

The information in the **updateddate** field represents the date on which the information in the status field was changed to its current state. Depending on the design of your site, you might want a query to return assets based on this date.

source

The **source** field is a default WebCenter Sites field that can identify where an asset originated. It is not required.

For example, the Burlington Financial sample site has sources named WireFeed, Asia Pulse, UPI, and so on. You add sources for your sites on the **Admin** tab in the tree. See Section 15.2.13, "Step 11: (Optional) Configure Sources."

If you use source with your assets, you can write your queries to use source as a parameter. In the previously mentioned News Wire Feed query example, the `AND Article.source='WireFeed'` statement ensures that only articles with WireFeed in their Source fields are selected by this query.

category

The **category** is a default WebCenter Sites field that can categorize assets according to a convention that works for your sites. It is not required.

For example, the Burlington Financial sample site has categories named Personal Finance, Banking and Loans, Rates and Bonds, News, and so on. You add categories for your sites on the Admin tab in the tree. See Section 15.2.12, "Step 10: (Optional) Configure Categories."

If you use category with your assets, you can write your queries to use category as a parameter. In the previously mentioned News Wire Feed query example, the `Article.category='n'` statement includes article assets from the News category.

pubid

A **pubid** is a unique value that identifies a site (or, in old terminology, a publication). When an asset is created, WebCenter Sites writes information about that asset to several database tables, one of which is the `AssetPublication` table.

An asset's row in the `AssetPublication` table includes the pubid of the site the asset was created for. If the asset is shared, the `AssetPublication` table has a row for each site that the asset is shared with. For example, if an article asset is available in two sites, there are two rows for that article in the `AssetPublication` table.

If you have only one WebCenter Sites site on your system or if your query results do not need to be site-specific, you do not need to code your queries to consider pubid. The Burlington Financial queries, however, are coded to restrict assets based on the pubid of Burlington Financial (`AssetPublication.pubid = 968251170475`) so that they do not return assets from another site.

startdate and enddate

Neither of the sample sites use the **startdate** and **enddate** fields but the WebCenter Sites database has columns to store this information. These fields exist so that you can assign time limits to assets. If your asset types use the startdate and enddate fields, you can create queries that select assets based on the dates stored in those fields.

26.2.4 Before You Begin

Before you begin creating query assets, consider the following:

- Query assets that are used on assets other than collections are not required to have templates. You can either create template elements specifically for your query assets that identify, run, and display the results, or you can code the template elements for your page assets to do that.
- When you write a query for a collection, be sure to code it to select the fields that are required for that asset type. WebCenter Sites is programmed to expect information from an asset type's required fields so that it can display that information in the Build Collection form.

  For example, the **Name** and **Description** fields are required fields for a Burlington Financial article. (The **Description** field is renamed and displayed as **Headline** in the form.) Therefore, the queries for Burlington Financial collections that hold Burlington Financial articles select the **Name** and **Description** fields. Those queries also select several other fields, but WebCenter Sites requires at least the **Name** and **Description** fields to present the assets returned by the queries in the **Build Collection** forms correctly.
- Query assets that are used only for collections have no need for templates. The template element assigned to the collection formats the assets in a collection's list of assets.
- For performance reasons, be sure to create efficient queries. For example:
  - Include as much logic as possible in the query rather than in the element that runs and displays the results of the query. For example, if you want to filter or constrain a list of articles, be sure the query performs the filtering or constraining step so that the list returned to the element is complete rather than coding the query to return the entire list and using the element code to constrain the list.
  - Be sure your queries return only the information that the element displays.
- Query assets that are for collections must return assets of one type only.

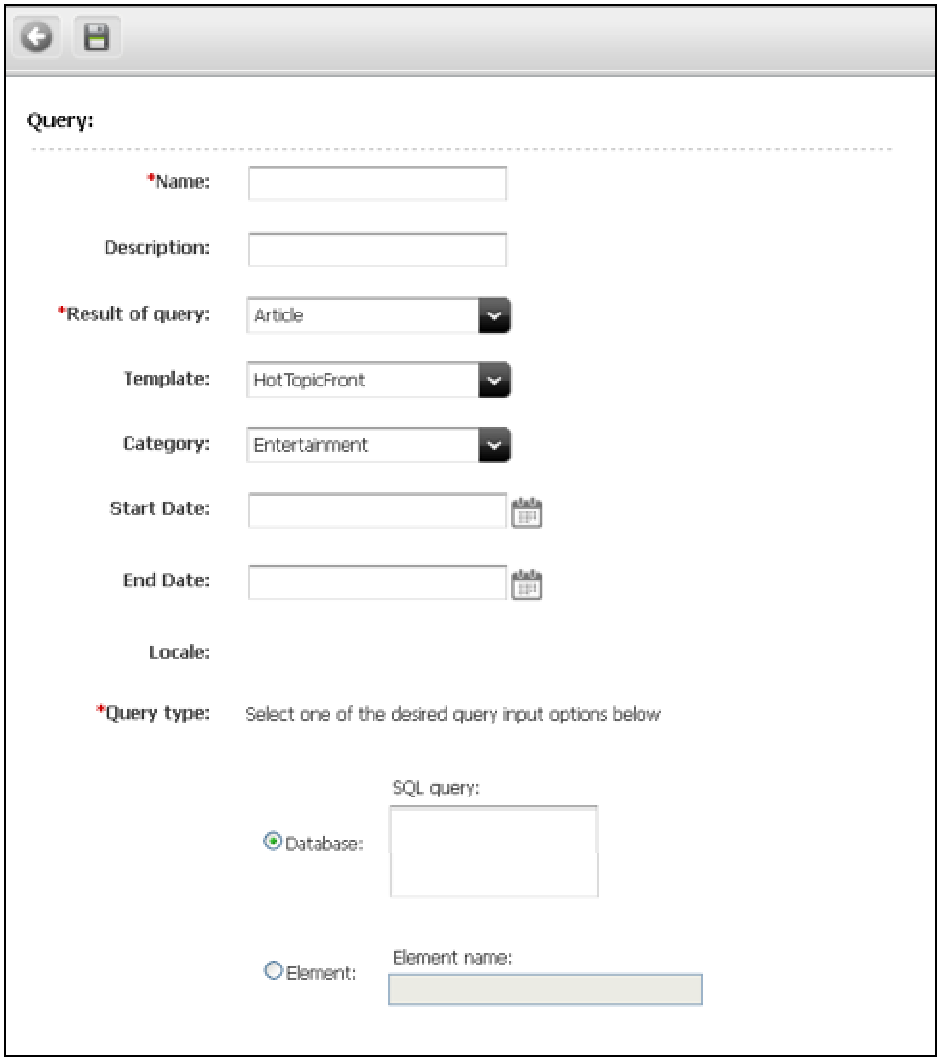
26.2.5 Creating Query Assets

To create a query asset

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Click **New** on the button bar.

5. Select **New Query** from the list of asset types. (Query asset types must be enabled for your site.)

   The Query form appears:



Query:

*Name:

Description:

*Result of query: Article

Template: HotTopicFront

Category: Entertainment

Start Date:

End Date:

Locale:

*Query type: Select one of the desired query input options below

● Database: SQL query:

○ Element: Element name:

6. (Required) In the **Name** field, type a descriptive name for the query asset. You can enter up to 64 alphanumeric characters, but the first character must be a letter. Underscores (_) and hyphens (-) are acceptable, but tab and space characters are not.

7. In the **Description** field, type a brief description of the query. You can enter up to 128 characters.

8. In the **Template** field, select a Template asset from the drop-down list.

9. In the **Category** field, select a category from the drop-down list. (If you do not select a category, the first item on the list is selected by default.)

10. In the **Result of query** field, select the type of asset that this query returns. (The query can return assets of one type only if this asset is to be used by a collection.)

11. Do one of the following:

    – If you want to store the query directly in this asset, select **Database**, click in the **SQL query** field, and then write your query.

   - If you wrote the query in an element, select **Element** and then enter the entire name of the element in the **Element name** field.

12. Click **Save**.

26.2.6 Sharing Query Assets

If you plan to share a query asset with another site, consider the following tips:

- If you want your query results to be site-specific, be sure to include a `WHERE` clause for `pubid` so that the query does not return assets to a site where those assets have not been shared.
   - For example, in either a query for a collection or a query for a static site, you can use the following statement:

     ```
     WHERE AssetPublication.pubid = SessionVariables.pubid
     ```

     because `SessionVariables.pubid` is always set when you are building a collection or using the Export to Disk function.
   - If the query is to be used on a dynamic site, you can use that same statement as long as you code your elements to either pass in the identify of `pubid` to the `ExecuteQuery` element or to set the `SessionVariables.pubid` variable.
- Because page assets cannot be shared, you should not share query assets if they return page assets.
- As with any shared asset, if the query has a template, be sure that the template assigned to the query is also shared with the other site.

26.2.7 Previewing and Approving Query Assets

First, remember that not all query assets have their own templates. If a query asset was designed to be used on a page asset and it is the page asset's template that actually formats the query, you must preview the page in order to preview the query.

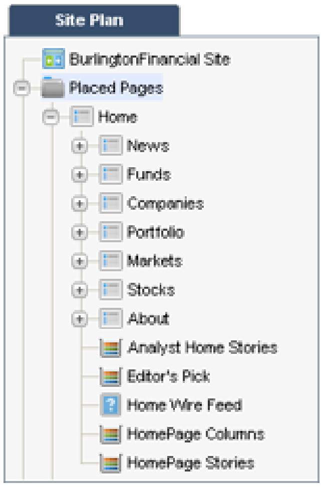
If your online site is a dynamic site (that is, you use the Mirror to Server publishing method) a query asset might return different assets on the management system than it does on the delivery system, depending on which assets have been published.

Therefore, if you preview your query to determine whether you should approve it or not, remember that the assets that it returns on the management system (where you are previewing it) could be different than the assets that it will return on the delivery system after it is published.

26.3 Page Assets

Page assets are site design assets that store references to other assets, organizing them according to the design that you and the other designers are implementing.

Open WebCenter Sites and examine the page assets listed in the site tree for the Burlington Financial sample site:



These page assets represent sections of the site, in essence the structure or organization of the site. They do not represent each and every rendered page that can possibly be served. This structural organization is primarily for the benefit of your WebCenter Sites users. This is not the only way of organizing your site, but it is convenient for your editors to see a structure that resembles your finished website.

Typically, you create page assets once: when you design the site. You associate collections, queries, articles, and so on with page assets and you code template elements that format the types of assets you want to associate with the page asset.

Before you can select the correct content for your page assets, you must be familiar with how your site is structured and what your template elements for page assets are designed to do. That is why you and other site developers (the people who are coding elements and creating Template assets) typically create the page assets for a site.

This section contains the following topics:

- Section 26.3.1, "Page Asset Model"
- Section 26.3.2, "Designing Page Attributes"
- Section 26.3.3, "Creating a Page Asset"
- Section 26.3.4, "Placing Page Assets"
- Section 26.3.5, "Moving Page Assets in the Site Tree"
- Section 26.3.6, "Placing Page Assets and Workflow"
- Section 26.3.7, "Editing Page Assets"
- Section 26.3.8, "Deleting Page Assets"

26.3.1 Page Asset Model

Page asset model is similar to the flex asset model. This provides option to change the data structure of the Page asset. Page asset model is made of a family of asset types.They are:

- Page attribute
- Page definition
- Page

Following are some general characteristics of the page asset model:

- Page assets are described by the page attributes that you select for them.
- The page attributes that characterize page assets are themselves assets. This means that attributes can be passed through workflow, edited, monitored by revision tracking, and subjected to all other content management operations.
- If you ever need to add attributes to your asset types in the future (a common occurrence with products), you just create the new attribute and assign it to the appropriate definitions.
- This asset model supports assets that have many, many attributes, which means that you can support large sets of data.

Page asset model does not have parent definition asset type and parents asset type like the typical flex family and it does not support data inheritance.

26.3.2 Designing Page Attributes

See Section 16.1.4, "Designing Flex Attributes."

Note: Inheritance is not applicable for page assets.

Designing Page Definition

See Section 16.1.6, "Designing Parent Definition and Flex Definition Assets."

Note: The Flex Parent Definition is not supported for page assets and inheritance is not applicable.

Creating Page Attributes

See Section 16.3.6, "Step 4: Create Flex Attributes."

(Optional) Create Page Filter Assets

See Section 16.3.7, "Step 5: (Optional) Create Flex Filter Assets."

Create Page Definition Assets

See Section 16.3.9, "Step 7: Create Flex Definition Assets."

(Optional) Create Page Associations

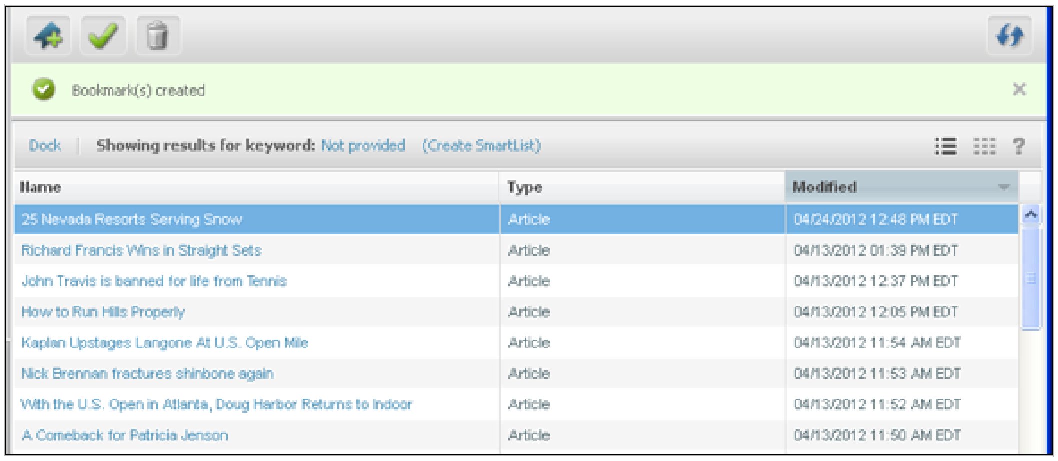
See Section 16.3.14, "Step 12 (Optional): Create Flex Asset Associations."

26.3.3 Creating a Page Asset

Follow these steps to create a page asset:

1. Log in to WebCenter Sites, select the site you want to work with and the icon for the WebCenter Sites Contributor interface.
2. Find and bookmark the assets (articles, queries, images, collections, and so on) you wish to include on the page. Do the following:
   a. In the **Search** field, enter criteria identifying the desired asset(s) and then click the magnifying glass button. A search tab opens displaying the results of your search.
   b. In the search results list, select (Ctrl+click) the assets you want to bookmark.
   c. In the search tab's toolbar, click the **Bookmark** icon.
   d. Repeat this step until you have bookmarked all of the desired assets and then continue with the next step.

A tab opens displaying the results of your search:



Bookmark(s) created

Dock | **Showing results for keyword:** Not provided (Create SmartList)

Name	Type	Modified
25 Nevada Resorts Serving Snow	Article	04/24/2012 12:48 PM EDT
Richard Francis Wins in Straight Sets	Article	04/13/2012 01:39 PM EDT
John Travis is banned for life from Tennis	Article	04/13/2012 12:37 PM EDT
How to Run Hills Properly	Article	04/13/2012 12:05 PM EDT
Kaplan Upstages Langone At U.S. Open Mile	Article	04/13/2012 11:54 AM EDT
Nick Brennan fractures shinbone again	Article	04/13/2012 11:53 AM EDT
With the U.S. Open in Atlanta, Doug Harbor Returns to Indoor	Article	04/13/2012 11:52 AM EDT
A Comeback for Patricia Jenson	Article	04/13/2012 11:50 AM EDT

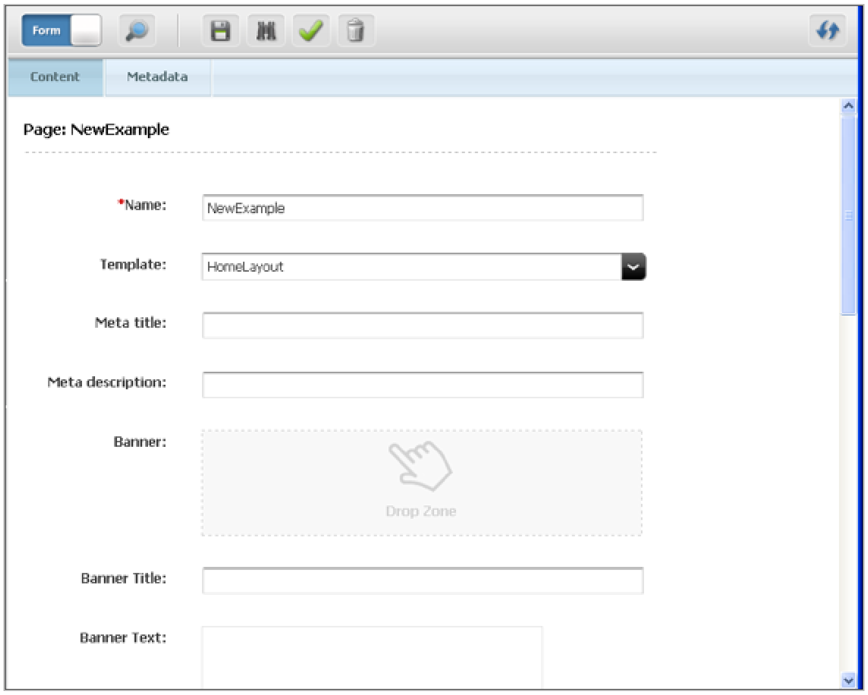
WebCenter Sites displays a confirmation message and also lists the bookmarked assets under the **Bookmarks** node in the **My Work** tree.

3. In the menu bar, select **Content**, then **New** , and then ***Page Asset***. In this example, we use the Page (Home) asset type in the avisports sample site.

Note: If you are using the avisports sample site, assets of type Page are configured to open in Web Mode. Therefore, when you select to create a "Page (Home)" asset, the "Create Page (Home)" dialog box is displayed. To follow this procedure (and create a Page (Home) asset in Form Mode):

In the "Create Page (Home)" dialog box, do the following:

1. In the **Select Layout** field, select the layout you wish to assign to the page.
2. In the **Name** field, enter a name for the page. For information about naming the page, see step 4 substep a. Then click **Continue**.
3. In the avisports sample site the page definition is chosen when you select the type of page asset you will be creating. However, if this is not the case in the site you are working with (for example, FirstSiteII), you will see a Page Definition field. Use this field to specify the page definition for the page you are creating and then click **Continue**.
4. Switch to Form Mode. In the asset's toolbar, click the **Mode** switch.
5. Continue to step 4 substep b.
6. The asset's Create view is displayed in Form Mode.



Page: NewExample

*Name: NewExample

Template: HomeLayout

Meta title:

Meta description:

Banner:

Banner Title:

Banner Text:

4. Create the Page asset.

   a. In the **Name** field, enter a name for the page. type a descriptive name for the page. You can enter up to 64 alphanumeric characters, but the first character must be a letter. Underscores (_) and hyphens (-) are acceptable, but tab and space characters are not.

   b. **Page Definition** (if applicable): In the avisports sample site the page definition is chosen when you select the type of page asset you will be creating. However, if this is not the case in the site you are working with (for example, FirstSiteII), you will see a **Page Definition** field. Use this field to specify the page definition for the page you are creating and then click **Continue**.

   c. In the **Template** field, select a template from the drop-down list.

   d. To add items, select the desired assets from the **Bookmarks** node in the My Work tree and then drag and drop the selected items into the desired field.

5. In the asset's toolbar, click the **Save** icon.

   The page is saved. It now appears in the **Site Tree** under the **Unplaced Pages** pages node. To place the page, see Section 26.3.4, "Placing Page Assets."

26.3.4 Placing Page Assets

After you create a page asset, you position it in the appropriate location in the site tree by using the **Place** function.

To place a page asset

1. Log in to WebCenter Sites as an administrator.

2. Select the site in which you want to work.

3. Select the **Admin** interface.

4. Click the **Site Plan** tab, where you should see the site tree with the new page asset in the **Unplaced Pages** list, as shown here:

5. Expand the list of **Placed Pages** in the site tree.
6. Select a parent for the page you are placing by doing one of the following:
   - If you want to place a page at the top-most level in the tree, right-click on the **Placed Pages** icon.
   - Otherwise, right-click on the placed page under which you want to insert the new unplaced page, and choose **Place Page** from the context menu.

   The place page form appears in the work area on the right. It lists all child pages that are placed under the parent page. It also lists all pages that have not yet been placed in the site tree:
7. Place the page:

   In the list of unplaced pages, type a number in the Rank field to designate the new page's position in the list of child page assets. Position numbering starts at 1, the top of the list. (In this example, it makes sense to type 1.)
8. Click **Save**.

   The unplaced page asset moves to the site tree, to its assigned rank. (To view the page asset in its new location, you may need to right-click in the site tree and choose **Refresh All** from the context menu.)
9. Preview both the parent page and the placed child page. See Section 26.1, "Collection Assets" for instructions.

26.3.5 Moving Page Assets in the Site Tree

In addition to placing unplaced pages, you can also use the place page form to:

- Change the order of child pages within the same parent page.
- Move a child page from one parent page to another.

Re-ordering Child Pages

To re-order children of the same parent page:

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Click the **Site Plan** tab and expand the list of **Placed Pages** in the site tree.
5. Right-click on a placed page that has more than one child page, and choose **Place Page** from the context menu.

   The place page form appears in the work area on the right.
6. In the list of placed child pages, type new values in the **Rank** column to re-order the child pages.
7. Click **Save**.

   The child pages move to their new positions in the site tree.

Changing Parent Pages

To move a child page from one parent page to another:

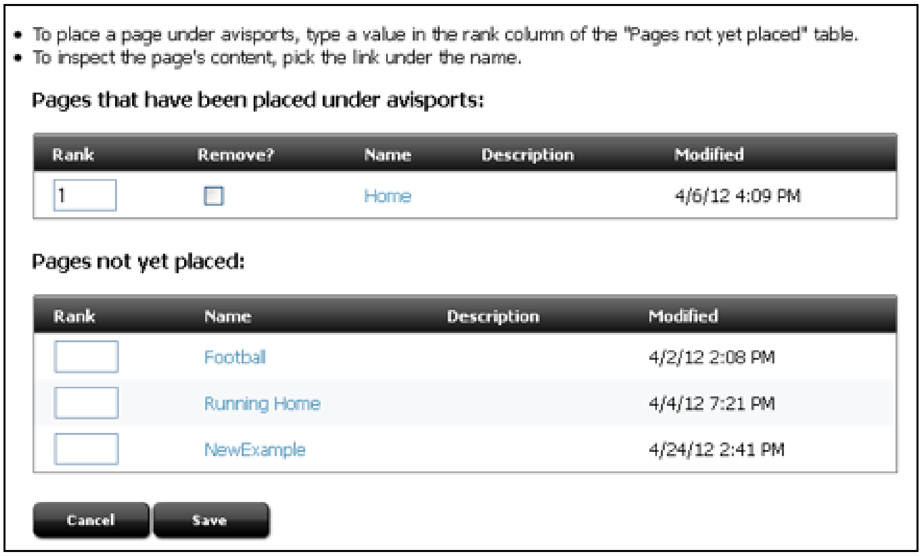
1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.

3. Select the **Admin** interface.
4. Click the **Site Plan** tab and expand the list of **Placed Pages** in the site tree.
5. Remove the page asset from its parent page:

   a. Right-click on the placed page whose child page you want to move, and choose Place Page from the context menu.

      The place page form appears in the work area on the right.

   b. In the list of placed child pages, select the **Remove** check box next to the child page that you want to move.



- To place a page under avisports, type a value in the rank column of the "Pages not yet placed" table.
- To inspect the page's content, pick the link under the name.

Pages that have been placed under avisports:

Rank	Remove?	Name	Description	Modified
1	□	Home		4/6/12 4:09 PM

Pages not yet placed:

Rank	Name	Description	Modified
	Football		4/2/12 2:08 PM
	Running Home		4/4/12 7:21 PM
	NewExample		4/24/12 2:41 PM

   c. Click **Save**.

      The child page moves to the list of **Unplaced Pages** in the site tree.

6. Place the page asset under its new parent page:

   a. In the site tree, right-click on the placed page where you want to insert the unplaced child page, and choose Place Page from the context menu.

      The place page form appears in the work area on the right.

   b. In the list of unplaced pages, type a number in the **Rank** field to designate the new page's position in the list of child page assets. Position numbering starts at 1, the top of the list. (In this example, it makes sense to type 1.)

   c. Click **Save**.

      The previously unplaced page asset moves to the site tree, to its assigned rank.

26.3.6 Placing Page Assets and Workflow

WebCenter Sites has a workflow feature that controls the flow of assets as they pass from one team member to another; for example, from author to editor to approver to publisher. The workflow administrator can create processes that control who can place page assets in the site tree and during which workflow step they can do so. Note the following:

- The **Place Page** workflow privilege controls all place page functions: **Place Pages**, **Remove**, and **Rank**.

- You must have the proper privileges for both the parent page on which you invoke **Place Pages**, and for any child page that you want to **Rank** or **Remove**.

For information about the workflow process, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

26.3.7 Editing Page Assets

In general, there are two ways to edit an existing page asset:

- Change the assets, but not the asset types, that are included on the page. For example, move new assets to the **Contains** list from your Bookmarks; select a different collection, query, or article from a named association field; or rebuild a collection already associated with the page asset to include different assets.
- Create a new association or change the actual structure of the page asset in some way.

Although you may frequently change the content in the collections or queries on a page at regular intervals, you are less likely to change the associations, asset types, or structure of a page after the site goes live. This may also require you to edit the code in the template element that formats the page.

26.3.8 Deleting Page Assets

During your site design phase, it is likely that you will create and delete many page assets. However, before deleting a page asset from a site that you have published, be sure that you understand the consequences. For example:

- Have you removed references to the page from other page assets?
- Are any of your other page templates coded to extract and use information about this page asset in any way?

Before you delete a page asset, be sure to remove any references to it from any other elements or pages. It is a good idea to unplace a page asset before you delete it.

27

Best Practices for Creating Future Site Preview Assets and Templates

To successfully implement Future Site Preview functionality, content creators must understand how to create sets of time-sensitive assets. Developers must also include the tag that filters assets by date in templates that will display those assets.

This chapter contains the following sections:

- Section 27.1, "Considerations when Implementing Future Site Preview"
- Section 27.2, "Creating Sets of Assets"
- Section 27.3, "Writing Templates for Future Site Preview"
- Section 27.4, "Caching Considerations"

27.1 Considerations when Implementing Future Site Preview

Using the Future Site Preview feature, content contributors can preview assets in the Contributor interface as they will appear at a future time. This functionality utilizes asset start date and end date attributes. These two attributes determine the date range that assets are available on the website. Content creators can specify start dates and end dates in Edit screens in the Contributor interface.

The following requirements must be met in order to properly implement the Future Site Preview feature:

- Content contributors must create an appropriate set of assets. See Section 27.2, "Creating Sets of Assets" for more information.
- The administrator must update the templates that render the assets to include the tag `asset:filterassetsbydate`. See Section 27.3, "Writing Templates for Future Site Preview" for more information.

27.2 Creating Sets of Assets

When different versions of the same item will be displayed on different dates, content creators need to create multiple assets.

For example, suppose you are creating a page in a website. You want to run a special version of the page for a one-day New Year's Day sale event. You would need three separate assets to accomplish this task. Here is how you would create the one-day sale:

- Create the regular page asset, setting its end date to the date before the sale day, December 31st 23:59:59.

- Create the sales event asset, setting the start date to the beginning of the sale day, January 1st 00:00:00, and the end date to the end of the sale day, January 1st 23:59:59.
- Duplicate the regular page asset, setting the start date to the day after the sale, January 2nd 00:00:00.

After the three assets are created, they can be passed as input to the `asset:filterassetsbydate` tag (see Section 27.3, "Writing Templates for Future Site Preview" for more information), which will return the asset to render based on the given date.

For ease of use in searching for the related group of assets for editing, publishing, and filtering in templates, we recommend that developers designate an attribute (such as `name`) which content contributors fill in using a naming convention when creating related sets of time-sensitive assets.

27.3 Writing Templates for Future Site Preview

Appropriate usage of the `asset:filterassetsbydate` tag is critical to Future Site Preview functionality. This tag filters assets according to a given date. The proper use of this tag is discussed below. More information about the `asset:filterassetsbydate` tag can be found in the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

This section contains the following topics:

- Section 27.3.1, "The asset:filterassetsbydate tag"
- Section 27.3.2, "The Input List"

27.3.1 The asset:filterassetsbydate tag

The filter tag has the following format:

```
<asset:filterassetsbydate
inputList="inputListName"
outputList="outputListName"
[date="date value in either yyyy-MM-dd HH:mm:ss OR yyyy-MM-dd format"]>
```

Notice that the filter tag has two input attributes (`inputList` and `date`) and one output attribute (`outputList`):

- The `inputList` attribute specifies the list of assets to be filtered based on the given date.
- The `date` attribute is optional. The `date` attribute is expected to be in either `yyyy-MM-dd HH:mm:ss` or `yyyy-MM-dd` format. The `date` attribute should be coded to accept the date value passed from the date picker in the preview screen (use the `__insiteDate` variable for this).
- The tag produces an output list (`outputList`) which contains assets whose start/end dates enclose the given date.

The tag performs the following steps:

1. Checks for the `cs.sitepreview` property to determine if the template is stored on a content management system.
2. If Future Site Preview is turned off, the system date passes into the date field.
3. If Future Site Preview is turned on, the tag routine:

- Disables caching of the current page being rendered (see Caching Considerations for more on caching).
- Accepts the date parameter (passed from the date picker).
- Checks for the appropriate format of the date passed into the tag.
- Filters the input list of assets based on the given date to produce the output list.

27.3.2 The Input List

The asset:filterassetsbydate tag requires the input list to contain two columns named assetid and assettype. This input list can be constructed in a variety of ways.

The simplest way to build the input list, and the way we recommend, is for content contributors to follow a naming convention when filling in the attribute of your choice (usually the name attribute). The list-building code could then be written using the asset:search tag or other similar tags to search that attribute for the agreed upon string and construct the list from the search results.

Another method for locating assets to place into the input list is to use the Recommendation asset type to hold a list of assets of interest.

Whichever method you use to determine the assets that need to be filtered, use the listobject tag to construct the input list for the filter tag, as shown in the sample code below.

This sample code creates a list object inputListName and adds a row containing two columns: assetid and assettype. The listobject:tolist then creates the input list called assetInputList. This list is now ready to be passed as input to the filter tag.

```
<listobject:create name="inputListName" columns="assetid,assettype" />
<listobject:addrow name="inputListName">
<listobject:argument name="assetid" value='<%=ics.GetVar("assetIdVar")%>' />
<listobject:argument name="assettype" value='<%=ics.GetVar("assetTypeVar")%>' />
</listobject:addrow>
<listobject:tolist name="inputListName" listvarname="assetInputList" />
```

Note that for the sake of simplicity, the code snippet explains the creation of an input list containing only one row. In practice users typically have multiple rows (usually read off from the results of asset:search tag or some other list) added to the list with each row representing an asset that needs to be filtered by a given date.

After creating the input list of assets to be filtered, use the asset:filterassetsbydate tag as follows:

```
<asset:filterassetsbydate inputList="assetInputList" outputList="assetOutputList"
date='<%=ics.GetVar("dateValueVariable")%>' />
```

To pass input from the Future Site Preview date picker to the date attribute, replace the generic dateValueVariable with _insiteDate.

The tag produces an output list assetOutputList. Read through the list for assets that clear the filter by date test as follows:

```
<ics:if condition='<%=ics.GetList("assetOutputList")!=null &&
ics.GetList("assetOutputList").hasData()'%>
<ics:then>
<ics:listloop listname=assetOutputList>
<ics:listget listname=assetOutputList fieldname=assetid output=id />
<ics:listget listname=assetOutputList fieldname=assettype output=type/>
```

```
<!--
Perform your usual asset load, asset get and other rendering functions using
WebCenter Sites tags here
-->
</ics:listloop>
</ics:then>
</ics:if>
```

27.4 Caching Considerations

In a delivery system, in order for web pages to be displayed appropriately for the current date, asset entries must be removed from the cache at the proper times. Therefore, WebCenter Sites includes start and end dates in the factors it uses to calculate the expiry time for cached pages.

Note: WebCenter Sites does not support the use of start and end dates with Export to Disk publishing. When assets are exported to disk, start and end date attributes are also exported to disk. However, the Export to Disk publishing method has no mechanism similar to the cache cleaning process (which, in other publishing methods, automatically removes expired assets from disk).

On a content management system, when a page is previewed, the `asset:filterassetsbydate` tag disables page caching for that page. This ensures that the page being served always displays assets filtered by the date being passed from the future preview date picker, rather than serving pages from the page cache, which may have been generated using different date input.

28

Coding Elements for Templates and CSElements

Elements provide the code that identifies, extracts, and displays your content. In a WebCenter Sites system, your content is stored as assets. Therefore, much of the XML or JSP code in your elements is dedicated to identifying the appropriate asset for the appropriate context and then extracting and displaying that asset's data.

This chapter describes how to code the elements that you make for your template and CSElement assets. For information about creating the assets themselves, see Chapter 23, "Creating Template, CSElement, and SiteEntry Assets."

This chapter contains the following sections:

28.1 About Dependencies

Your WebCenter Sites system tracks and relies on two kinds of dependencies to function correctly:

- **Approval dependencies**. These are conditions that determine whether an approved asset can be published.

  The approval system calculates the approval dependencies for an asset when it is approved. If there are dependent assets that also need to be approved, the parent asset will not be published.

- **Compositional dependencies**, that is, page composition dependencies. These are dependencies between assets and the pages and pagelets that they are rendered on that determine whether a page needs to be regenerated.

  The ContentServer servlet logs compositional dependencies when it renders pages. CacheManager consults the dependency log to determine when to regenerate the cached pages. The Export to Disk publishing method consults the dependency logs to determine when an exported page file must be regenerated.

> **Note:** One of your responsibilities while coding elements is to ensure that your code logs compositional dependencies accurately, and, if you are designing a static site, that it sets approval dependencies appropriately, as well.

This section contains the following topics:

- Section 28.1.1, "The Publishing System and Approval Dependencies"
- Section 28.1.2, "Page Generation and Compositional Dependencies"

28.1.1 The Publishing System and Approval Dependencies

The publishers, editors, and content providers who work on your management system **approve** assets to be published to a target destination. The publishing system then publishes the approved assets automatically, as a background process, according to the schedule that your administration team set up for your WebCenter Sites system.

An asset can be published only if it meets all specified approval dependencies, that is, all associated assets must have been either approved or previously published. If not, the asset is **held** from being published until the dependencies are met: the dependent (related) assets must themselves be approved for publishing to the same destination.

This approval process frees your content and editorial team from the responsibility of manually checking asset dependencies and then publishing a large number of related assets. It also ensures that there can be no broken links on your online site after assets are published.

If an asset is subsequently changed, the asset is no longer considered to be approved, and it must be approved again before it can be re-published.

This section contains the following topics:

- Section 28.1.1.1, "Calculating Approval Dependencies"
- Section 28.1.1.2, "Exists vs. Exact vs. None"
- Section 28.1.1.3, "Approval Templates for Export to Disk"
- Section 28.1.1.4, "Subtypes, Flex Definitions, and Approval Templates"

28.1.1.1 Calculating Approval Dependencies

Approval dependencies are recorded at the time the asset is approved. They are written to the `ApprovedAssetDeps` table in the WebCenter Sites database.

The approval status of an asset is determined by its dependency relationships, which include the approval status of all asset items associated with a particular asset item, as well as the dependency relationships of those associated items.

What is the basis for the dependency calculation? That depends on the publishing method:

- For **Export to Disk**, the approval system renders the asset using either the template that is assigned to it or, if there is one specified, the default approval templates for assets of this type. The tags in the template code set approval dependencies that determine the appropriate dependents for the approved asset.

The dependent assets must be in an appropriate approval state before the current asset can be published.

- For **Mirror to Server** or **Export Assets to XML**, the approval process examines the data relationships between asset types. Basic assets have associations. Flex assets have family relationships. Both of these relationships create approval dependencies for these publishing methods. For example, if you approve a flex asset, it will be held from a publishing session unless its parent assets are in an appropriate approval state.

28.1.1.2 Exists vs. Exact vs. None

Approval dependencies can be **exists**, **exact**, and **none**. This section defines each kind of approval type.

You cannot change the approval dependency type for CSElements and SiteEntry assets, embedded links and pagelets, or the Engage visitor data assets. With the exception of flex attributes, whose dependency type you set when you create the attribute, you also cannot change the approval dependency type for the flex family asset types. For basic asset types, you set the type of approval dependency for their associated assets when you configure the association fields.

When your publishing method is Export to Disk, the tags that set compositional dependencies when pages are rendered also create approval dependencies when the approval system calculates whether an asset can be published. When your code sets approval dependencies on pagelets generated for other assets, you can set the approval type to exists, exact, or none.

> **Note:** For information about the types of approval dependencies created by the relationships between assets of the various types, see the publishing chapter in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

Exists

With an **exists** dependency, the dependent asset must merely exist on the target, the version of the asset does not matter. An **exists** dependency means that an approved parent asset can be published even if a child asset changes (which means that the child asset is no longer approved), as long as the child asset was previously approved and published to that same destination.

For example, in the following sequence, a collection asset has an **exists** relationship with its ranked children:

1. A collection and all of its ranked articles are approved and published to a target.
2. One of the ranked articles is edited again, but not approved.
3. The collection itself is edited again, approved, and published to the destination.

   The collection is not held back from publishing by the changed but unapproved article, because a prior version of the article already exists.

However, in the following example, a collection with an **exists** dependency relationship to its articles cannot be published:

1. A collection and all of its ranked articles are approved but not published.
2. One of the ranked articles is edited again.

Because the edited article was never published to the destination, it does not yet exist for that destination, which means that the collection cannot be published. The collection asset is **held** and both the collection and the edited article must be approved before the collection can be published.

The exists approval type is generally useful for links.

Exact

With an **exact** dependency, the dependent asset must be the exact version on the target. No other previously approved version will do. An **exact** dependency means that the parent asset cannot be published if the version of the parent and child assets on the destination do not match.

In the following example, a page asset has an **exact** dependency with its article assets:

1. A page asset and all of its article assets are approved and published to a destination.
2. One of the articles is edited again, but is not re-approved.
3. The page asset is edited and is re-approved.

   The page asset is held, and the resulting form in the WebCenter Sites interface displays a link that points to a list of the assets that must be approved before the page asset can be published. This list shows the article that was edited but not re-approved.
4. The edited article is approved.

   The page asset has already been approved and can now be published because the version stamps of the article and the page asset match.
5. Another article asset associated with the page asset is edited.
6. Both the page asset and the edited article asset must be re-approved because the version stamps of the two do not match:
   - The article must be re-approved because it was edited but not yet re-approved.
   - The page asset must be re-approved because it was previously approved with a dependency on a different version of the article.

The exact approval type is generally useful for embedded content.

None

A **none** dependency means that the approved asset can be published no matter what approval state the dependent asset is in. You can set the approval dependency type to `none` by adding the `DEPTYPE` parameter to a tag that sets an approval dependency and setting that parameter to `none`.

Note that setting `DEPTYPE` to `none` effects the approval dependency only. When the Export to Disk process generates the page and invokes the tag, a compositional dependency is logged. But when the approval system invokes the tag during its calculation, no approval dependency is logged.

28.1.1.3 Approval Templates for Export to Disk

When assets are approved for a publishing destination that uses the Export to Disk publishing method, the approval system examines the template assigned to the asset to determine its dependencies.

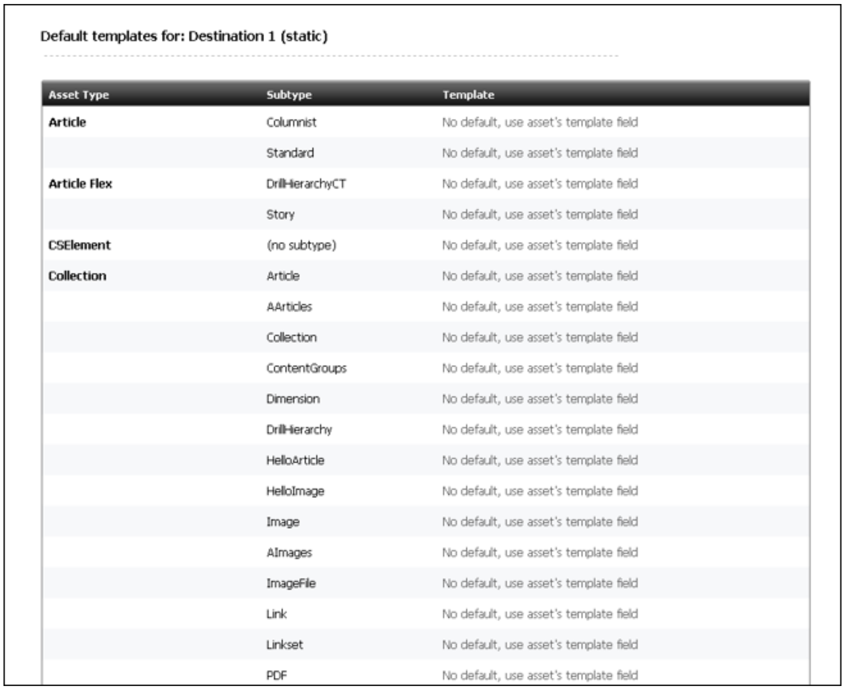
However, when Export to Disk actually publishes the asset, it does not necessarily use the template that is assigned to the asset. Why? Because the code in another element could determine that a different template is used for that asset in certain cases.

Consider the Burlington Financial sample site. An article asset from this sample site can be rendered by several different templates, depending on the context.

So when you approve an article asset for the Burlington Financial sample site, which template should the approval process use to determine the dependencies for the article? The one that contains the most representative set of dependencies for all of the templates. For an example, see the Burlington Financial template article named Full. You may decide to create a special template that contains all the possible dependencies for assets of each type.

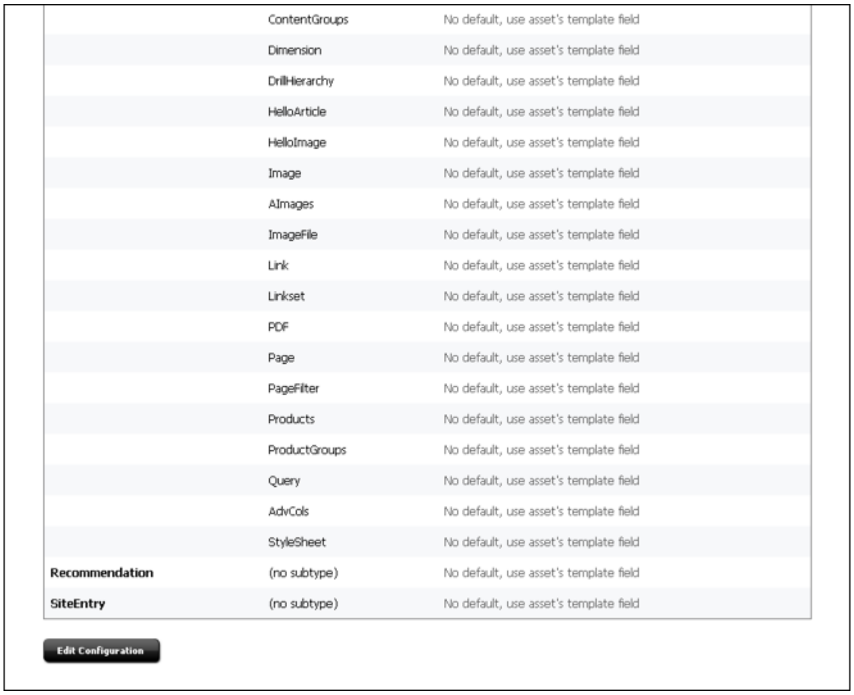
What if the template that contains the most representative set of dependencies is not the template that you want to assign to the asset? Set it as the **Default Approval Template** for assets of that type.

You can set Default Approval Templates for each asset type and for each publishing destination. This feature is located in the tree by selecting the **Admin** tab, then selecting **Publishing**, then **Destinations**, then *MyStaticDestinationName* and then **Set Default Templates**.



Default templates for: Destination 1 (static)

Asset Type	Subtype	Template
Article	Columnist	No default, use asset's template field
	Standard	No default, use asset's template field
Article Flex	DrillHierarchyCT	No default, use asset's template field
	Story	No default, use asset's template field
CSElement	(no subtype)	No default, use asset's template field
Collection	Article	No default, use asset's template field
	AArticles	No default, use asset's template field
	Collection	No default, use asset's template field
	ContentGroups	No default, use asset's template field
	Dimension	No default, use asset's template field
	DrillHierarchy	No default, use asset's template field
	HelloArticle	No default, use asset's template field
	HelloImage	No default, use asset's template field
	Image	No default, use asset's template field
	AImages	No default, use asset's template field
	ImageFile	No default, use asset's template field
	Link	No default, use asset's template field
	Linkset	No default, use asset's template field
	PDF	No default, use asset's template field



	ContentGroups	No default, use asset's template field
	Dimension	No default, use asset's template field
	DrillHierarchy	No default, use asset's template field
	HelloArticle	No default, use asset's template field
	HelloImage	No default, use asset's template field
	Image	No default, use asset's template field
	AImages	No default, use asset's template field
	ImageFile	No default, use asset's template field
	Link	No default, use asset's template field
	Linkset	No default, use asset's template field
	PDF	No default, use asset's template field
	Page	No default, use asset's template field
	PageFilter	No default, use asset's template field
	Products	No default, use asset's template field
	ProductGroups	No default, use asset's template field
	Query	No default, use asset's template field
	AdvCols	No default, use asset's template field
	StyleSheet	No default, use asset's template field
Recommendation	(no subtype)	No default, use asset's template field
SiteEntry	(no subtype)	No default, use asset's template field

Note: If you specify a default approval template for an asset type on a destination that uses the Mirror to Server publishing method, that template is used when you preview the asset on the Asset Status screen, but not when the asset is approved or published.

28.1.1.4 Subtypes, Flex Definitions, and Approval Templates

If you are using flex assets for a static site, you can assign more than one default approval template to the flex asset type in the family. You can designate a different default approval template for each flex definition.

For basic assets, the Subtype feature provides a way to further categorize assets of a single asset type. You can use this feature to assign more than one default approval template for assets of a specific type, based on some other organizing construct.

For example, perhaps the approval template for sports articles should be different than the approval template for world news articles. You can create a sports subtype and a world news subtype for the article asset type and then assign different approval templates for each subtype of the asset type.

You create subtypes for basic assets either in the asset descriptor file when you create the asset type or by using the **Asset Types** option on the **Admin** tab if you decide you need subtypes after the asset types were created. You assign a subtype to an asset by using the New and Edit asset forms. As mentioned, flex assets already have subtypes: their flex definitions.

For more information about configuring subtypes for basic assets, and about subtypes in general, see Chapter 15, "Designing Basic Asset Types."

28.1.2 Page Generation and Compositional Dependencies

Compositional dependencies are recorded in different ways:

- When the Export to Disk publishing method renders a page, it logs compositional dependencies to the appropriate publishing tables. Then, when it's time to publish again, Export to Disk can determine which pages need to be regenerated based on which assets are being published, it generates all the pages that have logged the assets as compositional dependents.
- When WebCenter Sites renders and caches a page, it logs the dependencies in the `SystemItemCache` table at the time a page is rendered and cached. Each row in this table holds the ID of an asset and the cache key or ID of the generated page that the asset was rendered on.

CacheManager and the Page Caches

The CacheManager maintains the WebCenter Sites page caches. As assets are changed, it consults the `SystemItemCache` table to determine which cached pages those assets were rendered on. Then it works through the `SystemPageCache` table, flushing and regenerating the appropriate pages.

After it makes changes to the WebCenter Sites page cache, CacheManager communicates that information to all the Satellite Servers participating in your WebCenter Sites system, the co-resident Satellite Server and any remote Satellite Servers that are installed in your system. The Satellite Server applications then update the Satellite page caches.

Note: If you have the appropriate permissions, you can examine the data in the `SystemItemCache` and `SystemPageCache` tables, but, as with any other system table in the WebCenter Sites database, do not alter the information stored in these tables in any way.

CacheManager and Dynamic Publish Sessions

The CacheManager interacts with the publishing system during Mirror to Server publishing session. When a Mirror to Server publishing session ends, the publishing system provides a list of all the IDs of all the assets that were included in the publish operation to the CacheManager servlet on the destination system.

The CacheManager compares that list to the compositional dependencies logged for the pages in the cache to determine which pages and pagelets need to be flushed from the page cache and regenerated. It updates the WebCenter Sites page cache accordingly, and then sends the list of pages to the co-resident and remote Satellite servlets so they can flush those same pages and get new versions from the WebCenter Sites page cache.

CacheManager and the Preview Function

When you preview an asset (on the development or management system), the WebCenter Sites interface executes the page name of the template for the asset. ContentServer renders the page, caches the page, and logs the compositional dependencies between the rendered page and the asset.

CacheManager updates the cached versions of previewed pages when assets are saved. That is, when someone clicks **Save**, CacheManager compares the object ID of that asset to the compositional dependencies logged for the pages in the cache. It then clears and refreshes the appropriate pages in the page cache and communicates the information about the changed pages to the Satellite servlets.

28.2 About Coding to Log Dependencies

While you are coding elements, one of your responsibilities is to include code that logs dependencies accurately.

There are several tags that log compositional dependencies. When the tag is executed, WebCenter Sites logs a dependency between the rendered page and the asset by writing this information in the `SystemItemCache` table.

Note that for a static site using the Export to Disk publishing method, the tags that log compositional dependencies can also log approval dependencies. When an asset is approved, the approval system renders that asset to determine whether it can be published. It logs the results of these tags to the `ApprovedAssetDep` table unless the tag sets the approval dependency type to none. See Section 28.1.1.2, "Exists vs. Exact vs. None" for more information about the none dependency type.

This section presents the tags that log dependencies in alphabetic order. For more information about these and any other tag, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

This section contains the following topics:

- Section 28.2.1, "ASSET.LOAD and asset:load"
- Section 28.2.2, "The ASSETSET (assetset) Tag Family"
- Section 28.2.3, "RENDER.GETPAGEURL and render:getpageurl"
- Section 28.2.4, "RENDER.LOGDEP (render:logdep)"
- Section 28.2.5, "RENDER.FILTER and render:filter"
- Section 28.2.6, "RENDER.UNKNOWNDEPS and render:unknowndeps"

28.2.1 ASSET.LOAD and asset:load

When WebCenter Sites executes an `ASSET.LOAD` tag (or `asset:load`), it automatically logs a compositional dependency for the asset that is loaded. For example:

```
<ASSET.LOAD TYPE="Page" NAME="target" FIELD="name" VALUE="Home"/>
```

That line of code marks a compositional dependency between the page asset named Home and the rendered page that is displaying this asset.

Setting the Approval Dependency Type

When an asset is approved for an Export to Disk destination and the approval system renders this tag, the tag also logs an approval dependency between the assets that are in play.

By default, the approval dependency for `ASSET.LOAD` is set to `exact`. You can set the dependency to `exists` or to `none` by using the `DEPTYPE` parameter. For example:

```
<ASSET.LOAD TYPE="Page" NAME="target" FIELD="name" VALUE="Home" DEPTYPE="exists"/>
```

28.2.2 The ASSETSET (assetset) Tag Family

You use the `ASSETSET` tag family to create a set of one or more flex assets. The following tags create assetsets and define compositional dependencies for the assets in the set:

```
ASSETSET.SETASSET and assetset:setasset
ASSETSET.SETEMPTY and assetset:setempty
ASSETSET.SETLISTEDASSETS and assetset:setlistedassets
ASSETSET.SETSEARCHEDASSETS and assetset:setsearchedassets
```

When an asset from the assetset is rendered, the compositional dependency is logged.

The first three tags define the following compositional dependencies:

- A dependency between each flex asset in the assetset and the rendered page.
- A dependency between the flex asset's parents and the rendered page. Because flex assets inherit values from their flex parent assets, a change to a parent can mean a change to the flex asset and that means the pages that hold the asset may no longer be accurate.

The fourth tag, `assetset:setsearchedassets`, creates an assetset from the results of a search state. Search states are queries, which means there is no way to predict the identities of the assets in the set. Therefore, the `ASSETSET.SETSEARCHEDASSETS` tag defines the compositional dependency as unknown. When a compositional dependency is unknown, it means the page must be regenerated during each Export to Disk publishing session and updated in the page caches after each Mirror to Server publishing session, whether it needs it or not.

If you have a search state that describes a fixed set of assets whose identities will not change, you instruct WebCenter Sites to set compositional dependencies for the assets in the assetset by setting the optional `fixedlist` property to true.

For example:

```
<assetset:setsearchedassets name=as assettypes=Products constrain=ss
fixedlist=true />
```

This example defines that there is a compositional dependency between each product asset in the assetset named as and the rendered page.

For more information about asset sets and search states, see Section 28.5.1, "Assetsets" and Section 28.5.2, "Searchstates."

Setting the Approval Dependency Type

If you are using flex assets for a static site, be aware that when the approval system invokes an assetset tag, the approval dependency type is set to `none` by default. To change this value to exists or exact, you use the `deptype` parameter.

For example:

```
<assetset:setsearchedassets name=as assettypes=Products constrain=ss
fixedlist=true deptype=exists />
```

Note that setting an approval type for the `assetset:setsearchedassets` tag is meaningful only if the `fixedlist` parameter is set to `true`.

28.2.3 RENDER.GETPAGEURL and render:getpageurl

The `RENDER.GETPAGEURL` tag creates a URL for assets that are not blobs. This tag logs an exists *approval* dependency, but not a compositional dependency, between the asset being approved (rendered) and the asset referred to by the tag. This means that it creates a dependency only when your publishing method is Export to Disk.

In this example, the template assigned to article ABC has the following code in it:

```
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Page/Home"
cid="Variables.pageid"
c="Page"
OUTSTR="referURL"/>
```

That code fragment both creates a URL (that is returned in the variable created by the `OUTSTR` parameter) and logs an **exists** approval dependency between the asset identified in the `cid` variable and article ABC.

Then, when article ABC is approved, the page identified by the `cid` variable must either be approved or must already have been published or article ABC is held from being published.

28.2.4 RENDER.LOGDEP (render:logdep)

There are several situations in which your code can obtain an asset's data without actually loading the asset. When this is the case, be sure to log the compositional dependency yourself with the `render:logdep` tag.

Example 1

When you call a CSElement from a Template asset or other CSElement asset, you do not load the asset to determine the identity of the element file to execute. Instead, you use the `RENDER.CALLELEMENT` or `render:callelement` tag and invoke the element directly by name. For example:

```
<render:callelement name=BurlingtonFinancial/Common/HeaderText/>
```

Because you didn't use the `asset:load` tag to access the CSElement, the compositional dependency between the CSElement asset and the page it is being rendered on is not automatically logged for you. Instead, you must set it yourself.

At the beginning of the element for each CSElement asset, you include the following line of code:

```
<render:logdep cid="Variables.eid" c="CSElement"/>
```

At the beginning of the element for a Template asset, the `render.logdep` statement would be as follows:

```
<render:logdep cid="Variables.tid" c="template"/>
```

Note that if you use the CSElement form or the template form in the WebCenter Sites interface to start coding the element, WebCenter Sites automatically includes an appropriate `render:logdep` statement in the stub code that it seeds into the element for you.

Example 2

For basic assets, when you use an `ASSET.LOAD` tag on a parent asset (basic asset) and then use an `ASSET.CHILDREN` tag, you have access to the children assets' data without having to load it. In this case, you should include a `RENDER.LOGDEP` statement to log the compositional dependency.

For example:

```
<ASSET.CHILDREN NAME="PlainListCollection" LIST="theArticles"
OBJECTTYPE="Article" ORDER="nrank" CODE=-/>
<LOOP LIST="theArticles">
<RENDER.LOGDEP cid="theArticles.id" c="Article"/>
...
```

Setting the Approval Dependency Type

When an asset is approved for an Export to Disk destination and the approval system invokes this tag, the tag also creates an exact approval dependency between the asset and the rendered page.

You can change the approval dependency type to exists or none by setting the `DEPTYPE` argument. For example:

```
<RENDER.LOGDEP cid="theArticles.id" c="Article" DEPTYPE="exists"/>
```

28.2.5 RENDER.FILTER and render:filter

You use the `RENDER.FILTER` tag for lists of assets created by queries. This tag filters out any unapproved assets from a list or a query. It also sets a compositional dependency of unknown. (The unknown compositional dependency is explained in Section 28.2.6, "RENDER.UNKNOWNDEPS and render:unknowndeps."

You use this tag when you do not want an approved asset that has an approval dependency on the results of a query (a collection or query asset, for example) to be held from being published when there are unapproved assets in the list that is returned by the query. For example, say that the element is coded to provide appropriate formatting for any number of article assets that are passed to it so it doesn't matter if only two of the five articles included in a collection cannot be published. Because this tag tells Export to Disk to filter out the unapproved assets, a page using the query can be published while the unapproved assets remain unpublished.

You might use this tag in the following places:

- Templates for query assets
- Templates for collection assets
- `SELECTTO` statements and `EXECSQL` queries

For example:

```
<RENDER.FILTER LIST="ArticlesFromWireQuery" LISTVARNAME="ArticlesFromWireQuery"
LISTIDCOL="id"/>
```

28.2.6 RENDER.UNKNOWNDEPS and render:unknowndeps

The `RENDER.UNKNOWNDEPS` tag signals that there are dependent assets but that there is no way to predict the identities of those assets because they came from a query or change frequently. This tag logs a compositional dependency of unknown for the rendered page. This tag does not set an approval dependency for the Export to Disk publishing method.

When a compositional dependency is set to unknown, it means the page must be regenerated during each Export to Disk publishing session and updated in the page caches after each Mirror to Server publishing session, whether it needs it or not.

> **Note:** You must use this tag carefully because the more pages that must be regenerated, the longer it takes to publish your site.

You use this tag to cover those coding situations in which you truly cannot determine what the dependent assets might be. For example, queries are dynamic and can retrieve a different resultset every time they are run. When you use queries of any kind, query assets, SELECTTO statements, EXECSQL, and so on, you should use the RENDER.UNKNOWNDEPS tag.

28.3 Calling CSElement and SiteEntry Assets

When your design requires that your code call a CSElement or SiteEntry asset, there is no need to load the asset itself. From a coding point of view, you are not interested in the CSElement or SiteEntry as an asset, you are solely interested in the element or page entry that the asset represents. Therefore, your code can directly invoke the element or page entry with the appropriate tag.

If a CSElement does not have a corresponding SiteEntry asset (which means its output is cached according to the cache criteria set for the calling page), or, if you don't need a separate pagelet at this invocation, you invoke it by name with the RENDER.CALLELEMENT (render:callelement) tag. For example:

```
<render:callelement name="BurlingtonFinancial/Common/SetHTMLHeader"/>
```

When CSElement does have a corresponding SiteEntry asset, you invoke the element by calling the page name of its SiteEntry asset with the RENDER.SATELLITEPAGE (render:satellitepage) tag. For example:

```
<render:satellitepage pagename="BurlingtonFinancial/Pagelet/Common/SiteBanner"/>
```

Note: When you use Sites Explorer to examine SiteCatalog and ElementCatalog entries, they are presented as folders and subfolders that visually organize the pages and pagelets. However, these entries are simply rows in a database table, there is no actual hierarchy. Therefore your code must always call a page entry or an element entry by its entire name. You cannot use a relative path.

Additionally, the chain of called elements should not be more than 20 levels deep. Otherwise, the system will perform poorly when displaying the assets.

Also, if you edit using Sites Explorer, save the asset in the asset's editorial form (in the WebCenter Sites interface) to ensure that the cache is updated to reflect your edits. (Sites Explorer does not automatically update the cache.)

28.4 Coding Elements to Display Basic Assets

To code an element for a template, you need to understand how the asset type it formats is designed. Having a preliminary understanding of data design and site design will prevent you from having to re-code templates in order to re-display assets when they are updated. (That is why this developer's guide covers both data design and site design in the same book.)

For example, the Home page asset for the Burlington Financial site has four collections and one query assigned to it through named associations:

- TopStory collection
- SideBarTop collection
- SideBarMiddle collection

- SideBarBottom collection
- Wirefeed Query

The Home page asset has a template, also called Home. The Home template is coded to identify the collections and the query related to the Home page through these named associations and to display the assets in the collection and the assets returned by the query.

Because the Home template is coded to handle any collection or query that is associated with the Home page through these associations (rather than hard-coded to extract specific articles), the assets that are displayed on the page can be updated as often as necessary but the code does not need to be changed.

Content providers can change the articles in the collections, and the wire feed service can make daily updates to the articles that the Wirefeed Query obtains. And no matter which articles are selected in the collections or returned by the query, they are always formatted in the same way.

This section provides:

- Information that you should keep in mind while you code templates for basic asset types.
- Code fragments and examples for various situations, including managing the dependencies between assets so that approval can be calculated correctly for static sites and so that the page cache can be cleared when appropriate for dynamic sites.

Before you begin, be sure to read the chapters in the Programming Basics section of this book, especially Chapter 4, "Programming with Oracle WebCenter Sites."

For information about the tags used in the code examples in this chapter, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

For more code samples that display basic assets, see Chapter 29, "Template Element Examples for Basic Assets."

This section contains the following topics:

- Section 28.4.1, "Assets That Represent Simple Content"
- Section 28.4.2, "Associations"
- Section 28.4.3, "ImageFile Assets or Other Blob Assets"
- Section 28.4.4, "Basic Assets That Can Have Embedded Links"
- Section 28.4.5, "Collections"
- Section 28.4.6, "Query Assets"
- Section 28.4.7, "Page Assets"

28.4.1 Assets That Represent Simple Content

Template elements for content assets generally extract one specific article, advertising copy, special offer, image, and so on from the database, then obtain information from the relevant fields such as headline, body, and byline (for example), and then display that information online.

Consider the following simple template element designed for a Burlington Financial article asset:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
```

```
<FTCS Version="1.1">
<!-- Article/VeryBasic
-
- INPUT
- Variables.c - asset type (Article)
- Variables.cid - id of the asset to display
- Variables.tid - template used to display the page(let)
- OUTPUT
-
-->
<!-- log the template as a dependent of the pagelet being rendered, so changes to
the template will force regeneration of the page(let) -->

<IF COND="IsVariable.tid=true">
<THEN>
<RENDER.LOGDEP cid="Variables.tid" c="Template"/>
</THEN>
</IF>
<!-- asset load will mark the asset as an 'exact' dependent of the pagelet being
rendered -->

<ASSET.LOAD NAME="anAsset" TYPE="Variables.c" OBJECTID="Variables.cid"/>

<!-- get all the primary table fields of the asset -->

<ASSET.SCATTER NAME="anAsset" PREFIX="asset"/>

<!-- display the description -->
<ics.getvar name=asset:description/>

<!-- display the contents of the urlbody file -->

<ics.getvar name="asset:urlbody" encoding="default"
output="bodyvar"/>
<RENDER.STREAM VARIABLE="bodyvar" /><br/>

</FTCS>
```

This code in this template does the following things:

- Logs a compositional dependency between the Template asset and the page being rendered with the element with the `RENDER.LOGDEP` tag.
- If the approval system is evaluating this code for an Export to Disk target, logs an approval dependency.
- Loads the article asset with an `ASSET.LOAD` tag, which logs a compositional dependency between the article asset and the page being rendered.
- Extracts all the values from all the fields of the article with an `ASSET.SCATTER` tag.
- Displays the contents of the description column with a `CSVAR` tag. The description column corresponds to the **Headline** field in the **New** or **Edit** article forms in the WebCenter Sites interface.
- Displays the contents of the urlbody column with the `ics.getvar` and `RENDER.STREAM` tags. The urlbody column corresponds to the **Headline** field in the **New** or **Edit** article forms in the WebCenter Sites interface.

Notice the difference in the code that displays the value from `description` column and the code that displays the value from the `urlbody` column. The `urlbody` column can contain embedded links and whenever a field can contain embedded links, you ensure that the links are rendered correctly by using the RENDER.STREAM tag rather than the CSVAR tag.

For a more complex example of an article template, examine the Burlington Financial template named Full. You can examine it in two ways:

- Search for and then inspect it in the WebCenter Sites interface.
- Use Oracle WebCenter Sites Explorer to open the template element called:

ElementCatalog/BurlingtonFinancial/Article/Full.
This template element provides the format for an article when it is displayed, in full, on a page in a browser.

28.4.2 Associations

You identify the assets that are associated with other assets through association fields with the ASSET.CHILDREN tag. To specify which associated asset, you use the CODE parameter to specify the association field.

For example, say that the following code fragment is inserted right before the </FTCS> tag in the preceding example:

```
<!-- display the Main Image -->
<ASSET.CHILDREN NAME="anAsset" LIST="associatedImage"
CODE="MainImage"/>
<IF COND="IsList.associatedImage=true">
<THEN>
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/ImageFile/TeaserSummary" ARGS_
cid="associatedImage.oid"/>
</THEN>
</IF>
```

The code in this fragment does the following things:

- Extracts the imagefile asset that is specified in the Main Image field for this article asset (named anAsset) with the ASSET.CHILDREN tag and the CODE parameter set to MainImage.
- Passes the identity of that imagefile to the page entry for the TeaserSummary template with the RENDER.SATELLITEPAGE tag. The page entry is identified with the PAGENAME parameter and the imagefile is identified with the ARGS_cid parameter. The TeaserSummary template than renders the imagefile into a pagelet and passes the pagelet back to this page, where it is displayed with the article.

28.4.3 ImageFile Assets or Other Blob Assets

The imagefile asset type stores uploaded image files. In other words, the imagefile asset type is a **b**inary **l**arge **obj**ect (blob), served from the WebCenter Sites database. You use the BlobServer servlet to serve and display imagefiles and other blobs.

A template element for an imagefile or other blob can use the RENDER.SATELLITEBLOB tag to create and return an HTML tag that tells the browser how to access the blob and how to format and display it. If you need a BlobServer URL only, without it being embedded in an HTML tag, you can use the RENDER.GETBLOBURL tag.

For more information about coding links to blobs, see Section 28.7, "Creating URLs for Hyperlinks."

28.4.4 Basic Assets That Can Have Embedded Links

The **Body** field of the Article asset and other assets that have fields with a data type of TEXTAREA allow editors to create embedded hyperlinks within the text field. To ensure that these links are rendered properly, use the RENDER.STREAM tag to retrieve the contents of the field, as shown in the following example:

```
<asset:load name="TestArticle" type="<%=ics.GetVar("c")%>"
 objectid='<%=ics.GetVar("cid")%>' />
<asset:scatter name="MainArticle" prefix="articleAsset" />
<!-- display the contents of the urlbody file -->

<ics:getvar name="articleAsset:urlbody" encoding="default"
 output="bodyvar"/>
<render:stream variable="bodyvar"/><br/>
```

If Web Mode is enabled on your management system, note that the insite:edit tag also manages embedded links appropriately when it retrieves the contents of a field that has embedded links in it.

28.4.5 Collections

Templates for collection assets typically extract the assets in the collection from the database with an ASSET.CHILDREN tag. For example:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="PlainListCollection"/>
<ASSET.SCATTER NAME="PlainListCollection" PREFIX="asset"/>
<ASSET.CHILDREN NAME="PlainListCollection" LIST="theArticles"
OBJECTTYPE="Article"/>
```

After the children are identified, the template code can then display parts of these assets in a list on a rendered page.

Sometimes the template for a collection is coded to handle the first item in the collection differently than the rest. You can single out the highest ranking asset in a collection by coding the element to order the items in the list according to their rank, as shown here:

```
<ASSET.CHILDREN NAME="HomePageStories" LIST="theArticles"
OBJECTTYPE="Article" ORDER="nrank"/>
```

For a longer example, examine the Burlington Financial template named MainStory List. You can examine it in two ways:

- Search for and then inspect it in the WebCenter Sites interface.
- Use Oracle WebCenter Sites Explorer to open the template element called:

  ```
  ElementCatalog/BurlingtonFinancial/Collection/MainStoryList
  ```

This template element calls two page entries for two other templates. The root element for the first of the two page entries displays the highest ranked article from the collection. The root element for the second displays the rest of the articles.

This section contains the following topics:

- Section 28.4.5.1, "Collection Templates and Approval Dependencies"
- Section 28.4.5.2, "Collection Templates and Compositional Dependencies"

28.4.5.1 Collection Templates and Approval Dependencies

When your publishing method is Export to Disk, you can use the RENDER.FILTER tag in your collection templates. This tag filters out any unapproved assets from the collection both when the approval dependencies are calculated and when the publish process renders the site.

The following code fragment, taken from the Burlington Financial StoryList template, illustrates this tag:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="StoryListCollection"/>
<ASSET.SCATTER NAME="StoryListCollection" PREFIX="asset"/>
<ASSET.CHILDREN NAME="StoryListCollection" LIST="theArticles"
ORDER="nrank" CODE="-"/>

<!-- Get only the articles that are approved for export -->

<RENDER.FILTER LIST="theArticles"
LISTVARNAME="ApprovedArticles"
LISTIDCOL="oid"/>

<!-- Display only the articles that are approved-->

<IF COND="IsList.ApprovedArticles=true">
<THEN>
<LOOP LIST="ApprovedArticles">
<RENDER.SATELLITEPAGE
PAGENAME="BurlingtonFinancial/Article/Summary"
ARGS_cid="ApprovedArticles.oid"
ARGS_p="Variables.p"/>
</LOOP>
</THEN>
</IF>
```

28.4.5.2 Collection Templates and Compositional Dependencies

In the preceding code example that illustrates the RENDER.FILTER tag, the ID of each of the child assets in the collection is passed to the Summary template.

The first line of code in the Summary template is an ASSET.LOAD statement, which means that the dependency between article asset that it loads and the page that is rendered with the Summary template is logged.

What if the code in the template for the collection also formats the child articles? In that case, you must carefully consider the code and determine whether you need to log the dependency with the RENDER.LOGDEP tag.

For example, when you use the OBJECTYPE parameter in an ASSET.CHILDREN tag, the resulting list is a join of the AssetRelationTree table and the asset table for the type specified and includes information from both tables. For example

```
<ASSET.CHILDREN NAME="StoryListCollection" LIST="theArticles"
OBJECTTYPE=Article ORDER="nrank" CODE="-"/>
```

You can then access the children asset's information without using subsequent ASSET.LOAD tags. If you do, be sure to include the RENDER.LOGDEP tag for each child so that the compositional dependencies between those assets and the rendered page can be tracked correctly.

For another example, see Section 29.2, "Example 2: Coding Links to the Article Assets in a Collection Asset."

28.4.6 Query Assets

Query assets can execute SQL code or they can run an element that contains query code. You use them in collections, on page assets, and so on:

- You build a collection by running a query in the Build Collection form and then selecting and ordering the assets you want from the resulting list. The collection is a static list of assets selected from the query's resultset.
- You select queries for a page asset either through unnamed relationships or through associations. You select queries for assets like articles through associations.

  In these cases, the page or article assets do not themselves invoke the query: you code the query template element to invoke a standard WebCenter Sites element called `OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery`. This element runs the query asset when the page asset or article asset is rendered.

Elements for query templates invoke the `ExecuteQuery` element and typically include code that loops through the items returned in the list object that the query created, extracts bits of information from those items, and then displays it.

The following example loads a query asset and passes it to the `ExecuteQuery` element:

```
<ASSET.LOAD TYPE="Query" NAME="Wirefeed" OBJECTID="Variables.id"/>
<CALLELEMENT NAME="OpenMarket/Xcelerate/AssetType/Query/
ExecuteQuery">
<ARGUMENT NAME="list" VALUE="ArticlesFromWireFeed"/>
<ARGUMENT NAME="assetname" VALUE="WireFeed"/>
<ARGUMENT NAME="ResultLimit" VALUE="-1"/>
</CALLELEMENT>
```

For a longer example, examine the Burlington Financial query template named `PlainList`. You can examine it in two ways:

- Search for and then inspect it in the WebCenter Sites interface.
- Use Oracle WebCenter Sites Explorer to open the template element called:

`ElementCatalog/BurlingtonFinancial/Query/PlainList.`
This element invokes the `ExecuteQuery` element to run the PlainListQuery query asset, filters out any unapproved asset if the publishing method is Export to Disk, and then loops through the resulting list, obtaining a dynamic URL for each item in the list and creating a hyperlink for it.

For information about hyperlinks, see Section 28.7, "Creating URLs for Hyperlinks."

Queries and Compositional Dependencies

The first line of code in the `ExecuteQuery` element is a `RENDER.UNKNOWNDEPS` tag, which alerts the Export to Disk publishing method and the CacheManager on a dynamic delivery system that the assets that will be retrieved by the query cannot be predicted and, therefore, no dependencies can be calculated and logged.

If you are using any other kind of query, for example, a `SELECTTO` statement, `CALLSQL`, or `EXECSQL`, you should include the `RENDER.UNKNOWNDEPS` tag.

Additionally, in the element that a query-generated list of assets is returned to, you must use the `RENDER.FILTER` tag if you are using the Export to Disk publishing method. For example:

```
<CALLELEMENT NAME="OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery">
<ARGUMENT NAME="list" VALUE="ArticlesFromTheQuery"/>
<ARGUMENT NAME="assetname" VALUE="PlainListQuery"/>
<ARGUMENT NAME="ResultLimit" VALUE="5"/>
</CALLELEMENT>

<!-- On export - filter out un-approved assets -->
<RENDER.FILTER LIST="ArticlesFromTheQuery" LISTVARNAME="ArticlesFromTheQuery"
LISTIDCOL="id"/>

<if COND="ArticlesFromTheQuery.#numRows!=0">
<then>
<LOOP LIST="ArticlesFromTheQuery">
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Article/
Variables.ct"
cid="ArticlesFromTheQuery.id"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
<A class="wirelink" HREF="Variables.referURL"
REPLACEALL="Variables.referURL"><ics.listget listname=ArticlesFromTheQuery
fieldname=subheadline/>
</A><P/>
```

For another example, see Section 29.4, "Example 4: Coding Templates for Query Assets."

28.4.7 Page Assets

Templates for page assets generally contain the following kinds of code:

- The framework for the page asset when it is a rendered page
- The logic for obtaining the content for the rendered page
- The logic for links to other rendered pages

The templates for content assets contain the formatting code for individual pieces of content. The page templates invoke the templates for the other assets, receive formatted assets from those template elements, and then place the formatted assets into the context of the page framework.

Following is the code for a simple template that formats a page asset:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Page/CollectionsAndQuery
-
- INPUT
- Variables.c - asset type (Page)
- Variables.cid - id of the asset to display
- Variables.tid - template used to display the page(let)
- OUTPUT
-
-->

<!-- log the template as a dependent of the pagelet being rendered, so changes to
the template will force regeneration of the page(let) -->

<IF COND="IsVariable.tid=true">
<THEN>
```

```
<RENDER.LOGDEP cid="Variables.tid" c="Template"/>
</THEN>
</IF>

<!-- asset load will mark the asset as an 'exact' dependent of the pagelet being
rendered -->

<ASSET.LOAD NAME="anAsset" TYPE="Variables.c"
OBJECTID="Variables.cid"/>

<!-- get all the primary table fields of the asset -->

<ASSET.SCATTER NAME="anAsset" PREFIX="asset"/>

<!-- get a list of id's of the child assets in the collection in order of their
rank -->

<!-- get the WireFeed query -->

<ASSET.CHILDREN NAME="HomeTextPage" LIST="WireFeedStories"
CODE="WireFeed"/>
<IF COND="IsList.WireFeedStories=true">
<THEN>
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Query/WireFeedFrontText"
cid="WireFeedStories.oid"
c="Query"
p="Variables.asset:id"
OUTSTR="referURL"/>
<P>
<A HREF="Variables.referURL"
REPLACEALL="Variables.referURL">From the Wires...</A>

</P>
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/Query/WireSummaryText"
ARGS_cid="WireFeedStories.oid"
ARGS_ct="WireStoryText"
ARGS_p="Variables.asset:id"/>
</THEN>
</IF>
</FTCS>
```

The code in this example does the following things:

- Logs a compositional dependency between the Template asset and the page being rendered with a RENDER.LOGDEP tag.
- Loads the page asset with an ASSET.LOAD tag, which logs a compositional dependency between the article asset and the page being rendered.
- Extracts the WireFeed query with an ASSET.CHILDREN tag and the CODE parameter set to WireFeed.
- Obtains a URL for a page that will display the stories from the WireFeed query with the RENDER.GETPAGEURL tag. The PAGENAME parameter specifies the page entry of the template to use to create that page and also determines part of the URL. The OUTSTR parameter creates a variable named referURL to hold the URL that RENDER.GETPAGEURL creates.
- Uses the URL from the referURL variable to build an <A HREF> link to the page.
- Passes the identity of the query asset to the page entry for the WireSummaryText template. The WireSummaryText template then creates a pagelet that displays the

summary text from each article returned by the Wire Feed query and passes the pagelet back to this page, where it is displayed.

For a more complex example of a page asset template, examine the Burlington Financial template named SectionFront. You can examine it in two ways:

- Search for and then inspect it in the WebCenter Sites interface.
- Use Oracle WebCenter Sites Explorer to open the template element called:

```
ElementCatalog/BurlingtonFinancial/Page/SectionFront
```

This element creates a Section Front page with a navigational bar on the top, a navigational area with links on the left, a list of stories, and so on.

28.5 About Coding Elements that Display Flex Assets

When you code templates for basic assets, you use the WebCenter Sites `ASSET` tag family. For example, when you want to extract and display a basic asset, you use the `ASSET.LOAD` tag, a tag that extracts data from the primary storage table for that asset type.

Because the database schema for flex assets is different than that for basic assets, WebCenter Sites provides tag families for flex assets that you use in place of the `ASSET` tags:

- `ASSETSET`. You use this tag family to specify a set of one or more flex assets.
- `SEARCHSTATE`. You use this tag family to create search constraints that filter the assets in an assetset.

> **Note:** The `ASSET.LOAD` tag will load a flex asset for you. However, using the `ASSET.LOAD` tag with flex assets is not supported: the code cannot be upgraded, and extracting the asset in this way is slower by orders of magnitude than using the `ASSETSET` tag family.

When you use the flex asset model to represent your content, your online site will use a mixture of flex and basic assets because the page asset type (which you are likely to use) is a basic asset type.

This section contains the following topics:

- Section 28.5.1, "Assetsets"
- Section 28.5.2, "Searchstates"
- Section 28.5.3, "Assetsets, Searchstates, and Flex Attribute Asset Types"
- Section 28.5.4, "Scope"

28.5.1 Assetsets

An **assetset** is a group of one or more flex assets or flex parent assets. You use the `ASSETSET` tags to create the set of assets and to extract the attribute values that you want to display.

You can retrieve the following information from an assetset:

- The values for one attribute for each of the flex assets in the assetset.
- The values for multiple attributes for each of the flex assets in the assetset.

- A list of the flex assets in the assetset
- A count of the flex assets in the assetset
- A list of unique attribute values for an attribute for all flex assets in the assetset
- A count of unique attribute values for an attribute for all flex assets in the assetset

You can create assetsets that include flex assets of more than one type, but only if those flex assets use the same flex attribute asset type.

The most commonly used ASSETSET tags are:

```
ASSETSET.SETASSET
ASSETSET.SETSEARCHEDASSETS
ASSETSET.GETMULTIPLEVALUES
ASSETSET.GETATTRIBUTEVALUES
ASSETSET.GETASSETLIST
ASSETSET.SORTLISTENTRY ...
```

All of the ASSETSET tags are described in the *Oracle Fusion Middleware WebCenter Sites Tag Reference* and several of them are used in the code samples in this chapter. For information about compositional dependencies and the assetset tags, see Section 28.2.2, "The ASSETSET (assetset) Tag Family."

28.5.2 Searchstates

How do you obtain the IDs of the flex assets that you want to display? With searchstate objects.

A **searchstate** is a set of search constraints based on the attribute values held in the _ Mungo table for the flex asset type. You **apply** searchstates to **assetsets**.

You build a searchstate by adding or removing constraints to narrow or broaden the list of flex assets that are described by the searchstate. For example, the GE Lighting sample site uses searchstates to create drill-down searching features that visitors use to browse through the product catalog.

An unconstrained searchstate applied to an assetset creates an unfiltered list of all the assets of that type. For example, the following code sample would create an assetset that contains all the products in the GE Lighting catalog:

```
<SEARCHSTATE.CREATE NAME=nolimits/>
<ASSETSET.SETSEARCHEDASSETS NAME=unconstrainedAssetSet
CONSTRAINT=nolimits ASSETTYPES=Products/>
```

To narrow the number of products in the assetset, you add constraints. For example, the following code sample would create an assetset that contains only the 40-watt light bulbs from the catalog:

```
<SEARCHSTATE.CREATE NAME=lightbulbs/>
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME=lightbulbs
 ATTRIBUTE=wattage VALUE=40/>
<ASSETSET.SETSEARCHEDASSETS NAME=40WattLightbulbs
CONSTRAINT=lightbulbs ASSETTYPES=Products/>
```

A constraint is a filter (restriction) that can be based on the value of an attribute or it can be based on another searchstate, which is called a nested searchstate.

A searchstate can search either the _Mungo table for the asset type database or the attribute indexes created by a search engine for that asset type. This means that you can mix database and rich-text (full-text through an index) searches in the same query.

To apply a constraint against a search engine index, use the `SEARCHSTATE.ADDRICHTEXTCONSTRAINT` tag.

> **Note:** Using SQL to query the flex asset database tables instead of using the `SEARCHSTATE` tag family is not supported.

The most commonly used `SEARCHSTATE` tags are as follows:

```
SEARCHSTATE.CREATE
SEARCHSTATE.ADDSTANDARDCONSTRAINT
SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT
SEARCHSTATE.ADDRANGECONSTRAINT
SEARCHSTATE.ADDRICHTEXTCONSTRAINT
SEARCHSTATE.TOSTRING
SEARCHSTATE.FROMSTRING
```

All of the `SEARCHSTATE` tags are described in the *Oracle Fusion Middleware WebCenter Sites Tag Referenc*e and several of them are used in the code samples in this chapter.

28.5.3 Assetsets, Searchstates, and Flex Attribute Asset Types

Because searchstates filter select assets based on attribute values, and assetsets are created by applying searchstates to the assets in the database, only those flex asset types that share the same attribute asset type can be included in the same assetset.

For example, in the GE Lighting sample site, you can create an assetset with both flex articles and flex images in it because they use the same attribute asset type; content asset attribute. However, because flex articles use the content attribute asset type and products use the product attribute asset type, you cannot create an assetset that contains both flex articles and product assets.

28.5.4 Scope

The scope of assetsets and searchstates is local; that is, they exist only for the current element (rendered page).

When you want to maintain the existing searchstate, you can use the `SEARCHSTATE.TOSTRING` tag to convert it to a string and then include that string as an argument in the URL for the next page.

For example:

```
<SEARCHSTATE.TOSTRING NAME=ss VARNAME=stringss/>
<RENDER.SATELLITEPAGE
pagename= SiteName/Products/Example
ARGS_search=Variables.stringss/>
```

And then, in the root element of this example page that receives the string, you code another searchstate:

```
<SEARCHSTATE CREATE NAME=ss/>
```

And upack the string that was passed to the example element with a `SEARCHSTATE.FROMSTRING` tag:

```
<SEARCHSTATE.FROMSTRING NAME=ss VALUE= Variables.search/>
```

28.6 Coding Templates That Display Flex Assets

When you code templates for an online site that uses the flex asset model, you are primarily concerned with the values of flex attributes, which are assets themselves.

A flex asset (a product, for example) or flex parent asset considered in the context of displaying it, is really an abstraction of attribute values.

You use searchstates to obtain the identity of the flex assets that you want to display, filtering the assets under consideration by their attribute values. The result is an assetset of flex assets or flex parent assets that is based on attribute values, and you can then display the attribute values for the assets in the assetset.

Be sure that you understand the data model of the flex family (or families) that you are using before you begin coding template elements for your flex assets. For more information, read Chapter 11, "Data Design: The Asset Models" and Chapter 16, "Designing Flex Asset Types."

This section contains the following topics:

- Section 28.6.1, "Example Data Set for the Examples in This Section"
- Section 28.6.2, "Examples of Assetsets with One Product (Flex Asset)"
- Section 28.6.3, "Special Cases: Flex Attributes of Type Text, Blob, and URL"
- Section 28.6.4, "Examples of Assetsets with More Than One Product (Flex Asset)"

28.6.1 Example Data Set for the Examples in This Section

The GE Lighting sample site and the Engage extensions to the Burlington Financial sample site illustrate the full power of the flex asset data model and the coding toolset delivered with WebCenter Sites. The templates and elements in the sample sites illustrate the code for fully functioning online sites that display a nearly real-world amount of data.

The code examples in this section are much simpler than the elements in the sample sites. These examples start with simple assetsets and searchstates (hello assetset and hello searchstate) that interact with a small, example data set.

The example data set used in these examples is based on the product flex family, as follows:

Flex Asset Type	External Name (as displayed in WebCenter Sites interface)	Internal Name (as used in the WebCenter Sites database)*
flex attribute	product attribute	`PAttributes`
flex asset	product	`Products`
flex parent	product parent	`ProductGroups`
Always use the internal name of the asset type when you use the `ASSETTYPES` parameter for an `ASSETSET` tag.	n/a	n/a

The example products in this example data set are pairs of blue jeans that have the following attributes:

Attribute	Data Type	Number of Values
sku	string	single
color	string	multiple
price	integer	single
style	text	single

There are four pairs of blue jeans, defined as follows:

sku	color	price	style
jeans-1	blue	35	wide
jeans-2	blue,black	30	straight
jeans-3	black,green	25	straight
jeans-4	green	20	wide

28.6.2 Examples of Assetsets with One Product (Flex Asset)

The code samples in this section do the following:

- Create an assetset that contains one pair of jeans, identified by its `sku` number
- Log a dependency between the product asset and the rendered page(let)
- Get and display the value for the `price` attribute and display it
- Get and display the values for the `color` attribute and display them
- Get and display the values for both the `price` and `color` attribute with the same tag (`ASSETSET.GETMULTIPLEVALUES`)

This section contains the following topics:

- Section 28.6.2.1, "Create a Searchstate and Apply It to an Assetset"
- Section 28.6.2.2, "Get the Price of the Product"
- Section 28.6.2.3, "Display the Price of the Product"
- Section 28.6.2.4, "Get the Colors for the Product"
- Section 28.6.2.5, "Display the Colors of the Product"
- Section 28.6.2.6, "Create a List Object for the ASSETSET.GETMULTIPLEVALUES tag"
- Section 28.6.2.7, "Get the Value for Both Price and Color with ASSETSET.GETMULTIPLEVALUES"
- Section 28.6.2.8, "Display the Value of Price and Color for the jeans-2 Product"

28.6.2.1 Create a Searchstate and Apply It to an Assetset

This line of code creates an unfiltered searchstate named `ss`:

```
<SEARCHSTATE.CREATE NAME="ss"/>
```

Next, we can narrow the unfiltered searchstate named `ss` so that it finds a specific product in the sample data set, by providing the `sku` of the product:

```
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="ss" TYPENAME="PAttributes"
```

```
ATTRIBUTE="sku" VALUE="jeans-2"/>
```

Now we can create an assetset named as, applying the searchstate named ss to it:

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" ASSETTYPES="Products"CONSTRAINT="ss"
FIXEDLIST="true"/>
```

Since the value of the sku attribute is unique for each product asset, there is only one product in the assetset: the one whose sku value is jeans-2.

Because this searchstate was created by querying for a hard-coded attribute value (a sku value of jeans-2) we know the exact contents of the assetset. That is why we set the FIXEDLIST parameter to true. Now the ASSETSET.SETSEARCHEDASSET tag logs a compositional dependency for the product asset.

28.6.2.2 Get the Price of the Product

Next, let's extract the price of this pair of jeans:

```
<ASSETSET.GETATTRIBUTEVALUES NAME="as" ATTRIBUTE="price" TYPENAME="PAttributes"
LISTVARNAME="pricelist"/>
```

Notice that even though price is a single-value attribute (which means the product only has one price), the ASSETSET.GETATTRIBUTEVALUES tag returns the value of the price attribute as a list variable (LISTVARNAME=pricelist).

28.6.2.3 Display the Price of the Product

Now the following line of code can display the price of the jeans-2 product:

```
Price: <ics.listget listname=pricelist fieldname=value/>
And this is the result:
Price: 30
```

28.6.2.4 Get the Colors for the Product

Next, let's determine which colors this pair of jeans is available in.

As specified above, the color attribute is a multiple-value attribute. Because the ASSETSET.GETATTRIBUTEVALUES tag works the same whether an attribute is a single-value or a multiple-value attribute, we use the tag exactly as we did for single-value price attribute:

```
<ASSETSET.GETATTRIBUTEVALUES NAME="as" ATTRIBUTE="color" TYPENAME="PAttributes"
LISTVARNAME="colorlist"/>
```

28.6.2.5 Display the Colors of the Product

Now the following code can display the colors for the jeans-2 product, and, because this product can have more than one color, the code loops through the list:

```
Colors: <LOOP LIST="colorlist">
<ics.listget listname=colorlist fieldname=value/>
&nbsp;
</LOOP>
```

And this is the result:

```
Colors: black blue
```

28.6.2.6 Create a List Object for the ASSETSET.GETMULTIPLEVALUES tag

In general, you should not use the ASSETSET.GETATTRIBUTEVALUES tag when you want to get the value for more than one attribute.

The ASSETSET.GETMULTIPLEVALUES tag gets and scatters the values from more than one attribute, for all the assets in an assetset. Because the tag makes only one call to the database for all the attribute values, it performs the query more efficiently than using multiple ASSETSET.GETATTRIBUTEVALUES tags.

Before you can use this tag, however, you must use the LISTOBJECT tags to create a list object that describes the attributes that the ASSETSET.GETMULTIPLEVALUES tag will return. The list object needs one row for each attribute that you want to get.

This next line of code creates a list object named lo that has columns named attributetypename, attributename, and direction.

```
<LISTOBJECT.CREATE NAME="lo"
COLUMNS="attributetypename,attributename,direction"/>
```

Then, this line adds a row to the list object for each attribute, color and price:

```
<LISTOBJECT.ADDROW NAME="lo" attributetypename="PAttributes" attributename="color"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributetypename="PAttributes" attributename="price"
direction="none"/>
```

The next line of code converts the list object to a list variable name lolist:

```
<LISTOBJECT.TOLIST NAME="lo" LISTVARNAME="lolist"/>
```

28.6.2.7 Get the Value for Both Price and Color with ASSETSET.GETMULTIPLEVALUES

And now we can get the values for both the price and the color attribute from our original assetset, named as:

```
<ASSETSET.GETMULTIPLEVALUES NAME="as" PREFIX="multi" LIST="lolist"
BYASSET="false"/>
```

28.6.2.8 Display the Value of Price and Color for the jeans-2 Product

Now that the values are stored in the list variable (lolist), the following code can display all the values for all the attributes:

```
<LOOP LIST="lolist">
<ics.listget listname=lolist fieldname=attributename
output=attrName/>
<ics.getvar name=attrName/> is
<LOOP LIST="multi:Variables.attrName">
<ics.listget listname=multi:Variables.attrName
fieldname=value/>&nbsp;
</LOOP><P/>
</LOOP>
```

This code sets up a nested loop that loops through all the attributes in the lolist variable, and then loops through all the distinct attribute values for each of the attributes in the lolist list variable.

And this is the result:

```
color is blue black
price is 30
```

28.6.3 Special Cases: Flex Attributes of Type Text, Blob, and URL

If you want to display the values held in flex attributes of type text or blob, use the methodologies described in this section.

This section contains the following topics:

- Section 28.6.3.1, "Flex Attributes of Type Text"
- Section 28.6.3.2, "Flex Attributes of Type Blob"
- Section 28.6.3.3, "Creating a BlobServer URL"
- Section 28.6.3.4, "Getting and Displaying the Value of the Blob"

28.6.3.1 Flex Attributes of Type Text

The ASSETSET.GETMULTIPLEVALUES tag does not retrieve the values for attributes of type text. This means that you must include a separate ASSETSET.GETATTRIBUTEVALUES tag for attributes of this type.

For example, if the color attribute in the sample data set used in these examples were an attribute of type text rather than type string, we could not have retrieved its values with the ASSETSET.GETMULTIPLEVALUES tag in the preceding examples.

28.6.3.2 Flex Attributes of Type Blob

As previously mentioned in Chapter 11, "Data Design: The Asset Models," the _Mungo table for a flex asset type stores the attribute values for the flex assets of that type and the ASSETSET tags query the asset type's _Mungo table for attribute values.

Attributes of type blob are an exception:

- WebCenter Sites stores all the values of all the attributes of type blob in the MungoBlobs table.
- A row in the _Mungo table (Products_Mungo, for example) for an attribute of type blob stores only the ID of the row in the MungoBlobs table that holds its value. That is, the blob column in a _Mungo table is a foreign key to the MungoBlobs table.

This means that for an attribute of type blob, the ASSETSET.GETATTRIBUTEVALUES and ASSETSET.GETMULTIPLEVALUES tags return the ID of the blob attribute's value, but not the actual value.

Once the ID of the attribute's value has been identified, you can do one of two things with it:

- Use the ID to obtain a BlobServer URL.
- Use the ID to extract the actual value of the blob.

28.6.3.3 Creating a BlobServer URL

To obtain a BlobServer URL for the value of the flex attribute blob, you do the following:

- Use the BLOBSERVICE tags to programmatically identify the MungoBlobs table and the appropriate columns in it.
- Pass that information to a RENDER.SATELLITEBLOB tag, if you want the URL in an HTML tag, or to a RENDER.GETBLOBURL tag if you need only the URL without the HTML tag.

Note: Be sure to use the `BLOBSERVICE` tags to programmatically identify the `MungoBlobs` table, as shown in the following example. By obtaining the value with the `BLOBSERVICE` tags rather than hard coding the name of the table into your code, your code will function properly even if the table name is changed in a future version of the product.

To illustrate the following blob examples, let's add the following attribute to the jeans products in our sample data set:

Attribute	Data Type	Number of Values
description	blob	single

First, let's create the assetset and log the dependency between the `jeans-2` product and the rendered page:

```
<SEARCHSTATE.CREATE NAME="ss"/>
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="ss" TYPENAME="PAttributes"
ATTRIBUTE="sku" VALUE="jeans-2"/>
<ASSETSET.SETSEARCHEDASSETS NAME="as" ASSETTYPES="Products" CONSTRAINT="ss"/>
<ASSETSET.GETASSETLIST NAME="as" LISTVARNAME="aslist"/>
<RENDER.LOGDEP cid="aslist.assetid" c="aslist.assettype"/>
The next line of code gets the ID of the jeans-2 asset's description attribute
(that attribute of type blob) and stores it in a list variable called descFile
<ASSETSET.GETATTRIBUTEVALUES NAME="as" TYPENAME="PAttributes"
ATTRIBUTE="description" LISTVARNAME="descFile"/>
```

The next lines of code use the `BLOBSERVICE` tags to obtain the table name and column names from the WebCenter Sites table that stores the attribute values for blob attributes and store them in variables named "`uTabname`", "`idColumn`", and "`uColumn`":

```
<BLOBSERVICE.GETTABLENAME VARNAME="uTabname"/>
<BLOBSERVICE.GETIDCOLUMN VARNAME="idColumn"/>
<BLOBSERVICE.GETURLCOLUMN VARNAME="uColumn"/>
```

Now we can pass the list variable named `descFile` and the `uTabname`, `idColumn`, and `uColumn` variables to a `RENDER.SATELLITEBLOB` tag, which returns a BlobServer URL in an HTML tag:

```
<RENDER.SATELLITEBLOB
BLOBTABLE="Variables.uTabname"
BLOBWHERE="descFile.value"
BLOBKEY="Variables.idColumn"
BLOBCOL="Variables.uColumn"
BLOBHEADER="application/pdf"
/> add service=a href ... download link...
```

The `RENDER.SATELLITEBLOB` tag returns a BlobServer URL in an HREF tag.

28.6.3.4 Getting and Displaying the Value of the Blob

To obtain and display the contents or data in the flex attribute blob after its ID has been returned, you use a `BLOBSERVICE.READDATA` tag, which loads the file name and URL data of the blob.

Under the same assumptions about the data set that we used for the preceding blob example, let's create the assetset, log the dependency between the `jeans-2` asset and the rendered page, and get the ID of the `description` attribute's value:

```
<SEARCHSTATE.CREATE NAME="ss"/>
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="ss" TYPENAME="PAttributes"
ATTRIBUTE="sku" VALUE="jeans-2"/>
<ASSETSET.SETSEARCHEDASSETS NAME="as" ASSETTYPES="Products" CONSTRAINT="ss"/>
<ASSETSET.GETASSETLIST NAME="as" LISTVARNAME="aslist"/>
<RENDER.LOGDEP cid="aslist.assetid" c="aslist.assettype"/>
<ASSETSET.GETATTRIBUTEVALUES NAME="as" TYPENAME="PAttributes"
ATTRIBUTE="description" LISTVARNAME="descFile"/>
```

This time, we want to get and then display the value (data) of the description attribute, so we have to use the `BLOBSERVICE.READDATA` tag:

```
<BLOBSERVICE.READDATA ID="descFile.value" LISTVARNAME="descData"/>
<ics.listget listname=descData fieldname=@urldata/>
```

28.6.4 Examples of Assetsets with More Than One Product (Flex Asset)

The code samples in this section do the following:

- Create an assetset that holds all the products (pairs of jeans) in the sample data set being used in this chapter.
- Get and display a count of the number of jeans in the assetset.
- Get and display all the values for the `color` attribute for all the pairs of jeans in the assetset.
- Get and display all the values for both the `color` and the `style` attributes for the jeans in the assetset.
- Get and display, in a table, all the attribute values for the jeans in the assetset.
- Add a search constraint that filters the assetset for the jeans whose price falls into a specific range.
- Replace the range constraint on the price attribute with a search constraint that filters the assetset for the jeans that are available in any color that begins with the letter b.
- Replace that color constraint with one that filters the assetset for the jeans that are available in either of two specific colors: blue or black

This section contains the following topics:

- Section 28.6.4.1, "Create a Searchstate and Apply it to an Assetset"
- Section 28.6.4.2, "Display the Number of Assets in the Assetset"
- Section 28.6.4.3, "Display the Colors That the Jeans Are Available In"
- Section 28.6.4.4, "Display Both the Colors and the Styles for the Jeans in the Assetset"
- Section 28.6.4.5, "Create a Table That Displays All the Jeans and Their Attribute Values"
- Section 28.6.4.6, "Search for Jeans Based on a Range of Prices"
- Section 28.6.4.7, "Search for Jeans with a Wildcard for Color"
- Section 28.6.4.8, "Search for Jeans with Specific Colors"

28.6.4.1 Create a Searchstate and Apply it to an Assetset

This line of code creates an unfiltered searchstate named ss:

```
<SEARCHSTATE.CREATE NAME="ss"/>
```

When you apply the unfiltered searchstate to an assetset, you get all the flex assets of the type specified (in this case, product assets):

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss" ASSETTYPES="Products"/>
```

28.6.4.2 Display the Number of Assets in the Assetset

These lines of code return and display a count of the number of assets in the assetset, which at this point represents the entire sample catalog:

```
<ASSETSET.GETASSETCOUNT NAME="as" VARNAME="count"/>
How many products are in the catalog?
<ics.getvar name=count/>
```

And this is the result:

```
How many products are in the catalog? 4
```

28.6.4.3 Display the Colors That the Jeans Are Available In

The next lines of code get and display the different colors for the jeans. In other words, the distinct values of the color attribute:

```
<ASSETSET.GETATTRIBUTEVALUES NAME="as" ATTRIBUTE="color" TYPENAME="PAttributes"
LISTVARNAME="colors"/>
What are the possible colors for any pair of jeans?<BR/>
<LOOP LIST="colors">
<ics.listget listname=colors fieldname=value/>
</LOOP><p/>
```

And this is the result:

```
What are the possible colors for any pair of jeans?
black blue green
```

28.6.4.4 Display Both the Colors and the Styles for the Jeans in the Assetset

Next, let's extract and display the values for both the color and the style attribute for the jeans in the assetset. This time we use the ASSETSET.GETMULTIPLEVALUES tag.

First, however, we need to create a list object for the resultset that the ASSETSET.GETMULTIPLEVALUES tag returns. The list object needs one row for each of the attributes, as follows:

```
<LISTOBJECT.CREATE NAME="lo" COLUMNS="attributename,attributetypename,direction"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="color" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="style" attributetypename="PAttributes"
direction="none"/>
```

The next line of code converts the list object to a list variable named lolist:

```
<LISTOBJECT.TOLIST NAME="lo" LISTVARNAME="lolist"/>
```

Now we can extract the attributes and store them in the list variable named lolist:

```
<ASSETSET.GETMULTIPLEVALUES NAME="as" LIST="lolist" PREFIX="distinct"
```

```
BYASSET="false"/>
```

Notice the `BYASSET` parameter in the preceding line of code. Because there is more than one asset in the assetset and we want to know the distinct values for the attribute rather than all the attribute values for each asset in the assetset, `BYASSET=false`. This way, we get only the unique attribute values and not every single attribute value.

The next lines of code loop through the list and display the unique values for each attribute:

```
Here are all the possible colors:
<LOOP LIST="distinct:color">
<ics.listget listname=distinct:color fieldname=value/>
</LOOP><P/>

Here are all the possible styles:
<LOOP LIST="distinct:style">
<ics.listget listname=distinct:style fieldname=value/>
</LOOP><P/>
```

And this is the result:

```
Here are all the possible colors: green blue black
Here are all the possible styles: wide straight
```

28.6.4.5 Create a Table That Displays All the Jeans and Their Attribute Values

You can also use the `ASSETSET.GETMULTIPLEVALUES` tag to obtain the attribute values that are distinct for each asset in the assetset. It creates a list of all the products and the values for their attributes that we can use to create a grid or table that displays all the products in the example catalog.

In this case, we have to do two additional things:

- Because we want the attribute values grouped by the asset that they belong to, the `BYASSET` parameter must be set to `true`.
- Because we need the IDs of the assets in this case, we need to use the `ASSETSET.GETASSETLIST` tag to obtain them.

First, this code creates a list object:

```
<LISTOBJECT.CREATE NAME="lo" COLUMNS="attributename,attributetypename,direction"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="color" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="style" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="price" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="sku" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.TOLIST NAME="lo" LISTVARNAME="lolist"/>
```

Next, we can get the attribute values:

```
<ASSETSET.GETMULTIPLEVALUES NAME="as" LIST="lolist" PREFIX="grid" BYASSET="true"/>
```

And then we use the `ASSETSET.GETASSETLIST` tag.

```
<ASSETSET.GETASSETLIST NAME="as" LISTVARNAME="aslist"/>
```

It returns a list with these columns:

- assettype

- assetid

By using both lists, we can create a grid that shows all of the products and all of their attribute values:

```
<TABLE>
<LOOP LIST="aslist">
<TR>
<TD><CSVAR NAME="grid:aslist.assetid:sku.value"/></TD>
<TD><CSVAR NAME="grid:aslist.assetid:price.value"/>
   </TD>
<TD><CSVAR NAME="grid:aslist.assetid:style.value"/>
</TD>
<TD>
<IF COND="IsList.grid:aslist.assetid:color=true"><THEN>
<LOOP LIST="grid:aslist.assetid:color">
<CSVAR NAME="grid:aslist.assetid:color.value"/>&nbsp;
</LOOP>
</THEN></IF>
</TD>
</TR>
</LOOP>
</TABLE>
```

And this is the result:

jeans-1	35	wide	blue
jeans-2	30	straight	black blue
jeans-3	25	straight	black green
jeans-4	20	wide	green

28.6.4.6 Search for Jeans Based on a Range of Prices

Up until now, we have been using the same assetset (`NAME=as`) that was created in the second line of code in this section. Next, let's filter the assetset by the price attribute, using a range constraint.

This line of code adds a range constraint to our original searchstate (`NAME=ss`) that was created in the first line of code in this section:

```
<SEARCHSTATE.ADDRANGECONSTRAINT NAME="ss" ATTRIBUTE="price" TYPENAME="PAttributes"
LOWER="0" UPPEREQUAL="30"/>
```

The range is from 0 to 30. Let's apply the modified searchstate against our assetset:

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss" ASSETTYPES="Products"/>
```

And check whether it worked, by obtaining and displaying a count of the jeans that are now in the assetset:

```
<ASSETSET.GETASSETCOUNT NAME="as" VARNAME="count"/>
How many jeans products are less than or equal to $30?
<ics.getvar name=count/>
```

Here's the result:

```
How many jeans products are less than or equal to $30? 3
```

28.6.4.7 Search for Jeans with a Wildcard for Color

Now let's replace the range constraint on the `price` attribute with a search constraint that filters the assetset for the jeans that are available in any color that begins with the letter b.

First this line of code deletes the range constraint for price:

```
<SEARCHSTATE.DELETECONSTRAINT NAME="ss" ATTRIBUTE="price"/>
```

And this line of code adds a new constraint for color, using the percentage (%) character as a wildcard with the VALUE parameter:

```
<SEARCHSTATE.ADDSIMPLELIKECONSTRAINT NAME="ss" ATTRIBUTE="color"
TYPENAME="PAttributes" VALUE="b%"/>
```

The `VALUE="b%"` statement means any color that begins with the letter b. Lets apply the modified searchstate against our same assetset (`as`):

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss" ASSETTYPES="Products"/>
```

And check whether it worked by obtaining and displaying a count of the number of jeans that are in the assetset now:

```
<ASSETSET.GETASSETCOUNT NAME="as" VARNAME="count"/>
How many jeans have a color that begins with the letter b?
<ics.getvar name=count/>
```

Here's the result:

```
How many jeans have a color that begins with the letter b? 3
```

28.6.4.8 Search for Jeans with Specific Colors

Finally, let's change the color constraint that filters the assetset for the jeans that are available in either of two specific colors: blue or black

This line of code deletes the color constraint from the searchstate:

```
<SEARCHSTATE.DELETECONSTRAINT NAME="ss" ATTRIBUTE="color"/>
```

Next, because we want to filter based on two values for the color attribute, we need to create a list object with those values:

```
<LISTOBJECT.CREATE NAME="lo" COLUMNS="value"/>
<LISTOBJECT.ADDROW NAME="lo" value="blue"/>
<LISTOBJECT.ADDROW NAME="lo" value="black"/>
<LISTOBJECT.TOLIST NAME="lo" LISTVARNAME="colorlist"/>
```

Now we can use the list variable named `colorlist` to create the searchstate:

```
<SEARCHSTATE.ADDSTANDARDCONSTRAINT NAME="ss" ATTRIBUTE="color"
TYPENAME="PAttributes" LIST="colorlist"/>
```

The `LIST=colorlist` statement is the equivalent of the `VALUE` statement in the preceding example. It means attribute values that match any of the colors in the list named `colorlist`. Let's apply the modified searchstate to our same assetset:

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss" ASSETTYPES="Products"/>
```

And check whether it worked by obtaining and displaying a count of the number of jeans that are in the assetset now:

```
<ASSETSET.GETASSETCOUNT NAME="as" VARNAME="count"/>
```

```
How many products have a color that is black or blue?
<ics.getvar name=count/>
```

Here's the result:

```
How many products have a color that is black or blue? 3
```

28.7 Creating URLs for Hyperlinks

Whether your site is dynamic or static, the fact that you are using a WebCenter Sites system indicates that your content changes regularly. That means that you cannot hard code URLs into hyperlinks. Your pages must be able to determine the identity of the assets they are providing links to when the page is rendered, either by the Export to Disk publish method or by a visitor's browser on a dynamic site.

WebCenter Sites provides three tags (each with an XML and a JSP version) that you can use to create your URLs:

- For URLs for assets that are not blobs, use RENDER.GETPAGEURL tag.
- For URLs for assets that are blobs, use either the RENDER.SATELLITEBLOB tag or the RENDER.GETBLOBURL tag.

This section contains the following topics:

- Section 28.7.1, "RENDER.GETPAGEURL (render:getpageurl)"
- Section 28.7.2, "RENDER.SATELLITEBLOB (render:satelliteblob)"
- Section 28.7.3, "RENDER.GETBLOBURL (render:getbloburl)"
- Section 28.7.4, "Using the referURL Variable"

28.7.1 RENDER.GETPAGEURL (render:getpageurl)

To obtain URLs for regular assets (that is, assets that are not blobs), use the RENDER.GETPAGEURL tag.

The RENDER.GETPAGEURL tag processes arguments passed in from the element that invokes it into a URL-encoded string that it returns as a variable that you name with the OUTSTR parameter. By convention, the name of that variable is referURL.

If rendermode is set to export, it creates a static URL (unless you specify that it should be dynamic). If rendermode is set to live, it creates a dynamic URL.

For example:

```
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Article/Full
cid="Variables.cid"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
```

You can now use the value in the referURL variable to create a hyperlink with an <A HREF> tag.

For more information about this tag, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

28.7.2 RENDER.SATELLITEBLOB (render:satelliteblob)

Binary large objects (blobs) that are stored in the WebCenter Sites database are served by the BlobServer servlet rather than the ContentServer servlet. The `RENDER.SATELLITEBLOB` tag returns an HTML tag with a BlobServer URL in it.

This tag takes a set of arguments that define the blob and an additional set of arguments that determine how to format the blob. For example, you can use it to create an `<IMG SRC>` tag or an `<A HREF>` tag, as follows:

```
<RENDER.SATELLITEBLOB
BLOBTABLE=ImageFile
BLOBKEY=id
BLOBCOL=urlpicture
BLOBWHERE=Variables.asset:id
BLOBHEADER=Variables.asset:mimetype
SERVICE=IMG SRC
ARGS_alt=Variables.asset:alttext
ARGS_hspace=5 ARGS_vspace=5/>
```

Note that there are additional coding steps if you are creating a URL for a flex attribute of type `blob`. For information, see Section 28.6.3.2, "Flex Attributes of Type Blob."

For a longer example, examine the Burlington Financial imagefile template named TeaserSummary. You can examine it in two ways:

- Search for and then inspect it in the WebCenter Sites interface.
- Use Oracle WebCenter Sites Explorer to open the template element called:

```
ElementCatalog/BurlingtonFinancial/ImageFile/TeaserSummary.
```

Even if you are not using Satellite Server, you should still use the `RENDER.SATELLITEBLOB` tag because the tag can create a BlobServer URL in an HTML tag even when Satellite Server is not present.

For more information about this tag, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

28.7.3 RENDER.GETBLOBURL (render:getbloburl)

If you need a BlobServer URL only, without it being embedded in an HTML tag, use the `RENDER.GETBLOBURL` tag.

For example, the Burlington Financial element named `SetHTMLHeader` uses the `RENDER.GETBLOBURL` element to obtain a BlobServer URL (stored as a variable named `referURL`) that it then passes on to JavaScript code that runs on the client side to determine which browser the visitor is using. In this case, the client-side JavaScript creates the HTML tag based on the browser it discovers, so it needs the BlobServer URL without an HTML tag.

The `SetHTMLHeader` element is the element for a CSElement. You can examine it in two ways:

- Use the WebCenter Sites interface to search for the BurlingtonFinanical/Common/SetHTMLHeader CSElement and then inspect it.
- Use Oracle WebCenter Sites Explorer to open the element called:

```
ElementCatalog/BurlingtonFinancial/Common/SetHTMLHeader
```

Note that there are additional coding steps if you are creating a URL for a flex attribute of type `blob`. For information, see Section 28.6.3.2, "Flex Attributes of Type Blob."

For more information about the RENDER.GETBLOBURL tag, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

28.7.4 Using the referURL Variable

The RENDER.GETPAGEURL, RENDER.GETBLOBURL, and RENDER.SATELLITEBLOB tags were introduced in the 3.6.x version of this product. Older versions of the product used elements named GetPageURL and GetBlobURL to obtain URLs; they are coded to return URLs in a variable named referURL.

By convention, all of the sample code in the sample sites that use the tags that replaced the GetPageURL and GetBlobURL elements use a referURL variable for the value of the URL.

Do not append or add any text to the value held in the referURL variable or any other variable returned by a RENDER.GETPAGEURL or RENDER.GETBLOBURL tag. URLs in this kind of variable are complete (whole). If you change the URL returned by the tag, you are likely to break it.

If you need to include additional arguments in a URL, use the RENDER.PACKARGS tag to URL-encode them ("pack" them) and then pass those encoded arguments to the RENDER.GETPAGEURL or RENDER.GETBLOBURL tag with the PACKEDARGS parameter.

For information about the RENDER.PACKARGS tag, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

28.8 Handling Error Conditions

While you code your elements, you should also include code that checks for error conditions. You decide which error conditions are serious and, when necessary, code a solution or alternate action. Sometimes the solution is to write a meaningful error message. As an additional step, you can include code that stops a broken page from being cached.

Note: While debugging your code, you can use the commons-logging.properties and log4j.properties files to enable loggers. Error and debugging messages are then written to the WebCenter Sites log file. For information about the debugging properties, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

This section contains the following topics:

- Section 28.8.1, "Using the Errno Variable"
- Section 28.8.2, "Ensuring that Incorrect Pages Are Not Cached"

28.8.1 Using the Errno Variable

The errno variable, a standard WebCenter Sites variable, holds error numbers that the WebCenter Sites XML and JSP tags report. When a WebCenter Sites tag cannot successfully execute, it sets errno to the value that best describes the reason why it did not succeed. For example, an errno value of -13004 means a CURRENCY tag couldn't read a number because it was not in the correct currency format. For a complete list of all the errno values and their descriptions, see the error conditions section in the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

The tags that are delivered with the WebCenter Sites modules and products clear errno before they execute so you do not need to set errno to 0 when you want to check for errors from these tags. Here's a code example that determines whether an `ASSET.LOAD` was successful before attempting to load the child assets:

```
<ASSET.LOAD NAME="topArticle" TYPE="Article"
OBJECTID="Variables.cid"/>
<IF COND="IsError.Variables.errno=false">
<THEN>
<ASSET.CHILDREN NAME="topArticle"
LIST="listOfChildren/>
</THEN>
</IF>
```

If you want to check the results of the tags that are delivered by WebCenter Sites, you must include code that clears the value errno before the tag whose results you want to check. For example:

```
<SETVAR NAME=errno VALUE=0/>
```

For a longer example, see the Burlington Financial CSElement named `BurlingtonFinancial/Util/Account/SignUp`. This CSElement provides the code that adds members to the site and updates existing member's information. It checks for several error conditions and provides appropriate responses to them.

The following code sample shows an error message that you could use while you are in the process of developing your templates:

```
<IF rendermode=preview>
<THEN>
<IF COND=IsError.Variables.errno=true>
<THEN>
<FONT COLOR=#FF0000>
Error <ics.geterrno/>
while rendering <ics.getvar name=pagename/>
with asset ID <ics.getvar name=cid/>.
</FONT>
</THEN>
</IF>
</THEN>
</IF>
```

28.8.2 Ensuring that Incorrect Pages Are Not Cached

If you can determine that the output from an element is incorrect, there is probably no need for WebCenter Sites or Satellite Server to cache the page. You can stop the page that is being generated from being cached with the `ics.disablecache` tag.

Example 1: Error Condition

To continue with the first example in Section 28.8.1, "Using the Errno Variable," if the article asset could not be loaded, there would also be no reason to cache the page. You could add the following `ELSE` statement to the `IF` condition in that code sample:

```
<ASSET.LOAD NAME="topArticle" TYPE="Article"
OBJECTID="Variables.cid"/>
<IF COND="IsError.Variables.errno=false">
<THEN>
<ASSET.CHILDREN NAME="topArticle"
LIST="listOfChildren/>
</THEN>
```

```
<ELSE>
<ics.disablecache/>
</ELSE>
</IF>
```

Example 2: Clear the Page From Cache if the Asset's Status is VO (Basic Assets Only)

As described in "CacheManager and Dynamic Publish Sessions" section of Section 28.1.2, "Page Generation and Compositional Dependencies," the CacheManager on the destination system regenerates all the pages and pagelets that were affected by a publishing session. Affected pages includes those whose dependent assets were deleted.

Deleted assets have their status set to VO. The ASSET.LOAD and asset:load tags do not check the status of an asset before they execute which means they can and will load a deleted asset. Typically this isn't a problem. Why? Because an asset cannot be deleted until all links to it from other assets are removed. Therefore, when the site is regenerated there are no longer any links to a page or pagelet that would display the deleted asset. But there is no need to leave a page or pagelet that displays a deleted asset in the cache.

The following code sample stops the page from being cached if the asset cannot be loaded or if the asset's status is deleted:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="WireStoryTextArticle"/>
<!-- if the asset cannot be loaded, then flush the pagelet from cache -->
<if COND="IsError.Variables.errno=true">
<then>
<ics.disablecache/>
</then>
</if>
<ASSET.SCATTER NAME="WireStoryTextArticle" PREFIX="asset"/>
<!-- if the asset is marked as void, then flush the pagelet from cache -->
<if COND="Variables.asset:status=VO">
<then>
<ics.disablecache/>
</then>
</if>
```

Note that you do not need to include code that checks the status of flex assets. The SEARCHSTATE and searchstate tags do not return assets that have a status of VO and the ASSETSET and assetset tags do not include assets that have a status of VO in the assetsets that they create.

29

Template Element Examples for Basic Assets

This chapter uses examples from the Burlington Financial sample site to illustrate the information presented in Chapter 28, "Coding Elements for Templates and CSElements."

This chapter contains the following sections:

- Section 29.1, "Example 1: Basic Modular Design"
- Section 29.2, "Example 2: Coding Links to the Article Assets in a Collection Asset"
- Section 29.3, "Example 3: Using the ct Variable"
- Section 29.4, "Example 4: Coding Templates for Query Assets"
- Section 29.5, "Example 5: Displaying an Article Asset Without a Template"
- Section 29.6, "Example 6: Displaying Site Plan Information"
- Section 29.7, "Example 7: Displaying Non-Asset Information"

All of the elements described in this section are from the Burlington Financial sample site.

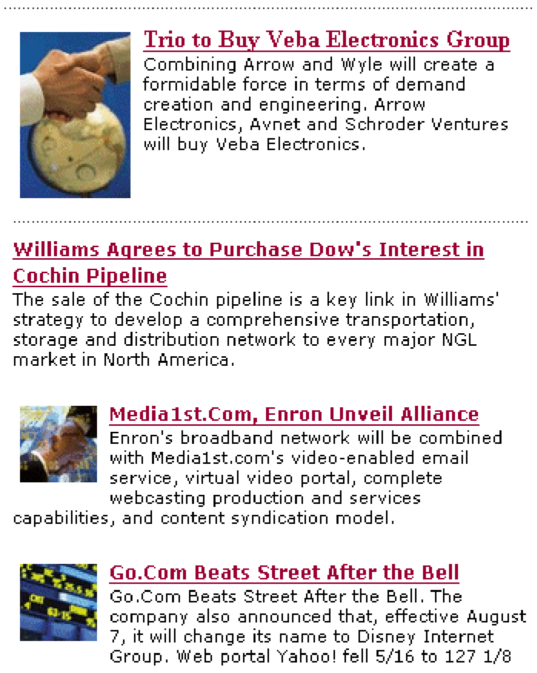
29.1 Example 1: Basic Modular Design

The Burlington Financial sample site is an example of a modular site design that takes advantage of common elements so that common code is written once but reused in several locations or contexts. Following is a description of how one area on the Burlington Financial home page is created from five separate elements.

First, open the WebCenter Sites interface, select the Burlington Financial site and preview the Home page asset. You can either search for the asset and select **Preview** from the drop-down list on the icon bar or you can expand the **Placed Pages** icon under the **Burlington Financial** node in the **Site Plan** tab, select the Home page, and then select **Preview** from the right-mouse menu.

WebCenter Sites displays the Burlington Financial home page in your browser.

Directly under the date, there is a column that displays the main stories of the day. There is a summary paragraph and byline for each story in the list. The titles of the stories are hyperlinks to the full story. Several of the stories, including the first story in the list, also present a photo:



Trio to Buy Veba Electronics Group
Combining Arrow and Wyle will create a formidable force in terms of demand creation and engineering. Arrow Electronics, Avnet and Schroder Ventures will buy Veba Electronics.

Williams Agrees to Purchase Dow's Interest in Cochin Pipeline
The sale of the Cochin pipeline is a key link in Williams' strategy to develop a comprehensive transportation, storage and distribution network to every major NGL market in North America.

Media1st.Com, Enron Unveil Alliance
Enron's broadband network will be combined with Media1st.com's video-enabled email service, virtual video portal, complete webcasting production and services capabilities, and content syndication model.

Go.Com Beats Street After the Bell
Go.Com Beats Street After the Bell. The company also announced that, effective August 7, it will change its name to Disney Internet Group. Web portal Yahoo! fell 5/16 to 127 1/8

This example describes how the first story in the list is identified, selected, positioned at the top of the list, and formatted.

These are the elements used to format the first story in the list:

- BurlingtonFinanical/Page/Home
- BurlingtonFinancial/Collection/MainStoryList
- BurlingtonFinancial/Article/LeadSummary
- BurlingtonFinancial/ImageFile/TeaserSummary

This section contains the following topics:

- Section 29.1.1, "First Element: Home"
- Section 29.1.2, "Second Element: MainStoryList"
- Section 29.1.3, "Third Element: LeadSummary"
- Section 29.1.4, "Fourth Element: TeaserSummary"
- Section 29.1.5, "Back to LeadSummary"
- Section 29.1.6, "Back to MainStoryList"
- Section 29.1.7, "Back to Home"

29.1.1 First Element: Home

The Home page of the of the Burlington Financial sample site uses a template that is also named Home. You can examine it in two ways:

- Search for and then inspect it in the WebCenter Sites interface.
- Use Oracle WebCenter Sites Explorer to open the template element called:

  ```
  ElementCatalog/BurlingtonFinancial/Page/Home
  ```

First, the `Home` element loads the home page asset, names it HomePage, and then scatters the information in its fields:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="HomePage"/>
<ASSET.SCATTER NAME="HomePage" PREFIX="asset"/>
```

The value for cid is passed in from the Burlington Financial URL and the value for c is available because it is set as a variable in the resarg1 column in the SiteCatalog page entry for the Home template.

Scroll down past several `callelement` and `RENDER.SATELLITEPAGE` tags to the following `ASSET.CHILDREN` tag:

```
<ASSET.CHILDREN NAME=HomePage LIST=MainStories CODE=TopStories/>
```

With this code, Home obtains the collection asset identified as the page asset's TopStories collection (`CODE=TopStories`) and creates a list named MainStories to hold it (`LIST=MainStories`).

Next, Home determines whether it successfully obtained the collection and then calls for the page entry of the MainStoryList template.

```
<IF COND = IsList.MainStories=true>
<THEN>
<RENDER.SATELLITEPAGE pagename=BurlingtonFinanical/Collection/MainStoryList
ARGS_cid=MainStories.oid
ARGS_p=Variables.asset:id/>
<THEN/>
<IF/>
```

Notice that Home passes the identity of the list that holds the collection to MainStories with `ARGS_cid` and the identity of the Home page asset with `ARGS_ p=Variables.asset:id`.

29.1.2 Second Element: MainStoryList

The MainStoryList page entry invokes its root element. Use Oracle WebCenter Sites Explorer to open and examine this element:

```
ElementCatalog/BurlingtonFinancial/Collection/MainStoryList.xml
```

The MainStoryList element is the template element (the root element) for the MainStoryList Template asset, a template that formats collection assets.

This element creates the framework for the Home page column that holds the main list of stories, and then fills that column with the articles from the TopStories collection. It uses two templates to format those articles:

- LeadSummary for the first article in the collection (the top-ranked article)
- Summary for the rest of the articles

Because the purpose of this example is to describe how the first story in the list is displayed, this example discusses only the LeadSummary template element.

MainStoryList loads and scatters the collection that Home passed to it:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="MainStoryListCollection"/>
<ASSET.SCATTER NAME="MainStoryListCollection" PREFIX="asset"/>
```

It then extracts the articles from the collection, creates a list to hold them, ordering them by their rank:

```
<ASSET.CHILDREN NAME="MainStoryListCollection" LIST="theArticles"
ORDER="nrank" CODE=-/>
```

And then it calls for the page entry of the LeadSummary template:

```
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/Article/LeadSummary"
ARGS_cid="theArticles.oid"
ARGS_ct="Full"
ARGS_p="Variables.p"/>
```

Once again, this element passes on several pieces of information:

- The identity of the list that holds the articles (ARGS_cid)
- The name of the template to use when creating the link to each of the articles (ARGS_ct)
- The identity of the originating page asset (ARGS_p), which is Home.

Because the list was ordered by rank and this code does not loop through the list, the value in ARGS_cid (theArticles.oid) is the object ID of the highest ranked article in the collection because that article is the first article in the list.

29.1.3 Third Element: LeadSummary

The LeadSummary page entry invokes its root element (which is the template element for the LeadSummary template). Use Oracle WebCenter Sites Explorer to open and examine it:

```
ElementCatalog/BurlingtonFinancial/Article/LeadSummary.xml
```

This element formats the first article in the TopStories collection. It does the following:

- Retrieves the image file associated with the first article through the TeaserImage association.
- Invokes the TeaserSummary element to obtain the formatting code for the image.
- Uses a RENDER.GETPAGEURL tag to obtain the URL for the first article in the collection.
- Displays the imagefile asset, the title of the article as a hyperlink to the full article, the summary paragraph, and the byline.

First LeadSummary loads the article and names it LeadSummaryArticle:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="LeadSummaryArticle"/>
<ASSET.SCATTER NAME="LeadSummaryArticle" PREFIX="asset"/>
```

It obtains the assets associated with the article as its Teaser imagefile asset, creating a list for that file named TeaserImage:

```
<ASSET.CHILDREN NAME="LeadSummaryArticle" LIST="TeaserImage"
CODE="TeaserImageFile"/>
```

Finally, it calls the page entry for the TeaserSummary template, passing it the ID of the imagefile asset held in the list:

```
<THEN>
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/ImageFile/TeaserSummary"
ARGS_cid="TeaserImage.oid"/>
</THEN>
</IF>
```

29.1.4 Fourth Element: TeaserSummary

The TeaserSummary page entry invokes its root element, the template element for the TeaserSummary template. Use Oracle WebCenter Sites Explorer to open and examine it:

```
ElementCatalog/BurlingtonFinancial/ImageFile/TeaserSummary
```

Because imagefile assets are blobs stored in the WebCenter Sites database, and blobs stored in the database must be served by the BlobServer servlet rather than the ContentServer servlet, this element obtains an HTML tag that uses a BlobServer URL.

Scroll down to the following RENDER.SATELLITEBLOB tag:

```
<RENDER.SATELLITEBLOB BLOBTABLE=ImageFile BLOBKEY=id BLOBCOL=urlpicture
BLOBWHERE=Variables.asset:id BLOBHEADER= Variables.asset:mimetype SERVICE=IMG SRC
ARGS_alt= Variables.asset:alttext ARGS_hspace=5 ARGS_vspace=5 />
```

The tag creates an HTML <IMG SRC> tag. The SRC is the blob in the ImageFile table identified through the ID passed in with BLOBWHERE=Variables.asset:id and both its horizontal and vertical spacing are at five pixels.

When TeaserSummary is finished, LeadSummary continues.

29.1.5 Back to LeadSummary

When LeadSummary resumes, having obtained the teaser image for the first article in the TopStories collection, it uses RENDER.GETPAGEURL to obtain the URL for that article:

```
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Article/Variables.ct"
cid="Variables.cid"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
```

Remember that when the MainStoryList element called the page entry for LeadSummary, it passed a ct variable set to Full. Therefore, the page name that LeadSummary is passing to RENDER.GETPAGEURL is really BurlingtonFinancial/Article/Full.

RENDER.GETPAGEURL creates the URL for the article based on the information passed in to it and then returns that URL to LeadSummary in a variable called referURL, as specified by the OUTSTR parameter.

LeadSummary uses the referURL variable in an HTML <A HREF> tag and then displays the link, the abstract of the article, and the byline:

```
<A class="featurehead" HREF="Variables.referURL" REPLACEALL="Variables.referURL">
<csvar NAME="Variables.asset:description"/></A>
&nbsp;<BR/>
<span class="thumbtext"><csvar NAME="Variables.asset:abstract"/>
</span><BR/>
<span class="thumbcredit"><csvar NAME="Variables.asset:byline"/>
</span><BR/>
```

Note the use of the REPLACEALL tag as an attribute in the HTML <A HREF> tag. You must use this tag as an attribute when you want to use XML variables in HTML tags.

Now that LeadSummary is finished, MainStoryList continues.

29.1.6 Back to MainStoryList

Next MainStoryList loops through the rest of the articles in the TopStories collection and uses the Summary template to format them.

If you are interested, use Oracle WebCenter Sites Explorer to open and examine it:

```
ElementCatalog/Article/Summary
```

When MainStoryList is finished, Home continues.

29.1.7 Back to Home

Home resumes, with a call to the WireFeedBox page entry.

29.2 Example 2: Coding Links to the Article Assets in a Collection Asset

When an element needs URLs to create a list of hyperlinks to dynamically served WebCenter Sites pages, use the RENDER.GETPAGEURL tag.

These are the elements referred to in this example:

- ElementCatalog/BurlingtonFinancial/Page/SectionFront
- ElementCatalog/BurlingtonFinancial/Collection/PlainList

For the purposes of this example, the code displayed is stripped of any error checking so that you can focus on how the links are created.

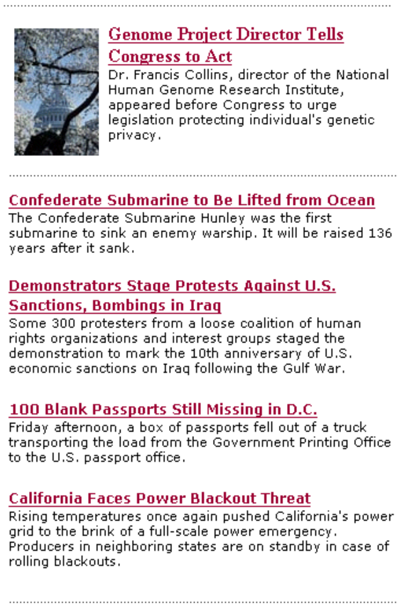
This section contains the following topics:

- Section 29.2.1, "First element: SectionFront"
- Section 29.2.2, "Second element: PlainList"

29.2.1 First element: SectionFront

SectionFront is the template element, the root element, of the SectionFront template which is assigned to the main section pages; News, Markets, Stocks, and so on. It is invoked when a visitor clicks a link to a section.

One section of a page formatted with the SectionFront element displays a list of links to articles from the Section Highlights collection that is associated with that page asset.



Genome Project Director Tells Congress to Act
Dr. Francis Collins, director of the National Human Genome Research Institute, appeared before Congress to urge legislation protecting individual's genetic privacy.

Confederate Submarine to Be Lifted from Ocean
The Confederate Submarine Hunley was the first submarine to sink an enemy warship. It will be raised 136 years after it sank.

Demonstrators Stage Protests Against U.S. Sanctions, Bombings in Iraq
Some 300 protesters from a loose coalition of human rights organizations and interest groups staged the demonstration to mark the 10th anniversary of U.S. economic sanctions on Iraq following the Gulf War.

100 Blank Passports Still Missing in D.C.
Friday afternoon, a box of passports fell out of a truck transporting the load from the Government Printing Office to the U.S. passport office.

California Faces Power Blackout Threat
Rising temperatures once again pushed California's power grid to the brink of a full-scale power emergency. Producers in neighboring states are on standby in case of rolling blackouts.

Use Oracle WebCenter Sites Explorer to open and examine the `SectionFront` element:

`ElementCatalog/BurlingtonFinancial/Page/SectionFront.`

First, `SectionFront` uses the variables `c` and `cid` to load and scatter the page asset, and names it SectionFrontPage:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="SectionFrontPage"/>
<ASSET.SCATTER NAME="SectionFrontPage" PREFIX="asset"/>
```

The values for `c` and `cid` are passed to the `SectionFront` element from the link that invoked it. That link could be from the home page or any one of several other locations.

In Oracle WebCenter Sites Explorer, scroll down past several `ASSET.CHILDREN` tags to the one that retrieves the Section Highlights collection:

```
<ASSET.CHILDREN NAME="SectionFrontPage" LIST="SectionHighlights"
CODE="SectionHighlight"/>
```

This code retrieves the collection with the `CODE=SectionHighlights` statement and stores it as a list, also named `SectionHighlights`.

Then SectionFront calls the page entry of the PlainList template (a collection template):

```
<RENDER.SATELLITEPAGE
pagename="BurlingtonFinancial/Collection/PlainList"
ARGS_cid="SectionHighlights.oid" ARGS_p="Variables.asset:id"/>
```

This code passes in the ID of the Section Highlights collection (`cid`) and the ID of the current page asset (`p`), which is the page asset assigned the name of SectionFrontPage.

29.2.2 Second element: PlainList

The PlainList page entry invokes its root element, the template element for the PlainList template. Use Oracle WebCenter Sites Explorer to open and examine it:

`ElementCatalog/BurlingtonFinancial/Collection/PlainList.`

PlainList extracts the articles from the collection and presents them in a list, by their rank, with the subheadline of the article. This element assumes that the assets in the collection are articles.

PlainList uses the values in `c` and `cid` (passed in from the SectionFront element) to load and scatter the collection:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="PlainListCollection"/>
<ASSET.SCATTER NAME="PlainListCollection" PREFIX="asset"/>
```

PlainList then sets the variable `ct` to `Full` because a value for this variable was not passed in (`Full` is the name of an article template):

```
<IF COND="IsVariable.ct!=true">
<THEN>
<SETVAR NAME="ct" VALUE="Full"/>
</THEN>
</IF>
```

Next PlainList creates a list of all the child articles in the collection asset, listing them by their rank, and naming the list theArticles.

```
<ASSET.CHILDREN NAME="PlainListCollection" LIST="theArticles" OBJECTTYPE="Article"
ORDER="nrank" CODE=-/>
```

Note that this `ASSET.CHILDREN` tag used the `OBJECTTYPE` parameter. If you use the `OBJECTTYPE` parameter with this tag, the resulting list of children is a join of the `AssetRelationTree` and the asset table for the type you specified (in this case, the `Article` table) and it contains data from both tables.

There is now no need for subsequent `ASSET.LOAD` tags because the data that the `PlainList` element is going to use to create the links to these articles is stored in the `Article` table.

`PlainList` loops through the list of articles, using the `RENDER.GETPAGEURL` tag to create a URL for each one. In this case (because the code does not use subsequent `ASSET.LOAD` tags for each of the children assets) the element includes a `RENDER.LOGDEP` tag in the loop:

```
<LOOP LIST="theArticles">
<RENDER.LOGDEP cid="theArticles.id" c="Article"/>
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Article/
Variables.ct"
cid="theArticles.id"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
```

`PlainList` passes a `cid` and `pagename` and the asset type with `ctype` for each article in the collection to the `RENDER.GETPAGEURL` tag. Because the variable `ct` was set to `Full`, the page name being passed to the tag is actually `BurlingtonFinancial/Article/Full`.

The `RENDER.GETPAGEURL` tag returns a `referURL` variable for each article in the collection, as specified by the `OUTSTR` parameter, and then `PlainList` uses the value in the `referURL` variable to create an HTML `<A HREF>` link for each article.

Because the `ASSET.CHILDREN` tag that obtained this collection created a join between `AssetRelationTree` and the `Article` table, `PlainList` can use the article's subheadline field to create the link:

```
<A class="wirelink" HREF="Variables.referURL"
 REPLACEALL="Variables.referURL">
<csvar NAME="Variables.theArticles:subheadline"/>
</A>
</LOOP>
```

Note the use of the REPLACEALL tag as an attribute for this HTML tag. You must use this tag as an HTML attribute when you want to use XML variables in an HTML tags. For more information about REPLACEALL, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

29.3 Example 3: Using the ct Variable

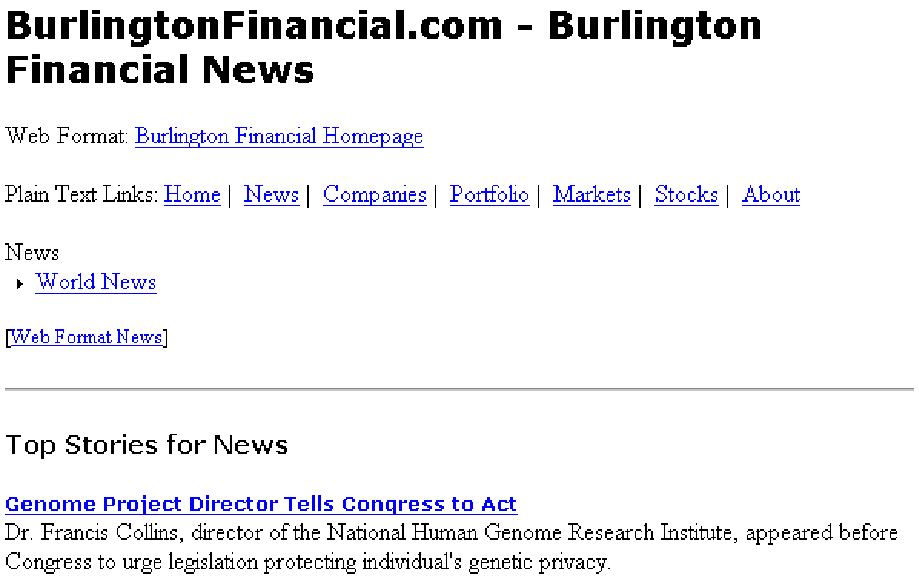
The ct variable represents the concept of a child template. Child templates are alternate templates. Because assets are assigned a template when they are created, the identity of an asset's template (which is not the same as a default approval template) is part of the information you obtain with an ASSET.LOAD or an ASSET.CHILDREN tag.

However, sometimes you want to use a template other than an asset's default template. In such a case, you supply the name of an alternate template with the ct variable.

For example, when a visitor browses the Burlington Financial site, there are text-only versions of most of the site available to that visitor. The text-only format is not the default format and content providers do not assign text-only formats to their assets. The Burlington Financial page elements are coded to provide the ID of the alternate, text-only template when it is appropriate to do so.

Open the WebCenter Sites interface, and preview both the Burlington Financial Columnists page and the News Page. In the upper right corner of these pages, the Plain Text link is displayed.

Click the **Plain Text** link on the Columnists page. Then click the Plain Text link on the News Page:



BurlingtonFinancial.com - Burlington Financial News

Web Format: Burlington Financial Homepage

Plain Text Links: Home | News | Companies | Portfolio | Markets | Stocks | About

News
- World News

[Web Format News]

Top Stories for News

Genome Project Director Tells Congress to Act

Dr. Francis Collins, director of the National Human Genome Research Institute, appeared before Congress to urge legislation protecting individual's genetic privacy.

Every page on the site uses the same element, the TextOnlyLink element, to determine the URL embedded in the Plain Text link for that page. The TextOnlyLink element returns the correct URL for each page because the Plain Text link on each page passes the TextOnly element the information that it needs:

- The ID of the page making the request
- the alternate, text-only template (that is, the child template) to use for the Plain Text link

These are the elements used in this example:

- ElementCatalog/BurlingtonFinancial/Page/SectionFront
- ElementCatalog/BurlingtonFinancial/Page/SectionFrontText
- ElementCatalog/BurlingtonFinancial/Common/TextOnlyLink
- ElementCatalog/BurlingtonFinancial/Page/ColumnistFront

This section contains the following topics:

- Section 29.3.1, "First Element: SectionFront"
- Section 29.3.2, "Second Element: TextOnlyLink"
- Section 29.3.3, "ColumnistFront"

29.3.1 First Element: SectionFront

Use Oracle WebCenter Sites Explorer to open and examine the Section Front element:

`ElementCatalog/BurlingtonFinancial/Page/SectionFront.`
SectionFront is the template element (root element) of the Template asset assigned to the standard section pages on the site, pages such as News, Money, Stocks, and so on.

Scroll down approximately two-thirds of the element to this `CALLELEMENT` tag:

```
<CALLELEMENT NAME="BurlingtonFinancial/Common/TextOnlyLink">
<ARGUMENT NAME="ct" VALUE="SectionFrontText"/>
<ARGUMENT NAME="assettype" VALUE="Page"/>
</CALLELEMENT>
```

`TextOnlyLink` is the element that creates the Plain Text Link. `SectionFront` passes it the name of the alternate template (`ct=SectionFrontText)` and the name of the asset type (`assettype=Page`).

29.3.2 Second Element: TextOnlyLink

The `TextOnlyLink` element executes. Use Oracle WebCenter Sites Explorer to open and examine it:

`ElementCatalog/BurlingtonFinancial/Common/TextOnlyLink`
First, `TextOnlyLink` checks to see whether there is a value for `ct`.

```
<IF COND="IsVariable.ct!=true">
<THEN>
<SETVAR NAME="ct" VALUE="Variables.asset:templateText"/>
</THEN>
</IF>
```

There is a value for `ct` because the `SectionFront` element passed in `ct=SectionFrontText`.

Next, TextOnlyLink uses a RENDER.GETPAGEURL tag to obtain a URL for the Plain Text link, passing in the page name by concatenating one based on the variables that were passed to TextOnlyLink by SectionFront.

```
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Variables.assettype/Variables.ct"
cid="Variables.asset:id"
c="Variables.assettype"
p="Variables.p"
OUTSTR="referURL"/>
```

TextOnlyLink knows that ct=SectionFrontText and that assettype=Page. Therefore BurlingtonFinancial/Variables.assettype/Variables.ct means BurlingtonFinancial/Page/SectionFrontText.

Now that TextOnlyLink has a URL (in the referURL variable specified by the OUTSTR parameter), it can create the Plain Text link with an HTML <A HREF> tag:

```
<A class="contentlink" HREF="Variables.referURL"
REPLACEALL="Variables.referURL">
<img src="/futuretense_cs/bf/images/TextOnly.gif" width="22"
height="14" border="0" HSPACE="3"/>Plain Text</A><BR/>
```

Note the use of the REPLACEALL tag as an attribute for this HTML tag. You must use this tag as an HTML attribute when you want to use XML variables in an HTML tag. For more information about REPLACEALL, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

And then TextOnlyLink clears the ct variable.

```
<REMOVEVAR NAME="ct"/>
```

When a visitor clicks the Plain Text link, the article is formatted with the SectionFrontText element and then displayed in the browser.

Note: If you are interested in the format of the plain text version of a section page, use Oracle WebCenter Sites Explorer to open and examine SectionFrontText:

```
ElementCatalog/BurlingtonFinancial/Page/SectionFrontText
```

29.3.3 ColumnistFront

Use Oracle WebCenter Sites Explorer to open and examine the ColumnistFront element:

```
ElementCatalog/BurlingtonFinancial/Page/ColumnistFront
```

This element formats the web format page that displays the stories supplied from the Burlington Financial columnists.

To create the Plain Text link in the upper right corner of a section page, ColumnistFront calls TextOnlyLink:

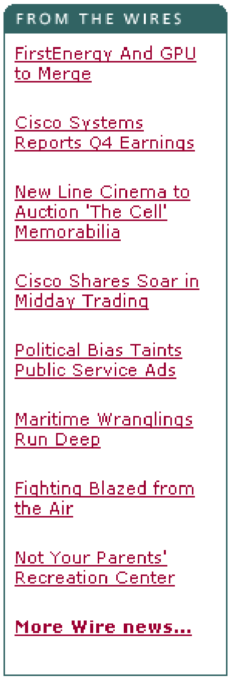
```
<CALLELEMENT NAME="BurlingtonFinancial/Common/TextOnlyLink">
<ARGUMENT NAME="ct" VALUE="ColumnistFrontText"/>
<ARGUMENT NAME="assettype" VALUE="Page"/>
</CALLELEMENT>
```

Based on the information passed in from ColumnistFront, this time TextOnlyLink creates a Plain Text link that takes the visitor to BurlingtonFinancial/Page/ColumnistFrontText.

29.4 Example 4: Coding Templates for Query Assets

When you use a query asset to obtain the assets that you want to display, you use the standard WebCenter Sites element name `ExecuteQuery` to run it.

Burlington Financial uses several query assets. This example describes a query asset named Home Wire Feed which is used to list wire feed stories on the Home page:



FROM THE WIRES

FirstEnergy And GPU to Merge

Cisco Systems Reports Q4 Earnings

New Line Cinema to Auction 'The Cell' Memorabilia

Cisco Shares Soar in Midday Trading

Political Bias Taints Public Service Ads

Maritime Wranglings Run Deep

Fighting Blazed from the Air

Not Your Parents' Recreation Center

More Wire news...

These are the elements used in this example:

- ElementCatalog/BurlingtonFinancial/Page/Home
- ElementCatalog/BurlingtonFinancial/Query/WireFeedBox
- ElementCatalog/OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery

This section contains the following topics:

- Section 29.4.1, "First Element: Home"
- Section 29.4.2, "Second Element: WireFeedBox"
- Section 29.4.3, "Third Element: ExecuteQuery"
- Section 29.4.4, "Back to WireFeedBox"

29.4.1 First Element: Home

Use Oracle WebCenter Sites Explorer to open and examine the template element for the Home page:

```
ElementCatalog/BurlingtonFinancial/Page/Home
```

First, Home loads the home page asset, names it HomePage, and then scatters the information in its fields:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="HomePage"/>
<ASSET.SCATTER NAME="HomePage" PREFIX="asset"/>
```

The values for `c` and `cid` are passed in from the Burlington Financial URL.

Scroll down past several `CALLELEMENT` and `RENDER.SATELLITEPAGE` tags to the following `ASSET.CHILDREN` tag:

```
<ASSET.CHILDREN NAME="HomePage" LIST="WireFeedStories"
CODE="WireFeed"/>
```

Notice that in this line of code, the `OBJECTTYPE` parameter is not used. `CODE=WireFeed` is enough information for WebCenter WebCenter Sites to locate and retrieve the query assigned to the Home page asset through the WireFeed association and there is no need to create a join between the `AssetRelationTree` and the `Query` table because all that Home needs is the ID of the query. The WireFeed query is retrieved and stored as `WireFeedStories`.

Next, Home calls the page entry of the WireFeedBox template, passing it the `cid` of the query stored as `WireFeedStories`:

```
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/Query/WireFeedBox"
ARGS_cid="WireFeedStories.oid"
ARGS_p=Variables.asset:id/>
```

Home passes on several pieces of information: the identity of the query with the `cid=WireFeedStories.oid` statement and the identity of the originating page asset, Home, with the `p=Variables.asset:id` statement.

29.4.2 Second Element: WireFeedBox

The WireFeedBox page entry invokes its root element, the template element for the WireFeedBox template. Use Oracle WebCenter Sites Explorer to open and examine it:

```
ElementCatalog/BurlingtonFinancial/Query/WireFeedBox
```

This element invokes the `ExecuteQuery` element to run the query and then displays a list of links to the article assets returned by the query.

First, `WireFeedBox` loads the query asset passed in from `Home`, names it `WireFeedBoxQuery`, and then retrieves the values from all of its fields with an `ASSET.SCATTER` statement:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="WireFeedBoxQuery"/>
<ASSET.SCATTER NAME="WireFeedBox" PREFIX="asset"/>
```

`Variables.cid` is the `WireFeedStories.oid` passed in from the Home element.

Then `WireFeedBox` calls the `ExecuteQuery` element:

```
<CALLELEMENT NAME="OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery">
<ARGUMENT NAME="list" VALUE="ArticlesFromWireQuery"/>
<ARGUMENT NAME="assetname" VALUE="WireFeedBoxQuery"/>
<ARGUMENT NAME="ResultLimit" VALUE="8"/>
</CALLELEMENT>
```

WireFeedBox passed in the query asset, the name of the list to create to hold the results of the query, and a limit of 8 so that no matter how many assets the query returns to `ExecuteQuery, ExecuteQuery` returns only 8 of them to WireFeedBox.

29.4.3 Third Element: ExecuteQuery

The `ExecuteQuery` element runs the query asset.

The query assets that can be assigned to a page asset as that page's Wire Feed query are coded to return field data rather than the IDs of assets only. Therefore, ExecuteQuery returns up to eight article assets and the data from several of their fields to WireFeedBox.

Use Oracle WebCenter Sites Explorer to open and examine ElementCatalog/OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery if you are interested in this element. Notice that the first line of code in the element is RENDER.UNKNOWNDEPS because there is no way of knowing which assets will be returned so there is no way to log dependencies for them.

When ExecuteQuery is finished, WireFeedBox resumes.

29.4.4 Back to WireFeedBox

WireFeedBox resumes, looping through the list of articles returned by ExecuteQuery, and obtaining a URL for each one by using a RENDER.GETPAGEURL tag.

Because there is no way of knowing which article assets will be returned by ExecuteQuery, there is a RENDER.FILTER tag included in the loop to filter out unapproved assets when the publishing method is Export to Disk:

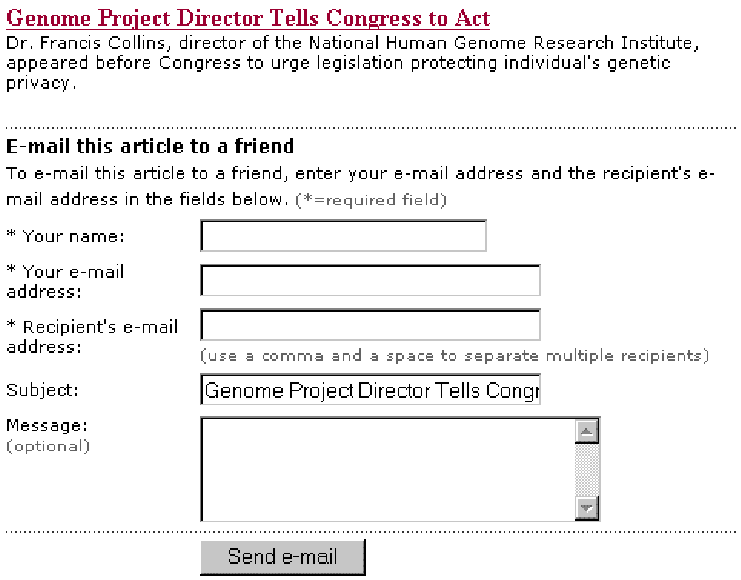
```
<RENDER.FILTER LIST="ArticlesFromWireQuery" LISTVARNAME="ArticlesFromWireQuery"
LISTIDCOL="id"/>
<if COND="ArticlesFromWireQuery.#numRows!=0">
<then>
<LOOP LIST="ArticlesFromWireQuery">
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Article/WireStory"
cid="ArticlesFromWireQuery.id"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
<A class="wirelink" HREF="Variables.referURL"
REPLACEALL="Variables.referURL"><csvar
NAME="ArticlesFromWireQuery.subheadline"/></A><P/>
</LOOP>
</then>
</if>
```

The RENDER.GETPAGEURL tag returns a URL for each article in the list in a variable named referURL. WireFeedBox uses the value from the referURL variable to create links to the articles, using the content from their subheadline fields (which is one of the fields that the Wire Feed query returned) as the hyperlinked text.

Note the use of the REPLACEALL tag as an attribute for this HTML tag. You must use this tag as an HTML attribute when you want to use XML variables in an HTML tag. For more information about REPLACEALL, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*

29.5 Example 5: Displaying an Article Asset Without a Template

Burlington Financial provides an *Email this article to a friend* function. Here is the email form for an article:



Genome Project Director Tells Congress to Act

Dr. Francis Collins, director of the National Human Genome Research Institute, appeared before Congress to urge legislation protecting individual's genetic privacy.

E-mail this article to a friend

To e-mail this article to a friend, enter your e-mail address and the recipient's e-mail address in the fields below. (*=required field)

* Your name:

* Your e-mail address:

* Recipient's e-mail address:

(use a comma and a space to separate multiple recipients)

Subject: Genome Project Director Tells Congr

Message: (optional)

Obviously the Burlington Financial developers do not want the Burlington Financial content providers to assign the email form to an article as the article's Display Style (template). Therefore, there is no Template asset that points to the email element that creates the article email form.

These are the elements used in this example:

- ElementCatalog/BurlingtonFinancial/Article/Full
- ElementCatalog/BurlingtonFinancial/Article/AltVersionBlock
- ElementCatalog/BurlingtonFinancial/Util/EmailFront

This section contains the following topics:

- Section 29.5.1, "First Element: Full"
- Section 29.5.2, "Second Element: AltVersionBlock"
- Section 29.5.3, "Third Element: EmailFront"

29.5.1 First Element: Full

Use Oracle WebCenter Sites Explorer to open and examine the template element for the `Full` template:

`ElementCatalog/BurlingtonFinancial/Article/Full`

This element provides the formatting code for articles when they are displayed in full. It displays the following items:

- A site banner
- The left navigation column
- A collection of related stories
- The text of the article
- A photo for the article
- A link that prints the story
- A link that emails the story

Scroll down past several RENDER.SATELLITEPAGE and CALLELEMENT tags to the following tag:

```
<CALLELEMENT NAME="BurlingtonFinancial/Article/AltVersionBlock"/>
```

29.5.2 Second Element: AltVersionBlock

Use Oracle WebCenter Sites Explorer to open and examine this element:

```
ElementCatalog/BurlingtonFinancial/Article/AltVersionBlock
```

AltVersionBlock is a short element with two RENDER.GETPAGEURL tags. The first RENDER.GETPAGEURL tag obtains the URL for the print version of an article. The second RENDER.GETPAGEURL tag obtains the URL for the email version of the story.

Because the Burlington Financial developers want a dynamic URL for the email version of the story even if the site is a static site, the second RENDER.GETPAGEURL tag uses the DYNAMIC parameter.

Scroll down to the second RENDER.GETPAGEURL tag:

```
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Util/EmailFront"
cid="Variables.asset:id"
c="Article"
DYNAMIC="true"
OUTSTR="referURL"/>
```

AltVersionBlock passes in the pagename for the EmailFront page entry, and a value for c, and cid, and sets the DYNAMIC parameter to true. The tag creates a dynamic URL for the article (even if the publishing method is Export to Disk) and returns it in a variable named referURL, as specified by the OUTSTR parameter.

29.5.3 Third Element: EmailFront

EmailFront is the pagename that AltVersionBlock passes to the RENDER.GETPAGEURL element. Because there is no corresponding template for EmailFront, WebCenter Sites did not create a page entry in the SiteCatalog for EmailFront by default. The Burlington Financial developers created the SiteCatalog entry for this element manually through Oracle WebCenter Sites Explorer.

Use Oracle WebCenter Sites Explorer to open and examine its root element:

```
ElementCatalog/BurlingtonFinancial/Util/EmailFront
```

This element creates a form that displays the first paragraph of the article that the visitor has chosen to email.

First, EmailFront loads the article asset:

```
<ASSET.LOAD TYPE="Article" OBJECTID="Variables.cid" NAME="EmailFront"/>
<ASSET.SCATTER NAME="EmailFront" PREFIX="asset"/>
```

Then it formats several parts of the page before creating the email form. Scroll down to the HTML FORM tag:

```
<FORM NAME="mailform" onSubmit="return checkEmail();" METHOD="POST" ACTION=...
```

EmailFront then calls the LeadSummary page entry to display a summary of the article in the form:

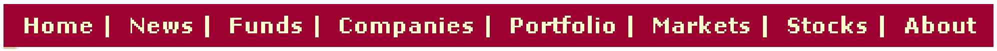
```
<RENDER.SATELLITEPAGE
ARGS_pagename="BurlingtonFinancial/Article/LeadSummary"
ARGS_cid="Variables.cid"
ARGS_ct="Full"
ARGS_p="Variables.p"/>
```

For information about the `LeadSummary` element, see Section 29.1, "Example 1: Basic Modular Design" or use Oracle WebCenter Sites Explorer to open and examine it.

29.6 Example 6: Displaying Site Plan Information

Because the developers of the Burlington Financial sample site used the Site Plan tab in the WebCenter Sites interface to order the basic structure of the Burlington Financial site, they are able to extract information from the `SitePlanTree` table to create navigational features for the site.

For example, the navigation bar at the top of the Burlington Financial home page is created by extracting information about the site's structure from the `SitePlanTree` table.


Home | News | Funds | Companies | Portfolio | Markets | Stocks | About

To extract information from the `SitePlanTree` table, you use the WebCenter Sites `SITEPLAN` tag family.

These are the elements used in this example:

- ElementCatalog/BurlingtonFinancial/Article/Home
- ElementCatalog/Pagelet/Common/SiteBanner
- ElementCatalog/BurlingtonFinancial/Site/TopSiteBar

This section contains the following topics:

- Section 29.6.1, "First Element: Home"
- Section 29.6.2, "Second Element: SiteBanner"
- Section 29.6.3, "Third Element: TopSiteBar"
- Section 29.6.4, "Back to SiteBanner"

29.6.1 First Element: Home

Use Oracle WebCenter Sites Explorer to open and examine the template element for the Home template:

```
ElementCatalog/BurlingtonFinancial/Page/Home
```

Scroll down to the first `RENDER.SATELLITEPAGE` tag:

```
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/Pagelet/Common/SiteBanner"/>
```

29.6.2 Second Element: SiteBanner

The `SiteBanner` pagelet invokes its root element. Use Oracle WebCenter Sites Explorer to open and examine it:

```
ElementCatalog/BurlingtonFinancial/Common/SiteBanner
```

`SiteBanner` gathers the images for the banner (the Burlington Financial logo and an advertising image) and then calls an element that creates the navigational links to the main sections of the site.

Scroll down to this `CALLELEMENT` tag:

```
<CALLELEMENT NAME="BurlingtonFinancial/Site/TopSiteBar"/>
```

29.6.3 Third Element: TopSiteBar

`TopSiteBar` executes, creating the navigational links to the main sections in the site. Use Oracle WebCenter Sites Explorer to open and examine TopSiteBar:

```
ElementCatalog/BurlingtonFinancial/Site/TopSiteBar
```

29.6.3.1 Creating the Link for the Home Page

First, `TopSiteBar` loads the Home page, names it `target`, gets the value from its ID field, and stores that value in the output variable `pageid`:

```
<ASSET.LOAD TYPE="Page" NAME="target" FIELD="name" VALUE="Home" DEPTYPE="exists"/>
<ASSET.GET NAME="target" FIELD="id" OUTPUT="pageid"/>
```

Note that the `ASSET.LOAD` tag changes the dependency type from its default of `exact` to `exists` with the `DEPTYPE` parameter. For a link like this one, a link in a navigational bar, it makes more sense for the dependency to be an `exists` dependency.

Then `TopSiteBar` uses the variable `pageid` to obtain a URL for the Home page from a `RENDER.GETPAGEURL` tag:

```
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Page/Home"
cid="Variables.pageid"
c="Page"
OUTSTR="referURL"/
```

Next `TopSiteBar` extracts the page asset's name from its **Name** field and uses it as the text for the hyperlink:

```
<ASSET.GET NAME="target" FIELD="name" OUTPUT="thepagename"/>
<A class="sectionlinks" HREF="Variables.referURL"
REPLACEALL="Variables.referURL"><csvar NAME="Variables.thepagename"/></A>
```

Note the use of the `REPLACEALL` tag as an attribute for this HTML tag. You must use this tag as an HTML attribute when you want to use XML variables in an HTML tag. For more information about `REPLACEALL`, see Section 4.6.4.4, "Using Variables in HTML Tags."

29.6.3.2 Creating the Links to the Home Page's Child Pages

In the next part of the code, TopSiteBar creates links for the child pages of the Home page. In order to determine the child pages of the Home page, it must first determine the node ID of the Home page.

The node ID of a page asset is different from its object ID:

- You use an object ID to extract information about an asset from asset tables.
- You use a node ID to extract information about a page asset from the `SitePlanTree` table.

First, TopSiteBar determines the node ID of the Home page:

```
<ASSET.GETSITENODE NAME="target" OUTPUT="PageNodeId"/>
```

Then it uses that information to load the Home page as a siteplan node object:

```
<SITEPLAN.LOAD NAME="ParentNode" NODEID="Variables.PageNodeId"/>
```

With the Home page node identified and loaded, TopSiteBar can then obtain the Home page's child nodes, storing them in a list that it names `PeerPages`, and ordering them according to their rank:

```
<SITEPLAN.CHILDREN NAME="ParentNode" TYPE="PAGE" LIST="PeerPages" CODE="Placed"
ORDER="nrank"/>
```

And now `TopSiteBar` loops through all the child nodes at the first level, using the `RENDER.GETPAGEURL` tag to create a URL for the link to each page:

```
<IF COND="IsList.PeerPages=true">
<THEN>
<LOOP LIST="PeerPages">&nbsp;|&nbsp;
<ASSET.LOAD NAME="ThePage" TYPE="Page"
 OBJECTID="PeerPages.oid"/>
<ASSET.GET NAME="ThePage" FIELD="name"
OUTPUT="thepagename"/>
<ASSET.GET NAME="ThePage" FIELD="template"
OUTPUT="pagetemplate"/>
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Page/
Variables.pagetemplate"
cid="PeerPages.oid"
c="Page"
OUTSTR="referURL"/>
<A class="sectionlinks" HREF="Variables.referURL"
REPLACEALL="Variables.referURL">
<csvar NAME="Variables.thepagename"/>
</A>
```

Notice how the page name is constructed in this example. The second `ASSET.GET` statement in the preceding piece of code obtains the name of the page's template from its template field. Here it is again:

```
<ASSET.GET NAME="ThePage" FIELD="template"
OUTPUT="pagetemplate"/>
```

Then, that information is used in the `PAGENAME` parameter passed to the `RENDER.GETPAGEURL` tag:

```
PAGENAME="BurlingtonFinancial/Page/Variables.pagetemplate"/>
```

Therefore, if the template for the page asset is SectionFront, this argument statement passes `pagename=BurlingtonFinancial/Page/SectionFront`. And if the template for the page asset is AboutUs, this argument statement passes `pagename=BurlingtonFinancial/Page/AboutUs`.

29.6.4 Back to SiteBanner

SiteBanner is finished after the call to `TopSiteBar`. The `SiteBanner` element is invoked on each page in the site.

Because `SiteBanner` has a page entry in the `SiteCatalog` table, the results of the navigational bar that `TopSiteBar` creates is cached the first time a visitor requests a page on the Burlington Financial site. This speeds up performance because the site does not have to re-invoke the `TopSiteBar` element for each and every page that the visitor subsequently visits.

29.7 Example 7: Displaying Non-Asset Information

Sometimes you need to render and display information that is not stored as an asset in the WebCenter Sites database. For example, the Burlington Financial site displays today's date on each page. The date is not information that can be stored as an asset.

These are the elements used in this example:

- ElementCatalog/BurlingtonFinancial/Article/Home
- ElementCatalog/Common/ShowMainDate

This section contains the following topics:

- Section 29.7.1, "First Element: Home"
- Section 29.7.2, "Second Element: ShowMainDate"

29.7.1 First Element: Home

Use Oracle WebCenter Sites Explorer to open and examine the template element for the Home template:

```
ElementCatalog/BurlingtonFinancial/Page/Home
```

Scroll down to the third `CALLELEMENT` tag, one that invokes the `ShowMainDate` element.

```
<CALLELEMENT NAME=BurlingtonFinancial/Common/ShowMainDate/>
```

29.7.2 Second Element: ShowMainDate

`ShowMainDate` executes. Use Oracle WebCenter Sites Explorer to open and examine it:

```
ElementCatalog/BurlingtonFinancial/Common/ShowMainDate
```

The main line of code is this one:

```
<span class="dateline"><csvar NAME="CS.Day CS.Mon CS.DDate, CS.Year"/></span>
```

It calculates the date and then returns that value to the `Home` element, which displays it at the top of the page, under the navigation bar and over the main list of stories.

This element performs a simple calculation and then outputs the value into the HTML code that is rendered in the browser window. There are no content assets that it formats or Template assets that use it as a root element. It also has no `SiteCatalog` entry because its result (the date) should be calculated each time the Home page is rendered.

30 Configuring Sites for Multilingual Support

This chapter explains how to configure multilingual support for a site.

This chapter contains the following sections:

30.1 Overview of Configuring a Site for Multilingual Support

When you configure a site for multilingual support, users in that site gain the ability to assign **locale** (language version) designations to assets, and to create translations of assets. You also have the option to create site-specific delivery rules for multilingual content that determine which language versions of assets will be shown on the online site, and what to do if a visitor's request is for a language version in which the content does not yet exist.

This chapter describes important topics you need to consider when planning and implementing the design of your site with respect to multilingual support. The chapter then goes on to describe the procedures necessary to configure your site to support multilingual assets and related features.

This section contains the following topics:

30.1.1 Dimensions

Locale designations in WebCenter Sites are implemented through the concept of **dimensions**. A dimension is an identifier that differentiates assets that are otherwise semantically identical. A locale (such as `en_US` for US English) is thus a type of dimension that differentiates two translations of the same content.

Dimensions are represented by assets of type Dimension. This asset type must be enabled by developers on a per-site basis so that users can create Dimension assets (of subtype Locale), and so that content providers can assign locale designations to assets they wish to translate.

> **Note:** Users cannot create translations of assets that have no locale designation assigned.

Each Dimension asset represents a locale in the site. For example, an `en_US` Dimension asset represents US English, and an `fr_CA` Dimension asset represents Canadian French. When a content asset is assigned a locale, the assignment is recorded in the `assetType_Dim` table for the corresponding asset type.

Publishing content in a given locale requires enabling the locale on the online site, that is, publishing the Dimension asset representing the locale to the delivery system, and including the locale in the site's dimension set. See Section 30.1.2, "Dimension Sets."

30.1.2 Dimension Sets

When you have created your locales, we recommend that you create at least one dimension set. A dimension set is a grouping of dimension assets (locales), which affects the delivery of content to the site visitor in the following ways:

- A dimension set defines which locales are designated as enabled for the online site. In other words, a dimension set determines the languages in which content will be shown to the visitors. For example, one dimension set would contain European languages, another Asian languages, and so on.
- A dimension set defines to filter content based on locale. For example, how to handle content that does not exist in the visitor's preferred language at the time the visitor requests it. If you do not publish a dimension set to the delivery system, locale filtering will not function. For information on locale filtering, see Section 30.2, "Working with Locale Filtering."

> **Note:** Dimension sets do not affect the operation of WebCenter Sites user interfaces.

For locale filtering to function on the online site, you must approve and publish to the delivery system both the DimensionSet assets and the Dimension assets referenced by each dimension set. (Because referenced Dimension assets are dependents of the DimensionSet assets referencing them, they must be approved with their respective DimensionSet assets.)

30.1.3 Cross-Site Multilingual Support

If you are setting up multilingual support in more than one site, you can choose to implement one of the following scenarios:

- **Create the locales, but no dimension sets**

  This option allows content providers to manage content in multiple languages, but does not enable render-time locale filtering. Use this option only if translations in all required languages will exist for each asset from the very beginning.

- **Create the locales and a dimension set, and share them across your sites**

This option provides the simplest way of enabling multilingual support on multiple sites. Sites set up in this way share the properties stored in the dimension set (locales enabled for display on the online site, the locale filtering method, and, if applicable, the fallback hierarchy (the path the Hierarchical filter traverses when looking up asset translations at render time). See Section 30.2.2.3, "The Hierarchical Filter" for details. If you are creating a bare-bones site that you will replicate into multiple target sites, it is best to share your locales to the target sites.

- **Create separate locales and dimension sets for each site**

  This option affords the most flexibility, at the cost of increased configuration complexity. Sites set up this way benefit from the fact that properties such as locale filter type or fallback hierarchy can be tailored to each site.

  > **Note:** While creating duplicate Dimension assets to represent the same language in multiple sites is possible, it is not recommended, as it introduces unnecessary complexity.

- **A mixture of the latter two options**

  This option provides the right balance between flexibility and configuration complexity. As a possible best practice, you would create a pool of unique locales, share the locales required by each site from that pool, and share or create dimension sets for each site as needed.

30.1.4 Master Assets, Translations, and Multilingual Sets

When an asset is assigned a locale for the first time, it gains **master**, or **dimension parent**, status. Master status allows the formation of a multilingual set (a group of assets whose content is semantically identical, but exists in different languages. (Note that this is not the same as a dimension set; a dimension set only affects the online site and the way assets are rendered.)

> **Note:** The terms master asset and dimension parent are equivalent. Master asset is displayed in WebCenter Sites's user interfaces; dimension parent is used in the *Oracle Fusion Middleware WebCenter Sites Tag Reference*, database table names (such as *assetType*_`DimP`), and element code.

For example, when you designate an Article asset as US English (`en_US`), and create translations of it in whichever locales are enabled in the site (such as French (`fr_CA`) and German (`de_DE`)), the translations point to their dimension parent (the US English asset) to indicate they are semantically equivalent to the master and one another.

When you create a translation of an asset with master status, WebCenter Sites copies the asset and assigns the locale of your choice to the copy. You then enter the translated content and save the translation as a new asset.

At this point, the source asset and its translation are linked into a multilingual set, and the translation adopts the source asset as its master, or dimension parent. Any member of the set that is not the master can be given master status; however, only one set member at a time can be the master.

The linking is accomplished through the *assetType*_`DimP` table for the asset type. The table stores the following information:

- ID of the master (dimension parent) asset
- ID of the translation asset
- ID of the locale dimension asset assigned to that translation

Even though WebCenter Sites interfaces allow you to initiate the creation of a translation from either the master asset or any of its existing translations, all translations in the multilingual set always point to the dimension parent (master) asset.

Note: If a locale-aware asset is being revision-tracked, changes to asset locale data (such as locale designation or master status) do not generate a new version of the asset.

30.1.5 Translations and Asset Relationships

The way asset relationships are handled when an asset is translated is summarized in the following table:

Relationship Type	Behavior
Associations	When an asset containing associations is translated, all assets associated with the source asset are automatically associated with the translation. You then have the choice to translate the associated assets and associate the translated versions with the translated parent asset.
Collections	When you create a translation of a Collection asset, the new Collection asset retains the member assets of the source asset. You then have the choice to translate the member assets and place the translated versions in the new Collection asset, replacing the member assets carried over from the old collection.
Static Lists Recommendations	When you create a new language version of a Static Lists recommendation, the new Recommendation asset retains the member assets of the source asset. You then have the choice to translate the member assets and place the translated versions in the new Recommendation asset, replacing the member assets carried over from the old collection.
Dynamic Lists Recommendations	Since Dynamic Lists recommendations are populated by element code, they are not affected.
Related Items Recommendations	When an asset containing Related Items associations is translated, all assets associated with the source asset are automatically associated with the translation. You then have the choice to translate the associated assets and associate the translated versions with the translated parent asset.
Asset-Type Attributes	When an asset containing associations through asset-type attributes is translated, all assets associated with the source asset are automatically associated with the translation. You then have the choice to translate the associated assets and associate the translated versions with the translated parent asset.
Embedded Links	Embedded links are not affected. When an asset containing embedded links is translated, you must manually update the links to point to the corresponding translations of the linked content (if such translations exist).

For information on handling asset relationships at render time, see Section 30.2, "Working with Locale Filtering."

30.1.6 Approval Dependencies

An approval dependency exists between two assets when editing one of the assets causes the other's approval status to change. The following table summarizes the approval dependencies affecting localized assets:

Dependency	Effect on Asset Approval
An Exists dependency exists between a localized asset and the Dimension asset representing the assigned locale.	To approve a localized asset for publishing, the corresponding Dimension asset must also be approved.
In a multilingual set, an Exists dependency exists between the master asset and each translation linked to it.	When you create the first translation of an asset, you must approve both the asset and its translation. To approve a translation, you must also approve the corresponding master asset, unless the master asset has already been approved. You must reapprove all members of the set if: ■ You add a new translation to, or delete an existing translation from the set. ■ You edit the set's master asset. ■ You designate another member of the set as the master.
An Exists dependency exists between a DimensionSet asset and the Dimension assets representing the locales enabled in that dimension set.	To approve a dimension set, the corresponding Dimension assets must also be approved.

30.2 Working with Locale Filtering

Ideally, a multilingual site would be complete in terms of its content before it is launched. That is, all assets and their relatives would exist in all of the required languages. However, in many situations, this is not so. In such cases, you would use a locale filter to decide at render time which language version of an asset to show on the site depending on the circumstances and the language preference specified by the visitor. For example, you could let the business logic decide what to do if a requested asset does not exist in the requested language.

By using locale filtering, you can also spread editorial work over time by allowing content providers to create the required translations after the original content is published to the online site. Locale filtering allows the site to automatically pick up the missing translations as soon as they are published to the delivery system.

Keep in mind that locale filtering introduces additional load on the delivery system. The amount of additional load depends on the complexity of the filtering logic.

This section contains the following topics:

- Section 30.2.1, "Handling Asset Relationships Through Locale Filtering"
- Section 30.2.2, "Included Locale Filters"
- Section 30.2.3, "Custom Locale Filters"
- Section 30.2.4, "Compositional Dependencies"
- Section 30.2.5, "Adding Filtering Support to Your Site"

30.2.1 Handling Asset Relationships Through Locale Filtering

The way you choose to implement locale filtering will have an influence on how asset relationships on your site are structured, and vice versa, depending on the way you want the online site to behave.

You can choose to implement one of the following options:

- **Maintain different asset relationship trees for each locale**

  When rendering assets, this model renders whatever assets are associated with the requested asset.

  For example, if an asset exists in English and French, and each version has a unique set of associated assets, each version is rendered with its respective associated assets. Filtering is only used to look up and deliver a version of the requested asset matching the language preference specified by the visitor; the associated assets are expected to already have been translated into all required languages.

  This model allows for completely independent content for each language. It is used in the First Site II sample site.

- **Use the same asset relationship tree for all locales**

  When rendering assets, this model substitutes the associated assets of the requested asset with the assets associated with a specific language version of the requested asset, regardless of the language preference specified by the visitor.

  For example, if an asset exists in English and French, each version has a unique set of associated assets, and the visitor specified French as their language preference, filtering will look up and deliver the French version of the requested asset, but it will substitute the associated assets of the English version in place of those of the French version (assuming the language version from which filtering is to derive associations is English).

  This model ensures consistent content across all languages.

- **A mixture of the two models**

  Allows for the greatest amount of flexibility and customization for your site. The optimal proportion between the two models will depend on the intended behavior of your site.

30.2.2 Included Locale Filters

WebCenter Sites ships with the following locale filters:

- The Simple Filter
- The SimpleLookup Filter
- The Hierarchical Filter (also known as the Fallback filter)

You also have the option to implement custom locale filters, if desired. See Section 30.2.3, "Custom Locale Filters" for more information on custom filters.

Note that depending on the type of filter you choose to implement, the assets being filtered must satisfy one, or both of the following conditions:

- Assets must have locale designations assigned. Assets without locale designations will be ignored by locale filters.

- Assets that are translations of one another must be linked into multilingual sets (that is, designated as translations of one another through a master asset). Otherwise, the filters will not be able to perform the necessary translation lookups.

30.2.2.1 The Simple Filter

The Simple filter is a possible choice for a site that should only be rendered in one language, but whose content exists in multiple languages. The filter checks the following:

- Whether the requested asset is in the language specified by the visitor
- Whether the locale of the asset is listed in the site's dimension set

If both conditions are met, the filter passes the asset to the template for rendering; otherwise, nothing is rendered.

The Simple filter has the least impact on delivery system performance, but increases the amount of editorial work that needs to be done, as assets must exist in the required language versions or they will not be displayed on the online site.

30.2.2.2 The SimpleLookup Filter

The SimpleLookup filter is ideal for a site that should only be rendered in one language, but whose content may exist in multiple languages, and for which there is no guarantee that all of the necessary translations exist at render time. The filter checks the following:

- Whether the requested asset is in the language specified by the visitor
- Whether the locale of the asset is listed in the site's dimension set

If the requested asset is not in the visitor's preferred language, the filter looks up a suitable replacement by checking the asset's translations. If the filter finds a matching translation, it passes it to the template; otherwise, nothing is rendered. (The filter will also return nothing if the locale of the translation is not included in the site's dimension set.)

This filter offers a reasonable balance between performance and functionality. While the lookup queries slightly increase the load on the delivery system, the amount of editorial work done to create assets can be reduced, as the required translations can be created after the original content is published to the online site. The lookup mechanism will pick up the missing translations as soon as they are published to the delivery system.

The FirstSiteII sample site uses this filter as the default locale filter.

30.2.2.3 The Hierarchical Filter

The Hierarchical filter checks whether the locale of the requested asset matches the locale requested by the visitor. If the locales do not match, the filter checks the asset's translations to see if a suitable replacement exists. If the filter finds a matching translation, it passes it to the template; otherwise, it substitutes translations of the requested asset according to the fallback hierarchy you set up when you configure the site's dimension set. The fallback hierarchy determines which language versions the filter should substitute for the requested asset, and in what order.

For example, consider the following hierarchy:

- · **en_US** (US English)
  - · **de_DE** (German)

    - · **de_CH** (Swiss German)
    - · **de_AT** (Austrian German)
- · **fr_FR** (French)
  - · **fr_BE** (Belgian French)
  - · **fr_CA** (Canadian French)
- · **en_UK** (British English)

In our example, when the visitor requests an asset in Swiss German (de_CH), the filter looks up the asset's translations and if it finds a Swiss German version of the asset, it passes that version to the template. If the filter cannot find a Swiss German version, it falls back to the next best locale in the hierarchy path, German (de_DE). If, in turn, no German translation exists, the filter follows the path specified in the hierarchy until it reaches the top of the tree. If no match is found in the process, nothing is rendered.

Note that the above example describes a situation in which the visitor specifies a single preferred language. If the user specifies multiple preferred languages (in most cases, in the form of an ordered list), the filter attempts to find a match in the fallback hierarchy for the visitor's most preferred language. If no match is found, the filter checks the next language on the visitor's list, until a match in the fallback hierarchy is found. When that happens, the filter attempts to substitute translations of the requested asset by tracing a path from the matching locale to the top of the fallback tree, as described earlier.

For example, if the user's preferred languages are Japanese, French, and English (in that order), the filter attempts to locate Japanese in its fallback hierarchy. Since Japanese is not in the hierarchy, the filter then attempts to locate French. French is in the hierarchy, so the filter traces a path from French to the root node of the tree, and attempts substitution according to that path, as illustrated by the example earlier in this section.

While powerful and convenient, the hierarchical filter has the following drawbacks:

- The additional database queries run by the filter tax the performance of the delivery system. To minimize the performance hit, editorial work should be done to ensure that content exist in as many of the required languages as possible, so that the filter's activity is minimized. (You may also choose to use a different filter.)
- Control over which assets to display on the online site is put exclusively in the hands of the site developer or administrator. This is because the filter follows the fallback tree configured in the dimension set, rather than the preference order specified by the site visitor (assuming the site is set up to accept multiple language preferences from each visitor).

30.2.3 Custom Locale Filters

Depending on the design of your site, you may decide to create custom filters. For example, your site design might call for a hierarchical (fallback) filter that favors the locale priority specified by the visitor, rather than the one defined in the dimension set. In such cases, a field in the Edit form for the DimensionSet asset lets you specify a custom filter class.

30.2.4 Compositional Dependencies

If you decide to incorporate locale filtering on your site, you must account for the additional compositional dependencies that are introduced as a result. Compositional dependencies determine how pages are cached on your delivery system.

30.2.4.1 Asset Lookup Chain

When using locale filtering to look up a translation of an asset, the following factors determine how pages are cached, based on which assets are loaded during the lookup process:

- The filtering logic employed
- The page and asset from which the lookup request originates
- The language preference specified by the visitor

A cached page containing the requested asset is dependent on all assets loaded during the lookup process. Thus, if an asset that is loaded during the lookup process is modified, the affected page is flushed from the cache.

For example, consider the SimpleLookup filter and the following multilingual set:

· **en_US** (master)

- · **fr_FR** (translation)
- · **de_DE** (translation)

If a visitor requests a page containing the French version, but the visitor's language preference is German, the lookup chain is as follows:

fr_FR > **en_US** > **de_DE**

In this example, all three assets are loaded, because the filter must first load the master asset linked to the French version, and then use the master asset to look up the German version. Thus, if any of these three assets is modified, the affected page is flushed from the cache.

If, on the other hand, the user requested the US English version, which is the master asset of the set, then the lookup chain would be shorter:

fr_FR > en_US

In such case, the French and US English versions are loaded, but the German version is not. Thus, modifying the German version would not cause the corresponding page to be flushed from the cache, but modifying the French or US English versions would.

For a detailed explanation of the lookup mechanisms employed by locale filters included with WebCenter Sites, see Section 30.2.2, "Included Locale Filters."

Section 30.2.4.2, "Caching Rules" explains the caching rules applicable to multilingual assets.

30.2.4.2 Caching Rules

Once the translation lookup occurs and the affected page is cached, the page is flushed from the cache whenever one of the following occurs:

- A new language is added to the multilingual set
- A translation that was part of the lookup chain when the page was rendered is edited
- A translation that is a member of the multilingual set is deleted
- The set's master asset is edited
- Another member of the set is designated as the master

30.2.5 Adding Filtering Support to Your Site

To add support for locale filtering to your site, you must modify the templates and element code used on your site.

When the template code fetches an asset via the asset's c/cid values, the locale filter executes its business logic on the incoming c/cid values and returns the resulting c/cid values (or nothing) to the template for rendering.

The structure of your site will influence how you implement locale filtering in your templates, and vice versa. It will also determine the behavior of your site in different scenarios.

For example, imagine five articles, each existing in two languages, en (English), and fr (French). The articles would be a1en, a1fr, a2en, a2fr, and so on. We can decide to put these articles into a collection and implement locale filtering in one of the following ways:

- Create an English collection, c1en, and assign all of the English articles to it. This way, before we render the collection, we would simply filter the c/cid of the c1en asset, then render its children without filtering their c/cid values, because we trust the c1en collection to be in a single language.
- Create a multilingual collection (without assigning a locale to it) and add the articles in whatever languages are desired. Then, when rendering each article, filter the article c/cid values so that the article is rendered in the locale specified by the visitor.

The rest of this section provides code examples based on the FirstSiteII sample site. We recommend that you examine the FirstSiteII code to get an idea of how it multilingual support.

30.2.5.1 Adding Filtering to Templates

Usually, you would place your filter code into a utility element and call the element at the top of the template to process the c/cid values.

The following example shows how the FSIILayout template calls the filter code stored in the FSIICommon/Multilingual/Filter element asset via the render:lookup tag:

```
<%-- Execute the Dimension filter to look up the translated asset that corresponds
to the locale that the visitor requested. --%>
<render:lookup site='<%=ics.GetVar("site")%>' varname="Filter" key="Filter"
match=":x" tid='<%=ics.GetVar("tid")%>' />
<render:callelement elementname='<%=ics.GetVar("Filter")%>' scoped="global"/>
```

30.2.5.2 Obtaining and Maintaining a Visitor's Locale Preference

For filtering to work, you have to allow the visitor to specify a preferred language (locale). This preference must then be propagated throughout the entire site (that is, passed to all the templates).

The example below shows how this is accomplished in the FSIIWrapper element. The last section of this example shows how the locale variable is set by taking the value from the session variable (earlier in the example, we ensure that the session variable exists).

```
<%-- The session variable locale refers to the id of the dimension with the
subtype of Locale that specifies which language the site is to be rendered in.
Users can select the locale of their choice from a menu on every page of the
site, and once selected, it is stored in session. A default locale is mapped to
this CSElement and is set if it has not already been set. --%>
```

```
<ics:if condition='<%=ics.GetSSVar("preferred_locale") == null%>'>
<ics:then>
<render:lookup site='<%=ics.GetVar("site")%>' varname="default:locale:name"
key='DefaultLocale' ttype="CSElement" tid='<%=ics.GetVar("eid")%>' match=":x"/>
<asset:load name="defaultLocale" type="Dimension" field="name"
value='<%=ics.GetVar("default:locale:name")%>'/>
<asset:get name="defaultLocale" field="id" output="default:locale:id"/>
<ics:setssvar name="preferred_locale"
value='<%=ics.GetVar("default:locale:id")%>'/>
</ics:then>
</ics:if>
<%-- Call the wrapped child page. There is no need to look up the template or to
enable any special PageBuilder functionality, so we can use the
render:satellitepage tag in this situation. --%>
<render:satellitepage pagename='<%=ics.GetVar("childpagename")%>'
packedargs='<%=ics.GetVar("packedargs")%>'>
<render:argument name='c' value='<%=ics.GetVar("c")%>'/>
<render:argument name='cid' value='<%=ics.GetVar("cid")%>'/>
<render:argument name='p' value='<%=ics.GetVar("p")%>' />
<render:argument name="locale" value='<%=ics.GetSSVar("preferred_locale")%>'/>
</render:satellitepage>
```

30.2.5.3 Filtering Search Results

If your online site contains search functionality, you may choose to filter the search results returned to the visitor, based on the visitor's language preference.

The following example shows how the `Page/SearchDetailView` template filters an `IList` of search results so that the query can go against all languages but only return the desired results:

```
<%-- look up the dimension set and filter the ProductList results --%>
<asset:load name="GlobalDimSet" type="DimensionSet" field="name"
value='<%=ics.GetVar("GlobalDimSet")%>' />
<dimensionset:filter name="GlobalDimSet" tofilter="ProductList"
list="ProductList">
<dimensionset:asset assettype="Dimension" assetid='<%=ics.GetVar("locale")%>'/>
</dimensionset:filter>
```

For more information, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*. Additionally, examine the code in the FirstSiteII sample site to see how it implements multilingual support.

30.3 Planning Multilingual Support for a Site

Before you start configuring a site for multilingual support, make sure you have satisfied the following prerequisites. It is best to make these decisions in agreement with your site administrators.

1. Determine how many languages to initially implement on your site (or sites), based on your organization's content management needs. You can either:
   - Build a site (or sites) to support a single language initially, and add support for additional languages as the need arises.
   - Plan ahead for all the languages you expect to incorporate across all of your sites and create the appropriate locales in advance.

2. Determine whether you will share existing locales and dimension sets to the new site (or sites), or create separate ones. For more information, see Section 30.1.3, "Cross-Site Multilingual Support."

3. Decide how asset relationships are going to be handled at render time with respect to locales, and choose the locale filtering method(s) appropriate to your decision. The choices you make will have to strike a balance between the desired levels of automation, delivery system performance, and editorial workload, and are thus best made in agreement with your site administrators. For more information, see Section 30.2, "Working with Locale Filtering." Note the following:

   – Different filtering methods provide different levels of automation at the cost of a performance hit on the delivery system. The more complex the filter, the higher the performance hit. For example, the SimpleLookup filter provides better performance than the Hierarchical filter.

   – Depending on the filtering method you implement, editorial work on site content can be spread over time, as translations can be created after the original content is created and published to the live site. The filtering logic you implement will decide what to do content that does not yet exist in the required language versions.

4. If you are converting a monolingual site to a multilingual site, obtain the element code you will use to assign the default locale to the assets in the site. Sample code based on the FirstSiteII sample site is provided in the section, Section 30.4.10.1, "Sample Element Code for Bulk-Assigning a Default Locale."

> **Note:** When replicating a site containing multilingual sets, make sure the master assets are available on the target site (by either sharing or copying). Otherwise, the set members will no longer be linked as translations of each other on the target site.

30.4 Configuring Multilingual Support for a Site

This section describes the procedures necessary to configure a site for multilingual support.

This section contains the following topics:

- Section 30.4.1, "Configuration Quick Reference"
- Section 30.4.2, "Enabling the Dimension and DimensionSet Asset Types"
- Section 30.4.3, "Enabling the Locale Subtype of the Dimension Asset Type"
- Section 30.4.4, "Creating a Locale"
- Section 30.4.5, "Sharing a Locale to Another Site"
- Section 30.4.6, "Creating and Configuring a Dimension Set"
- Section 30.4.7, "Sharing a Dimension Set to Another Site"
- Section 30.4.8, "Configuring a Locale Filter"
- Section 30.4.9, "Configuring the Fallback Hierarchy of the Hierarchical Filter"
- Section 30.4.10, "Bulk-Assigning a Default Locale to Assets in a Site"

30.4.1 Configuration Quick Reference

This section provides an overview of the steps necessary to configure multilingual support for a site. Use this list as a quick reference during the configuration process.

To configure multilingual support for a site

1. Make the necessary decisions and preparations as described in Section 30.3, "Planning Multilingual Support for a Site."

2. Enable the Dimension and DimensionSet asset types on the site. For instructions, see Section 30.4.2, "Enabling the Dimension and DimensionSet Asset Types."

3. Enable the Locale subtype of the Dimension asset type on the site. For instructions, see Section 30.4.3, "Enabling the Locale Subtype of the Dimension Asset Type."

4. Create or share the desired locales. For instructions, see the following sections:

   – For creating new locales, see Section 30.4.4, "Creating a Locale."

   – For sharing existing locales, see Section 30.4.5, "Sharing a Locale to Another Site."

   For help in determining whether to create new locales or share existing ones, see Section 30.1.3, "Cross-Site Multilingual Support."

5. Create or share a dimension set. For instructions, see the following sections:

   – For creating a new dimension set, see Section 30.4.6, "Creating and Configuring a Dimension Set."

   – For sharing an existing dimension set, see Section 30.4.7, "Sharing a Dimension Set to Another Site."

   For help in determining whether to create a new dimension set or share an existing one, see Section 30.1.3, "Cross-Site Multilingual Support."

6. (Optional) If you are converting an existing monolingual site to a multilingual site, execute element code that assigns a default locale to each asset in the site. For instructions, see Section 30.4.10, "Bulk-Assigning a Default Locale to Assets in a Site." The section includes sample code which you can customize for your site.

7. Modify the templates used in the site to include support for the locale filter you selected when you configured the dimension set. For an overview of the process, see Section 30.2.5, "Adding Filtering Support to Your Site."

   Note: In addition to locale filtering, you will have to implement the following site functionality:

   - Allow the visitor to specify their language preference
   - Propagate the visitor's language preference throughout the site (by passing it to all templates on the site)
   - Maintain the visitor's language preference for the duration of the session.

30.4.2 Enabling the Dimension and DimensionSet Asset Types

Before you can create Dimension and DimensionSet assets in your site, you must enable the corresponding asset types and subtypes. This procedure describes how to enable the Dimension and DimensionSet asset types on your site. The next procedure describes how to enable the Locale subtype of the Dimension asset type on your site.

To enable the Dimension and DimensionSet asset types on your site

1. Log in to the WebCenter Sites interface and select the site in which you want to enable the asset types.
2. Select the **Admin** interface.
3. In the tree, select the **Admin** tab.
4. In the **Admin** tab, drill down the following hierarchy:

   a. Expand the **Sites** node.

   b. Under the **Sites** node, expand the node corresponding to the desired site.

   c. Under the desired site node, expand the **Asset Types** node.

   d. Under the **Asset Types** node, double-click the **Enable** node.

   WebCenter Sites displays the Enable Asset Types form.
5. In the Enable Asset Types form, select the check boxes next to the **Dimension** and **DimensionSet** asset types.
6. Click **Enable Asset Types**.

   WebCenter Sites displays the Start Menu Selection form.
7. In the Start Menu Selection form, select all of the check boxes, and click **Enable Asset Types**.

   WebCenter Sites displays a message confirming the asset types have been enabled for the site.

30.4.3 Enabling the Locale Subtype of the Dimension Asset Type

Before you can assign locales to assets in your site, you must enable the Locale subtype of the Dimension asset type on your site.

To enable the Locale subtype of the Dimension asset type on your site

1. Log in to the WebCenter Sites interface and select the site in which you want to enable the subtype.
2. Select the **Admin** interface.
3. In the tree, select the **Admin** tab.
4. In the **Admin** tab, drill down the following hierarchy:

   a. Expand the **Asset Types** node.

   b. Under the **Asset Types** node, expand the **Dimension** node.

   c. Under the **Dimension** node, double-click the **Subtypes** node.

   WebCenter Sites displays the Subtypes for Asset Type: Dimension form.
5. In the form, click the **Edit** (pencil) icon next to the **Locale** subtype.

   WebCenter Sites displays the Edit Dimension Subtype: Locale form.
6. In the **Sites** field, **Ctrl+click** the site in which you want to enable the Locale subtype.

> **Note:** You must **Ctrl+click** the name of your site in order to keep the existing site selections intact; if you simply click on the site name, other selected sites (if any) will be deselected.

7. Click **Save**.

30.4.4 Creating a Locale

To add a new locale to your site, create a Dimension asset of subtype Locale representing the desired locale, by performing the following steps:

> **Note:** If the locale you want to create designates a language that is already represented by a Dimension asset in another site in your WebCenter Sites system, share the existing Dimension asset representing that language to your current site instead to avoid redundancy.

To create a locale

1. In the button bar, click **New**.
2. In the list of asset types, click **New Dimension**.
3. WebCenter Sites displays the New Dimension form.
4. In the New Dimension form, do the following:
   a. In the **Name** field, enter a descriptive name for the locale. It is recommended that you use the following convention as a best practice:

      `xx_YY`

      where:

      – `xx` is the two-letter ISO 639-1 language code (for example, `fr` for French)

      – `YY` is the two-letter ISO country code (for example, `CA` for Canada)

      To complete the above example, the name `fr_CA` would denote Canadian French.

      a. In the **Description** field, enter a description of the language this locale represents.

      b. In the Subtype drop-down list, select **Locale**.

      c. Click **Save**.

30.4.5 Sharing a Locale to Another Site

To share a locale to another site, you must share the corresponding Dimension asset.

To share a locale to another site

1. Log in to the WebCenter Sites interface and select the site containing the Dimension assets for the locales you want to share.
2. Select the Admin interface.
3. Find the desired Dimension asset and open its Inspect form:

  a. In the button bar, click **Search**.
  b. In the list of asset types, click **Find Dimension**.
  c. Enter the desired search criteria (if any), and click **Search**.
  d. In the list of search results, navigate to the desired asset and click its name.

     WebCenter Sites opens the asset in the Inspect form.

4. In the action bar, select **Share Dimension**.

   WebCenter Sites displays the Share Dimension form.

5. In the Share Dimension form, select the check boxes next to the sites to which you want to share the Dimension asset. (To share the asset to all sites on your WebCenter Sites system, select the All Sites check box.)
6. Click **Save Changes**.
7. WebCenter Sites displays a message confirming the asset is now available in the sites you selected.

30.4.6 Creating and Configuring a Dimension Set

To create and configure a new dimension set

1. Add the Dimension assets (locales) you want to include in the dimension set to your Bookmarks by doing the following:
   a. In the button bar, click **Search**.
   b. In the Search form, click **Find Dimension**.
   c. Enter the desired search criteria (if any) and click **Search**.
   d. In the list of search results, navigate to the desired **Dimension** assets and select their check boxes.
   e. Click **Add to My Bookmarks**.
2. Create and configure the dimension set by doing the following:
   a. In the button bar, click **New**.
   b. In the New asset list, click **New DimensionSet**.

      WebCenter Sites displays the New DimensionSet form.

   c. In the **Name** field, enter a descriptive name for the dimension set.
   d. In the tree, select the **Bookmarks** tab.
   e. In the **Bookmarks** tab, select a locale you want to add to the dimension set and click **Add Selected Items**. Repeat this step for each additional locale you want to add.
   f. In the **Dimension Filter Class** field, select the desired locale filter type. The **Advanced** option lets you specify a custom filter class.

      For more information on locale filter types, see Section 30.2, "Working with Locale Filtering."

   g. (Optional) If you selected **Advanced** in step 2, enter the name of the custom filter class into the text box that appears.
   h. When you are finished, click **Save Changes**.

- **i.** (Optional) If you selected the **Hierarchical** filter in step 2, complete the steps in Section 30.4.9, "Configuring the Fallback Hierarchy of the Hierarchical Filter" to configure the filter's fallback hierarchy.

30.4.7 Sharing a Dimension Set to Another Site

To share a dimension set to another site, you must share the corresponding DimensionSet asset.

To share a dimension set to another site

1. Log in to WebCenter Sites and select the site containing the DimensionSet asset you want to share.
2. Select the **Admin** interface.
3. Find the desired DimensionSet asset and open its Inspect form:
   - **a.** In the button bar, click **Search**.
   - **b.** In the list of asset types, click **Find DimensionSet**.
   - **c.** Enter the desired search criteria (if any), and click **Search**.
   - **d.** In the list of search results, navigate to the desired asset and click its name.

     WebCenter Sites opens the asset in the Inspect form.
4. In the action bar, select **Share DimensionSet**.

   WebCenter Sites displays the Share DimensionSet form.
5. In the Share DimensionSet form, select the check boxes next to the sites to which you want to share the DimensionSet asset. (To share the asset to all sites on your WebCenter Sites system, select the All Sites check box.)
6. Click **Save Changes**.
7. WebCenter Sites displays a message confirming the asset is now available in the sites you selected.

30.4.8 Configuring a Locale Filter

Usually, you configure the locale filter when you create the dimension set for your site. To make changes to the locale filter configuration in an existing dimension set, do the following:

To configure a locale filter

1. Find the dimension set whose locale filter you want to configure and open in the Inspect form:
   - **a.** In the button bar, click **Search**.
   - **b.** In the Search form, click **Find DimensionSet**.
   - **c.** Enter the desired search criteria (if any) and click **Search**.
   - **d.** In the list of search results, navigate to the desired asset and click its name.

     WebCenter Sites opens the asset in the Inspect form.
2. In the **Dimension Filter Class** field, select the option next to the desired filter type. The **Advanced** option lets you specify a custom filter class.

For more information on locale filters, see Section 30.2, "Working with Locale Filtering".

3. (Optional) If you selected **Advanced** in step 2, enter the name of the custom filter class into the text field that appears.
4. Click **Save Changes**.
5. (Optional) If you selected the **Hierarchical** filter in step 2, complete the steps in Section 30.4.9, "Configuring the Fallback Hierarchy of the Hierarchical Filter" to configure the filter's fallback hierarchy.

30.4.9 Configuring the Fallback Hierarchy of the Hierarchical Filter

If you selected the Hierarchical (Fallback) locale filter when configuring your dimension set, perform the following steps to configure the filter's fallback hierarchy:

To configure the fallback hierarchy of the Hierarchical locale filter

1. If you plan to add new locales or rearrange existing locales in the fallback hierarchy, add the Dimension assets representing the locales to be included in the hierarchy to your Bookmarks by doing the following:
   a. In the button bar, click **Search**.
   b. In the Search form, click **Find Dimension**.
   c. Enter the desired search criteria (if any) and click **Search**.
   d. In the list of search results, navigate to the desired Dimension assets and select their check boxes.
   e. Click **Add to My Bookmarks**.
2. Find and open in the Inspect form the dimension set that contains the Hierarchical filter you want to configure:
   a. In the button bar, click **Search**.
   b. In the Search form, click **Find DimensionSet**.
   c. Enter the desired search criteria (if any) and click **Search**.
   d. In the list of search results, navigate to the desired DimensionSet asset and click its hyperlinked name.

      WebCenter Sites displays the DimensionSet asset in the Inspect form.
3. Configure the fallback hierarchy:

   Note: When configuring the fallback hierarchy, note the following:

   - A locale can appear in the fallback hierarchy only once.
   - If you need to delete a locale from the hierarchy, click the **Delete** (trash can) icon next to the locale node.
   - If you need to change the position of a locale in the hierarchy, delete it, then add it under the desired parent node.

   a. In the **Dimension Filter Class** field, click **Configure Locale Hierarchy**.

      WebCenter Sites displays the Configure Locale Hierarchy form.
   b. In the Configure Locale Hierarchy form, click **Edit**.

WebCenter Sites displays an editable version of the form.

c. In the tree, select the **Bookmarks** tab.

d. (Optional) If the hierarchy is empty, select in the **Bookmarks** tab the locale you want to designate as the top node of the fallback hierarchy, then click **Add Selected Items**.

e. Select a parent node for the locale you want to add to the hierarchy.

   If you are building a hierarchy from scratch, your only choice will be the top-level node you added in step d.

   When building your hierarchy, keep in mind the direction in which the fallback process occurs (from most-specific to least-specific; that is, towards the root node of the tree).

f. In the **Bookmarks** tab, select the locale you want to appear under the parent node you selected in step e, then click **Add Selected Items**.

g. Repeat steps e and f for each additional locale you want to add to the hierarchy.

h. When your fallback hierarchy is complete, click **Save Changes**.

30.4.10 Bulk-Assigning a Default Locale to Assets in a Site

If you are converting a monolingual site to a multilingual site, you must assign a default locale to all assets in the site. The fastest way to accomplish this is to execute an element that assigns the default locale to the assets.

> **Note:** For your convenience, sample element code for this procedure is provided Section 30.4.10.1, "Sample Element Code for Bulk-Assigning a Default Locale." The sample code is intended as an example, and will have to be customized for your site.

To bulk-assign a default locale to assets in a site

1. Create a CSElement asset to hold the element code that will assign a default locale to your assets.

2. Create a SiteEntry asset that references the CSElement asset you created in step 1.

3. Call the SiteEntry asset you created in step 2 in a URL, as follows:

```
http://<host>:<port>/<context>/ContentServer?pagename=
<siteentry_name>
```

where:

- `<host>` is the host of your WebCenter Sites system
- `<port>` is the port number on which WebCenter Sites is listening for connections
- `<context>` is the application context root assigned to the WebCenter Sites application.
- `<siteentry_name>` is the name of the SiteEntry asset you created in step 2.

When the element code completes execution, check to make sure that your assets have the desired locale assigned. If not, check the element code for possible errors.

30.4.10.1 Sample Element Code for Bulk-Assigning a Default Locale

This section contains sample element code written for the FirstSiteII sample site. The code does the following:

1. Creates a Dimension asset named `en_US` to represent your default locale designation within the site (US English in this example).
2. Assigns this default locale to all Page assets within the site.

> **Note:** The code in this section is provided as an example. If you decide to use it, be sure to customize it for your site. Test the code before deploying it; no error checking is included in this example.

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="user" uri="futuretense_cs/user.tld"%>

<cs:ftcs>

<%-- Record dependencies for the SiteEntry and the CSElement --%>
<ics:if condition='<%=ics.GetVar("seid")!=null%>'>
<ics:then>
<render:logdep cid='<%=ics.GetVar("seid")%>' c="SiteEntry"/>
</ics:then>
</ics:if>

<ics:if condition='<%=ics.GetVar("eid")!=null%>'>
<ics:then>
<render:logdep cid='<%=ics.GetVar("eid")%>' c="CSElement"/>
</ics:then>
</ics:if>

<%-- log in as firstsite--%>
<user:login username="firstsite" password="firstsite"/>
<%-- create the Dimension asset (this can be done manually) --%>
<asset:create name="en_US" type="Dimension"/>
<asset:setsubtype name="en_US" value="Locale"/>
<asset:set name="en_US" field="name" value='en_US'/>
<asset:set name="en_US" field="description" value='US English'/>
<%-- enter your site's pubid below --%>
<ics:setvar name="primarypubid" value="1112198287026"/>
<asset:save name="en_US"/>

<%-- look up the id of the Dimension asset you just created --%>
<asset:get name="en_US" field="id" output="en_US.id"/>

<%-- get a list of all Content_C assets in the site, and assign a dimension to
each of them --%>
<asset:list type="Content_C" list="allContentAssets" pubid="1112198287026"/>
<ics:listloop listname="allContentAssets">
<ics:listget listname="allContentAssets" fieldname="id" output="id"/>
<asset:load type="Content_C" objectid='<%=ics.GetVar("id")%>' name="tempName"
editable="true"/>
<asset:adddimension name="tempName" dimensionid='<%=ics.GetVar("en_US.id")%>'/>
<asset:save name="tempName"/>
</ics:listloop>
```

```
</cs:ftcs>
```

31

User Management on the Delivery System

WebCenter Sites provides authentication functionality through the `USER` tags, user profile management through the `DIR` tags, and enforces security on database tables and rendered pages through access control lists (ACLs). You use these user management and security mechanisms to manage users and control visitor access on your distribution system and on your WebCenter Sites development and management systems.

This chapter contains the following sections:

- Section 31.1, "The Directory Services API"
- Section 31.2, "Controlling Visitor Access to Your Online Sites"
- Section 31.3, "Creating Login Forms"
- Section 31.4, "Creating User Account Creation Forms"
- Section 31.5, "Visitor Access in the Burlington Financial Sample Site"

31.1 The Directory Services API

The Directory Services API enables your WebCenter Sites system to connect to directory servers that contain authentication information, user information, and so on.

WebCenter Sites delivers three directory services plug-ins, one of which was installed when your WebCenter Sites systems were installed:

- The WebCenter Sites directory services plug-in, which uses the native WebCenter Sites user management tables; that is, the `SystemUsers` and `SystemUserAttrs` tables
- The LDAP plug-in, which actually supports any JNDI server
- The NT plug-in, which retrieves user credentials and login name from the NT directory but gets all other user information from the `SystemUserAttrs` table

The plug-in is installed during the installation of your WebCenter Sites systems and it is configured by setting properties in the `dir.ini` file. For information about configuring your user management setup, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

This section contains the following topics:

- Section 31.1.1, "Entries"
- Section 31.1.2, "Hierarchies"
- Section 31.1.3, "Groups"

- Section 31.1.4, "Directory Services Tags"
- Section 31.1.5, "Directory Operations"
- Section 31.1.6, "Error Handling"
- Section 31.1.7, "Troubleshooting Directory Services Applications"

31.1.1 Entries

A directory entry is a named object with assigned attributes, in particular, user and group type entries:

- A user type object has a distinguished name and a set of attributes such as commonname, username, password and email.
- A group type object, similar to a WebCenter Sites ACL, also has a distinguished name and a set of attributes.

Names reflect the hierarchy in which they are associated; to ensure portability across directory implementations, names should be treated as opaque strings.

31.1.2 Hierarchies

Some directory databases organize entries using a hierarchical structure. With WebCenter Sites's directory services API, an entry's attributes and its place in the hierarchy are distinct. As a result, retrieving an entry's attributes does not yield information about its children.

Support for hierarchies depends on the underlying directory implementation; for example, LDAP directories support a hierarchical structure, while WebCenter Sites's native directory database does not support a hierarchical structure.

To ensure portability across directory implementations, your code should not assume support for hierarchical data.

Group hierarchies do not affect internal WebCenter Sites permissions.

31.1.3 Groups

WebCenter Sites's directory services API does not enforce referential integrity. When you delete a user with the directory tags, your application must ensure that group memberships are also deleted, by first removing the user from the groups that he is associated with.

When a member is added to a group, the JNDI implementation always builds a fully distinguished name for the value of the `uniquemember` attribute, regardless of the name passed into the `addmember` tag.

31.1.4 Directory Services Tags

WebCenter Sites delivers the `DIR` tag family, with both XML and JSP versions, that you can use to invoke the Directory Services API.

The `DIR` tags are as follows:

Tag	Description
`DIR.ADDATTRS` `dir:addattrs`	Adds attributes to an existing entry (which can be either a user or a group).

Tag	Description
`DIR.ADDGROUPMEMBER` `dir:addgroupmember`	Adds a member to a group (usually a user).
`DIR.CHILDREN` `dir:children`	Retrieves the child entries for a specified parent in a list variable.
`DIR.CREATE` `dir:create`	Creates a directory entry.
`DIR.DELETE` `dir:delete`	Deletes a directory entry.
`DIR.GETATTRS` `dir:getattrs`	Gets the attribute values for a specified entry in a list variable.
`DIR.GROUPMEMBERS` `dir:groupmembers`	Lists the members of a specified group.
`DIR.GROUPMEMBERSHIPS` `dir:groupmemberships`	Lists all the groups that an entry (either a group or a user) belongs to.
`DIR.LISTUSERS` `dir:listusers`	Returns a list of all the users in the directory.
`DIR.REMOVEATTRS` `dir:removeattrs`	Deletes an attribute value for an entry.
`DIR.REMOVEGROUPMEMBER` `dir:removegroupmember`	Removes an entry from a group.
`DIR.REPLACEATTRS` `dir.replaceattrs`	Replaces the value of an attribute for an entry (either a user or a group).
`DIR.SEARCH` `dir:search`	Searches the directory for entries who match the specified search criteria.

Regardless of whether the directory is implemented with LDAP, WebCenter Sites only, or any other directory server, the code you write with the `DIR` tags is very similar.

For more information about these tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*. For code samples, see Section 31.1.5.4, "Directory Services Code Samples."

31.1.5 Directory Operations

Some of the WebCenter Sites Directory Services tags write information to the database. If your database administrators will be handling all of the website's write operations, such as adding user information to the database, restrict use of the directory tags to read-only operations. This policy avoids synchronization issues with third-party directory administration tools.

The read-only operations are presented in this section. Operations are performed using the credentials and read permissions of the currently authenticated user.

31.1.5.1 Searching

Due to limitations in some directory servers, search is not allowed from the top organizational level. To avoid portability issues, always specify the context attribute on the DIR.SEARCH tag.

31.1.5.2 Lookup

Looking up a user generally involves two steps:

1. Call `DIR.SEARCH` on the `userid` to get the entry name.
2. Call `DIR.GETATTRS` to get the attributes of the user in question.

31.1.5.3 Listing Users

It is recommended that you use one of the following three methods to list users:

- For small user databases, use the `DIR.LISTUSERS` tag, which recursively lists all users under the `peopleParent` property. This tag is inefficient on large user databases.
- For large user databases, use the `DIR.CHILDREN` tag to walk the hierarchy. The `DIR.CHILDREN` tag is best used for group types and not for user types.
- For user databases with a flat hierarchy, narrow results with a search

31.1.5.4 Directory Services Code Samples

The following JSP code sample illustrates some possible directory operations:

```
<%
String sMainTestUserName = "ContentServer";
String sMainTestUserPW="FutureTense";

String sPeopleParent = ics.GetProperty("peopleparent", "dir.ini", true);
String sGroupParent = ics.GetProperty("groupparent", "dir.ini", true);
String sUsername = ics.GetProperty("username", "dir.ini", true);
String sCommonName = ics.GetProperty("cn", "dir.ini", true);
IList mylist;
%>

<user:su username='<%=sMainTestUserName%>' password='<%=sMainTestUserPW%>'>

<H2>List All Users</H2>

<ics:clearerrno/>
<dir:listusers list='mylist'/>
<br>
<b>dir:listusers errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br><ics:listget listname='mylist' fieldname='NAME'/>
</ics:listloop>

<H2>Look Up the ContentServer User by Username</H2>

<ics:clearerrno/>
<dir:search list='mylist' context='<%=sPeopleParent%>'>
<dir:argument name='<%=sUsername%>' value='ContentServer'/>
</dir:search>
<br><b>dir:search errno: <ics:getvar name='errno'/></b>

<%
```

```
mylist = ics.GetList("mylist");
if(mylist.numRows() != 1) {
out.print("<br>Error finding entry.");
}
mylist.moveTo(1);
ics.SetVar("ContentServerDn", mylist.getValue("NAME"));
%>

<H2>Show ContentServer Attributes</H2>

<ics:clearerrno/>
<dir:getattrs list='mylist'
name='<%=ics.GetVar("ContentServerDn")%>'/>
<br><b>dir:getattrs errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br>
<ics:listget listname='mylist' fieldname='NAME'/>=
<ics:listget listname='mylist' fieldname='VALUE'/>
</ics:listloop>

<H2>Show Group Memberships for ContentServer</H2>

<ics:clearerrno/>
<dir:groupmemberships name='<%=ics.GetVar("ContentServerDn")%>'
list='mylist'/>
<br><b>dir:groupmemberships errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br>
<ics:listget listname='mylist' fieldname='NAME'/>
</ics:listloop>

<H2>Lookup the SiteGod Group by CommonName</H2>

<ics:clearerrno/>
<dir:search list='mylist' context='<%=sGroupParent%>'>
<dir:argument name='<%=sCommonName%>' value='SiteGod'/>
</dir:search>
<br><b>dir:search errno: <ics:getvar name='errno'/></b>

<%
mylist = ics.GetList("mylist");
if(mylist.numRows() != 1) {
out.print("<br>Error finding entry.");
}
mylist.moveTo(1);
ics.SetVar("SiteGodDn", mylist.getValue("NAME"));
%>

<H2>Show SiteGod Attributes</H2>

<ics:clearerrno/>
<dir:getattrs list='mylist' name='<%=ics.GetVar("SiteGodDn")%>'/>
<br>
<b>dir:getattrs errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br>
<ics:listget listname='mylist' fieldname='NAME'/>=
<ics:listget listname='mylist' fieldname='VALUE'/>
</ics:listloop>
```

```
<H2>Show SiteGod Group Members</H2>

<ics:clearerrno/>
<dir:groupmembers name='<%=ics.GetVar("SiteGodDn")%>' list='mylist2'/>
<br>
<b>dir:groupmembers errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist2'>
<br>
<ics:listget listname='mylist2' fieldname='NAME'/>
</ics:listloop>

<H2>Children of groupparent </H2>

<ics:clearerrno/>
<dir:children name='<%=sGroupParent%>' list='mylist'/>
<br>
<b>dir:children errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br>
<ics:listget listname='mylist' fieldname='NAME'/>
</ics:listloop>

</user:su>
```

31.1.6 Error Handling

Any of the directory tags can cause a range of directory errors to be set. See the *Oracle Fusion Middleware WebCenter Sites Tag Reference* for a comprehensive list of directory services error messages.

Your directory services code should handle every one of the error codes listed for a given tag call. This is necessary to support the J2EE JNDI interface.

31.1.7 Troubleshooting Directory Services Applications

The first step in troubleshooting directory services applications is to check the error log (in the WebCenter Sites log file).

You enable directory services logging by setting the `log.filterLevel` property (found in the `logging.ini` property file). There are seven levels of error messages that you can view:

- `fatal`, which logs fatal level messages
- `severe`, which logs severe and fatal level messages
- `error`, which logs error and fatal level messages
- `warning`, which logs warning and fatal level messages
- `info`, which logs warning, error, severe, and fatal level messages
- trace, which logs trace messages
- detail, which logs all types of messages

During troubleshooting, `trace` is the most verbose setting, and as a result, has the highest performance impact.

Directory services log entries use the following format:

```
[<timestamp>][Directory-<severity>-<errno>]
[<class>:<method>][<message>][<session id>]
```

For example:

```
[Jan 17, 2002 1:49:44 PM][Directory-T]
[BaseFactory:instantiateImplementation(ICS,String,Class[],
Object[])][Instantiating:com.openmarket.directory.common.Factory]
[PEccxyF1Ueh7zYvjNgg4D6bqZzf0llfWMaiBimIN9H1Z9KomDcPy]
```

The previous message is a trace (T), and thus has no associated `errno` value.

For information about error logging, see Chapter 10, "Error Logging and Debugging."

A common problem for LDAP implementations is incorrectly specified permissions on the directory server. If the error log indicates a permission problem, ensure that the authenticated user has permissions to execute the requested operation by checking the permission settings on the directory server. Try logging into the directory server directly (outside of WebCenter Sites) and performing the same action to ensure that permissions are correctly set. After checking the log and permissions, you can often resolve a configuration error by examining the property files.

For property descriptions and values, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

31.2 Controlling Visitor Access to Your Online Sites

WebCenter Sites manages users through access control lists (ACLs). By using ACLs, you can restrict access to tables in the WebCenter Sites database and the rendered pages served on your sites by WebCenter Sites.

If you design an online site where visitors log in with user names and passwords, you can associate registered visitors with one or more ACLs.

When a visitor first visits a site, WebCenter Sites creates a session and implicitly logs in the visitor as the standard default user, **DefaultReader**. The identity of a visitor is updated (and any associated ACLs go into effect) when a `USER.LOGIN` command is used and the visitor is authenticated against a password.

This section contains the following topics:

- Section 31.2.1, "ACL Tags"
- Section 31.2.2, "User Tags"
- Section 31.2.3, "WebCenter Sites and Encryption"

31.2.1 ACL Tags

WebCenter Sites provides a set of access control list tags (both XML and JSP versions) that you can use to create ACLs. You can use either the WebCenter Sites interface on the delivery system or the WebCenter Sites ACL tags to create the ACLs that you need for your visitor accounts on your delivery system.

The following table lists the ACL tags:

Tag	Description
`ACL.CREATE` `acl:create`	Creates an ACL
`ACL.DELETE` `acl:delete`	Deletes an ACL

Tag	Description
`ACL.GATHER` `acl:delete`	Gathers fields into an ACL
`ACL.GET` `acl:get`	Copies a field from an ACL
`ACL.LIST` `acl:list`	Retrieves a list of ACLs
`ACL.LOAD` `acl:load`	Loads an ACL
`ACL.SAVE` `acl:save`	Saves an ACL
`ACL.SCATTER` `acl:scatter`	Scatters a field from an ACL
`ACL.SET` `acl:set`	Sets a field in an ACL

For more information:

- ACL tags: See the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.
- ACLs in general: See the *Oracle Fusion Middleware WebCenter Sites: Administrator's Guide*.

31.2.2 User Tags

WebCenter Sites also provides the following `USER` tags (both XML and JSP versions) that you use on pages that log users in and out.

The `USER` tags are as follows:

Tag	Description
`USER.LOGIN` `user:login`	Logs a user in.
`USER.LOGOUT` `user:logout`	Logs a user out.
`USER.SU` `user:su`	Logs the user in as a specific user in order to perform an operation such as creating an account or edit a user profile.

For more information about these tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

31.2.3 WebCenter Sites and Encryption

WebCenter Sites includes a default key for encrypting passwords and other sensitive information. You can specify your own encryption key by using the `Utilities` class `encryptString` method. See the *Oracle Fusion Middleware WebCenter Sites Java API Reference* for information about Java methods that deal with encryption.

WebCenter Sites also supports Secure Sockets Layer (SSL), which allows encryption of information going to and from your web servers. For more information about WebCenter Sites and SSL, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

31.3 Creating Login Forms

This section provides simple code samples that illustrate how to code login forms that prompt a visitor to log in and then authenticate the user name and password.

This example presents code from the following elements:

- `PromptForLogin`: XML element that displays a form that requests a username and password
- `Login`: XML element that authenticates the username and password combination passed to it from the `PromptForLogin` element

This section contains the following topics:

- Section 31.3.1, "Prompt for Login (PromptForLogin.xml)"
- Section 31.3.2, "Root Element for the Login Page"

31.3.1 Prompt for Login (PromptForLogin.xml)

The `PromptForLogin` element displays a form that asks a visitor to enter two pieces of information: username and password.

The code that creates the form follows:

```
<DIV ALIGN="center">
<FORM ACTION="ContentServer" METHOD="post">
<INPUT TYPE="hidden" NAME="pagename" VALUE="CSGuide/Security/Login"/>

<TABLE CELLPADDING="5" CELLSPACING="5">

<TR>
<TH ALIGN="right">Name</TH>
<TD><INPUT TYPE="text" NAME="username" SIZE="16"/></TD>
</TR>

<TR>
<TH ALIGN="right">Password</TH>
<TD><INPUT TYPE="password" NAME="password" SIZE="16"/></TD>
</TR>

<TR>
<TD>&nbsp;</TD>
<TD ALIGN="center"><INPUT TYPE="submit" NAME="doit" VALUE="Login"/></TD>
</TR>

</TABLE>

</FORM>
</DIV>
```

The visitor fills in the form and clicks the **Submit** button. The information gathered in the form and the page name of the `Login` page (see the first `input type` statement, above) is sent to the browser. The browser sends the page name to WebCenter Sites, WebCenter Sites looks it up in the SiteCatalog table and then invokes that page entry's root element.

31.3.2 Root Element for the Login Page

There can only be one root element for a WebCenter Sites page (that is, an entry in the SiteCatalog table). The root element for the Login page is the Login.xml element.

The Login element attempts to log the visitor in and then to authenticate the visitor by using the USERISMEMBER tag to determine whether the visitor has any of the required ACLs.

This element does the following:

- Logs the visitor in with the USER.LOGIN tag and checks to see if there was an error.
- Sets a variable list that holds a list of ACLs to compare the visitor's credentials against.
- Checks the visitor's ACLs assignments against the list variable with the USERISMEMBER tag

The following sample code authenticates the visitor:

```
<SETVAR NAME="errno" VALUE="0"/>

<USER.LOGIN USERNAME="Variables.username" PASSWORD="Variables.password"/>

<IF COND="Variables.errno=0">
<THEN>
<H3>Welcome <CSVAR NAME="Variables.username"/></H3>

<!-- Next, create a variable named aclToCheck and set it to hold the ACLs that the
visitor needs to progress any further on the site. This variable can be set to one
ACL or to a comma-separated list of ACLs, as in this example-->

<SETVAR NAME="aclToCheck" VALUE="ContentCustomer,SiteGod"/>
<SETVAR NAME="errno" VALUE="0"/>

<USERISMEMBER GROUP="Variables.aclToCheck"/>
<IF COND="Variables.errno=1">
<THEN>
<P>You are a member of the at least one of the following acls:
<CSVAR NAME="Variables.aclToCheck"/></P>
</THEN>
    <ELSE>
<P>Sorry, you are not a member of any of the following acls:
<CSVAR NAME="Variables.aclToCheck"/></P>
    </ELSE>
    </IF>

</THEN>
<ELSE>

    <H3>Sorry, can't find your credentials.</H3>

</ELSE>
</IF>
```

31.4 Creating User Account Creation Forms

This section provides simple code samples that illustrate how to code forms that prompt a visitor to register (obtain a user account) and then create a user account for that visitor.

This example presents code from the following elements:

- `PromptForNew Account`: XML element that displays a form that requests the visitor to enter the user name and password that he or she would like to use
- `CreateAccount`: JSP element that creates the new account

This section contains the following topics:

- Section 31.4.1, "PromptForNewAccount"
- Section 31.4.2, "Root Element for the CreateAccount Page"

31.4.1 PromptForNewAccount

The `PromptForNewAccount` element displays a form that prompts the visitor to enter a user name and password and to re-enter the password to confirm it.

Here's the code that creates the form:

```
<div align="center">
<h3>Create a New Account</h3>
<FORM ACTION="ContentServer" METHOD="post">
<input type="hidden" name="pagename" value="CSGuide/Security/CreateAccount"/>

<table cellpadding="5" cellspacing="5">

<tr>
<th align="right">Pick a username</th>
<td><input type="text" name="username" size="16"/></td>
</tr>

<tr>
<th align="right">Pick a password</th>
<td><input type="password" name="password" size="16"/></td>
</tr>

<tr>
<th align="right">Confirm your new password</th>
<td><input type="password" name="confirm_password" size="16"/></td>
</tr>

<tr>
<td>&nbsp;</td>
<td><input type="submit" name="doit" value="Create Account"/></td>
</tr>
</table>

</FORM>
</div>
```

The visitor fills in the form and clicks the **Submit** button. The information gathered in the form and the page name of the `CreateAccount` page (see the first `input type` statement, above) is sent to the browser.

The browser sends the page name to WebCenter Sites, WebCenter Sites looks it up in the `SiteCatalog` table and then invokes that page entry's root element.

31.4.2 Root Element for the CreateAccount Page

There can only be one root element for a WebCenter Sites page (that is, an entry in the SiteCatalog table). The root element for the CreateAccount page is the CreateAccount.jsp element.

Only someone with SiteGod or ContentEditor ACLs can create a new user account. Because of this restriction, the CreateAccount element does the following:

- Logs the visitor in as a privileged user, without the knowledge of the visitor.
- Creates the account.
- Assigns the new user the appropriate ACLs (every user must belong to at least one ACL)

Here's the code that creates the new user account:

```
<SETVAR NAME="errno" VALUE="0"/>

<!-- switch temporarily to a privileged user -->
<!-- The username and password for the privileged user should be encrypted in a
property file. You should obtain them from the property file, decrypt them, then
pass them it. For this example, they are hard-coded. -->

<USER.SU USERNAME="jumpstart" PASSWORD="jumpstart">

<USERISMEMBER GROUP="UserEditor"/>
<IF COND="Variables.errno!=1">
<THEN>
<h3>An error has occurred creating the account (no UserEditor
privs).  Contact the webmaster</h3>
</THEN>
<ELSE>
    <IF COND="Variables.password!=Variables.confirm_password">
<THEN>
<h3>Your passwords do not match. Click the Back button and
try again.</h3>
</THEN>
<ELSE>

<!-- Get the parameters from the property file -->

<ics.getproperty name="username" file="dir.ini"
output="unameattr"/>
<ics.getproperty name="password" file="dir.ini"
output="passattr"/>

<!-- create the user's name in the right format for the dir tags -->

<ics.getproperty name="peopleparent" file="dir.ini"
output="namebase"/>
<name.makechild context="Variables.namebase" output="iname">
<name.argument name="Variables.unameattr"
value="Variables.username"/>
</name.makechild>

<!-- create the user -->

<dir.create name="Variables.iname">
<dir.argument name="Variables.unameattr"
value="Variables.username"/>
```

```
<dir.argument name="Variables.passattr"
value="Variables.password"/>

<!-- additional parameters can be added here but for the example we won't -->
<!-- In particular, if you are using LDAP, you will have to spin through and set
the values of the properties in the property requiredPeopleAttrs in dir.ini. -->

        </dir.create>

        <IF COND="Variables.errno=0">
        <THEN>

<!-- give the new user an acl and format it correctly for dir.addgroupmember -->

<ics.getproperty name="groupparent" file="dir.ini"
output="groupparent"/>
<ics.getproperty name="cn" file="dir.ini" output="cn"/>
<name.makechild context="Variables.groupparent"
output="groupid">
<name.argument name="Variables.cn" value="Browser"/>
</name.makechild>

<!-- add the acl -->

<dir.addgroupmember name="Variables.groupid"
member="Variables.iname"/>

<IF COND="Variables.errno=0">
<THEN>
                <h3>Success!</h3>
</THEN>
<ELSE>
<h3>User created but error adding user to group.
Contact the webmaster</h3>
</ELSE>
</IF>

</THEN>
<ELSE>
<h3>Error creating user!  Contact the webmaster.</h3>
</ELSE>
</IF> <!-- create success check -->

</ELSE>
</IF> <!-- passwords match -->

</ELSE>
</IF>

</USER.SU>
```

31.5 Visitor Access in the Burlington Financial Sample Site

The Burlington Financial sample site includes a membership component that uses elements to sign up new members and log in existing members.

These visitor registration elements are not robust enough for use on a real-world website, but can give you a starting point for your own designs. For example, Burlington Financial has sample visitor account screens, allowing visitors to register and set their own preferences, but does not use this information to restrict visitor access to certain web pages, or to make recommendations based on a member's profile.

This section contains the following topics:

- Section 31.5.1, "Membership Table"
- Section 31.5.2, "Users and Passwords"
- Section 31.5.3, "Member Accounts"
- Section 31.5.4, "Membership Processing Elements"

31.5.1 Membership Table

Burlington Financial uses a table named `bfmembers` to implement the membership component. (This table is created for the sample site when it is installed. None of the WebCenter Sites modules or products use this table.) Although the membership elements add a row to the `bfmembers` database table for each new registered member's profile information, they do not add a row to the `SystemUsers` table.

31.5.2 Users and Passwords

There is one generic user, BFUser, for all Burlington Financial members. The name and password are the same (`BFUser/BFUser`) and should not be changed. The member login code in Burlington Financial sets a session variable for the visitor, which is then used to identify that visitor.

Because Burlington Financial is a sample site, members' passwords are stored in the `bfmembers` table as plain text. A real website would store passwords in encrypted format. Burlington also grants Visitor, BFMember, and Browser ACL privileges to entries added to the `bfmembers` table.

31.5.3 Member Accounts

There are currently no elements for managing the Burlington Financial accounts. If you want to try editing or deleting members' accounts, use Oracle WebCenter Sites Explorer to modify the `bfmembers` table.

31.5.4 Membership Processing Elements

There are several elements that handle processing requests for Burlington Financial members. If you have installed the Burlington Financial sample site, you can use Oracle WebCenter Sites Explorer to open and examine them. All but one is located here:

```
ElementCatalog/BurlingtonFinancial/Util/Account
```

The `AccountAccessScript` element is located here:

```
ElementCatalog/BurlingtonFinancial/Util
```

AccountAccess.xml

This is a page template that calls pagelet elements for the header, footer, navigation menu, and the account content.

AccountAccessScript.xml

This file contains three JavaScript routines (`checkSignupForm`, `checkProfileForm`, and `checkLoginForm`) that perform basic error checking on the HTML account forms. This is called from `Login.xml`, `Profile.xml`, and `SignUp.xml` elements.

Benefits.xml

This page calls the `Block.xml` article template to render an article of text about the Burlington Financial site. On a real website, the article would contain benefits information.

Login.xml

This page displays the login screen for registered members and calls `LoginPost.xml` to handle the login form input. It also calls `Benefits.xml`, and `SignUp.xml` for non-members.

LoginPost.xml

This pagelet element calls `ProcessLogin.xml` to display a login message.

Profile.xml

This page displays an editable profile form if the visitor is registered, or else calls `SignUp.xml` if the visitor is not registered.

ProcessLogin.xml

This pagelet element displays an appropriate login message, depending on whether the visitor who submitted the form is a registered member.

SignUp.xml

This page displays the sign-up screen for non-registered visitors and calls `the catalogmanager` to add a row to the `bfmembers` table for a new user, or to update the `bfmembers` table for an existing user.

32 The HelloAssetWorld Sample Site

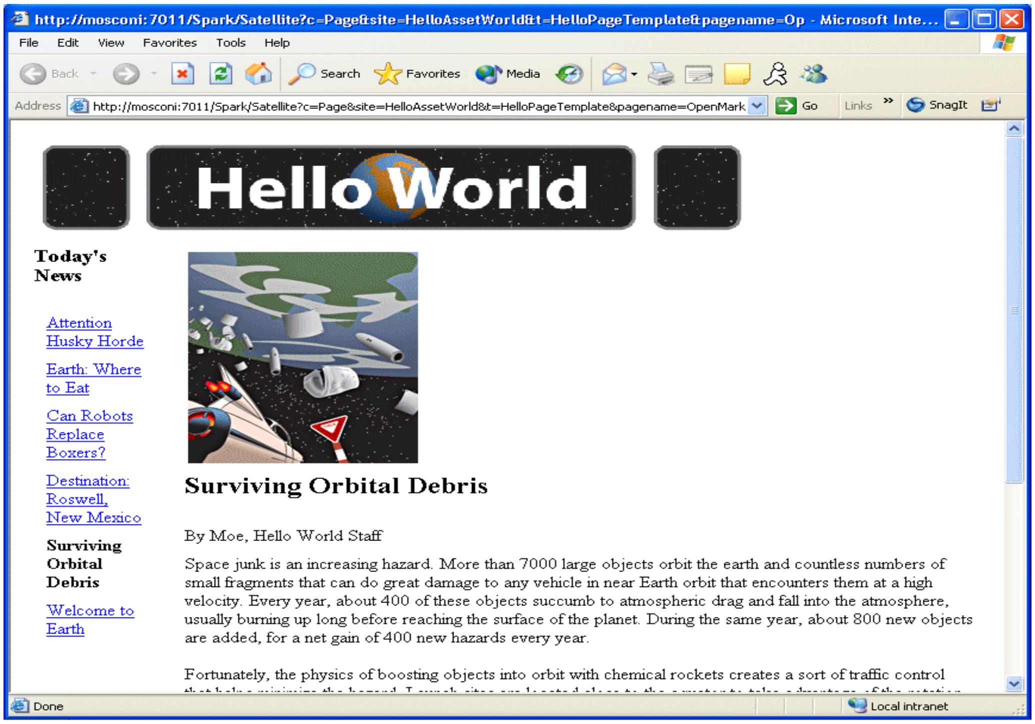
The HelloAssetWorld sample site is a sample website built using WebCenter Sites. It is meant to provide a simple entry point into the process of building a website with WebCenter Sites. This chapter focuses on the steps that a developer would take in creating this simple website; further information on HelloAssetWorld's configuration and users are in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

This chapter contains the following sections:

- Section 32.1, "Overview of the HelloAssetWorld Sample Site"
- Section 32.2, "Modified Asset Types"
- Section 32.3, "HelloAssetWorld Templates"
- Section 32.4, "The HelloQuery Asset"

32.1 Overview of the HelloAssetWorld Sample Site

The HelloAssetWorld site has a simple design; it is composed of one page, as shown in the following screen capture:


http://mosconi:7011/Spark/Satellite?c=Page&site=HelloAssetWorld&t=HelloPageTemplate&pagename=Op - Microsoft Inte...
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Media
Address
http://mosconi:7011/Spark/Satellite?c=Page&site=HelloAssetWorld&t=HelloPageTemplate&pagename=OpenMark
Go
Links
SnagIt
Hello World
Done
Local intranet

Today's News

Attention Husky Horde

Earth: Where to Eat

Can Robots Replace Boxers?

Destination: Roswell, New Mexico

Surviving Orbital Debris

Welcome to Earth

Surviving Orbital Debris

By Moe, Hello World Staff

Space junk is an increasing hazard. More than 7000 large objects orbit the earth and countless numbers of small fragments that can do great damage to any vehicle in near Earth orbit that encounters them at a high velocity. Every year, about 400 of these objects succumb to atmospheric drag and fall into the atmosphere, usually burning up long before reaching the surface of the planet. During the same year, about 800 new objects are added, for a net gain of 400 new hazards every year.

Fortunately, the physics of boosting objects into orbit with chemical rockets creates a sort of traffic control

The stories that appear on this web page change depending upon the article that you choose to view, but the layout of the page remains the same.

To view the HelloAssetWorld sample site yourself, enter the following URL into your web browser:

```
http://server_
name/servlet/ContentServer?pagename=HelloAssetWorld/Page/HelloPageTemplate
```

This section contains the following topics:

- Section 32.1.1, "HelloAssetWorld Templates"
- Section 32.1.2, "HelloAssetWorld Asset Types"

32.1.1 HelloAssetWorld Templates

The HelloAsset World web page is composed of three templates:

- The HelloArticle template, which displays the article that you select and that article's associated image.
- The HelloCollection template, which displays hyperlinks to a collection of articles that you can view.
- The HelloPage template, which is the containing page. It displays the HelloAssetWorld banner graphic and calls the HelloArticle and Hello Collection templates.

32.1.2 HelloAssetWorld Asset Types

The asset types used in the HelloAssetWorld site are modified from asset types used in the Burlington Financial sample site, described in Chapter 33, "The Burlington Financial Sample Site" A list of the asset types used in HelloAssetWorld follows:

- The Page asset type, which performs several functions:
  - It allows uses to create a site hierarchy by **placing** the page. Placing a page gives it an entry in the SitePlanTree table and allows it to be viewed under the Placed Pages node of the Site Plan Tree.
  - It lets you associate assets of various types with it. For example, you can associate Collection assets and Article assets with a Page asset.

    The instance of the Page asset type used in the HelloAssetWorld site, HelloPage, has a collection called HelloCollectionHello associated with it. The Page asset type is a core asset type which is provided with WebCenter Sites, and has not been modified.
- The HelloArticle asset type, which contains an article. HelloArticle assets can have HelloImage assets associated with them. The HelloArticle asset type has been modified from the Article asset type that is provided with Burlington Financial.
- The HelloImage asset type, which contains an image. The HelloImage asset type has been modified from the ImageFile asset type that is provided with Burlington Financial.
- The Query asset type, which queries the database and returns the HelloArticle assets that display on the website. This is a core asset type, which is provided with WebCenter Sites and has not been modified.

- The Collection asset type, which orders the results that the query asset returns. This is a core asset type, which is provided with WebCenter Sites and has not been modified.
- The Template asset type, which renders the various asset types. This is a core asset type, which is provided with WebCenter Sites and has not been modified.

32.2 Modified Asset Types

Most of the asset types used in the HelloAssetWorld sample site are core asset types, and hence cannot be modified. The HelloArticle and HelloImage asset types, however, are simplified versions of the Article and ImageFile asset types that are provided with WebCenter Sites. Each asset type has a new asset descriptor file that is based on the asset descriptor files for the Article and ImageFile asset types. The simplified asset descriptor files are shown in the following sections.

This section contains the following topics:

- Section 32.2.1, "The HelloArticle Asset Type"
- Section 32.2.2, "The HelloImage Asset Type"

32.2.1 The HelloArticle Asset Type

The ASSET tag, shown in the following code, is the standard opening for all asset descriptor files. Among other things, it names the new asset type and specifies the asset's DEFDIR, the default directory where uploaded items are stored.

```
<ASSET NAME="HelloArticle" DESCRIPTION="HelloArticle"
MARKERIMAGE="/Xcelerate/data/help16.gif" PROCESSOR="4.0"
DEFDIR="c:\FutureTense\Storage\HelloArticle">
```

The next code section create a text field for the article's headline.

```
<PROPERTIES>
<PROPERTY NAME="Headline" DESCRIPTION="Headline">
<STORAGE TYPE="VARCHAR" LENGTH="255" />
<INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="255" REQUIRED="YES" />
<SEARCHFORM DESCRIPTION="Headline contains" TYPE="TEXT" WIDTH="48" MAXLENGTH="255"
/>
</PROPERTY>
```

The next code section create a text field for the article's byline.

```
<PROPERTY NAME="Byline" DESCRIPTION="Byline">
<STORAGE TYPE="VARCHAR" LENGTH="100" />
<INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="100" REQUIRED="YES" />
<SEARCHFORM DESCRIPTION="Byline contains" TYPE="TEXT" WIDTH="48" MAXLENGTH="100"
/>
</PROPERTY>
```

The following code create an upload field where content editors and authors can type in the content of an article's body. This content will be stored in the DEFDIR specified in the ASSET tag.

```
<PROPERTY NAME="urlBody" DESCRIPTION="Body">
<STORAGE TYPE="VARCHAR" LENGTH="2000" />
<INPUTFORM TYPE="TEXTAREA" COLS="300" ROWS="300" REQUIRED="YES" />
</PROPERTY>
</PROPERTIES>
```

```
</ASSET>
```

32.2.2 The HelloImage Asset Type

The ASSET tag, shown below, is the standard opening for all asset descriptor files. Among other things, it names the new asset type and specifies the asset's DEFDIR, the default directory where uploaded items are stored.

```
<ASSET NAME="HelloImage" DESCRIPTION="HelloImage" MARKERTEXT="*" PROCESSOR="4.0"
DEFDIR="c:\FutureTense\Storage\HelloImage">
```

Then, the next code section creates an upload field for the image file.

```
<PROPERTIES>
<PROPERTY NAME="urlfile" DESCRIPTION="Image File">
<STORAGE TYPE="VARCHAR" LENGTH="255"/>
<INPUTFORM TYPE="UPLOAD" WIDTH="36" REQUIRED="NO" LINKTEXT="HelloImage"/>
</PROPERTY>
```

The following code creates a drop-down select and specifies how the search field for mimetypes will appear on the Advanced Search form. The SQL statement supplied as a value for the SQL parameter for the INPUTFORM tag queries the database to supply mimetypes for the text of the drop-down.

```
<PROPERTY NAME="mimetype" DESCRIPTION="Mimetype">
<STORAGE TYPE="VARCHAR" LENGTH="36"/>
<INPUTFORM TYPE="SELECT" SOURCETYPE="TABLE" TABLENAME="mimetype"
OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype" SQL="SELECT mimetype,
description FROM mimetype WHERE keyword = 'image' AND isdefault = 'y'"
INSTRUCTION="Add more options to mimetype table with isdefault=y and
keyword=image"/>
```

The next code section specifies how the mimetype field will appear on the Advanced Search form. As shown above, the SQL supplied here queries the database for mimetypes to fill the drop-down select with.

```
<SEARCHFORM DESCRIPTION="Mimetype" TYPE="SELECT" SOURCETYPE="TABLE"
TABLENAME="mimetype" OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype"
SQL="SELECT mimetype, description FROM mimetype WHERE keyword = 'image' AND
isdefault = 'y'"/>
</PROPERTY>
```

The following code create a text field that allows users of the management system to input alternate text for the image. The SEARCHFORM tag specifies how the *Alt Text contains* field will appear on the Advanced Search form.

```
<PROPERTY NAME="alttext" DESCRIPTION="Alt Text">
<STORAGE TYPE="VARCHAR" LENGTH="255"/>
<INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="255" REQUIRED="NO"/>
<SEARCHFORM DESCRIPTION="Alt Text contains" TYPE="TEXT" WIDTH="48"
MAXLENGTH="255"/>
</PROPERTY>

</PROPERTIES>
</ASSET>
```

32.3 HelloAssetWorld Templates

The HelloAssetWorld sample site uses three Template assets to render the assets that were described previously. The following sections describe these Template assets.

This section contains the following topics:

- Section 32.3.1, "The HelloArticle Template"
- Section 32.3.2, "The HelloCollection Template"
- Section 32.3.3, "The HelloPage Template"

32.3.1 The HelloArticle Template

The HelloArticleTemplate renders HelloArticle assets. The template uses the following variables:

Table 32–1 HelloArticle Template

Variable	Value	Source
`tid`	The current template's ID.	The `tid` variable is set in the resargs1 field of the SiteCatalog table. The value is set automatically when the template is created.
`c`	The type of content that the template displays.	The `c` variable is set in the resargs1 field of the SiteCatalog table. The value is set using the Asset Type field on the New Template form.
`cid`	The ID of the asset to load.	The `cid` variable is passed in by the HelloPage template.
`picture:oid`	The object ID of a HelloImage asset that is associated with the HelloArticle asset.	The `picture:oid` variable is obtained by loading the current HelloArticle asset and using the ASSET.CHILDREN tag to find information on associated HelloImage assets.
`picture:alttext`	The alternate text for the associated `HelloImage` asset.	The picture:alttext variable is obtained by loading the current HelloArticle asset and using the ASSET.CHILDREN tag to find information on associated HelloImage assets.
`picture:mimetype`	The mimetype of the associated HelloImage asset.	The `picture:mimetype` variable is obtained by loading the current HelloArticle asset and using the `ASSET.CHILDREN` tag to find information on associated HelloImage assets.
`asset:headline`	The value in the Headline field of this HelloArticle asset.	The `asset:headline` variable is obtained by scattering the information in the HelloArticle asset.
`asset:byline`	The value in the Byline field of this HelloArticle asset.	The `asset:headline` variable is obtained by scattering the information in the `HelloArticle` asset.

Table 32–1 (Cont.) HelloArticle Template

Variable	Value	Source
`artID`	The ID of the article to display in the HelloArticle template.	On the first page view, this is set in the `resdetails` field of the ElementCatalog entry. On subsequent viewings, this is passed in by the HelloCollection template.

The following code is the standard beginning for an article template. They appear when you click the XML or JSP buttons on the New Template form in the WebCenter Sites user interface.

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!--  HelloArticle/HelloArticleTemplate
-
- INPUT
-
- OUTPUT
-
-->
```

The `RENDER.LOGDEP` tag marks the template as a cache dependency item. This means that when the template is modified, any outdated copies of the template will be removed from the WebCenter Sites and Satellite Server caches and replaced with current versions automatically.

```
<IF COND="IsVariable.tid=true">
<THEN>
<RENDER.LOGDEP   cid="Variables.tid" c="Template"/>
</THEN>
</IF>

<table border="0" cellspacing="2" cellpadding="2">

<tr>
<td>
```

The following `ASSET.LOAD` tag loads a HelloArticle asset using the asset's ID, which is stored in Variables.cid. This value is passed in to the HelloArticle template by the HelloPage template.

```
<!-- asset load will mark the asset as an 'exact' dependent of the pagelet being
rendered -->

<ASSET.LOAD NAME="helloArticleAsset" TYPE="Variables.c" OBJECTID="Variables.cid"/>
```

The next code section uses the `ASSET.CHILDREN` tag to load any HelloImage assets that are associated with the article. `ASSET.CHILDREN` creates a list which contains the information necessary to display the HelloImage asset.

```
<ASSET.CHILDREN NAME="helloArticleAsset" LIST="picture" TYPE="HelloImage"/>
```

This code checks to see if there is a HelloImage asset associated with the current article. If there is no associated HelloImage asset, the template only displays the text of the article.

```
<!--Check to see if the list (which contains information to display an image)
```

```
exists--if the list doesn't exist, display the article text only. -->
<IF COND="IsList.picture=true">
  <THEN>
  <!--Log the image as a dependency.-->
  <RENDER.LOGDEP   cid="picture.oid" c="HelloImage"/>
```

This code uses the RENDER.SATELLITEBLOB tag to display the associated image.

```
  <!--Display the image.-->
  <RENDER.SATELLITEBLOB BLOBTABLE="HelloImage" BLOBKEY="id" BLOBCOL="urlfile"
BLOBWHERE="picture.oid" BLOBHEADER="picture.mimetype" SERVICE="IMG SRC" ARGS_
ALT="picture.alttext" ARGS_ALIGN="left"/>
  </THEN>
</IF>
</td>
</tr>

<tr>
<td>
```

Then this ASSET.SCATTER tag gets all of the HelloArticle asset's primary fields.

```
<!-- get all the primary table fields of the asset -->
<ASSET.SCATTER NAME="helloArticleAsset" PREFIX="asset"/>
```

The following CSVAR tag displays the contents of the asset's fields.

```
<!-- display the headline-->

<h3><CSVAR NAME="Variables.asset:headline"/></h3>
</td>
</tr>

<tr>
<td>
<CSVAR NAME="Variables.asset:byline"/>
</td>
</tr>

<tr>
<td>
```

Because the Body field may contain an embedded link, it must be retrieved using the ics.getvar tag and displayed using the RENDER.STREAM tag, as shown in the following code:

```
<!-- display the body-->
<ics.getvar name="asset:urlbody" encoding="default" output="bodyvar"/>
<RENDER.STREAM VARIABLE="bodyvar" /><br/>
</td></tr>
</table>

</FTCS>
```

32.3.2 The HelloCollection Template

The collection template displays the HelloCollectionHello collection. It uses the following variables:

Table 32–2 HelloCollection Template

Variable	Value	Source
`tid`	The current template's ID.	The `tid` variable is set in the `resargs1` field of the SiteCatalog table. The value is set automatically when the template is created.
`c`	The type of content that the template displays.	The `c` variable is set in the `resargs1` field of the SiteCatalog table. The value is set using the `Asset Type` field on the New Template form.
`cid`	The ID of the asset to load.	The `cid` variable is passed in by the HelloPage template.
`tid`	The current template's ID.	The `tid` variable is set in the `resargs1` field of the `SiteCatalog` table. The value is set automatically when the template is created.
`p`	The ID of the page that generates the hyperlink. In this case, the current template.	n/a
`ApprovedArticles:id`	The ID of the current HelloArticle in the collection.	n/a
`ApprovedArticles:name`	The name of the current HelloArticle in the collection.	n/a
`artID`	The `artID` variable contains the ID of the HelloArticle being displayed by the HelloArticleTemplate. The value is the ID of the current article in the collection.	If the current article in the list is not the article being displayed by the HelloArticle template, then a URL is generated to create a hyperlink to that article. The URL is created by the `RENDER.GETPAGEURL` tag and appends the artID variable to the URL that it creates. This variable contains the ID of the article to display.
`pageID`	The ID of the HelloPage asset.	n/a
`referURL`	The URL generated by the `RENDER.GETPAGEURL` tag.	n/a

```
<IF COND="IsVariable.tid=true">
<THEN>
<RENDER.LOGDEP   cid="Variables.tid" c="Template"/>
</THEN>
</IF>
```

The following `ASSET.LOAD` tag loads the HelloCollectionHello asset, based upon the value of its ID, contained in Variables.cid. Variables.cid is passed in by the HelloPage template. `ASSET.LOAD` also names the asset HelloCollection. This does not change the name of the asset in the database; rather it sets the name which the rest of the code in the template uses to refer to the collection asset.

```
<ASSET.LOAD NAME="HelloCollection" TYPE="Collection" OBJECTID="Variables.cid"/>
<ASSET.SCATTER NAME="HelloCollection" PREFIX="asset"/>
```

Then the following ASSET.CHILDREN tag loads a list containing the HelloArticles that compose the collection.

```
<ASSET.CHILDREN NAME="HelloCollection" LIST="theArticles"
OBJECTTYPE="HelloArticle"/>
```

The next codes section uses the RENDER.FILTER tag to filter out articles that are not approved for Export to Disk Publishing. This allows the template to be used for both Mirror Publishing and Export to Disk Publishing.

```
<!--Filter out assets which aren't approved for export to disk publishing.-->
<RENDER.FILTER LIST="theArticles"LISTVARNAME="ApprovedArticles" LISTIDCOL="oid"/>
```

In the following code, the LOOP tag loops through the list of approved article and the RENDER.LOGDEP tag logs each item in the list as a cache dependency.

```
<LOOP LIST="ApprovedArticles">
<RENDER.LOGDEP cid="ApprovedArticles.id" c="Article"/>
```

The following code uses an IF tag to check whether the current article in the list is the article being displayed by the HelloArticle template. The ID of the article being displayed by the HelloArticle template is contained in Variables.artID. This variable is passed in by the HelloPage template. If the article IDs are the same, the name of the article is displayed in bold text and is not a hyperlink.

```
<IF COND="Variables.artID=ApprovedArticles.id">
<THEN>
<B><CSVAR NAME="ApprovedArticles.name"/></B><P/>
</THEN>
```

If the current article in the list is not the article being displayed by the HelloArticle template, then a URL is generated to create a hyperlink to that article. The URL is created by the RENDER.GETPAGEURL tag in the following code. The RENDER.GETPAGEURL tag appends the artID variable to the URL that it creates. This variable contains the ID of the article to display.

```
<ELSE>
<RENDER.GETPAGEURL PAGENAME="HelloAssetWorld/Page/HelloPageTemplate"
cid="Variables.pageID"
c="Page"
p="Variables.p"
OUTSTR="referURL"
ARGS_artID="ApprovedArticles.id"/>
```

The code example below displays the hyperlink. The REPLACEALL argument evaluates Variables.referURL, which contains the URL for the hyperlink. The CSVAR tag, shown below, displays the name of the article that the hyperlink links to.

```
<A HREF="Variables.referURL" REPLACEALL="Variables.referURL">
<CSVAR NAME="ApprovedArticles.name"/>
</A><P/>
</ELSE>
</IF>
</LOOP>
</FTCS>
```

32.3.3 The HelloPage Template

The HelloPage template acts as a containing page. It renders the HelloPage asset, displays a header graphic, and calls the HelloCollection and HelloArticle templates, creating the finished page layout.

The HelloPage template uses the following variables:

Table 32–3 HelloPage Template

Variable	Value	Source
`tid`	The current template's ID.	The `tid` variable is set in the `resargs1` field of the SiteCatalog table. The value is set automatically when the template is created.
`c`	The type of content that the template displays.	The `c` variable is set in the `resargs1` field of the `SiteCatalog` table. The value is set using the `Asset Type` field on the New Template form.
`cid`	The ID of the asset to load.	The cid variable is set in the `resargs1` field of the template's SiteCatalog entry.
`topImg:ID`	The ID of the TopImage ImageFile asset.	The `topImg:ID` variable is obtained by scattering the information in the TopImage image file asset.
`topImg:alttext`	The alternate text for the TopImage asset.	The `topImg:alttext` variable is obtained by scattering the information in the TopImage image file asset.
`topImg:mimetype`	The mimetype of the TopImage asset	The `topImg:mimetype` variable is obtained by scattering the information in the TopImage image file asset.
`asset:ID`	The ID of the Page asset that this template renders.	The `asset:ID` variable is obtained by scattering the information in the HelloPage asset.
`theCollection.oid`	The ID of the HelloCollectionHello collection, to be passed to the HelloCollection template for display.	The collection, and hence its ID, are associated with the Page asset.
`artID`	The ID of the article to display in the HelloArticle template.	On the first page view, this is set in the `resdetails` field of the ElementCatalog entry. On subsequent viewings, this is passed in by the HelloCollection template.

The code for the HelloPage template follows, along with a description of what it does:

```
<IF COND="IsVariable.tid=true">
<THEN>
<RENDER.LOGDEP cid="Variables.tid" c="Template"/>
</THEN>
```

```
</IF>

<!--Table for formatting-->
<table border="1" cellpadding="5" cellspacing="5">

<tr>
<td colspan="2">
```

This code loads a HelloImage asset and display that asset using the `RENDER.SATELLITEBLOB` tag.

```
<!--Embedded Image Asset-->

<!-- The following two lines of code load the image file and scatter the
information in its fields. -->
<ASSET.LOAD NAME="TopImage" TYPE="HelloImage" OBJECTID="1024605735822"/>
<ASSET.SCATTER NAME="TopImage" PREFIX="topImg"/>

<!-- This code creates a URL to display the image file.-->
<RENDER.SATELLITEBLOB BLOBTABLE="HelloImage" BLOBKEY="id" BLOBCOL="urlfile"
BLOBWHERE="Variables.topImg:id" BLOBHEADER="Variables.topImg:mimetype"
SERVICE="IMG SRC" ARGS_alt="Variables.topImg:alttext" ARGS_WIDTH="600" ARGS_
HEIGHT="90"/>
</td>
</tr>


<tr>
<td>
```

Next, the code loads the HelloPage asset based on the value of `Variables.cid`, which is set in the `resargs1` field of the template's SiteCatalog entry.

```
<!-- This loads the HelloPage asset and names it HelloPage. -->
<ASSET.LOAD NAME="HelloPage" TYPE="Variables.c" OBJECTID="Variables.cid"/>

<!-- This scatters the fields of the HelloPage asset for use later in the element.
-->
<ASSET.SCATTER NAME="HelloPage" PREFIX="asset"/>

<!-- This finds the collection asset associated with the HelloPage asset and puts
the information the collection into a list. -->
<ASSET.CHILDREN NAME="HelloPage" LIST="theCollection" />
<RENDER.LOGDEP C="Collection" CID="theCollection.oid"/>
<!-- This checks to see whether ASSET.CHILDREN really generated a list. -->
<IF COND = "IsList.theCollection=true">

   <THEN>
         <!-- This displays the HelloCollectionTemplate template and passes the
template the ID of the collection to display and the ID of the current page. -->
         <RENDER.SATELLITEPAGE
PAGENAME="HelloAssetWorld/Collection/HelloCollectionTemplate" ARGS_
cid="theCollection.oid" ARGS_p="Variables.asset:id" ARGS_artID="Variables.artID"/>
   </THEN>

</IF>
</td>

<td>
<!-- This displays the HelloArticleTemplate template and passes the template the
ID of the article to display and the ID of the current page. The artID variable is
```

```
passed in by the URL.-->
<RENDER.SATELLITEPAGE PAGENAME="HelloAssetWorld/HelloArticle/HelloArticleTemplate"
ARGS_cid="Variables.artID" ARGS_p="Variables.asset:ID"/>
</td>
</tr>
</table>
</FTCS>
```

32.4 The HelloQuery Asset

The HelloAssetWorld site uses a query asset named HelloQuery to retrieve HelloArticles from the database. A content provider then creates and builds a collection, ranking the items that the HelloQuery asset returns in the order that they will be displayed.

33
The Burlington Financial Sample Site

The Burlington Financial sample site demonstrates site design best practices using WebCenter Sites assets.

Content management can be highly abstract, especially when described in terms of assets, queries and templates. Burlington Financial helps you to understand what the end result of your application can be—a live, functioning website. Developers and content managers have immediate and easy access to the sample site's code and queries.

This chapter contains the following sections:

- Section 33.1, "Overview of the Burlington Financial Sample Site"
- Section 33.2, "Navigation Features"
- Section 33.3, "Best Practices"

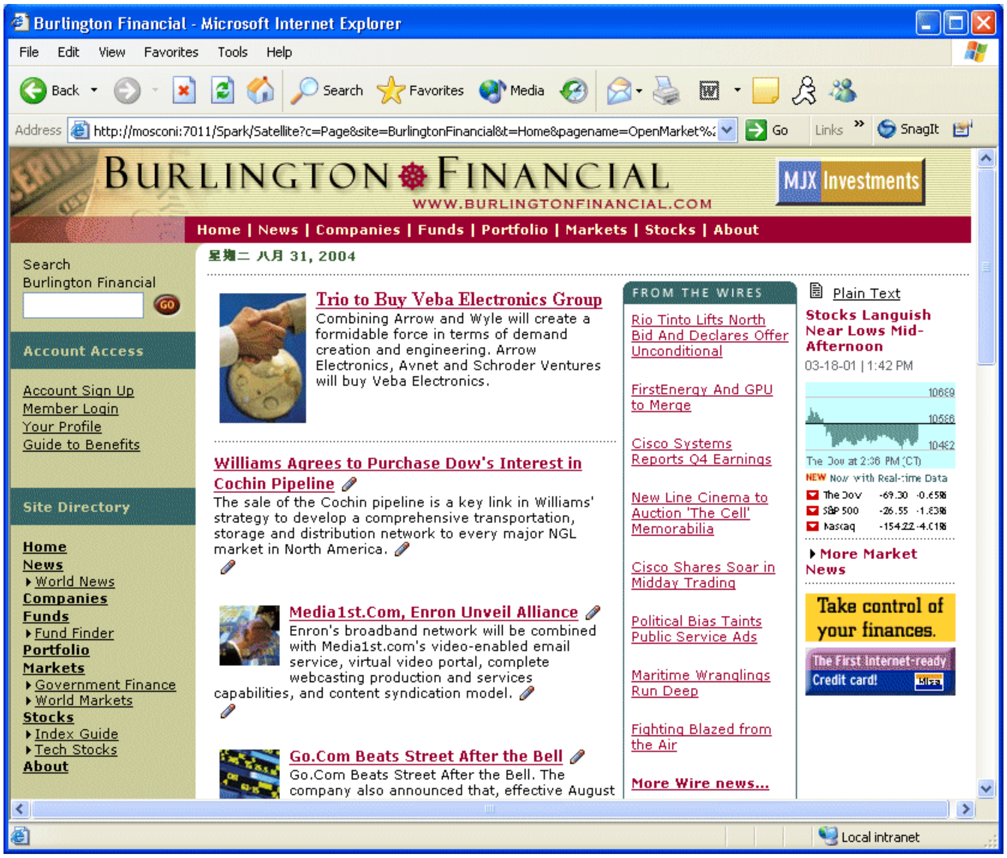
33.1 Overview of the Burlington Financial Sample Site

Burlington Financial is a fictitious financial news site. The site emphasizes navigation between sections, the site's hierarchy, how the site works with asset types, and a real-world look-and-feel. Burlington Financial also has about five hundred articles and over a hundred images, or enough real-world content to populate several sections.

Burlington Financial is a fully functional sample site with the following features:

- Includes search, member login, printer-friendly articles, email to a friend, topic directory, and stylesheets
- Demonstrates a hierarchy of website sections
- Supports component caching and Satellite Server
- Demonstrates the use of assets created with AssetMaker
- Demonstrates the use of Flex Assets and Engage Segment Assets
- Includes a meaningful amount of content
- Approximates a real-world site that developers can learn from

The Burlington Financial home page is shown in the following figure:


Burlington Financial - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://mosconi:7011/Spark/Satellite?c=Page&site=BurlingtonFinancial&t=Home&pagename=OpenMarket%2
Go Links SnagIt
BURLINGTON FINANCIAL
WWW.BURLINGTONFINANCIAL.COM
MJX Investments
Home | News | Companies | Funds | Portfolio | Markets | Stocks | About
Search Burlington Financial
GO
Account Access
Account Sign Up
Member Login
Your Profile
Guide to Benefits
Site Directory
Home
News
World News
Companies
Funds
Fund Finder
Portfolio
Markets
Government Finance
World Markets
Stocks
Index Guide
Tech Stocks
About
星期二 八月 31, 2004
FROM THE WIRES
Rio Tinto Lifts North Bid And Declares Offer Unconditional
FirstEnergy And GPU to Merge
Cisco Systems Reports Q4 Earnings
New Line Cinema to Auction 'The Cell' Memorabilia
Cisco Shares Soar in Midday Trading
Political Bias Taints Public Service Ads
Maritime Wranglings Run Deep
Fighting Blazed from the Air
More Wire news...
Plain Text
Stocks Languish Near Lows Mid-Afternoon
03-18-01 | 1:42 PM
10689
10586
10482
The Dow at 2:36 PM (CT)
NEW Now with Real-time Data
The Dow -69.30 -0.65%
S&P 500 -26.55 -1.63%
Nasdaq -154.22 -4.01%
More Market News
Take control of your finances.
The First Internet-ready Credit card!
Visa
Local intranet

Trio to Buy Veba Electronics Group

Combining Arrow and Wyle will create a formidable force in terms of demand creation and engineering. Arrow Electronics, Avnet and Schroder Ventures will buy Veba Electronics.

Williams Agrees to Purchase Dow's Interest in Cochin Pipeline

The sale of the Cochin pipeline is a key link in Williams' strategy to develop a comprehensive transportation, storage and distribution network to every major NGL market in North America.

Media1st.Com, Enron Unveil Alliance

Enron's broadband network will be combined with Media1st.com's video-enabled email service, virtual video portal, complete webcasting production and services capabilities, and content syndication model.

Go.Com Beats Street After the Bell

Go.Com Beats Street After the Bell. The company also announced that, effective August

Burlington Financial takes advantage of cacheable pagelets. These individual pagelets can be cached and managed independently, giving developers greater performance and flexibility on the site.

It is recommended that you design your site using cacheable pagelets. For more information, see Chapter 5, "Page Design and Caching."

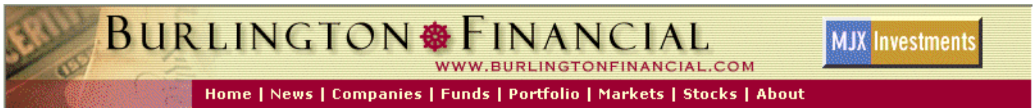
33.2 Navigation Features

The following three elements are used to display the primary navigation bars in Burlington Financial (which you can look at using Oracle WebCenter Sites Explorer).

This section contains the following topics:

- Section 33.2.1, "BurlingtonFinancial/Site/TopSiteBar.xml"
- Section 33.2.2, "BurlingtonFinancial/Site/LeftSideSiteBar.xml"
- Section 33.2.3, "BurlingtonFinancial/Site/BottomNavFooter.xml"
- Section 33.2.4, "Breadcrumbs"

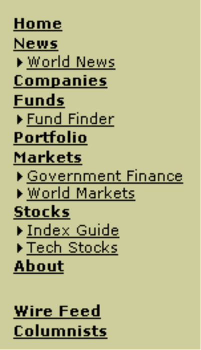
33.2.1 BurlingtonFinancial/Site/TopSiteBar.xml


BURLINGTON FINANCIAL
WWW.BURLINGTONFINANCIAL.COM
MJX Investments
Home | News | Companies | Funds | Portfolio | Markets | Stocks | About

This element draws the hyperlinks to the home page and its top-level children at the top of the page, just under the Burlington Financial logo.

It intentionally displays only the top-level children of the Home page, so that the row of hyperlinks does not wrap, breaking the design of the page. The Home page appears first and it is at the same level as its children.

33.2.2 BurlingtonFinancial/Site/LeftSideSiteBar.xml



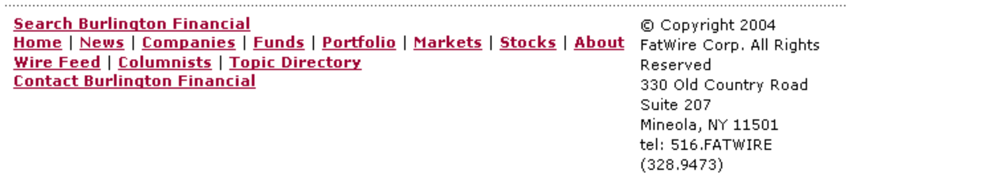
Home
News
▸ World News
Companies
Funds
▸ Fund Finder
Portfolio
Markets
▸ Government Finance
▸ World Markets
Stocks
▸ Index Guide
▸ Tech Stocks
About

Wire Feed
Columnists

This element draws a more detailed map of the major sections of the website and looks at the child and grandchild pages of the Home page. This element could be modified to go more than two levels deep, although the graphic design of the site limited the space available.

Notice that two other major pages are listed here that were not listed in the `TopSiteBar`; the Wire Feed and Columnists pages are independent of the Home page and its children, and are displayed separately.

33.2.3 BurlingtonFinancial/Site/BottomNavFooter.xml



Search Burlington Financial
Home | **News** | **Companies** | **Funds** | **Portfolio** | **Markets** | **Stocks** | **About**
Wire Feed | **Columnists** | **Topic Directory**
Contact Burlington Financial

© Copyright 2004
FatWire Corp. All Rights
Reserved
330 Old Country Road
Suite 207
Mineola, NY 11501
tel: 516.FATWIRE
(328.9473)

This element draws the hyperlinks at the bottom of every page. Like `LeftSideSiteBar`, the `BottomNavFooter` element also includes links to pages that are not children of the Home page.

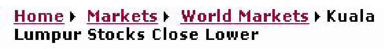
> **Note:** If you add another Page asset to the site, and it is not a descendant of the Home Page asset, then it will not automatically appear in any of the navigation bars used in Burlington Financial.

Although these navigation bars are computed dynamically, in a real-world website they would probably not change very often. For maximum performance, you could simply replace the dynamic code in the element with a static list of hard-coded links to the top-level pages. Later on, if you do need to change the site and add a new top-level section, you need only modify a few elements.

If you use dynamic navigation bar elements, you should set a long cache time-out for the navigation bar pagelets.

33.2.4 Breadcrumbs

This common feature in websites is a tiny map of the path to a particular item. In Burlington Financial, it is a conceptual path rather than the actual history of pages visited, and is located just under the top navigation bar:

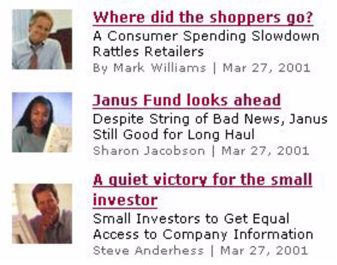


Home ▸ Markets ▸ World Markets ▸ Kuala Lumpur Stocks Close Lower

Because WebCenter Sites allows assets to be assigned to multiple parents at the same time, the same article can appear as a member of a collection on the Home page and on the News page at the same time. There is no way to identify the true parent of the article, so Burlington Financial passes the id of the parent Page anytime it draws a hyperlink to a child. That way, when the child asset draws itself, it already has the ID of the desired parent. This value is passed in the variable `p`.

Since our templates generate different HTML based on different values for `p`, those versions should be cached independently. So the variable `p` must be part of the page criteria variables listed in the page entry in the `SiteCatalog` for that template. Because this is such a common technique, WebCenter Sites automatically includes `p` in the list of page criteria variables for a Template asset's `SiteCatalog` page entry.

Sometimes you may want to override `p` and use a different parent asset. For example, the Burlington Financial Home page has links to articles by the following columnists:

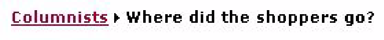


Where did the shoppers go?
A Consumer Spending Slowdown Rattles Retailers
By Mark Williams | Mar 27, 2001

Janus Fund looks ahead
Despite String of Bad News, Janus Still Good for Long Haul
Sharon Jacobson | Mar 27, 2001

A quiet victory for the small investor
Small Investors to Get Equal Access to Company Information
Steve Anderhess | Mar 27, 2001

However, these articles don't belong conceptually to the Home page, rather, they belong to the Columnists page. So you can load and pass the ID of the Columnists Page asset as the parent for those hyperlinks. This way, when a visitor clicks on them, the breadcrumb identifies Columnists as the parent. This behavior is consistent with clicking on the Columnists link in the navigation bar, and then clicking on one of the articles there:



Columnists ▸ Where did the shoppers go?

In other cases, it isn't immediately clear which asset should be the parent. For example, Burlington Financial treats the articles in the From the Wires box as belonging to the current page. Stories listed on the main Wire Feed section page belong to the Wire Feed section, and show Wire Feed as their parent.

33.3 Best Practices

The Burlington Financial sample site demonstrates WebCenter Sites best practices for several other important features found in real-world websites.

This section contains the following topics:

- Section 33.3.1, "Searching"
- Section 33.3.2, "Keywords"

33.3.1 Searching

The database search code in Burlington Financial is very similar to the search code in the WebCenter Sites application itself. It works with dynamic delivery, but not with exported static HTML. In this case, you'll need a different search mechanism for indexing the static HTML files.

The `BurlingtonFinancial/Util/SearchPost` element uses SQL searching against the Article table, or it can use the search engine index if it is installed and enabled for the Article asset type. SQL searching is case-sensitive. Using a search engine would allow more sophisticated search capabilities, such as case-insensitive searching, word variants and word stemming.

33.3.2 Keywords

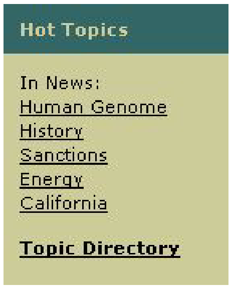
The Article asset type contains a field called **keyword** which lets editors associate specific terms with an Article for improved searching of the Article asset type. Burlington Financial Article assets have one or more keywords separated by commas, for example, Energy, Shell Oil, OPEC. Burlington Financial uses keywords to display lists of Hot Topics.

33.3.3 Hot Topics

Burlington Financial Hot Topics demonstrate one use of query assets.

On the left side of most pages, there is a list of Hot Topics for a particular section of the site. Hot topics are listed according to which section the visitor is viewing, as determined by the Page assets.

In the following example, the Hot Topics in the News section are Human Genome, History, Sanctions, Energy and California:



Hot Topics

In News:
Human Genome
History
Sanctions
Energy
California

Topic Directory

The element `BurlingtonFinancial/Common/LeftNavColumn` includes the pagelet `BurlingtonFinancial/Query/ShowHotTopics`. This element receives the ID of a page asset, passed through the variable p. If it cannot find a value for p, it defaults to using the Home page. It loads that page asset and looks for the top stories collection associated with the page and loads it. The element then loops through each of the articles in the collection and builds a list of keywords, pulled from the keyword field of the article (multiple keywords for an article must be delimited by commas). After it has made a list of the keywords, the element loops over that list, listing each keyword as a hyperlink to a page that runs a query for that keyword, by rendering the query asset named HotTopics.

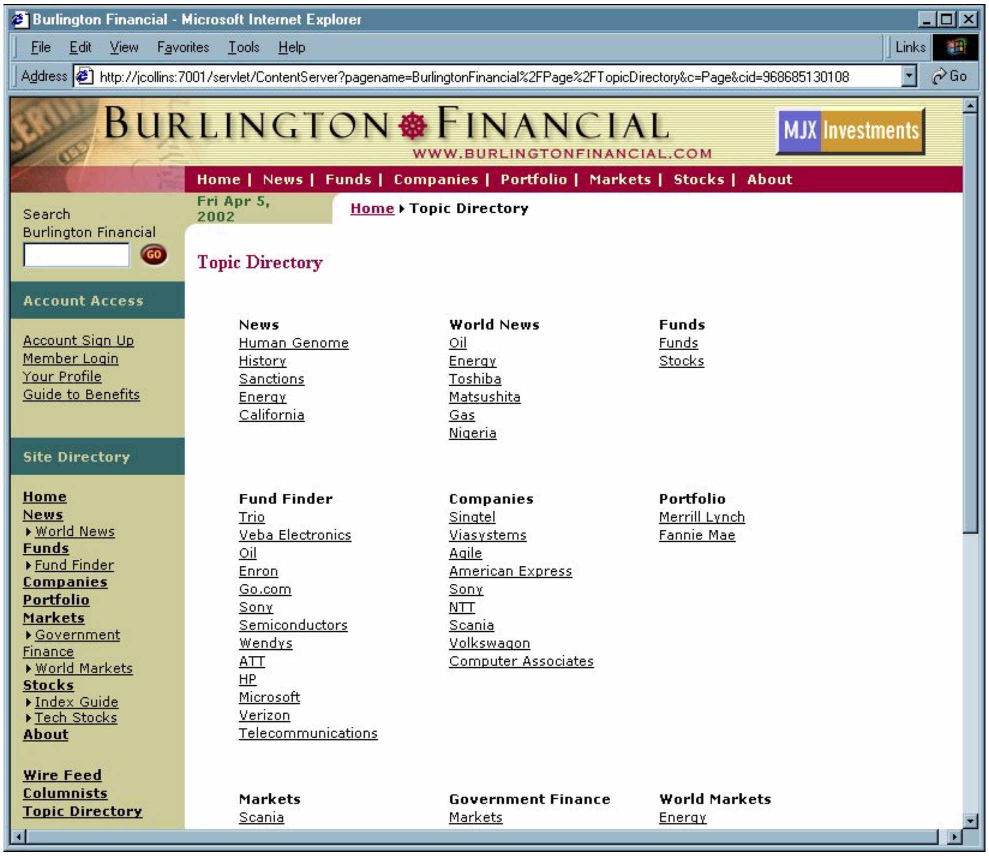
When visitors click a link, they are taken to a page that renders the query asset HotTopics, using the HotTopicFront template. The HotTopics query does a straight SQL match against articles that contain the selected keyword. The keyword search is not constrained in any way, it searches all articles in the Burlington Financial site, not just those in a particular section or category.

Each article returned by the HotTopics query inherits the parent ID of the page asset where the visitor first started looking at the keywords. Clicking on the Shell Oil story from the list of Energy stories under the News page causes the story to be displayed with News as its parent page. Clicking on the same Shell Oil story from the list of Energy stories under the World News page causes the story to be displayed with World News as its parent.

This design is not a problem in the dynamic live site; however it does cause duplicate files to be exported to a static delivery site.

33.3.4 Topic Directory

At the bottom of the left navigation column, and also in the navigation links at the bottom of each page, there is a hyperlink to the Topic Directory:



Address: http://jcollins:7001/servlet/ContentServer?pagename=BurlingtonFinancial%2FPage%2FTopicDirectory&c=Page&cid=968685130108

Topic Directory

News
Human Genome
History
Sanctions
Energy
California

World News
Oil
Energy
Toshiba
Matsushita
Gas
Nigeria

Funds
Funds
Stocks

Fund Finder
Trio
Veba Electronics
Oil
Enron
Go.com
Sony
Semiconductors
Wendys
ATT
HP
Microsoft
Verizon
Telecommunications

Companies
Singtel
Viasystems
Agile
American Express
Sony
NTT
Scania
Volkswagon
Computer Associates

Portfolio
Merrill Lynch
Fannie Mae

Markets
Scania

Government Finance
Markets

World Markets
Energy

This page consists of the Hot Topics pagelet for every section. Since the pagelet that is used here is also used in the left navigation column, it is displayed in the browser very quickly. Each topic inherits its parent page and passes it to the list of articles the query returns.

33.3.5 Related Stories

The query asset used for an article's Related Stories list is similar to the previously described HotTopics query, but instead of getting the keywords from another page, it gets the keyword from the article that the query is associated to. The Article template Full includes the Related Stories pagelet, and passes the Article's first keyword to the query asset. When the Related Stories query is executed, it looks for other articles with the same keyword.

For example, from the Home page, click on the Hot Topic Microsoft and choose the story Microsoft launches worldwide campaign against counterfeit software. This article (cid=984156689788) has the world Microsoft in its keyword field. It also has the Related Keyword query associated with it. When this Query is executed during rendering, the SQL looks up five other Articles with the term Microsoft in their keyword field. The query is designed to exclude the article that it is associated with, so you don't see the Microsoft Campaigns Against Counterfeit Software subheadline in the Related Stories.

In a dynamic environment, the list of articles returned by this query can change as articles are added to the site.

33.3.6 Text-Only Versions

Creating a text-only version of a web page (for printing it) is very easy to implement with WebCenter Sites. Using the WebCenter Sites rendering model, you can override a template that displays a given asset just by changing the page name used to render it. You do not need to pass the override template as a separate parameter. Creating a printer-friendly version of a page is simply a matter of adding a hyperlink that uses the plain-text version of the appropriate template.

For example, if you are pointing to an article:

```
http://myserver/servlet/ContentServer?pagename=BurlingtonFinancial/Article/Full&ci
d=987654321
```

You can change to the text-only version of the page by pointing to the text version of the template:

```
http://myserver/servlet/ContentServer?pagename=BurlingtonFinancial/Article/FullTex
t&cid=987654321
```

Burlington Financial uses the convention of adding Text to the end of a Template asset name to indicate different styles of templates and elements. Some examples:

- Web format: `BurlingtonFinancial/Article/Summary`
- Plain text: `BurlingtonFinancial/Article/SummaryText`
- Web format: `BurlingtonFinancial/Page/SectionFront`
- Plain text: `BurlingtonFinancial/Page/SectionFrontText`

33.3.7 Plain Text Parallel Site

In most websites, the text-only pages have hyperlinks that take you back to the full web format pages. Or there are simply no navigable links on the printer friendly page. Burlington Financial's templates, though, are designed to show the visitor an entire plain text version of the site.

After you switch to the plain text version, you can continue to navigate around the plain text pages. However, not every single page in the Burlington Financial site is represented in the plain text version of the site. And the plain text pages do not have all the same content or hyperlinks as their graphics-rich versions. This was done intentionally, as a plain-text visitor would probably prefer a less complex version of a site. Extending Burlington Financial to include other parallel styles (WAP templates, WebTV, PDF, XML) would be very straightforward.

33.3.8 Email This Story

Another very common feature on websites is the ability to email a story. This feature is relatively straightforward in Burlington Financial.

> **Note:** Before you can email users, you have to configure WebCenter Sites and Content Centre to use the email feature.
>
> First, in the `futuretense.ini` file, set these two properties:
>
> ```
> cs.emailreturnto=<your email address>
> cs.emailhost=po-1.example.com (or the emailhost is)
> ```
>
> Second, set this property in `futuretense_xcel.ini`:
>
> ```
> xcelerate.emailnotification=true
> ```

The BurlingtonFinancial/Util/EmailFront and BurlingtonFinancial/Util/EmailPost elements call the article pagelet BurlingtonFinancial/Article/Summary to display the story in summary form. A more robust version of this code would check to make sure that the visitor entered a valid email address before submitting the form. A real site would also keep records of which stories have been emailed, the sender's email address, and the recipient's email address.

33.3.9 AssetMaker Asset Types

AssetMaker is a WebCenter Sites utility for constructing basic asset types. Two sample AssetMaker asset types are included in Burlington Financial: ImageFile and Stylesheet. These asset types use standard elements from AssetMaker without modification. Both have file upload fields for storing files in the database.

Burlington Financial includes JavaScript at the top of every page to do client-side browser detection and then load one of four corresponding Stylesheet assets. The element BurlingtonFinancial/Common/SetHTMLHeader, called at the top of each full-page template, uses the WebCenter Sites element GetBlobURL to get four different BlobServer URLs, one for each of the four different Stylesheet assets used by Burlington Financial. The actual .css file from the Stylesheet asset is served via the BlobServer, even though it isn't binary data.

33.3.10 Mimetype

Both the imagefile and stylesheet asset types are served to the browser using the BlobServer servlet. To ensure that the browser knows how to handle arbitrary chunks of content, a mimetype code is saved and passed to the browser. WebCenter Sites includes a MimeType table for storing these codes.

The AssetMaker also allows asset types to define their own mimetype fields. Both the ImageFile and Stylesheet asset types include mimetype fields as part of their asset descriptor files. You can add your own mimetype codes and extensions to the MimeType table using Oracle WebCenter Sites Explorer or some other database tool.

33.3.11 Collections of Collections

The collection asset type is how WebCenter Sites arranges content into manageable groups. Burlington Financial also demonstrates the use of a collection whose child asset type is a collection. This allows editors to easily rearrange groups of content on a web page simply by re-ranking a collection. These sub-collections can be used to build a library of favorite stories, breaking news, hot topics, etc.

In Burlington Financial, a named asset association called "HomeStoryGroups" was created from the page asset type to the collection asset type. The page template SectionFront looks for a collection in the StoryGroups slot. It forces the collection to be displayed using the BurlingtonFinancial/Collection/StoryGroups template:

```
<ASSET.CHILDREN NAME="SectionFrontPage" LIST="StoryGroups" CODE="HomeStoryGroups"/
<IF COND="IsList.StoryGroups=true"
<THEN
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/Collection/StoryGroups"
ARGS_cid="StoryGroups.oid"
ARGS_p="Variables.asset:id"/
</THEN
</IF
The StoryGroups template then loops over each collection in the asset:
<LOOP LIST="theGroups"
```

```
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/Collection/PlainList"
ARGS_cid="theGroups.oid"
ARGS_p="Variables.p"/
<img src="/futuretense_cs/bf/images/dot_rule_125.gif" width="125" height="1"/<P/
</LOOP
```

This can be seen on the news page. There is a collection called NewsGroup that is associated with the news page. This collection contains one child collection, called Energy List, which itself contains three articles. This lets an editor add or remove items from the News page by re-ranking the NewsGroup collection.

33.3.12 Membership

The Burlington Financial sample site includes a membership component that has elements to sign up new members and log in existing members. These visitor registration elements are not robust enough for use on a real-world website, but can give you a starting point for your own designs. For example, Burlington Financial has sample visitor account screens, allowing visitors to register and set their own preferences, but does not use this information to restrict visitor access to certain web pages, or to make recommendations based on a member's profile.

For more information about visitor registration in Burlington Financial and about security in general, see Chapter 31, "User Management on the Delivery System."

33.3.13 Wire Feed

Both the home page template and the section front page template include a list of stories called From the Wires. This represents content that flows automatically onto the site. Large sites often subscribe to wire feed services or other content aggregators. Stories from these sources are moved onto a site with little human intervention.

Each of the page assets that make up the major sections of Burlington Financial are associated with a query asset. For the wire feed section, this query asset contains a query to look for article assets whose source field is set to WireFeed. The queries also look at the category field of each article. Each section in Burlington Financial contains certain categories of stories, so the wire feed queries try to match those categories.

The page asset named WireFeed contains a query to return wire feed stories regardless of their category.

33.3.14 Featured Funds

The fund section front page template includes a list of funds called *Featured Funds*. This list contains funds that are selected using a Engage segment asset. Segment assets divide visitors into groups based on common characteristics.

You build a segment asset by first creating visitor data assets. A visitor data asset stores a single piece of information about visitors to the website; a zip code, for example. Segments are built by selecting visitor data assets to base them on, and then setting qualifying values for those criteria. For example, you can create a zip code segment that uses the value in the zip code visitor data asset to display advertisements for local businesses.

The Featured Funds list displays funds based upon whether the website visitor belongs to the `BFfrequentvisitor` segment or the `Highriskinvestor` segment.

33.3.15 Fund Finder

Fund Finder is a form that lets you search for funds based on the criteria that you select. Some of the Fund Finder form drop-downs are hard-coded into the form; the Fund Families that the form searches, however, are listed dynamically, using the concept of assetsets and searchstates.

A *searchstate* is a set of search constraints based on the attribute values

```
<SEARCHSTATE.CREATE NAME="ss"/>
<ASSETSET.SETSEARCHEDASSETS
  NAME="as" ASSETTYPES="ProductGroups"
  CONSTRAINT="ss"/>
<ASSETSET.GETATTRIBUTEVALUES
  NAME="as" TYPENAME="PAttributes"
  ATTRIBUTE="FundFamily"
  LISTVARNAME="fflist"/>

<P>
<SELECT name="FundFamily" SIZE="3" MULTIPLE="1">
<OPTION SELECTED="" VALUE="NoPreference"/>No Preference
<LOOP LIST="fflist"><OPTION/><csvar NAME="fflist.value"/>
</LOOP>
</SELECT>
</P>
```

33.3.16 Page Cache Parameters

By default, WebCenter Sites sets the cacheinfo property to `cs.pgcachefolder`,* for any `SiteCatalog` page entries that it creates when a you save a Template asset. However, there are times when you may not want pages to be cached.

Pages like `BurlingtonFinancial/Util/LoginPost` and `BurlingtonFinancial/Page/AccountAccess` are specifically set to `cs.nevercache`. This is necessary since they are visitor-specific, and you don't want one visitor to see another visitor's cached page results. Real customer sites need cache fine tuning for their pages.

WebCenter Sites has a set of default page criteria for creating SiteCatalog entries. You can also add additional page criteria variables, but the defaults should not be removed. For more information about caching and page criteria, see Chapter 5, "Page Design and Caching."

34 Customizing the WebCenter Sites Admin Interface

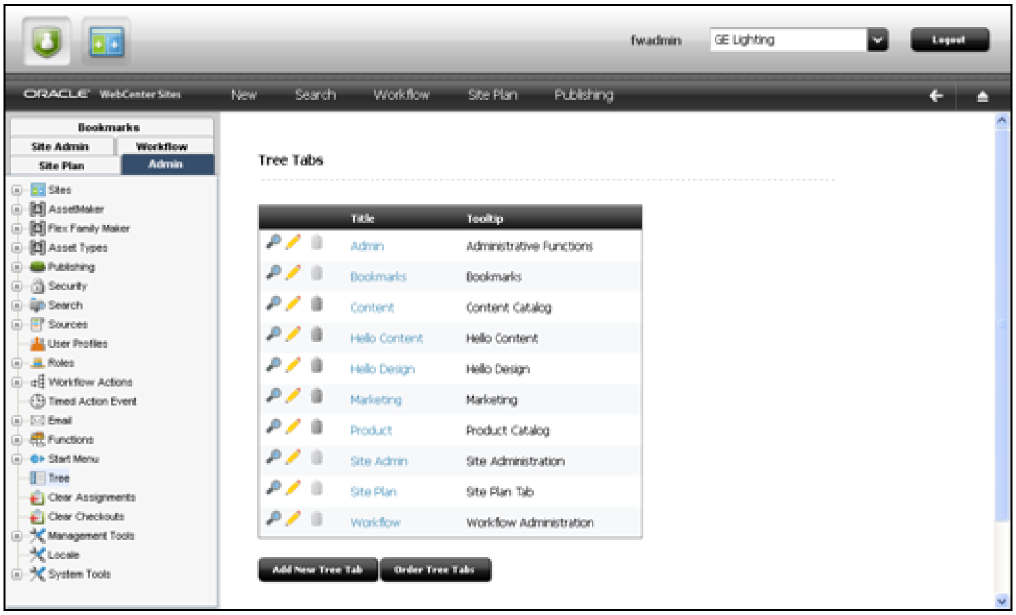
Administrative and editorial users of WebCenter Sites interact with the product through various trees that display in the Admin interface. You can customize the WebCenter Sites Admin interface by modifying these trees.

This chapter describes how to modify trees and contains the following sections:

- Section 34.1, "Overview of the Tree"
- Section 34.2, "Trees and Security"
- Section 34.3, "Tree Error Logging"

34.1 Overview of the Tree

The tree appears as a set of tabs in the left pane of the WebCenter Sites Admin interface, as shown in the following screen capture:



Tree Tabs

Title	Tooltip
Admin	Administrative Functions
Bookmarks	Bookmarks
Content	Content Catalog
Hello Content	Hello Content
Hello Design	Hello Design
Marketing	Marketing
Product	Product Catalog
Site Admin	Site Administration
Site Plan	Site Plan Tab
Workflow	Workflow Administration

WebCenter Sites tree tabs are created by the tree applet. You can create or modify your own trees by setting various parameters that will be passed to the tree applet. The tree applet accepts several kinds of parameters:

- Applet-wide parameters, which control the overall appearance and behavior of the applet.
- Tree-specific parameters, which control the appearance and behavior of the tree.
- Node parameters, which control the appearance and behavior of individual nodes on the tree.
- OpURL Node parameters, which allow the tree to communicate with WebCenter Sites.

A set of tree tab tables in the database stores information about tree configuration, including tab names, what roles have access to a tab, and the path to the element that populates the tree tab with data. You enter information into these tables via the Tree Tabs screens, which are accessed via the Admin interface by selecting the **Admin** tab and then clicking the **Tree** node.

This section contains the following topics:

- Section 34.1.1, "Loading the Tree Tabs"
- Section 34.1.2, "Refreshing the Tree"

34.1.1 Loading the Tree Tabs

For most of the default tree tabs supplied with WebCenter Sites, requests for tree data pass through the `OpenMarket/Gator/UIFramework/LoadTab` element. The `LoadTab` element performs several basic tasks, such as checking for session timeout.

For example, the Product tab, found in the GE sample site, completes the following steps as it loads:

1. Java code in the Product tab calls the `LoadTab` element.
2. The `LoadTab` element queries the TreeTab database tables to retrieve the elements that will load the data for the Product tree's top-level nodes. In this case, the elements are the `OpenMarket/Xcelerate/ProductGroups/LoadTree` element and the `OpenMarket/Xcelerate/Product/LoadTree` element.
3. The `OpenMarket/Xcelerate/ProductGroups/LoadTree` element and the `OpenMarket/Xcelerate/Product/LoadTree` element query the database for assets that correspond to the tree nodes and stream back node data to the tree applet.
4. The tree applet parses the node data and displays the nodes.
5. Java code in the Product tab calls an element to initialize its global context menu, the `OpenMarket/Gator/UIFramework/LoadGlobalPopup` element. This element sends a `GetTypes` command to each tree loading element called by the Products tab. When the tree loading elements receive this command, they return a list of asset types whose start menu items that should appear in the global context menu.
6. The `OpenMarket/Gator/UIFramework/LoadGlobalPopup` element finds the start menu items for the specified asset types and streams that information back to the tree.

Note that each asset type in the system must have a LoadTree element. The LoadTree element is a pointer to another element that actually loads the tree. If an asset type can have children, each of those children must have a LoadTree element. LoadTree elements have the following path:

```
OpenMarket/Xcelrate/AssetType/MyAssetType
/LoadTree
```

where MyAssetType is the name of the asset type that the LoadTree element refers to.

LoadTree elements are called based on the asset type set in the `Section` field of the Manage Tree form.

Core asset types use one of several elements to load their trees. The following table contains a list of these elements:

Table 34–1 Asset Type Elements

Asset Type	Location	Description
Flex Groups	`OpenMarket/Gator/UIFramework/LoadGroupNodes`	Displays a FlexGroup parent hierarchy and FlexAsset children
Flex Assets	`OpenMarket/Gator/UIFramework/LoadOrphanNodes`	Displays flex assets that do not belong to a flex group
Site Plan Tree	`OpenMarket/Xcelerate/AssetType/Page/LoadSiteTree`	Displays the SitePlan tree
Site Plan Associations	`OpenMarket/Gator/UIFramework/LoadChildren`	Displays asset associations in the SitePlan tree
Bookmarks	`OpenMarket/Gator/UIFramework/LoadActiveList`	Displays the Bookmarks tree
Administrative Tree	`OpenMarket/Gator/UIFramework/LoadAdminTree`	Displays the Administrative tree
Administrative Tree Helper Elements	`OpenMarket/Gator/UIFramework/Admin`	Loads helper elements for the Administrative tree
Asset Types	`OpenMarket/Gator/UIFramework/LoadAdministrationAsset`	Displays an asset type node at the top level of the tree and the names of all assets of that type on lower levels of the tree

If you want to change the appearance or behavior of nodes in your tree, create a new tree loading element based on one of these standard elements. Your website administrator can then specify the element's name and the path to that element in the Section Name and Element Name fields of the New Tree form, located off the Tree Tabs form. See the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide* for more information about adding trees and the New Tree form.

See Section 34.1.1.3, "Node Parameters" for more information about modifying tree nodes.

34.1.1.1 Applet-Wide Parameters

Applet-wide parameters are set in the `TreeAppletParams.xml` element. To modify the tree applet's behavior, change the parameter values set there, as shown in the following table:

Table 34–2 Applet-Wide Parameters

Parameter	Description
`Debug`	Turns debugging on and off. Valid values are `true` and `false`. If `Debug` is set to `true`, Java console debug and error messaging is turned on.

Table 34–2 (Cont.) Applet-Wide Parameters

Parameter	Description
ServerBaseURL	Sets the base string to which all the node data URL strings will be appended. For example, if the ServerBaseURL is set to file://localhost, and the value of the LoadURL parameter is NodeReader.test, the URL used for loading the tree's child nodes will be as follows: file://localhost/NodeReader.test
BackgroundColor	Sets the background color of the tree using a decimal RGB value. If this parameter is not set, the background color defaults to the color of the HTML frame in which the tree is embedded.
TotalPanes	Sets the number of tree tabs that will be displayed. This value is set automatically.
URLTarget	The target frame in which to display node links. The default value is XcelAction (name of the pane on the right side of the browser window).

34.1.1.2 Tree-Specific Parameters

Tree-specific parameters are set by the Add New Tree Tab form and the OpenMarket\Gator\UIFramework\TreeTabAdd.xml element that creates the Add New Tree Tab form. To modify the tree's appearance or behavior, change the parameter values shown in the following table by using the using the form or by altering the TreeTabAdd element.

Table 34–3 Tree-Specific Parameters

Parameter	Description
Title	Sets the text that is displayed on the tab. This value is set in the Title field of the Manage Tree form, found on the Admin tab.
ToolTip	Sets the text that is displayed when the mouse pointer hovers over the tab index. This value is set in the Tool Tip field of the Manage Tree form, found on the Admin tab.
LoadURI	The URI of the page to call to retrieve a node's children. This value is set in the TreeTabAdd element.
ActionURL	The URL of the page that performs a context menu action for a node in the tree. The default value points to the OpURL.xml element. This value is set in the TreeTabAdd element.
OpenIcon	The path to the icon to use when depicting an expanded node. The default is a plus sign (+). This value is set in the TreeTabAdd element.
CloseIcon	The path to the icon to use when depicting an unexpanded node. The default is a minus sign (-). This value is set in the TreeTabAdd element.
LineStyle	Sets whether or not lines connect the nodes of the tree. Valid values are Angled and blank; Angled is the default. If the parameter is set to Angled, lines connect the nodes. If the value is left blank, no lines connect the nodes. This value is set in the TreeTabAdd element.

Table 34–3 (Cont.) Tree-Specific Parameters

Parameter	Description
RootID	Sets the ID of the root node. This string is used for specifying the node path. It defaults to the value of the Title parameter. This value is set in the TreeTabAdd element.
GlobalItems	This value is set in the GlobalItems field of the Manage Tree form, found on the Admin tab.
NodeItems	This value is set in the NodeItems field of the Manage Tree form, found on the Admin tab.

34.1.1.3 Node Parameters

The node parameters determine the appearance and behavior of the nodes in your tree. To define the appearance and behavior of these nodes, you write an element which sets the node parameters (shown in the following table) and passes their values to the BuildTreeNode.xml element, which creates the tree nodes.

Table 34–4 Node Parameters

Parameter	Description
Label	Specifies the text to be displayed for this node. The value does not have to be unique. The default value is "".
ID	A string identifier that is unique within the tree, used by WebCenter Sites to express selection paths. The ID is specified by WebCenter Sites.
ExecuteURL	The URI value of the page to be displayed when completing the Execute action. This value will have the value of ServerBaseURL prepended to it. If the node is not executable, do not include this parameter in the node data.
URLTarget	The frame target for ExecuteURL. If ExecuteURL is not included in the node data, it defaults to the target specified in the Applet-wide parameters.
Description	An alternative to the string specified in Label, if you choose this option on the tree-wide context menu. The default value is "".
Level	The relative level of this node, represented by a number >= 0. A value of 0 indicates that the node is an immediate child of the node requesting the data. To load more than one level of nodes at a time, set this value to a number greater than zero. The default value is 0.
Image	The URI for the image to be prepended to the label. If this field is not included in the node data, no image will be displayed for that node.

Table 34–4 (Cont.) Node Parameters

Parameter	Description
`LoadURL`	The URI for the subtree hierarchy. If this field is not included in the node data, this node requires no additional loading. The URL specified in this parameter must contain enough information so that the tree applet can find that node's children. For example, if your hierarchy is as follows: Product Tab / Reebok / Running Shoes the value of `LoadURL` is as follows: `ContentServer?pagename=OpenMarket/Gator/UIFramework/LoadTab&AssetType=ProductGroups&populate=OpenMarket/Xcelerate/AssetType/ProductGroups/LoadTree&op=load&parent=Variables.parentid` where `parentid` is the assetid of the "Running Shoes" asset, and `op` and `populate` are used by LoadTab to route to your tree load element.
`OKAction`	An action that will be displayed in the node's context menu. This string may appear multiple times in the same node data set.
`OpURL`	The URL to execute a given action on the server. This value will be prepended with the value of the `ServerBaseURL` parameter. Include this parameter in the node data unless the value of the `NodeItems` parameter is a null string, and thus has no `OKAction` specified.
`RefreshKeys`	Creates a key or set of set of keys which can be used to refresh the tree. Set the value to the ID of the current node.

The following excerpt from the `LoadAdministrationAsset` element sets the values of the node parameters and passes those values to the `BuildTreeNode` element.

The `ListofAsset` list referred to in this excerpt is a list of information on assets of a given type. This list was generated by a SQL query that is executed elsewhere in the element.

```
<CALLELEMENT NAME="OpenMarket/Gator/UIFramework/BuildTreeNode">
  <ARGUMENT NAME="Label"
    VALUE="ListofAsset.name"/>
  <ARGUMENT NAME="Description"
    VALUE="ListofAsset.description"/>
  <ARGUMENT NAME="ID"
    VALUE="Variables.TreeNodeID"/>
  <ARGUMENT NAME="OpURL"
    VALUE="ContentServer?pagename=
    OpenMarket/Gator/UIFramework/TreeOpURL&#38;
    AssetType=Variables.AssetType"/>

  <ARGUMENT NAME="ExecuteURL"
    VALUE="ContentServer?pagename=
    OpenMarket/Gator/UIFramework/TreeOpURL&#38;
    AssetType=Variables.AssetType&#38;n0_=
    Variables.packedTreeNodeID&#38;op=displayNode"/>

  <ARGUMENT NAME="OKActions"
    VALUE="Status;Inspect;Edit;Delete;refresh"/>
  <ARGUMENT NAME="Image"
    VALUE="Xcelerate/OMTree/TreeImages/AssetTypes/Variables.AssetType.gif"/>
  <ARGUMENT NAME="RefreshKeys"
```

```
    VALUE="ListofAsset.id"/>
</CALLELEMENT>
```

To customize the appearance or behavior of tree nodes, copy one of the standard elements and modify the node arguments. Note that tree loading elements are passed the following variables, so any tree loading element that you create or customize must take these variables into account:

Variables Passed in by the LoadTree element:

- AssetType, which is set to the section name that was created using the New Tree form
- op, which is set to init

Variables Passed in by the LoadGlobalPopup element:

- command, which is set to GetTypes
- AssetType, which is set to the section name that was created using the New Tree form
- varname, which you set with a comma-separated list of asset types that you want to display start menu items for
- popupvar, which you set to either true, if you want to add items to the global context menu, or false, if you do not need to add items to the context menu

34.1.1.4 Node Context Menu Commands

Each node on the tree has a menu that appears when the user right-clicks with the mouse. Commands on this menu allow you to refresh the node or load pages in the right side of the browser window. You can add commands to a node context menu that allow you to load forms such as the status and publish forms. Any form that can be called using an asset type and ID is a good candidate for being called by a node context menu command.

Add a new command to the node context menu by completing the following steps:

1. Add the new command, exactly as you want it to appear, into the node's OKActions field.
2. Into the element that the node's OpURL refers to (usually the TreeOpURL element), add a new IF statement that calls the form you want to load.

   For example, the following code from the TreeOpURL element displays a node:

```
<IF COND="Variables.op=displayNode">
     <THEN>
      <callelement NAME="OpenMarket/Gator/UIFramework/TreeIDFromPath">
      <argument NAME="TreePath" VALUE="Variables.TreeNodePath"/>
      </callelement>
         <setvar NAME="id" VALUE="Variables.ID"/>
         <callelement NAME="OpenMarket/Xcelerate/UIFramework/ApplicationPage">
         <argument NAME="ThisPage" VALUE="ContentDetailsFront"/>
         <argument NAME="contentfunctions" VALUE="true"/>
         <argument NAME="AssetType" VALUE="Variables.AssetType"/>
     </callelement>
    </THEN>
```

34.1.2 Refreshing the Tree

Elements that can alter the tree are responsible for refreshing the tree so that it displays current data. There are three different types of refresh action that you can specify:

- Self, which refreshes the children of the specified node
- Parent, which refreshes the specified node and its children
- Root, which refreshes the entire tree

There are two steps to refreshing the tree:

1. Code your tree customization elements so that the tree nodes that you wish to refresh have RefreshKeys. RefreshKeys are keys (usually the asset ID of the current node) which allow the refresh to take place.
2. Call the OpenMarket/Xcelerate/UIFramework/UpdateTreeOMTree element, and pass the element the _TreeRefreshKeys_ variable, specifying the type of refresh you want in the variable value.

You set the RefreshKeys for a node by passing the RefreshKeys argument to the BuildTreeNode element, as shown in the code sample in Section 34.1.1.3, "Node Parameters."

To refresh the tree, call the OpenMarket/Xcelerate/UIFramework/UpdateTreeOMTree element, as shown in the following example:

```
<CALLELEMENT NAME="OpenMarket/Xcelerate/UIFramework/UpdateTreeOMTree">
    <ARGUMENT NAME= "_TreeRefreshKeys_" VALUE= "Root:ActiveList"/>
</CALLELEMENT>
```

34.2 Trees and Security

WebCenter Sites uses roles to control access to the tree in the Sites Admin interface. The system-defined tabs Admin, Site Admin, and Workflow also require their users to have the xceladmin ACL.

Additional control is available by setting properties in futuretense_xcel.ini. For example, xcelerate.showSiteTree determines whether the tree is displayed by default; xcelerate.restrictSiteTree determines which users can display or hide the tree. For more information about trees and security, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*

34.3 Tree Error Logging

All tree-related error and debug messages are logged to the Java Console. You can turn debugging on and off by supplying a value for the Debug parameter when you create a tree.

Note that enabling debug affects performance, so error logging should generally be turned off on the delivery system.

35

Customizing Workflow

A WebCenter Sites workflow process is the series of states an asset moves through on its way to publication. The asset moves from one state to the next by taking a workflow step. Each step that the asset takes can be associated with a timed action, such as sending an email to a user when an asset is assigned to them, or a workflow step condition, which prevents an asset from moving on to the next step if certain conditions are not fulfilled.

You must create the workflow step condition elements which specify the conditions that an asset must meet to move on to the next state, and the workflow action elements which perform various actions as the asset moves from one state to the next. This chapter describes these elements in greater detail and provide sample code for each element type.

This chapter contains the following sections:

- Section 35.1, "Workflow Step Conditions"
- Section 35.2, "Workflow Actions"

35.1 Workflow Step Conditions

A workflow process is composed of one or more workflow states. Workflow steps move the asset from one workflow state to the next. Sometimes, however, there are conditions under which the asset should not move on to the next workflow state. You must create the element that defines the condition or conditions that prevent the asset from moving on to the next state.

This element receives the following data when it is called:

- An `IWorkflowable` object called `Object`, which represents the asset whose state is being changed
- An `IWorkflowStep` object called `Step`, which represents the current workflow step
- The `StepUser` variable, which contains the ID of the user attempting the step
- Variables specified as name-value pairs when a StepCondition is defined in the WebCenter Sites user interface. For more information about defining StepConditions, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

The workflow step condition element should check for a condition and return a Boolean value. If the value is false, the step will not proceed.

The following code comes from a sample workflow step condition element:

Example 35–1

```
1.  <?xml version="1.0" ?>
2.  <!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
3.  <FTCS Version="1.1">
4.  <!-- OpenMarket/Xcelerate/Actions/Workflow/StepConditions/ExampleStepCondition
5.  -
6.  - INPUT
7.  -
8.  - OUTPUT
9.  -
10. -->
11. <csvar NAME="This step condition will check if step can be taken"/><br/>
```

Line 1 in the following code sets an empty `ReturnVal` variable. In lines 4 and 9 of Example 35–6, this variable will be set with the reasons why the step cannot proceed.

Example 35–2

```
1.  <setvar NAME="ReturnVal" VALUE="Variables.empty"/>
2.
3.  <!--change the value of ReturnVal to a non-empty string later on, if you want
    to stop the step -->  <!-- most of the stuff below are debugging statements and
    also show you some items available to you to set up a condition for stopping
    the step-->
```

Line 2 in the following code uses the `WORKFLOWABLEASSET.GETDISPLAYABLENAME` tag to get the name of the asset that is in workflow.

Example 35–3

```
1.  <!-- get asset -->
2.  <WORKFLOWABLEOBJECT.GETDISPLAYABLENAME OBJECT="Object"
    VARNAME="assetdisplayablename"/>
3.  Object:<csvar NAME="Variables.assetdisplayablename"/><br/>
```

Line 2 in the following code creates a variable called `StepUser` which will contain the ID of the user attempting to take the step. Line 3 uses the `USERMANAGER.GETUSER` tag to load the user's ID into the `StepUser` variable. Line 4 uses the `CCUSER.GETNAME` tag to retrieve a human-readable user name, and line 5 uses the `csvar` tag to display that user name.

Example 35–4

```
1.  <!-- get userid -->
2.  Userid: <csvar NAME="Variables.StepUser"/><br/>
3.  <USERMANAGER.GETUSER OBJVARNAME="myUserObj" USER="Variables.StepUser"/>
4.  <CCUSER.GETNAME NAME="myUserObj" VARNAME="uname"/>
5.  Username: <csvar NAME="Variables.uname"/><br/>
```

Line 2 in the following code uses the WORKFLOWSTEP.GETID tag to get the ID of the current workflow step. The WORKFLOWSTEP.GETNAME tag, used in line 4, loads the step with the specified name.

Example 35–5

```
1. <!-- getstep -->
2. <WORKFLOWSTEP.GETID NAME="Step" VARNAME="sid"/>
3. Stepid: <csvar NAME="Variables.sid "/>
4. <WORKFLOWSTEP.GETNAME NAME="Step" VARNAME="sname"/>
5.  Stepname: <csvar NAME="Variables.sname"/><br/><br/>
```

The following code defines the conditions that will stop the change of step from taking place. The forcestop and notalloweduser variables that the conditionals check were set as arguments when the sample step condition was defined in the WebCenter Sites interface. In a real step condition, you would test for the condition of your choice here. for example, seeing whether an article asset has an associated image.

Example 35–6

```
<!-- This is the actual condition to stop the step. The following is just an
example. -->
<if COND="Variables.forcestop=true">
<then>
<setvar NAME="ReturnVal" VALUE="You can not take this step because
forcestop=true"/>
</then>
<else>
<if COND="Variables.uname=Variables.notalloweduser">
<then>
<setvar NAME="ReturnVal" VALUE="You are not allowed to take this step"/>
</then>
</if>
</else>
</if>
</FTCS>
```

35.2 Workflow Actions

As an asset moves through workflow, it can trigger a **workflow action**. A workflow action can do anything from send an email to alert a user that he has a new asset to evaluate to breaking a deadlock after a specified period of time has elapsed. There are five types of workflow actions:

- Step actions, which are executed as part of a transition between workflow states.
- Timed actions, which are triggered by deadlines when the asset is in a given state, thus associating the asset with a specific assignment.

- Deadlock actions, which are executed when an asset needs a unanimous vote in order to move to the next state, but the voters differ on which step the asset should take. The deadlock action will be executed whenever users choose different steps for the asset to move to.
- Group deadlock actions, which are executed when the assets in a workflow group need a unanimous vote in order to move to the next state, but the voters choose different steps, creating a deadlock.
- Delegation actions, which are executed when an asset is delegated. The delegated asset remains in its current workflow state, but is assigned to a new user.

Your workflow administrator must first define workflow actions using the WebCenter Sites user interface. Then you must create the elements that accomplish these workflow actions. WebCenter Sites provides several sample workflow action definitions for you to look at. For more information about defining workflow actions, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

The following sections describe sample workflow action elements:

- Section 35.2.1, "Step Action Elements"
- Section 35.2.2, "Timed Action Elements"
- Section 35.2.3, "Deadlock Action Elements"
- Section 35.2.4, "Group Deadlock Action Elements"
- Section 35.2.5, "Delegation Action Elements"

35.2.1 Step Action Elements

A Step Action element receives the following data when it is called:

- A `WorkflowEngine` object called `WorkflowEngine`.
- An `ObjectTotal` variable, which represents the total number of assets whose state is being changed.
- An `IWorkflowable` object called `Object`*`nnn`*, which represents the assets whose state is being changed. *`nnn`* is a number between `0` and `ObjectTotal -1`.
- An `IWorkflowStep` object called Step, which represents the workflow step being considered.
- A `StepTargetUser` variable, which is a comma-separated list of the step's target users.
- A `StepUser` variable, which contains the ID of the user attempting the step.
- A `Group` variable, which contains the ID of the workflow group to which the assets belong (if you are using workflow groups).
- Any variables that your workflow administrator has created in the definition for this Step Action.

The following Step Action element approves assets for publish; most other Step Action elements send an email to the assignees.

Example 35–7

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
```

```
4.  <!-- OpenMarket/Xcelerate/Actions/Workflow/StepActions/ApproveForPublish
5.  -
6.  - INPUT
7.  -     Variables.ObjectTotal - number of loaded workflowasset objects
8.  -  Object[n] - loaded workflowasset objects, where n = 0 - Variables.ObjectTotal
9.  -targets - one or more comma separated names of PubTargets for which to approve the asset
10. -
11. - OUTPUT
12. -
13. -->
14.
15. <!-- This is an action element called by step actions ApproveForPublish-->
16. This step action element will approve an asset for publish.<br/>
```

Line 2 in the following code uses the SETCOUNTER tag to create a counter which keeps track of the number of assets to approve. Lines 3 through 9 use the LOOP tag to loop through the assets and retrieve the asset types and IDs.

Example 35–8

```
1. <!-- get the id and assettype of the asset(s) to approve -->
2. <SETCOUNTER NAME="count" VALUE="0"/>
3. <LOOP COUNT="Variables.ObjectTotal">
4. <WORKFLOWASSET.GETASSETTYPE OBJECT="ObjectCounters.count" VARNAME="assettype"/>
5. <WORKFLOWASSET.GETASSETID OBJECT="ObjectCounters.count" VARNAME="assetid"/>
6. <SETVAR NAME="idCounters.count" VALUE="Variables.assetid"/>
7. <SETVAR NAME="typeCounters.count" VALUE="Variables.assettype"/>
8. <INCCOUNTER NAME="count" VALUE="1"/>
9. </LOOP>
```

Line 2 in the following code uses the STRINGLIST tag to create a comma-separated list of publish target names. Lines 6 through 21 loop through this list, using the PUBTARGET.LOAD and PUBTARGET.GET tags to load information about the publish targets from the PubTarget table. This information and information about the assets to be approved are passed to the ApprovePost element for further processing in line 12.

Example 35–9

```
1. <!-- approve for each destination -->
2. <STRINGLIST NAME="publishTargets" STR="Variables.targets" DELIM=","/>
3.
4. <if COND="IsList.publishTargets=true">
5. <then>
6. <LOOP LIST="publishTargets">
7. <PUBTARGET.LOAD NAME="pubtgt" FIELD="name" VALUE="publishTargets.ITEM"/>
```

```
<if COND="IsError.Variables.errno=false">
<then>
Approving for publish to <CSVAR NAME="publishTargets.ITEM"/><br/>
<PUBTARGET.GET NAME="pubtgt" FIELD="id" OUTPUT="pubtgt:id"/>
<CALLELEMENT NAME="OpenMarket/Xcelerate/PrologActions/ApprovePost">
<ARGUMENT NAME="targetid" VALUE="Variables.pubtgt:id"/>
<ARGUMENT NAME="assetTotal" VALUE="Counters.count"/>
</CALLELEMENT>
</then>
<else>
Cannot approve for publish to destination: <CSVAR NAME="publishTargets.ITEM"/>,
Error: <CSVAR NAME="Variables.errno"/>
</else>
</if>
</LOOP>
</then>
<else>
Cannot approve for publish. This step action requires a targets argument with
one or more comma separated publishing destination names.
</else>
</if>

</FTCS>
```

35.2.2 Timed Action Elements

Timed Action elements receive the following data when they are called:

- A `WorkflowEngine` object called `WorkflowEngine`.
- A `WorkflowAssignmentTotal` variable, which contains the total number of assignments for which this action applies.
- An `IWorkflowAssignment` object called `WorkflowAssignment`*nnn*, which represents assignments to apply the action to. *nnn* is a number greater than zero.
- An optional `Group` variable, which contains the ID of the workflow group to which the assets belong (if you are using workflow groups)
- Any variables that your workflow administrator has created in the definition for this Timed Action.

The following excerpt is from a Timed Action element that sends an email. The text of the subject and body of this email are set in the Workflow Email forms that you access from the Admin tab in the WebCenter Sites user interface. The body text expects the following variables:

- Variables.assetname, which contains the name of the current asset
- `Variables.assigner`, which is the name of the user who completed the previous state in the workflow process
- `Variables.instruction`, which is the text that the assigner puts in the `Action to Take` text box as he or she completes an assignment

In lines 2 and 5 of the following code, the variables in the email object, `subject` and `body`, are replaced by their values.

Example 35–10

```
1.  <!-- translate subject -->
2.  <EMAIL.TRANSLATESUBJECT NAME="emailobject"
    PARAMS="assetname=Variables.assetname" VARNAME="subject"/>
3.
4.  <!-- translate body -->
5.  <EMAIL.TRANSLATEBODY NAME="emailobject"
    PARAMS="assetname=Variables.assetname&#38;time=Variables.time" VARNAME="body"/>
6.
7.  <!-- send mail -->
8.  <sendmail TO="Variables.EmailAddress" SUBJECT="Variables.subject"
    BODY="Variables.body"/>
9.  </THEN>
10. <ELSE>
11. Email address: None<br/>
12. </ELSE>
13. </IF>
14.
15. <inccounter NAME="COUNT" VALUE="1"/>
16. </loop>
17. </then>
18. </if>
19.
20.
21. </FTCS>
```

35.2.3 Deadlock Action Elements

Deadlock Action elements receive the following data when they are called:

- A `WorkflowEngine` object
- An `ObjectTotal` variable, which represents the total number of deadlocked assets
- An `IWorkflowable` object called `Objectnnn`, which represents the deadlocked assets
- An `IWorkflowStep` object called `Step`, which represents the workflow step
- A `StepTotal` variable, which contains the number of steps chosen by individual users
- A `StepUser` variable, which contains the ID of the user attempting the step
- An optional `Group` variable, which contains the ID of the workflow group to which the assets belong (if you are using workflow groups)
- Any variables that your workflow administrator has created in the definition for this Deadlock Action.

The following Deadlock Action element sends an email to the users who approve the asset.

The text of the subject and body of this email are set in the Workflow Email forms in the WebCenter Sites administrative user interface. The body text expects the following variables:

- `Variables.assetname`, which contains the name of the current asset
- `Variables.header` and `Variables.message`, which contain the text of the email's body

Example 35–11

```
1.  <?xml version="1.0" ?>
2.  <!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
3.  <FTCS Version="1.1">
4.  <!-- OpenMarket/Xcelerate/Actions/Workflow/DeadlockActions/SendEmailToAssignees
5.  -
6.  - INPUT
7.  -
8.  - OUTPUT
9.  -
10. -->
11.
12. <!-- This is an action element called by step actions SendAssignmentEmail and
    SendRejectionEmail-->
13.
14. <csvar NAME="This deadlock action element will send emails"/><br/>
```

Line 2 in the following code uses the `EMAILMANAGER.LOAD` tag to load an email object.

Example 35–12

```
1.  <!-- load email object -->
2.  <EMAILMANAGER.LOAD NAME="Variables.emailname" OBJVARNAME="emailobject"/>
```

Lines 1 through 9 in the following code create a `NumOfSteps` variable, which contains either the total number of assets being delegated or zero.

Example 35–13

```
1.  <!-- get total steps -->
2.  <if COND="IsVariable.StepTotal=true">
3.    <then>
4.    <setvar NAME="NumOfSteps" VALUE="Variables.StepTotal"/>
5.    </then>
6.    <else>
7.    <setvar NAME="NumOfSteps" VALUE="0"/>
8.    </else>
```

```
</if>

<removevar NAME="Step"/>
<setvar NAME="Header" VALUE="The following users have chosen the corresponding
steps that has resulted in a deadlock. Please take appropriate actions to
resolve deadlock:"/>
<setvar NAME="Message" VALUE="Variables.empty"/>
```

The code in the following three examples (Example 35–14, Example 35–15, and Example 35–16) loop through the list of users who have put the asset in deadlock, creating an email for each one. Line 10 in the following code uses the `USERMANAGER.GETUSER` tag to load the user information of the user specified in the ID. Lines 11 and 12 use `CCUSER` tags to get the user's name and email address.

Example 35–14

```
1.  <!-- For each assignment object, get asignee -->
2.  <setcounter NAME="COUNT" VALUE="0"/>
3.  <if COND="Variables.NumOfSteps!=0">
4.  <then>
5.  <loop FROM="0" COUNT="Variables.NumOfSteps">
6.  <!-- get assigner -->
7.
8.  <setvar NAME="userid" VALUE="Variables.StepUserCounters.COUNT"/>
9.  <!-- get email address --->
10. <USERMANAGER.GETUSER USER="Variables.userid" OBJVARNAME="userobj"/>
11. <CCUSER.GETNAME NAME="userobj" VARNAME="user_name"/>
12. <CCUSER.GETEMAIL NAME="userobj" VARNAME="EmailAddress"/>
```

Lines 1 through 6 of the following code use the `WORKFLOWSTEP` and `WORKFLOWSTATE` tags to retrieve the asset's starting and ending steps and states.

Example 35–15

```
1.  <WORKFLOWSTEP.GETNAME NAME="StepCounters.COUNT" VARNAME="stepname"/>
2.  <WORKFLOWSTEP.GETSTARTSTATE NAME="StepCounters.COUNT" VARNAME="startstate"/>
3.  <WORKFLOWSTEP.GETENDSTATE NAME="StepCounters.COUNT" VARNAME="endstate"/>
4.
5.  <WORKFLOWSTATE.GETSTATENAME NAME="Variables.startstate"
    VARNAME="startstatename"/>
6.  <WORKFLOWSTATE.GETSTATENAME NAME="Variables.endstate" VARNAME="endstatename"/>
7.
8.  <setvar NAME="Message" VALUE="Variables.Message Variables.user_name:
    Variables.stepname - "/>
9.  <!--
10. user:<csvar NAME="Variables.user_name"/><br/>
11. step name:<csvar NAME="Variables.stepname"/><br/>
12. startstate name:<csvar NAME="Variables.startstate"/><br/>
```

```
13. endstate name:<csvar NAME="Variables.endstate"/><br/>
14. -->
15.
16. <!-- get asset -->
17. <WORKFLOWABLEOBJECT.GETDISPLAYABLENAME NAME="Variables.ObjectCounters.COUNT"
    VARNAME="assetname"/>
```

In lines 3 and 6 of the following code, the variables in the email object, `subject` and `body`, are replaced by their values.

Example 35–16

```
1.  <!-- translate subject -->
2.  <SETVAR NAME="params" VALUE="username=Variables.user_
    name&#38;header=Variables.Header&#38;message=Variables.Message&#38;assetname=Va
    riables.assetname"/>
3.  <EMAIL.TRANSLATESUBJECT NAME="emailobject" PARAMS="Variables.params"
    VARNAME="subject"/>
4.
5.  <!-- translate body -->
6.  <EMAIL.TRANSLATEBODY NAME="emailobject" PARAMS="Variables.params"
    VARNAME="body"/>
7.
8.  <!-- send mail -->
9.  <sendmail TO="Variables.EmailAddress" SUBJECT="Variables.subject"
    BODY="Variables.body"/>
10.
11. <inccounter NAME="COUNT" VALUE="1"/>
12. </loop>
13. </then>
14. </if>
15. email message:<csvar NAME="Variables.Header Variables.Message"/><br/>
16.
17. </FTCS>
```

35.2.4 Group Deadlock Action Elements

Group Deadlock action elements receive the following data when they are called:

- A `WorkflowEngine` object called `WorkflowEngine`.
- An `ObjectTotal` variable, which represents the total number of deadlocked assets.
- An `IWorkflowable` object called `Object`*nnn*, which represents the deadlocked asset. *nnn* is a number greater than zero.
- An `IWorkflowStep` object called `Step`, which represents the workflow step.
- A `StepTotal` variable, which contains the number of steps chosen by individual users
- A `StepUser` variable, which contains the ID of the user attempting the step.

- A `Group` variable, which contains the ID of the workflow group that is deadlocked.
- Any variables that your workflow administrator has created in the definition for this Group Deadlock Action.

The following Group Deadlock Action element sends an email to the users who approve the asset.

The text of the subject and body of this email are set in the Workflow Email forms in the WebCenter Sites administrative user interface. The body text expects the following variables:

- `Variables.assetname`, which contains the name of the current asset
- `Variables.header` and `Variables.message`, which contain the text of the email's body

Example 35–17

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- OpenMarket/Xcelerate/Actions/Workflow/GroupActions/SendEmailToAssignees
-
- INPUT
-
- OUTPUT
-
-->

<!-- user code goes here -->

<csvar NAME="This group deadlock action element will send emails"/><br/>
<!-- load email object -->
<EMAILMANAGER.LOAD NAME="Variables.emailname" OBJVARNAME="emailobject"/>
<!-- get group -->
<WORKFLOWENGINE.GETGROUPID ID="Variables.Group" OBJVARNAME="grpobj"/>
<WORKFLOWGROUP.GETNAME NAME="grpobj" VARNAME="GroupName"/>

<!-- get total steps -->
<if COND="IsVariable.StepTotal=true">
<then>
<setvar NAME="NumOfSteps" VALUE="Variables.StepTotal"/>
</then>
<else>
<setvar NAME="NumOfSteps" VALUE="0"/>
</else>
</if>

```

```
<removevar NAME="Step"/>
<setvar NAME="Header" VALUE="The following users have chosen the corresponding
steps that has resulted in a deadlock for the group: Variables.GroupName.
Please take appropriate actions to resolve deadlock:"/>
<setvar NAME="Message" VALUE="Variables.empty"/>
<!-- For each assignment object, get asignee -->
<setcounter NAME="COUNT" VALUE="0"/>
<if COND="Variables.NumOfSteps!=0">
<then>
<loop FROM="0" COUNT="Variables.NumOfSteps">
<!-- get assigner -->
<setvar NAME="userid" VALUE="Variables.StepUserCounters.COUNT"/>
<!-- get email address --->
<USERMANAGER.GETUSER USER="Variables.userid" OBJVARNAME="userobj"/>
<CCUSER.GETNAME NAME="userobj" VARNAME="user_name"/>
<CCUSER.GETEMAIL NAME="userobj" VARNAME="EmailAddress"/>

<WORKFLOWSTEP.GETNAME NAME="StepCounters.COUNT" VARNAME="stepname"/>
<WORKFLOWSTEP.GETSTARTSTATE NAME="StepCounters.COUNT" VARNAME="startstate"/>
<WORKFLOWSTEP.GETENDSTATE NAME="StepCounters.COUNT" VARNAME="endstate"/>

<WORKFLOWSTATE.GETSTATENAME NAME="Variables.startstate"
VARNAME="startstatename"/>
<WORKFLOWSTATE.GETSTATENAME NAME="Variables.endstate" VARNAME="endstatename"/>

<!-- get asset -->
<WORKFLOWABLEOBJECT.GETDISPLAYABLENAME NAME="Variables.ObjectCounters.COUNT"
VARNAME="assetname"/>

<!-- set message -->
<setvar NAME="Message" VALUE="Variables.Message Asset: Variables.assetname,
User: Variables.user_name, Step: Variables.stepname -- "/>

<!-- translate subject -->
<SETVAR NAME="params" VALUE="username=Variables.user_
name&#38;header=Variables.Header&#38;message=Variables.Message&#38;assetname=Va
riables.assetname"/>
<EMAIL.TRANSLATESUBJECT NAME="emailobject" PARAMS="Variables.params"
VARNAME="subject"/>

<!-- translate body -->
<EMAIL.TRANSLATEBODY NAME="emailobject" PARAMS="Variables.params"
VARNAME="body"/>

<!-- send mail -->
```

```
<sendmail TO="Variables.EmailAddress" SUBJECT="Variables.subject"
BODY="Variables.body"/>

<inccounter NAME="COUNT" VALUE="1"/>
</loop>
</then>
</if>
email message:<csvar NAME="Variables.Header Variables.Message"/><br/>

</FTCS>
```

35.2.5 Delegation Action Elements

Delegation action elements receive the following data when they are called:

- A `WorkflowEngine` object called `WorkflowEngine`.
- A `CurrentUser` variable, which contains the ID of the user who is delegating the asset.
- An optional `Group` variable, which contains the ID of the workflow group. All objects to be delegated must be in the same workflow group.
- An `ObjectTotal` variable, which represents the total number of assets being delegated.
- An `IWorkflowable` object called `Object`*`nnn`*, which represents the assets being delegated. nnn represents a number greater than zero.

Delegation action elements should be coded like other Workflow Action elements.

36

Understanding Web Services

This chapter introduces web services and explains how they work with WebCenter Sites. It describes what is supplied. As a standard part of the WebCenter Sites product, web services require no additional installation or configuration.

This chapter contains the following sections:

- Section 36.1, "What Are Web Services?"
- Section 36.2, "SOAP and Web Services"
- Section 36.3, "Supported SOAP Version"
- Section 36.4, "Supported WSDL Version"
- Section 36.5, "Related Programming Technologies"

> **Note:** The following feature is deprecated in WebCenter Sites 11*g* Release 1 (11.1.1.8.0): SOAP-based web services. This feature is replaced by REST services.

36.1 What Are Web Services?

WebCenter Sites enables you to create, deploy, and publish your own web services, as well as consume web services from other applications. Web services, which are a collection of operations that are accessible via standard XML messaging over the Internet, enable data exchange independent of the programming language, operating system, or hardware used by a given target system. With regard to WebCenter Sites, web services provide a standard means to expose content and functionality for consumption by remote applications, including ERP (enterprise resource planning), CRM (customer relationship management), and commerce systems.

36.2 SOAP and Web Services

Integration between WebCenter Sites and other applications is accomplished by transforming data using HTTP and XML data formats. The key XML format for web services is SOAP (Simple Object Access Protocol). SOAP is a W3C specification that extends HTTP to enable distributed applications to make remote-procedure calls (RPCs) over the Internet. As a language, SOAP defines the XML elements used to describe the parameters, return values, and so on required for RPC-style interactions. SOAP messages are transmitted point-to-point and handled in request-and-response fashion. WebCenter Sites, which supports SOAP, can exchange data with any application that has a SOAP interface. When processing SOAP requests, WebCenter

Sites leverages its native support for XML and its efficient page-evaluation and delivery mechanisms

---

Note: WebCenter Sites uses the JWSDP (Sun) implementation for web services. If the application server where WebCenter Sites is installed also has its own web services implementations, there will be conflicts with the SOAP `MessageFactory`. To resolve the conflicts, you must set a default `MessageFactory` via VM parameters before starting the application server. With the default set, WebCenter Sites can determine which `MessageFactory` to use for WebCenter Sites web services.

For example:

JBoss comes with its own web services. If you wish to use JBoss's `MessageFactory` for other reasons and also use WebCenter Sites web services, you must set the default `MessageFactory` to that of JBoss as follows:

```
-Djavax.xml.soap.MessageFactory=org.jboss.ws.core.soap.MessageFacto
ryImpl
```

If you need only WebCenter Sites web services, set the default `MessageFactory` as follows:

```
-Djavax.xml.soap.MessageFactory=com.sun.xml.messaging.saaj.soap.ver
1_1.SOAPMessageFactory1_1Impl
```

36.3 Supported SOAP Version

WebCenter Sites supports SOAP 1.1. You will need to know the capabilities and limitations of the SOAP protocol to write web services for WebCenter Sites. SOAP standard syntax is described in detail at the W3C website:

`http://www.w3.org`

36.4 Supported WSDL Version

WSDL (web services description language) is an XML format that describes distributed services on the Internet. A WSDL file describes the location of the service and the data to be passed in messages for particular operations. With regard to WebCenter Sites, these messages contain procedure-oriented information.

WebCenter Sites supports WSDL 1.1. You will need to understand the web services description language to write web services for WebCenter Sites and use the predefined WSDL files shipped with WebCenter Sites. The SOAP standard syntax is described in detail at the following W3C website:

`http://www.w3.org/TR/wsdl.html`

36.5 Related Programming Technologies

To write web services for WebCenter Sites, you should have a basic understanding of some or all of the following related technologies:

- XML

- SOAP
- WSDL
- JSP
- Java
- J2EE (Java 2)
- .Net

37

Creating and Consuming Web Services

This chapter explains how to integrate WebCenter Sites with remote applications over the Internet using web services protocols. In the context of the WebCenter Sites development framework, it teaches you how to create and consume basic web services.

This chapter contains the following sections:

- Section 37.1, "Using Predefined Web Services"
- Section 37.2, "Creating Custom Web Services"

> **Note:** The following feature is deprecated in WebCenter Sites 11*g* Release 1 (11.1.1.8.0): SOAP-based web services. This feature is replaced by REST services.

37.1 Using Predefined Web Services

WebCenter Sites includes a complete array of asset-delivery functions implemented as web services. These services can be accessed by any technology that can produce a web-services-enabled client. Supplied web-service capabilities are comparable to existing WebCenter Sites APIs (XML, JSP, Java, and COM).

Each supplied service is represented by a predefined WSDL (web services description language) file that contains descriptions of multiple web-services operations. Individual WSDL files define the interface and methods for web-services operations that correspond to WebCenter Sites functions. Related operations are grouped and collectively described according to function.

The WSDL file is used to generate the code required to interact with WebCenter Sites from a client application. You can generate client code automatically using various third-party applications that read WSDL files, or manually by examining the WSDL description and writing the client code from scratch. The resulting client stub constitutes a suitable interface for interaction with WebCenter Sites. When executed, the code creates a SOAP request based on the WSDL operation.

Most times you will have control over the client interaction with WebCenter Sites. For access by potentially unknown client applications, however, the supplied WSDL files can also be posted to a URL and registered via UDDI (universal description, discovery, integration) for remote access.

This section contains the following topics:

- Section 37.1.1, "Accessible Information"
- Section 37.1.2, "WSDL File Location"

- Section 37.1.3, "Process Flow for Predefined Web Services"
- Section 37.1.4, "Consider Your Data for Predefined Web Services"
- Section 37.1.5, "Generating the Client Interface"
- Section 37.1.6, "Writing Client Calls"

37.1.1 Accessible Information

Any web services client that supports SOAP and follows the predefined WSDL specifications can access the following information from the WebCenter Sites database:

- Site map of a content management site
- Blob data, such as a PDF file
- List of all the valid asset types and asset subtypes at a content management site
- List of assets that match specified search criteria
- Metadata associated with a particular asset

37.1.2 WSDL File Location

Predefined WSDL files for WebCenter Sites are automatically installed with the WebCenter Sites application in the following location:

```
http://install_dir/futuretense_cs/Xcelerate/wsdl/*.wsdl
```

37.1.3 Process Flow for Predefined Web Services

The following general steps describe how a request from a web services client program is processed using a supplied WSDL file:

1. The supplied WSDL file includes a description of the format for the request (input data expected by WebCenter Sites) and the format for the return data. The WSDL file maps standard data types for applications written in Java, Visual Basic, or other programming languages to XML schema data types.
2. The client program uses instructions in the WSDL to transform data from an input source (for example, a structured file) to an XML schema that is consistent with what the WebCenter Sites web service interface expects.
3. The client generates a SOAP envelope that includes the required data and transmits it to the content management site.
4. WebCenter Sites receives the SOAP message.
5. An XML parser and transformation utility map the data in the SOAP message to the format required by WebCenter Sites.
6. WebCenter Sites invokes the appropriate seed classes.
7. Seeds invoke the specified WebCenter Sites action.
8. WebCenter Sites returns requested data (name/value pairs) in the output format defined by the WSDL file to the client application.
9. The SOAP processor for the client application maps the XML schema data types to native data types for the specific programming language used.

37.1.4 Consider Your Data for Predefined Web Services

Data for the predefined WSDL operations is passed using RPC-style interactions (versus exchanging entire XML documents) to your program. These are mostly strings, but WebCenter Sites also includes classes that handle native objects with complex data types; for example, SearchStates and ILists.

37.1.5 Generating the Client Interface

Use the WSDL files to generate an interface for your client application. A variety of tools that generate client code from WSDL files are available. These tools support output for different programming languages. Choose a tool that produces code in the target language for your client application, and run it on the WSDL file that describes the operations you need. The resulting client stub makes all WebCenter Sites operations available to your client program.

37.1.6 Writing Client Calls

The code generated from the WSDL file provides an interface to WebCenter Sites functions. Once available, you can call the functions from your application, as needed.

37.2 Creating Custom Web Services

With WebCenter Sites, you can create web services that map data from any WebCenter Sites functions that you want to expose. Because of its support for XML, Java, and JSP, the existing WebCenter Sites development environment provides a familiar platform for developing web services. A supplied tag set enables you to build a SOAP response and stream SOAP encapsulated data to and from applications. As with the prepackaged web services, the WebCenter Sites delivery capability and page-evaluation pipeline are used to process SOAP requests. For web services, the client is a program, not a browser.

This section contains the following topics:

- Section 37.2.1, "Process Flow for Creating Custom Web Services"
- Section 37.2.2, "Consider Your Data for Custom Web Services"
- Section 37.2.3, "Creating a WebCenter Sites Page"
- Section 37.2.4, "Writing a WebCenter Sites Element"
- Section 37.2.5, "Creating a WSDL File"

37.2.1 Process Flow for Creating Custom Web Services

The following general steps describe how a request from a web services client program is processed:

1. The client program wraps whatever inputs are required in SOAP and passes them to WebCenter Sites.
2. The client uses instructions in the WSDL file to transform data from an input source (for example, a structured file) to an XML schema that is consistent with what the WebCenter Sites web service interface expects.
3. The client generates a SOAP envelope that includes the required data and transmits it to the content management site.
4. WebCenter Sites receives the SOAP message.

5. An XML parser and transformation utility map the data in the SOAP message to the format required by WebCenter Sites.
6. WebCenter Sites invokes the appropriate seed classes.
7. Seeds invoke the specified WebCenter Sites action.
8. WebCenter Sites returns requested data (name/value pairs) in the output format defined by the WSDL file to the client application.
9. SOAP processor for the client application maps the XML schema data types to native data types for the specific programming language used.

37.2.2 Consider Your Data for Custom Web Services

Data is passed using RPC-style interactions (versus exchanging entire XML documents) to your program. Consider your data and verify that you will be dealing with simple XSI data types. WebCenter Sites supports all W3C XSI primitive data types without modification.

Note: Support for complex web-services data types is possible in WebCenter Sites but requires that you create your own Java classes and deploy them on the application server. If you plan to create your own data types, it is recommended that you consult with WebCenter Sites technical support before doing so.

37.2.3 Creating a WebCenter Sites Page

Each web service requires page entry in the `SiteCatalog` table. The page entry to the SiteCatalog is a name that points to the element that calls the WebCenter Sites function described by your web service. The SiteCatalog stores all valid entries for pages at your site, including those that invoke web services.

The page is invoked by a request from the client. In turn, the response from WebCenter Sites is encapsulated in SOAP and returned to the client. Remember that for web services the client is a program (instead of a browser), and the response is XML (instead of HTML).

To create a WebCenter Sites page for web services

1. Start Oracle WebCenter Sites Explorer and log in to WebCenter Sites. For instructions, refer to the online help or the instructions in this guide.
2. In the left pane, open the `SiteCatalog` table.
3. Select the folder for your site.
4. In the **pagename** field, create an entry for the last part of the page name for your web service in the `SiteCatalog` table.
5. In the **root element** field, create a root entry for your web service. (Including the SiteCatalog entry in the SiteCatalog root avoids a table lookup and ensures that the element name of the first child element is mapped to the specified pagename.)
6. In either of the **resargs** fields, add the following optional arguments, if appropriate:
   - `cs.session=false` bypasses application server session management for the life of the SOAP request without using existing session objects or creating new session objects on behalf of the current request. This improves performance by reducing the application server load for native requests and requests from

clients that do not require session persistence. Although supported on any page, the `cs.session=false` resarg is mainly intended for use with SOAP services.

– `cs.contenttype=text/xml` prepares the root element for processing. Specifically, it causes the XML engine to properly respect namespace on tags and prevents default HTML compression from occurring. This is required only if you expect the request to come through a browser. Unless the web services request is always received as a SOAP request, you must include this resarg in each SiteCatalog entry to override the HTML compression. Provided that your input XML is well formed, you can be sure that the content output will be proper XML.

7. In the **csstatus** field, enter `Live`, or the appropriate status at this time.
8. Choose **File**, then **Save All**, or click the **Save All** toolbar button to save your work.

37.2.4 Writing a WebCenter Sites Element

The WebCenter Sites element contains the code for the function you want to expose. The element handles data and formats the SOAP response. To format the SOAP response, include the SOAP XML tags supplied with WebCenter Sites in your code. WebCenter Sites automatically generates the XML for the SOAP envelope.

WebCenter Sites elements written for web services in XML and JSP must not contain extra whitespace or comments because, unlike HTML, XML and its SOAP implementation have stricter parsing requirements. Because XML and JSP pages are handled the same way as HTML pages and are not filtered by WebCenter Sites, extra whitespace or comments can corrupt the SOAP message. Comments (XML or JSP) should appear only after the `soap.message` tag.

To write an element for a web service

1. Start Oracle WebCenter Sites Explorer and log in to WebCenter Sites (if you have not already done so). For instructions, refer to the online help or this guide.
2. In the left pane, select and highlight the `ElementCatalog` table.
3. Create a new folder in the `ElementCatalog` table: choose **File**, then **New Folder**.
4. Select and highlight the new folder, and right-click anywhere in the right pane and select **New** from the context menu.

   A new row appears in the table.
5. In the **elementname** field, enter an appropriate name for your web service as the name of the element.
6. In the **description** field, enter a short description of the element.
7. Click in the **url** field, and click the button that appears.

   The **New File** dialog box displays.
8. In the **Type/Ext** field, select **XML** or **JSP** as the file type from the drop-down list.
9. Click **OK**. Oracle WebCenter Sites Explorer opens its default editor. WebCenter Sites creates a file containing the skeleton code required of all XML or JSP elements. Enter your element code, including the required WebCenter Sites SOAP tags. For example, you can use the following code for XML:

```
<?xml version="1.0" ?><!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.2">
<!-- WebServices/helloworld
```

```
-
- INPUT
-
- OUTPUT--->
<soap.message  ns="mynamespace">
      <soap.body tagname="HelloWorldOut">
      <echoStringOut xsi:type="xsd:string">
      <csvar NAME="Variables.echoString"/>
         </echoStringOut>
      </soap.body>
</soap.message>

</FTCS>
```

10. Choose **File**, then **Save All** to save your work.

> **Note:** Choosing **File**, then **Save** instead of **File**, then **Save All** saves your file, but does not make it available to the application server.

37.2.5 Creating a WSDL File

WSDL (web services description language) files describe the web service so that a basic web services client can be automatically generated based on information it contains. If you are not using the predefined services provided with WebCenter Sites, you can optionally create your own WSDL file to describe your web service.

A WSDL file includes the SOAP address, SOAP action, a description of the format for the request (input data expected by WebCenter Sites), and the format for the return data. The WSDL file maps standard data types for applications written in Java, Visual Basic, or other programming languages to XML schema data types. Use any of the predefined WSDL files for WebCenter Sites functions as a template to get started.

37.2.5.1 Writing WSDL File Elements

A WSDL file has four sections:

- `Types`: specifies the data format and schema definition for operations. The type correlates with return data.
- `Message`: names inputs and outputs. This describes what the WebCenter Sites page must send back.
- `Operation`: describes inputs and outputs.
- `Binding`: specifies the SOAP action and operations.
- `Service`: describes associated port and binding with a URL.

In these sections, specify the following key XML elements:

- target namespace
- service name
- port name
- operation name
- input parameters (corresponding to simple data types) for the operation. Simple XSI data types (string, integer, float, and so on) return a single value.

37.2.5.2 WSDL File Example

Each WSDL file is a collection of interrelated operations, logically grouped together according to their WebCenter Sites function. Completed web services return XML in the form of a SOAP encapsulated response. You will need to understand web services description language to write web services for WebCenter Sites. WebCenter Sites supports WSDL 1.1.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:s0="http://FatWire.com/someuri/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 targetNamespace="http://FatWire.com/someuri/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
   </types>
   <message name="HelloWorldIn">
      <part name="echoString" type="s:string"/>
   </message>
   <message name="HelloWorldOut">
      <part name="echoStringOut" type="s:string"/>
   </message>
   <portType name="HelloWorldPortType">
      <operation name="helloworld">
         <documentation>FOR LATER</documentation>
         <input message="s0:HelloWorldIn"/>
         <output message="s0:HelloWorldOut"/>
      </operation>
   </portType>
   <binding name="HelloWorldBinding" type="s0:HelloWorldPortType">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/
       http" style="rpc"/>
      <operation name="helloworld">
         <soap:operation soapAction="WebServices/" />
         <input>
            <soap:body use="encoded"/>
         </input>
         <output>
            <soap:body use="encoded"/>
         </output>
      </operation>
   </binding>
   <service name="HelloWorld">
      <port name="HelloWorldPort" binding="s0:HelloWorldBinding">
         <soap:address location="http://localhost:8080/servlet/ContentServer"/>
      </port>
   </service>
</definitions>
```

38

Creating Visitor Data Assets

Engage lets you design online sites that gather information about your site visitors and customers, and then to use that information to personalize the product placements and promotional offerings that are displayed for each visitor.

Customizing your online sites begins with visitor data. The definitions of visitor data types are treated as assets in the WebCenter Sites database. There are three kinds of visitor data assets: visitor attributes, history attributes, and history types.

This chapter describes the visitor data assets and presents procedures for creating them. It contains the following sections:

- Section 38.1, "About Visitor Data Assets"
- Section 38.2, "Creating Visitor Data Assets"
- Section 38.3, "Verifying Visitor Data Assets"
- Section 38.4, "Approving Visitor Data Assets"

38.1 About Visitor Data Assets

You create visitor data assets so that you can use them to group your site visitors into segments. There are three kinds:

- Visitor attributes
- History attributes
- History types

When you create visitor data assets, you create entries in the visitor data tables in the WebCenter Sites database and you reserve a place in the database to store information of that kind for your site visitors.

This section contains the following topics:

- Section 38.1.1, "Visitor Attributes"
- Section 38.1.2, "History Attributes and History Definitions"
- Section 38.1.3, "Segments"
- Section 38.1.4, "Developing Visitor Data Assets: Process Overview"

38.1.1 Visitor Attributes

Visitor attributes hold types of information that specify one characteristic only (scalar values). For example, you can create visitor attributes named "years of experience," "job description," or "number of children."

When the visitor changes the data, the new data overwrites the old data. Engage does not assign a timestamp to the data that is stored as a visitor attribute and does not store revisions. For example, if a visitor changes his entry for "job description" from "butcher" to "baker," the information that the visitor was once a butcher is overwritten. You cannot, for example, create a segment based on bakers who used to be butchers.

For historical data, you must use history types.

38.1.2 History Attributes and History Definitions

History attributes are individual information types that you group together to create a vector of information that Engage treats as a single record. This record is the **history definition**. For example, a history definition called "purchases" can consist of the history attributes "SKU," "itemname," "quantity," and "price."

Engage references data stored as a history definition as a whole or an aggregate. It assigns a timestamp to each instance of the recorded definition and keeps each of those records. This means that you can sum or count history definitions and you can determine the first time or the last time a history definition was recorded for a visitor. Using the example in the preceding paragraph, you can create a segment based on the amount of money a visitor spends on specific items during a set period of time.

History definitions store historical data.

38.1.3 Segments

Segments are assets that divide visitors into groups based on common characteristics. Segments are built by determining which visitor data assets to base them on and then setting qualifying values for those criteria.

When you create visitor data assets, you create fields. These fields can be used in two places:

- As criteria for segments. That is, as configuration options in the Engage Segment Filtering forms (because you define segments with the visitor data assets). In other words, the choices you make about the data types for the attributes determine their appearance and behavior in the Segment forms. When you create these assets, you are customizing the Segment forms.
- On your public site pages. That is, as form fields or hidden fields on registration pages and other pages.

Segments are the key to personalizing merchandising messages with Engage. When visitors browse your site, the information they submit is used to qualify them for segment membership. When the site displays a page with a recommendation or promotion, Engage determines which segments a visitor belongs to and displays the product recommendations or promotional messages that are designated for those segments.

38.1.4 Developing Visitor Data Assets: Process Overview

There are five general steps for creating and using visitor data assets (fields):

1. A cross-functional design team including developers and marketers determines what kind of data you want to gather about your site visitors.
2. You (the developers) create and define the necessary visitor attributes, history attributes, and history definitions by using the forms in Engage.

3. The marketers use the Segment Filtering forms in Engage to categorize groups of visitors based on these visitor attributes, history attributes, and history definitions.
4. You program the appropriate site pages with the Engage XML or JSP object methods to collect and store the data, using either server-side validation or Javascript to validate the input on the pages. For example, you can create an online registration form for visitors to fill out by using JavaScript to validate the input and the Engage XML or JSP tags to process and store that information in the WebCenter Sites database.
5. When visitors browse your site, the information they submit is used to qualify them for segment membership. If your site is using promotions and recommendations based on segments, the message displayed for the visitor is personalized based on the segments that he or she qualifies for.

38.2 Creating Visitor Data Assets

Before you begin creating visitor data assets, be sure that you have completed the following tasks:

- Met with the marketing and design teams to determine the kinds of data that you want to collect about visitors.
- Examined the Segment Filtering forms so that you understand the context in which the visitor data assets that you create are used by the marketers. Additionally, note that the **visitor data assets** are listed by their **descriptions** rather than by their names in the Segment Filtering forms.

You can use the following data definitions for your visitor and history attributes:

- string: can hold up to 255 characters
- boolean: true and false are the only legal values
- short: valid range of values is 0 through 255
- integer: valid range of values is 0 through 65,535
- long: valid range of values is 0 through 65,535
- double: valid range of values is 0 through 4,294,967,295
- date: format is `yyyy-mm-dd hh:mm:ss.s`
- money: format is currency; valid range of values is unlimited
- binary: for visitor attributes only; used for binary data such as image files or cart objects

> **Note:** Binary visitor attributes can record binary data for individual visitors. Visitor attributes of this type are not displayed in the Segment Filtering forms and cannot be used to define a segment. Creating an attribute of type binary reserves space in the WebCenter Sites database that you use to store objects by using the XML object method `VDM.SAVESCALAROBJECT` or its JSP equivalent `vdm:savescalarobject` to convert an object from the WebCenter Sites name space into a binary form.

This section contains the following topics:

- Section 38.2.1, "Creating Visitor Attributes"

- Section 38.2.2, "Creating History Attributes"
- Section 38.2.3, "Creating History Definitions"

38.2.1 Creating Visitor Attributes

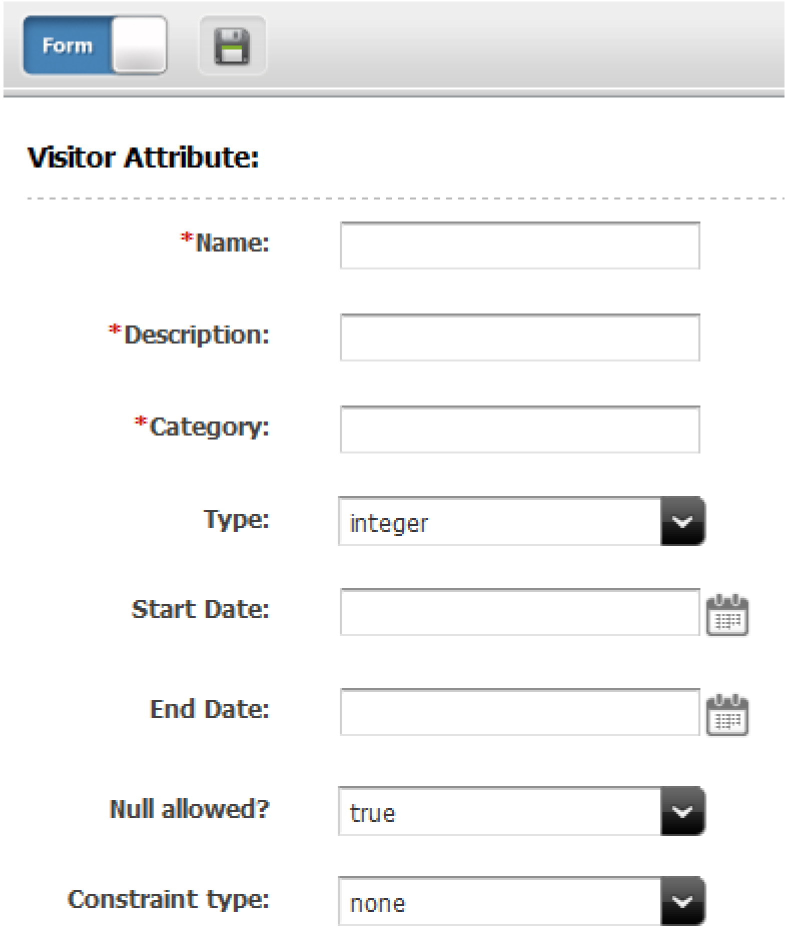
Use the procedures in this section to create visitor attributes with the Engage forms.

To create a new visitor attribute, complete the following steps:

1. Log in to WebCenter Sites as an administrator.
2. Select the site in which you want to work.
3. Select the **Admin** interface.
4. Click **New** on the button bar.
5. Select **New Visitor Attribute** from the list of asset types. (Site Visitor Attribute asset types must be enabled for your site.)

   The Site Visitor Attribute form displays.

Figure 38–1 Site Visitor Attribute



Visitor Attribute:

*Name:	
*Description:	
*Category:	
Type:	integer
Start Date:	
End Date:	
Null allowed?	true
Constraint type:	none

Note: If **Visitor Attribute** does not appear in the menu list, it means that your login password combination does not give you administrator rights. Contact the site administrator and request that the admin user profile be assigned to your user name.

1. In the **Name** field, enter a unique, descriptive name for the attribute (field). You can enter up to 32 alphanumeric characters, including spaces. The first character must be a letter.
2. In the **Description** field, enter a description of the attribute (field). Enter a value (alphanumeric characters) that will help you easily identify the attribute (attributes are listed by their descriptions rather than their names in the **Segment Filtering** forms).
3. In the **Category** field enter the category for the attribute. The text that you enter in this field determines where the attribute is listed in the Segment Filtering form. You can enter up to 32 alphanumeric characters.

Note: Categories for visitor attributes must be different from the categories for history definitions.

38.2.1.1 Configure the Data Type

1. In the Type field select a data type.
2. If you selected **string**, in the **Length** field enter the maximum number of characters allowed for input in the attribute (field). You can enter a value up to 255.
3. In the **Null allowed** field, select **true** to allow null values or **false** to require input for the attribute when it is used. For example, an attribute with a Boolean data type cannot allow a null value.
4. If you selected **false** in the **Null allowed** field, in the **Default Value** field enter a default value that is appropriate for the attribute's data type. For example, if the data type is "integer" the default value must be a number.

Note: If you selected **binary** as the data type, you cannot specify a default value for the attribute.

38.2.1.2 Configure the Constraint Criteria

The constraint options that are available for validating input into the attribute depend on the data type that you designated for the attribute.

Option 1: Configure the attribute to accept free-form text

In the **Constraint type** field select none from the drop-down list. For example, a visitor attribute named `residence` of type `string` might accept unconstrained text as input.

Option 2: Configure the attribute to accept input from a range of values

To configure the attribute to accept a specific range of values, the data type must be integer, short, long, double, or money.

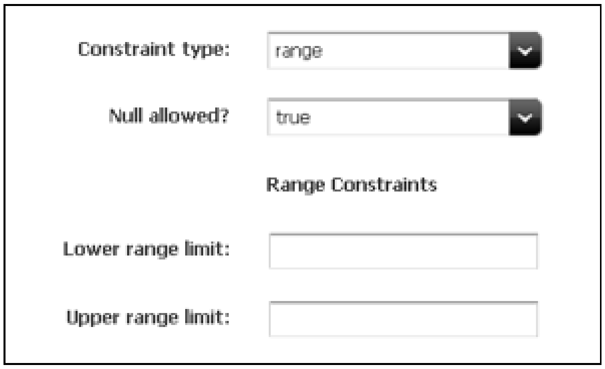
1. In the **Constraint type** field select **range**.

The form displays range fields.

2. In the **Lower range limit** field and specify the smallest possible value that can be accepted in the attribute when it is used as a field. This value cannot be a negative number.
3. In the **Upper range limit** field, enter the largest possible value that can be accepted in the attribute when it is used as a field. (For a short data type, you can enter a value up to 255; for integer, up to 65,535; for double, up to 4,294,967,295; for money, unlimited.)

   For example, an attribute named Age can be restricted to accept only values between 1 and 110.

Figure 38–2 Range Fields for the Constraint Type



Constraint type: range

Null allowed? true

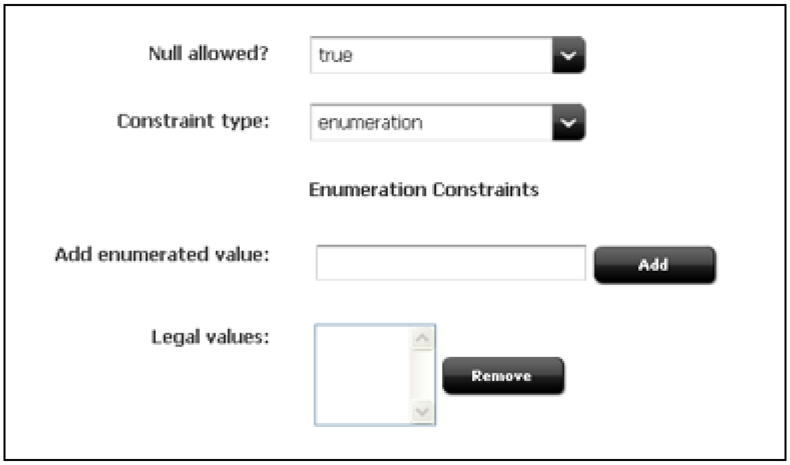
Option 3: Configure the attribute to offer a set list of values in a drop-down list

1. In the Constraint type field and select enumeration.

   The form displays text boxes for adding options.
2. In the **Add Enumerated Value** field, enter the name of the first option. For example, an attribute named "gender" can have "female" as an option.
3. Click **Add**.

   The option is moved to the list.

Figure 38–3 Enumeration Constraints



Null allowed? true

Constraint type: enumeration

4. Repeat these steps for each of the options that you want to make available for this attribute (field).

38.2.1.3 Save the Attribute

1. (Optional) If more than one site is set up and you have access to those sites, specify whether you want to share this attribute with the other sites. In the **More** menu on the Inspect form, select **Share Visitor Attribute**.
2. When you are finished configuring the visitor attribute, click **Save**.

   Engage creates an entry for this attribute in the visitor data asset tables in the WebCenter Sites database and reserves a place in the database to store information of that type for your site visitors.

   Engage displays a summary of the attribute in the Inspect form.

You can now use this visitor attribute in a segment.

> **Note:** After a visitor attribute is used to define a segment, deleting the attribute invalidates the segment. Be sure to correct your segments if you delete an attribute.

38.2.2 Creating History Attributes

The purpose of history attributes is different from the purpose of visitor attributes: you create history attributes to be used by history definitions. You cannot use them in the Segment Filtering forms until they are used to define a history definition.

Use the procedures in this section to create history attributes by using the Engage forms.

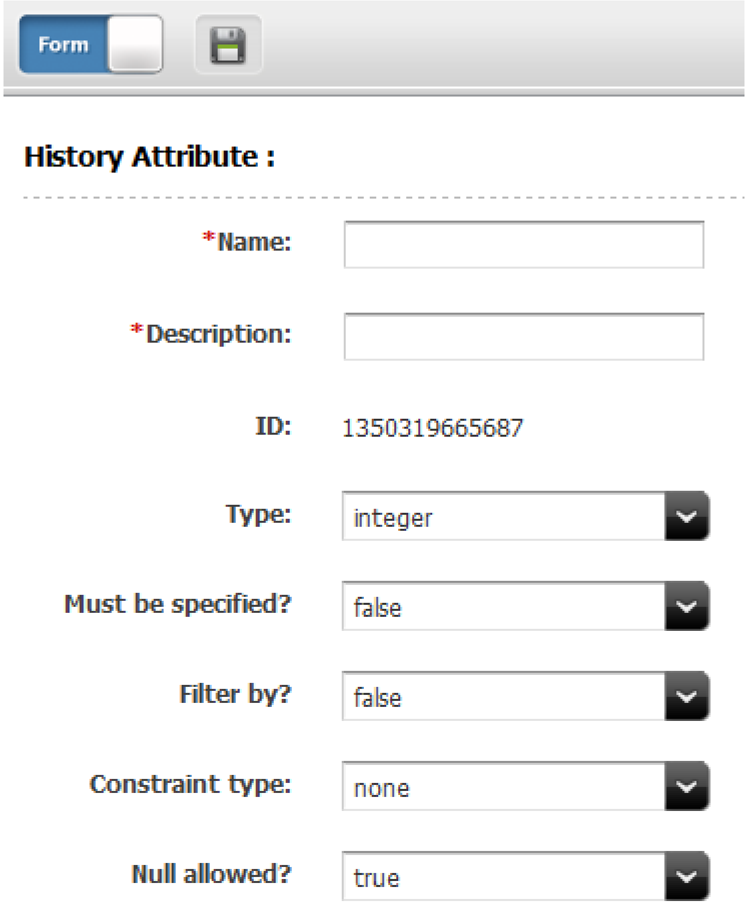
> **Note:** You cannot edit or delete a history attribute after it has been used to define a history definition. You also cannot remove it from the history definition. If you must change a history attribute after it has been used to define a history definition, it is best to stop using the history definition. Create a new history attribute, create a new history definition, and then start using the new history definition.

To create a new history attribute, complete the following steps:

1. Select the site in which you want to work.
2. Select the **Admin** interface.
3. Click **New** on the button bar and select **New History Attribute** from the list.

   The History Attribute form displays.

Figure 38–4 History Attribute Form



History Attribute :

*Name:	
*Description:	
ID:	1350319665687
Type:	integer
Must be specified?	false
Filter by?	false
Constraint type:	none
Null allowed?	true

Note: If History Attribute does not appear in the menu list, it means that your login/password combination does not give you administrator rights. Contact the site administrator and request that the admin user profile be assigned to your user name.

---

1. Log in to WebCenter Sites as an administrator.
2. In the **Name** field, enter a unique, descriptive name for the attribute (field). You can enter up to 32 alphanumeric characters, including spaces. The first character must be a letter.
3. In the **Description** field, enter a description of the attribute (field). Enter a value (alphanumeric characters) that will help you easily identify the attribute (attributes are listed by their descriptions rather than their names in the **Segment Filtering** forms).
4. In the **Type** field and select a data type.
5. If you selected **string**, in the **Length** field enter the maximum number of characters allowed for input in the attribute (field).
6. If you want this attribute to be a required field when the history definitions that use it are used to define a segment, select **true** in the **Must be specified** field.
7. Click in the **Filter by** field and select **true**.

If you do not set **Filter by** to true, the marketers cannot use the attribute (field) as a constraint for any history definition that it belongs to when they create segments.

If the data type for this attribute is numeric, then by default the attribute is included in the list of attributes that can be selected for a Total constraint in a segment, whether you set **Filter by** to true or to false. However, if you want to use a numeric attribute as a constraint in any other way, you must set **Filter by** to true.

8. Click in the **Null allowed** field and select **true** to allow null values or **false** to require input for the attribute when it is used. For example, an attribute with a Boolean data type cannot allow a null value.
9. If you selected **false** in the **Null allowed** field, in the **Default Value** field enter a default value that is appropriate for the attribute's data type. For example, if the data type is integer, the default value must be a number.

38.2.2.1 Configure the Constraint Criteria

The constraint options available for validating input into the attribute depend on the data type you designated for the attribute.

Configure the attribute to accept free-form text

Click in the **Constraint type** field and select none from the drop-down list. For example, a history attribute named `Street Name` of type `string` might accept unconstrained text as input.

Configure the attribute to accept input from a range of values

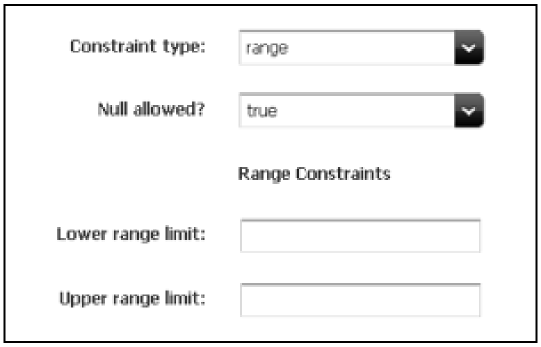
To configure the attribute to accept a specific range of values, the data type must be integer, short, long, double, or money.

1. In the **Constraint type** field, select range.

   The form displays range fields.
2. In the **Lower range limit** field, specify the smallest possible value that can be accepted in the attribute when it is used as a field. This value cannot be a negative number.
3. In the **Upper range limit** field, enter the largest possible value that can be accepted in the attribute when it is used as a field. (For a short data type, you can enter a value up to 255; for integer, up to 65,535; for double, up to 4,294,967,295; for money, unlimited.)

   For example, an attribute named "Number of Items" can be restricted to accept only values between 1 and 50.

Figure 38–5 Range Constraints and Related Fields



Constraint type: range

Null allowed? true

Range Constraints

Lower range limit:

Upper range limit:

Configure the attribute to offer a drop-down list of specific values

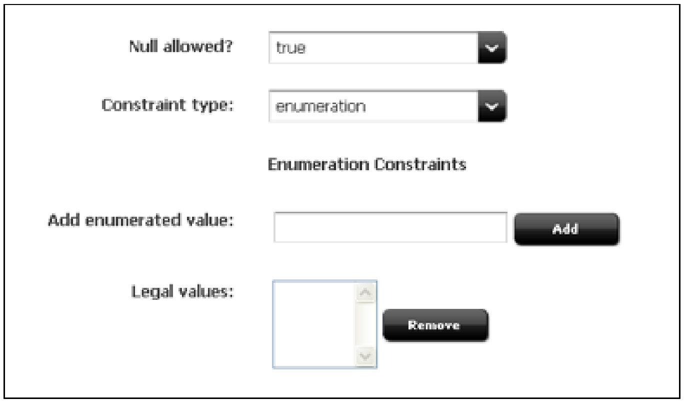
1. In the Constraint type field, select enumeration.

   The form displays text boxes for adding options.

2. In the **Add Enumerated Value** field, enter the name of the first option. .

3. Click **Add**.

Figure 38–6 Enumeration Constraints Fields



Null allowed? true

Constraint type: enumeration

Enumeration Constraints

Add enumerated value:

Legal values:

4. Repeat these steps for each of the options that you want to make available for this attribute.

38.2.2.2 Save the History Attribute

When you are finished configuring the history attribute, click **Save**.

Engage creates an entry for this attribute in the visitor data asset tables in the WebCenter Sites database and reserves a place in the database to store information of that type for your site visitors.

You can now use this history attribute to define a history definition.

38.2.3 Creating History Definitions

History definitions are made up of history attributes. Therefore, there must be at least one history attribute created before you can create a history definition.

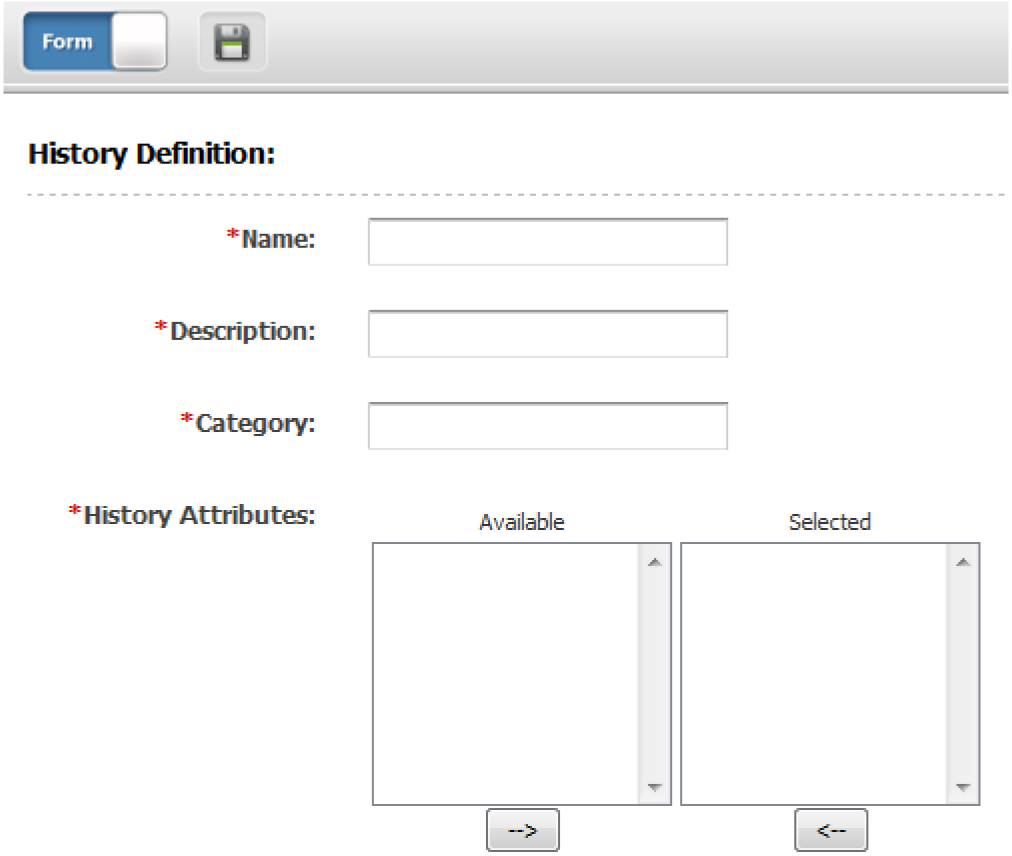
Use this procedure to create history definitions by using the Engage forms:

1. If Engage is not open, log in.

2. Click **New** and select **History definition** from the list.

   The History definition form appears:

Figure 38–7 History Definitions Fields



Note: If History definition does not appear in the menu list, it means that your login/password combination does not give you administrator rights. Contact the site administrator and request that the admin user profile be assigned to your user name.

3. In the **Name** field, enter a unique, descriptive name for the history definition (record). You can enter up to 32 alphanumeric characters, including spaces. The first character must be a letter.
4. In the **Description** field, enter a description of the history definition. Enter a value (alphanumeric characters) that will help you easily identify the history definition (history definitions are listed by their descriptions rather than their names in the **Segment Filtering** forms).
5. In the **Category** field, enter a category for the history definition. The text that you enter in this field determines how the history definition is sorted and displayed in the Segment Filtering forms. You can enter up to 32 alphanumeric characters.

Note: Categories for history definitions must be different from the categories for visitor attributes.

6. In the **Fields** area, select the history attributes that make up this history definition. Select an attribute and then click the right arrow to move it to the list on the right. Use control + click to select more than one attribute at the same time.

Note: After a history attribute is used to define a history definition, you can no longer edit or delete that history attribute.

7. Click **Save**.

   Engage creates an entry for this history definition (record) in the visitor data asset tables in the WebCenter Sites database and reserves a place in the database to store information of that type for your site visitors.

   Engage then displays a summary of the history definition in the Inspect form.

You can now use this history definition in a segment.

38.3 Verifying Visitor Data Assets

To determine that the visitor attributes, history attributes, and history definitions are properly set up, examine the Segment Filtering forms and decide whether the visitor assets that you created were configured correctly:

- Create segments that use each of the visitor attributes and history definitions that you created.
- Determine that the constraint definitions are correct and that the input ranges are accepting the correct range of input.

38.4 Approving Visitor Data Assets

When your visitor data assets are ready, approve them so that they can be published to the delivery system.

When a history definition is published, the history attributes that are used to define it are also published. That means that you have to approve all the history attributes in a history definition before the history definition can be published.

To approve any asset, select **Approve for Publish** from the drop-down list in the icon bar in the asset's Edit or Inspect form.

39

Recommendation Assets

Recommendations are assets that determine which products or content should be featured or "recommended" on a rendered page. These assets are a set of rules that might be based on the segments the visitors qualify for, and, in some cases, relationships between the product and/or content assets.

This chapter describes how recommendations work and how to create a custom element that returns assets to be recommended.

This chapter contains the following sections:

- Section 39.1, "Overview of Recommendation Assets"
- Section 39.2, "Development Process"
- Section 39.3, "Creating a Dynamic List Element"

39.1 Overview of Recommendation Assets

A recommendation asset collects, evaluates, and sorts product assets and content assets and then recommends the most appropriate flex assets (such as assets for the current visitor). How does it determine which are the most appropriate assets? By consulting the list of segments that the visitor belongs to.

The product assets and content flex assets are rated for their importance to each segment. When a recommendation asset is called from a template, Engage determines which segments the current visitor qualifies for, and then selects the assets that are identified by the recommendation as having the highest rating for those segments. These are the assets that are "recommended" to the visitor.

There are three kinds of recommendations:

- **Static Lists**: Return a static list of recommended items
- **Dynamic Lists**: Return a list of recommended items that is generated by a dynamic list element that you create
- **Related Items** : Return a list of recommended items based on relationships between flex assets, such as products.

Engage uses a recommendation's configuration options and the asset ratings to constrain the list when the list contains more items than the template is programmed to display. For related items recommendations, Engage also uses asset relationships to constrain the list. For all recommendations, Engage eliminates assets that are rated 0 for the current visitor.

The recommendation asset is the only Engage asset that can be assigned a template. You code your recommendation templates to render the items that the recommendation returns in an appropriate way on the rendered page.

The template tells the recommendation how many assets to return, and the recommendation asset determines which assets to select and return to the template based on the way it is configured and on the segments that the current visitor belongs to.

There are several XML and JSP object methods (tags) that you can use to code templates for recommendations. For information about coding templates when you are using Engage, see Chapter 40, "Coding Engage Pages." For information about all of the Engage tags see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*

39.2 Development Process

Following are the basic steps for setting up recommendations:

1. Developers and designers meet with the marketing team to define all the merchandising messages that you want to display on your site and to plan how to represent those messages using recommendation and promotion assets.
2. The developers and designers use the XML or JSP object methods to design and code templates for the recommendations. Chapter 40, "Coding Engage Pages" explains how to code these templates.
3. If the website will use dynamic list recommendations, the developers code the dynamic list elements that return the assets to recommend. Section 39.3, "Creating a Dynamic List Element" explains how to code dynamic list elements.
4. The marketing team uses the Engage recommendation wizard to create and then configure the recommendations. They assign the appropriate template to the appropriate recommendation.
5. Using the Engage product and content asset forms, the marketers rate how important the assets are to each segment, and, therefore, to the individual visitors who become members of those segments. (Typically, you assign ratings to flex parents, such as product parents, instead of to individual assets.)
6. For each "related items" recommendation, the marketers configure the relationships maintained by those recommendations by assigning related assets in the flex asset or flex parent forms. (Typically, you configure relationships among flex parents, such as product parents, instead of individual assets.)

39.3 Creating a Dynamic List Element

If your website uses dynamic list recommendations, you must code the dynamic list elements that return lists of recommended assets. A dynamic list element is an instance of the CSElement asset type; this ensures that the dynamic list element will be transferred to the delivery system when the website that uses it is published.

A dynamic list element must return a list named `AssetList`. The set of assets that becomes your `AssetList` must have the following traits:

- It must contain only assets of the types that you want to recommend.
- It must contain the IDs of the assets that you want to recommend.
- It should contain the assets' confidence ratings, although this is optional.

The following sample code is an excerpt from a dynamic list element. Line 2 in the following code adds a constraint to the ssprod searchstate, filtering it to find items with a browse category of Fund Type. Line 3 adds another constraint to the ssprod searchstate, creating an assetset composed entirely of Product assets. Finally, line 4 turns the assetset created in line 3 into the AssetList list.

```
1. <SEARCHSTATE.CREATE NAME="ssprod"/>
2. <SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="ssprod" TYPENAME="PAttributes"
   ATTRIBUTE="BrowseCategory" VALUE="Fund Type"/>
3. <ASSETSET.SETSEARCHEDASSETS NAME="asprod" CONSTRAINT="ssprod"
   ASSETTYPES="Products"/>
4. <ASSETSET.GETASSETLIST NAME="asprod" LISTVARNAME="AssetList"/>
```

When you have completed coding your dynamic list elements, provide their names and information about what sort of content they return to the users who will create the recommendation assets.

40 Coding Engage Pages

This chapter presents information about designing an online site that gathers visitor information and personalizes promotional messages for each visitor based on that information.

This chapter contains the following sections:

> **Note:** This chapter refers to specific XML tags that you use to accomplish the tasks being described. In all cases, there are also equivalent JSP tags. The XML and JSP tags are all documented in the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

40.1 Commerce Context and Visitor Context

During a visitor's session at an Engage site, a visitor context is created for that visitor. Five types of session objects are placed in the visitor context:

- Current shopping cart
- List of segments that the visitor belongs to
- List of promotions that the visitor qualifies for
- Time object that is used for calculating time-based rules for segments and promotions
- Utility object that gives you, the developer, access to product attributes

The commerce context encompasses the visitor context and gives you access to it.

There are two sets of XML and JSP object methods that serve as your interface to these contexts:

- Commerce context methods, which you use to place objects in the visitor context.

- Visitor Data Manager methods, which you use to gather, store, and retrieve visitor data and to associate a visitor's data with the correct visitor.

40.2 Identifying Visitors and Linking Sessions

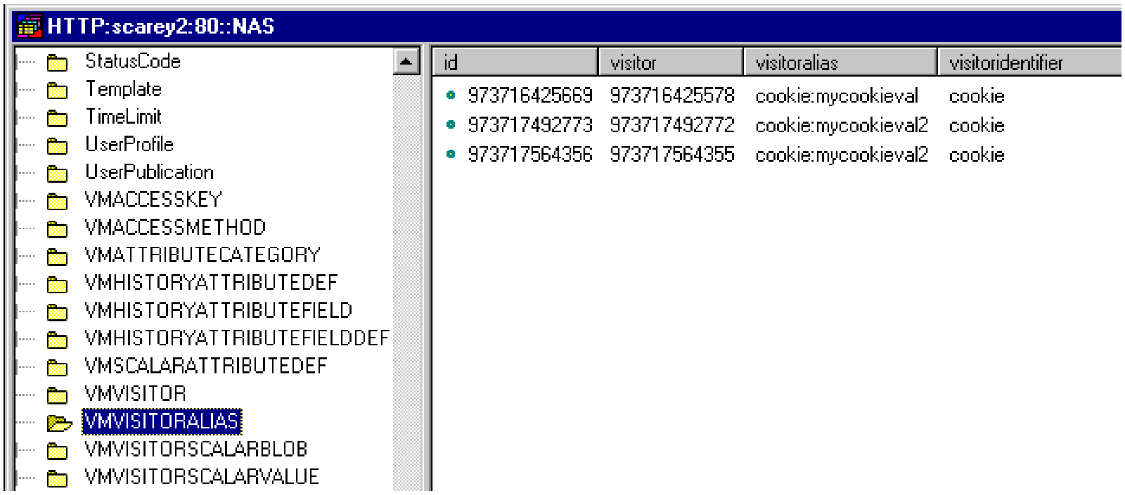
Engage creates a unique visitor ID for each visitor for each session. It stores these IDs in the `VMVISITOR` table in the WebCenter Sites database. The data gathered for a visitor during that session is identified by that visitor ID. To link the data gathered from one session to the data from another, your site pages must assign aliases that link those visitor IDs.

You use the following Visitor Data Manager object method to create an alias:

```
<VDM.SETALIAS KEY="keyvalue" VALUE="aliasvalue"/>
```

When you use this tag, Engage associates the visitor session ID with the alias, and writes them both to the `VMVISITORALIAS` table.

Figure 40–1 VMVISITORALIAS Table



id	visitor	visitoralias	visitoridentifier
973716425669	973716425578	cookie:mycookieval	cookie
973717492773	973717492772	cookie:mycookieval2	cookie
973717564356	973717564355	cookie:mycookieval2	cookie

The values in this table link the data that is gathered in separate sessions to the same visitor because the alias provides a link to the visitor IDs that are recorded for that visitor. In the illustration above, the data recorded in the session associated with the visitor ID 973717492772 is linked to the data associated with the visitor ID 973717564355 because they have aliases with the same key/value pair.

All visitor information that is associated with sessions that are linked through common aliases. That is, aliases with the same key/value pairs can be accessed during the current session. It is considered current visitor information.

You can create aliases with cookies, with login IDs, or with any other unique identifier that your site uses to recognize visitors.

The `VMVISITORALIAS` table grows quickly. For information about managing it and the other visitor data tables, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

40.3 Collecting Visitor Data

To collect visitor data, you must program your online pages to gather it, validate it, and then write it to the WebCenter Sites database.

There are three Visitor Data Manager object methods that write this information to the database:

- `<VDM.SETSCALAR ATTRIBUTE="attribute" VALUE="value"/>` records visitor attributes.
- `<VDM.RECORDHISTORY ATTRIBUTE="attribute" LIST="valuelist"/>` records history definitions.
- `<VDM.SAVESCALAROBJECT ATTRIBUTE="attribute" OBJECT="objectname"/>` records visitor attributes of type binary. The demo site delivered with Engage uses this method to store shopping carts across sessions and to store saved searches for visitors.

> **Note:** Because these tags write information to the database, they can be a factor in the performance of your delivery system. Be sure to use them efficiently.

These are the tables that store the visitor data:

Table 40–1 Object Methods and Database Tables

XML or JSP Object Method	Database Table That It Writes To
`VDM.SETSCALAR` `vdm:setscalar`	`VMVISITORSCALARVALUE`
`VDM.SAVESCALAROBJECT` `vdm:savescalarobject`	`VMVISITORSCALARBLOB`
`VDM.RECORDHISTORY` `vdm:recordhistory`	`VMz------------` (These tables are dynamically generated for each history definition. Engage creates a unique table for each one.)

These tables grow quickly. For information about managing them, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

There are also a number of Visitor Data Manager object methods that retrieve this information from the WebCenter Sites database. See Chapter 10, "Error Logging and Debugging."

> **Note:** For information about these and other Engage XML and JSP tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*

40.4 Coding Site Pages That Collect Visitor Data

This section presents an overview of the general steps that you follow when you code your site pages to collect visitor data:

1. Create forms to capture the data that you need your visitors to manually provide. It is a good practice to create form fields with names that match the names of the attributes that you created. See Chapter 38, "Creating Visitor Data Assets" for more information.

   Attributes are listed by their descriptions rather than by their names in the Engage Segment forms. Be sure that you do not confuse their attribute names with attribute descriptions when you are creating form fields or writing values to the WebCenter Sites database.

2. Create a "submit" page that validates the data that the visitor entered in the fields (either by using JavaScript or with a server-side validation method). The input data must comply with the constraints that you set for the attributes. For example, if you created a visitor attribute of type string with a length of 30, be sure that the form does not try to submit data from the form field with a length of 31.
3. Program the "submit" page to write the validated data to the WebCenter Sites database. Be sure to use the names of the attributes and history definitions and not their descriptions. Here are some examples:

This section contains the following topics:

- Section 40.4.1, "Example 1: Visitor Attributes"
- Section 40.4.2, "Example 2: History Definition"
- Section 40.4.3, "Example 3: Visitor Attribute of Type Binary"

40.4.1 Example 1: Visitor Attributes

```
<!-- Write the registration information to the database.-->
<VDM.SETSCALAR ATTRIBUTE="name" VALUE="Variables.name"/>
<VDM.SETSCALAR ATTRIBUTE="age" VALUE="Variables.age"/>
<VDM.SETSCALAR ATTRIBUTE="jobdesc" VALUE="Variables.jobdesc"/>
```

40.4.2 Example 2: History Definition

Because history definitions hold multiple values as an aggregate, you must create a list of the data before you can write it to the database. In this example, a form writes an order to the WebCenter Sites database:

```
<!-- Write the order details to a list. -->
<!-- assume that Variables.order_id is set to the order id -->
<!-- assume that Variables.wasCouponUsed is set to 1 (yes) or 0 (no) -->
<!-- assume that Variables.shippingtype is set to UPS or FedEx -->
<!-- assume that Variables.order_price is set to the total amount of the
order -->

<LISTOBJECT.CREATE NAME="histList" COLUMNS="orderid, shippingtype, price,
couponUsed"/>
<LISTOBJECT.ADDROW NAME="histList" orderid="Variables.order_id"
shippingtype="Variables.shippingtype" price="Variables.order_price"
couponUsed="Variables.wasCouponUsed"/>
<LISTOBJECT.TOLIST NAME="histList" LISTVARNAME="itemList"/>

<!-- Write the list to the history definition named visitorOrderHistory in the
WebCenter Sites database.-->
<VDM.RECORDHISTORY ATTRIBUTE="visitorOrderHistory" LIST="itemList"/>
```

And you can use that record to determine information about how many orders a visitor had made, when their first or last purchase was, and the total amount they've spent.

40.4.3 Example 3: Visitor Attribute of Type Binary

Binary visitor attributes allow you to convert an object from the WebCenter Sites name space into a binary form. The sample site delivered with Engage uses two visitor attributes of type binary: one to store shopping carts across sessions and one to store saved searches.

To see these examples, use Oracle WebCenter Sites Explorer to examine `ElementCatalog/OpenMarket/Demos/CatalogCentre/GE/Navigation/stylesheet.xml` and `ElementCatalog/OpenMarket/Demos/CatalogCentre/GE/myge.xml`

1. If you want to gather data about visitor behavior (clickstream information, for example), you can program your pages to gather that without using input forms. For example, the demo site delivered with Engage uses a history definition to record the number of times a visitor browses the site.

   For this examples, use Oracle WebCenter Sites Explorer to examine `ElementCatalog/OpenMarket/Demos/CatalogCentre/GE/Navigation/stylesheet.xml`

2. Whenever visitor data is written to the database, segments and promotions can also change. Therefore, after any change to visitor data, be sure to recalculate the segments and promotions lists. There are two Commerce Context object methods that you can use:

   ```
   COMMERCECONTEXT.CALCULATEPROMOTIONS
   ```

   ```
   COMMERCECONTEXT.CALCULATESEGMENTS
   ```

   `COMMERCECONTEXT.CALCULATEPROMOTIONS` recalculates both the segments that the visitor belongs to and the promotions that apply to those segments.

3. Whenever visitor data is written to the database, ratings for assets can also change. Therefore, after any change to visitor data, be sure to refresh the ratings of any assets that are in an existing asset set.

   Use the `ASSETSET.ESTABLISHRATINGS` tag to refresh the asset ratings of the assets in a set.

> **Note:** For information about these and other Engage XML and JSP object methods, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*

40.5 Templates and Recommendations

The key Commerce Context object method for invoking a recommendation asset is this one:

```
<COMMERCECONTEXT.GETRECOMMENDATIONS COLLECTION="recommendationname"
[LIST="inputlist" VALUE="rating" MAXCOUNT="assetcount"] LISTVARNAME="assetlist"/>
```

This method retrieves and lists the assets that match the recommendation constraints passed to the method. It uses the following arguments:

- `COLLECTION`: The name of the recommendation. If you plan to use the same template for several recommendations, code the template to supply the identity of the recommendation through a variable.
- `LIST`: The name of the list of assets; this is the name that you want to be used as the input for the calculation.

  You use this argument when the recommendation named by `COLLECTION` is a context-based recommendation. Columns are `assettype` and `assetid`. You can create this list by creating a list object and adding rows for each asset that you want to use as input. For an example, use Oracle WebCenter Sites Explorer to examine `ElementCatalog/OpenMarket/Demos/CatalogCentre/GE/cart.xml`.

- `VALUE`: The default rating for assets that do not have one. If you do not declare a value, unrated assets are assigned a default rating of 50 on a scale of 0-100. It is recommended that you keep this value set to 50.
- `MAXCOUNT`: (Optional.) The maximum number of assets to return. Use this value to constrain the list of recommended assets.
- `LISTVARNAME`: The name that you want to assign to the list of assets. Its columns are: `assettype` and `assetid`.

If the segment list and the promotion list have not yet been created and placed in the visitor context, this object method invokes the methods that calculate them. Remember that promotions do not have templates, they override the template that a recommendation is using. If there are any promotions that apply to the current visitor **and** that override the recommendation named by the `COLLECTION` argument, the object method returns the ID of the promotion asset rather than the items identified by the recommendation asset.

> **Note:** The `COMMERCECONTEXT.GETSINGLERECOMMENDATION` object method returns one recommended asset based on the recommendation criteria passed to the method. Typical uses for this method are to feature one product or to put one product on sale. See the *Oracle Fusion Middleware WebCenter Sites Tag Reference* for information about this object method and its JSP equivalent.

Before you begin coding the templates for recommendations, be sure to complete the following tasks:

- Meet with the marketing team to define all the merchandising messages that you want to display on your site and plan how to represent those messages in recommendations and promotions.

  For example, do you want to display a list of links to other products? What information should the link include? The product name only or also the price? What will be displayed when a recommendation returns a promotion rather than a list of assets?
- Determine where and on which pages the recommended assets from each recommendation will be displayed.

Creating Templates for Recommendations

To use templates to render items that are returned by recommendation assets, you must complete at least the following basic steps:

1. Create a template element that invokes a recommendation asset. Use the object method described in the preceding section.
2. Code the template to display the items that are returned by the recommendation. The returned items are stored in a variable designated by the `LISTVARNAME` argument. This list includes the asset IDs and asset types of those items. Use that information to extract the asset attributes that you want to display. (Name, price, SKU, for example.)

   You can use the `ASSETSET.SETLISTEDASSETS` and `ASSETSET.GETASSETLIST` object methods to sort and display the returned assets and their attributes.

   For an example, use Oracle WebCenter Sites Explorer to examine `ElementCatalog/OpenMarket/Demos/CatalogCentre/Templates/GE/recommendation.xml`.

3. Open Engage. Under New, select Template. Create a corresponding Template asset for this template element. Enter a name that describes what the element does so that when you create a recommendation asset you know which template to assign to it. Identify the path to the element (its location in the ElementCatalog) in the Element Name field.
4. Publish the Template asset when other assets are published.
5. Render the recommendations on the appropriate site pages.

Creating Templates for Recommendations Using Oracle Real-Time Decisions

Oracle Real-Time Decisions (RTD) is an engine that helps site visitors make decisions by recommending the best options when they make their choices. When the `commercecontext:getrecommendations` tag is invoked, it sends a list of recommendations and certain visitor profile information to Oracle RTD. Oracle RTD then ranks the recommendations and, using the `maxcount="<n>"` parameter, returns a refined list of *n* recommendations best suited for the visitor's profile.

> **Note:** This release of WebCenter Sites supports Oracle Real-Time Decisions (RTD) version 3.0 series. RTD 11*g* is not supported.

Oracle RTD (and other engines) can be integrated with WebCenter Sites by use of the following tags:

- The `commercecontext:getrecommendations` tag, where `engine` and `engineparameters` are generic parameters that can be set to invoke any engine. For integration with Oracle RTD, the engine parameter must be set to `rtd` and engineparameters must name a list of the following Oracle RTD-specific parameters: `advisor` (integration point), `attributes` (visitor profile), `sessionkey`, and `assetattributes`.
- The `commercecontext:inform` tag, which also takes the parameters `engine` and `engineparameters`. For this tag, engineparameters must name a list of the following parameters: `informant` (integration point), `attributes` (visitor profile), and `sessionkey`.

> **Note:** The `choices` parameter (choice of recommendations) was removed from this tag. The list of choices is now passed to RTD via the list `tag` attribute.

Once Oracle RTD learns the list of attributes from the `commercecontext:getrecommendations` tag, you, the developer, can use the `sessionkey` parameter to map visitors to their profiles for subsequent `commercecontext:inform` calls. For example, once a `commercecontext:getrecommendations` call has been made to Oracle RTD, the next call from the same visitor may pass an empty string for attributes and use the same `sessionkey` to get recommendations.

> **Note:** Using the `commercecontext:getrecommendations` and `commercecontext:inform` tags to invoke Oracle RTD requires adding the following properties to the `futuretense_xcel.ini` file: `rtd.inline.service.name` and `rtd.host`.

For more information about the commercecontext tags, parameter definitions, integration with Oracle RDT, and sample code, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*. For information about the rtd.inline.service.name property, rtd.host property, and rtd.choiceId.pattern property, see the *Oracle Fusion Middleware WebCenter Sites Property Files Reference*.

40.6 Shopping Carts and Engage

When you are using the shopping cart interface with Engage, there are a number of additional facts and tips to keep in mind while you code your shopping cart pages:

- If your site uses promotions, you must code your cart pages to apply the discounts from the promotions.

  Use the COMMERCECONTEXT.DISCOUNTCART and COMMERCECONTEXT.DISCOUNTTEMPCART object methods to apply promotional discounts to the shopping cart.
- It is good practice to clear existing discounts from the cart before applying them again.
- You can store carts across sessions by writing them to the database as a visitor attribute of type binary (a scalar object). Be sure to write the cart object to the database each time the cart is modified.
- If your site uses a visitor login feature, there can be conditions under which you should merge shopping carts. For example, a visitor adds products to her cart before she logs in. Then, when she logs in, Engage finds a stored cart that also has items in it. In such a case, you would want to merge the carts.

For information about the CART object methods and their JSP equivalents, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

For an example of a Engage shopping cart, use Oracle WebCenter Sites Explorer to examine ElementCatalog/OpenMarket/Demos/CatalogCentre/GE/cart.xml.

40.7 Debugging Site Pages

During your development phase, you must verify that session linking is set up correctly, that specific attributes obtain the value that you expect, and that recommendations return the items that you expect. There are several Engage object methods that you can use to retrieve and review information and values by writing information to a browser window or to the JRE log.

This section lists the Visitor Data Manager object methods that you will probably use the most. For information about these and any other XML and JSP object methods, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*

This section contains the following topics:

- Section 40.7.1, "Session Links"
- Section 40.7.2, "Visitor Data Collection"
- Section 40.7.3, "Recommendations and Promotions"

40.7.1 Session Links

Use the following Visitor Data Manager object methods to verify that pages that handle session linking are creating the aliases correctly:

- `<VDM.GETALIAS KEY="keyvalue" VARNAME="varname"/>`

  Retrieves an alias

- `<VDM.GETCOMMERCEID VARNAME="varname"/>`

  Retrieves the visitor's commerce ID from session data.

- `<VDM.GETACCESSID KEY="pluginname" VARNAME="varname"/>`

  Retrieves the visitor's access ID from session data.

40.7.2 Visitor Data Collection

Use the following Visitor Data Manager object methods to retrieve values stored for specific visitor attributes, history attributes, and history definitions (records):

- `<VDM.GETSCALAR ATTRIBUTE="attribute" VARNAME="varname"/>`

  Retrieves a specific visitor attribute.

- `<VDM.LOADSCALAROBJECT ATTRIBUTE= "attribute" VARNAME= "varname"/>`

  Retrieves (materializes) an object stored as a visitor attribute of type binary.

-
  ```
  <VDM.GETHISTORYCOUNT
  ATTRIBUTE="attribute"VARNAME="varname"[STARTDATE="date1"
  ENDDATE="date2"LIST="constraints"]/>
  ```

  Retrieves the number of history definition records that were recorded for the visitor that match the specified criteria.

-
  ```
  <VDM.GETHISTORYSUM ATTRIBUTE="attribute"
  VARNAME="varname"[STARTDATE="date1" ENDDATE="date2"
  LIST="constraints"]FIELD="fieldname"/>
  ```

  Sums the entries in a specific field for the specified history definition.

-
  ```
  <VDM.GETHISTORYEARLIEST VARNAME="varname"
  [STARTDATE="date1"ENDDATE="date2" LIST="constraints"]/>
  ```

  Retrieves the timestamp of the first time the specified history definition was recorded for this visitor.

-
  ```
  <VDM.GETHISTORYLATEST
  VARNAME="varname"[STARTDATE="date1"ENDDATE="date2" LIST="constraints"]
  />
  ```

  Retrieves the timestamp of the last time (that is, the most recent time) the specified history definition was recorded for this visitor.

40.7.3 Recommendations and Promotions

Use the following Commerce Context object methods to verify pages that display recommendations and promotions:

- `<COMMERCECONTEXT.CALCULATESEGMENTS/>`

  Lists the segments that the visitor belongs to. It examines the available visitor data, compares it to the data types that define the segments, and then lists the segments that are a match.

- `<COMMERCECONTEXT.GETPROMOTIONS LISTVARNAME="promotionlist"/>`

  Creates the list of promotions that the current visitor is eligible for

- `<COMMERCECONTEXT.GETRATINGS ASSETS="assetlist" LISTVARNAME="ratinglist" DEFAULTRATING="defaultrating"/>`

  Calculates the ratings of the assets in a named list according to how important the asset is to this visitor based on the segments that the visitor belongs to.

- `<COMMERCECONTEXT.GETSEGMENTS LISTVARNAME="segmentlist"/>`

  Retrieves the list of segments that the current visitor belongs to.

41

Memory-Centric Visitor Tracking

Site visitors who browse Engage assets typically provide information about themselves in a personal profile. Demographics information is also collected as visitors browse. To prevent overloading WebCenter Sites's database with large amounts of visitor data, and therefore to improve performance, memory-centric visitor tracking was developed for Engage assets.

This chapter describes the memory-centric method of tracking visitors and outlines requirements for implementing this method.

This chapter contains the following sections:

- Section 41.1, "Overview of Memory-Centric Visitor Tracking"
- Section 41.2, "Enabling Memory-centric Visitor Tracking"
- Section 41.3, "How Memory-centric Visitor Tracking Works"

41.1 Overview of Memory-Centric Visitor Tracking

When large numbers of visitors browse a website, storing all of their personal information in a single repository degrades the performance of the delivery system. For this reason, sites that collect demographic information on visitors often require an additional repository to help improve site performance. Additional performance gain can be achieved by preventing load on WebCenter Sites's database.

WebCenter Sites supports add-on repositories, enabling Engage developers to implement a repository of their own choice to store visitor scalar attribute values. Custom code must be written to store and retrieve visitor information to and from the repository. In addition, WebCenter Sites's memory-centric visitor tracking method must be enabled. Differences between memory- and database-centric methods are outlined in this overview. Information about enabling memory-centric tracking is provided in the rest of this chapter, followed by diagrams illustrating how memory-centric visitor tracking works.

This section contains the following topics:

- Section 41.1.1, "Database-centric Model"
- Section 41.1.2, "Memory-centric Model"

41.1.1 Database-centric Model

In database-centric visitor tracking, performance issues on Engage-enabled websites arise for the following reasons:

- All visitor information is stored in WebCenter Sites's database. Multiple database accesses are required to store and retrieve information.
- When dynamic recommendations are used and a large amount of data is returned, WebCenter Sites logs a greater number of `SystemItemCache` entries, which leads to performance degradation. To restore performance, the `render:overridedeps` tag was introduced as a way of limiting the number of dependencies that are logged in WebCenter Sites's database. This tag applies to both the database and memory-centric models.

41.1.2 Memory-centric Model

Memory-centric tracking improves system performance by reducing load on WebCenter Sites's database.

- All computations are performed in memory:
  - When custom code retrieves visitor scalar attribute values (such as age and gender) from the add-on repository, the memory-centric model stores the retrieved values in memory and uses them with history attribute values to compute segments to which the visitor belongs.
  - Memory-centric tracking computes statistics on various history attribute values in memory and then caches the statistics. Statistics include sums, counts, oldest, and newest.
- Alias and history attribute values are stored in WebCenter Sites's database, but in a way that reduces both the volume of visitor information and the number of calls required to access the information.

As in database-centric tracking, developers can use the `render:overridedeps` tag to specify the number and types of Engage asset dependencies to log in WebCenter Sites's database. Enabling memory-centric tracking requires setting a property in `visitor.ini` and writing supporting template code.

41.2 Enabling Memory-centric Visitor Tracking

This section contains the following topics:

- Section 41.2.1, "Visitor Tracking Property"
- Section 41.2.2, "Supporting Code"
- Section 41.2.3, "Batch-Saving History Attributes to the Database"

41.2.1 Visitor Tracking Property

Memory-centric visitor tracking is enabled by the property `vis.useSessionVisitorConnection`, which must be manually added to WebCenter Sites's `visitor.ini` file **on the delivery system**:

- Setting `vis.useSessionVisitorConnection=true` enables memory-centric visitor tracking. Supporting template code must also exist on the delivery system in order for Engage visitors to be correctly tracked; pre-existing code can be reused. For more information, see Section 41.2.2, "Supporting Code."
- Setting `vis.useSessionVisitorConnection=false`, leaving it blank, or omitting it from `visitor.ini` enables the database-centric method.

Note: On all content management systems, `vis.useSessionVisitorConnection` must be either set to `false` or omitted, thus enabling the database-centric method, which allows visitor attributes to be created and otherwise managed. (Visitor attribute management is supported only on content management systems that are enabled for database-centric tracking. It is not supported on delivery systems; visitor attributes must be published to the delivery system.)

41.2.2 Supporting Code

The skeleton template in this section shows the type of code that must be written to support memory-centric tracking.

Note: How the template code executes depends on the value of the `vis.useSessionVisitorConnection` property. If the value is set to `false`, the database method of tracking visitors is used. If the value is set to `true`, memory-centric tracking takes effect.

```
vdm:setalias
retrieve visitor scalar attributes from add-on repository
vdm:setscalar
commercecontext:calculatesegments
commercecontext:getrecommendations
render Engage assets
vdm:recordhistory
...
render:overridedeps
```

For more information about this template, see Section 41.3, "How Memory-centric Visitor Tracking Works."

41.2.3 Batch-Saving History Attributes to the Database

As of WebCenter Sites 7.5 Patch 2, history attributes are saved first to memory, then to the file system, and finally as a batch to WebCenter Sites's database. Batch-saving prevents excessive memory usage (which would otherwise occur when data is produced faster than it can be saved to the database). The batch-save process batches one `HistoryAttributeDef` table at a time from the file and saves the tables to the database in bulk.

Batch saving to the database requires Engage to have access to database information. Database access is enabled by adding a file named `<dataSourceName>.properties` to the classpath to specify the following information: `driver`, `url`, `user`, and `password`. For example, if the data source name is `csDataSource`, then a file named `csDataSource.properties` must be placed in the classpath. In our example, properties in the file are set as follows:

```
driver=com.jnetdirect.jsql.JSQLDriver
url=jdbc:JSQLConnect://localhost:1433/database=TomcatDB
user=tomcatuser
password=tomcatuser
```

The following Java JVM parameter was added to control the time interval at which threads are spawned for asynchronous batch-saving of history attributes from memory to WebCenter Sites's database via the file system:

```
-Dvisitor.SyncInterval=<seconds>
```

If left unspecified, the parameter value defaults to 30 seconds.

41.3 How Memory-centric Visitor Tracking Works

Diagrams in the rest of this chapter show how memory-centric tracking works.

> **Note:** In the diagrams that follow, the add-on repository is a system of your own choice, used to store and retrieve visitor scalar attribute values. Custom code must be written to store and retrieve the values.

This section contains the following topics:

- Section 41.3.1, "Visitor Detection"
- Section 41.3.2, "Retrieving Scalar Values"
- Section 41.3.3, "Collecting History Attribute Values"
- Section 41.3.4, "Computing Sums and Counts"
- Section 41.3.5, "Computing Segments"
- Section 41.3.6, "Determining and Displaying Recommended Assets"
- Section 41.3.7, "Logging Dependencies"

41.3.1 Visitor Detection

A first-time visitor enters the site and creates a personal profile. The following events occur:

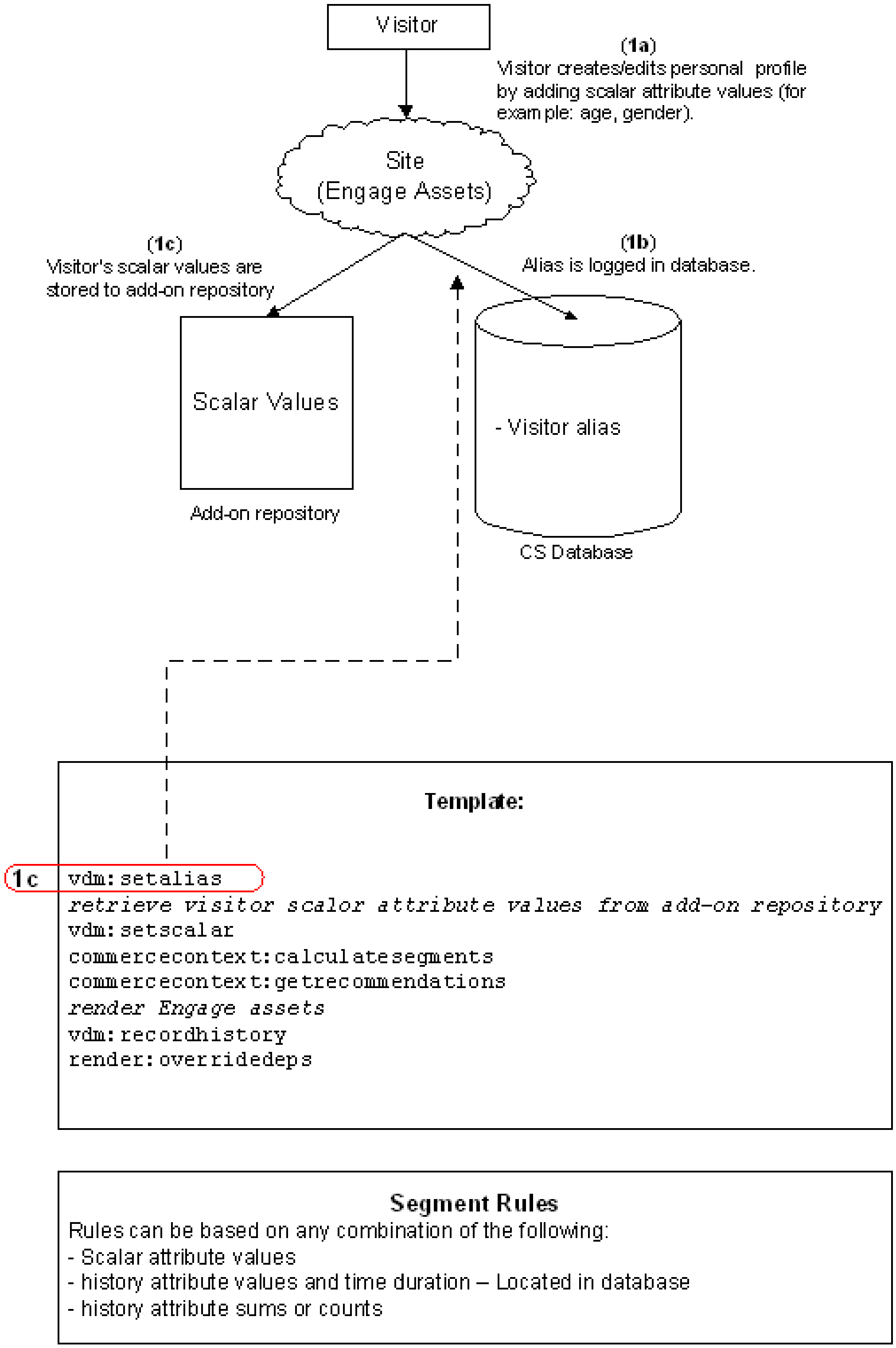
1. The `vdm:setalias` tag in the template code is used to log the visitor's alias to WebCenter Sites's database. In subsequent sessions, the visitor is automatically recognized. As a result, only a single ID unique to the visitor is kept in the database, which improves performance.

   Note that memory-centric visitor tracking logs a *unique ID per visitor*, whereas the existing database-centric method logs a *unique ID per visit* even for returning visitors. Extra IDs create extra load on the database and reduce performance.

2. The visitor's scalar values (personal information, such as age and gender) are stored to the add-on repository (by custom code).

In the remaining steps, WebCenter Sites gathers and computes information that it needs to determine the visitor's segments and therefore which Engage assets to display to the visitor.

Figure 41–1 Visitor Detection Flow


Visitor
(1a)
Visitor creates/edits personal profile by adding scalar attribute values (for exam ple: age, gender).
Site
(Engage Assets)
(1c)
Visitor's scalar values are stored to add-on repository
(1b)
Alias is logged in database.
Scalar Values
Add-on repository
- Visitor alias
CS Database
Template:
1c

```
vdm:setalias
retrieve visitor scalor attribute values from add-on repository
vdm:setscalar
commercecontext:calculatesegments
commercecontext:getrecommendations
render Engage assets
vdm:recordhistory
render:overridedeps
```

Segment Rules

Rules can be based on any combination of the following:

- Scalar attribute values
- history attribute values and time duration – Located in database
- history attribute sums or counts

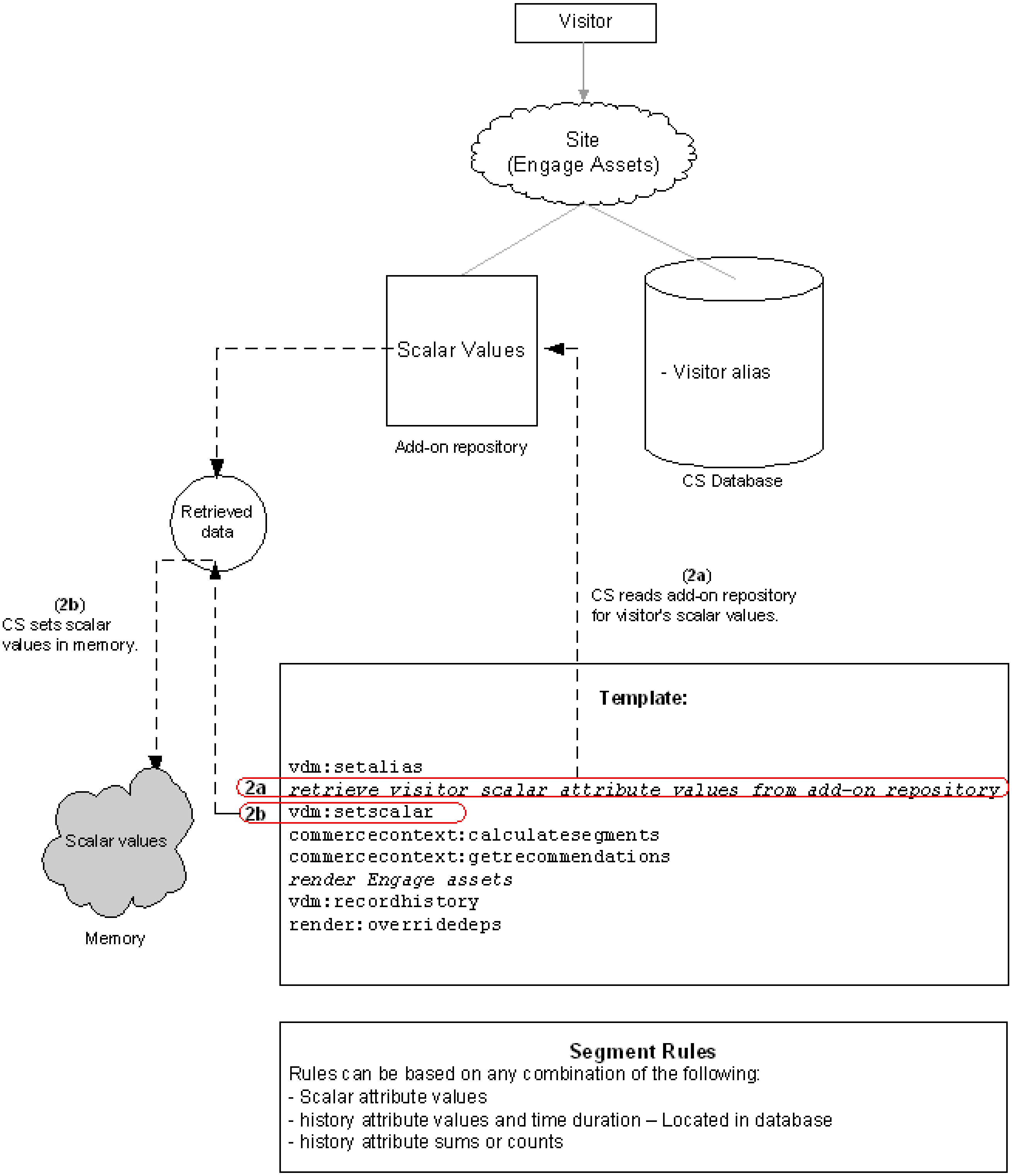
41.3.2 Retrieving Scalar Values

WebCenter Sites begins computing the visitor's segments. Starting with scalar values:

1. WebCenter Sites retrieves scalar values from the add-on repository (using custom code).
2. WebCenter Sites sets retrieved scalar values in memory, using the `vdm:setscalar` tag.

   The session-based implementation stores visitor information (scalar values) only in the http session. Therefore, when a new session begins, visitor information is made available through the add-on repository, instead of WebCenter Sites's database. A new level of caching has been added to optimize the performance of querying for visitor scalar values.

Figure 41–2 Scalar Value Retrieval Flow



```
vdm:setalias
retrieve visitor scalar attribute values from add-on repository
vdm:setscalar
commercecontext:calculatesegments
commercecontext:getrecommendations
render Engage assets
vdm:recordhistory
render:overridedeps
```

Segment Rules

Rules can be based on any combination of the following:

- Scalar attribute values
- history attribute values and time duration – Located in database
- history attribute sums or counts

41.3.3 Collecting History Attribute Values

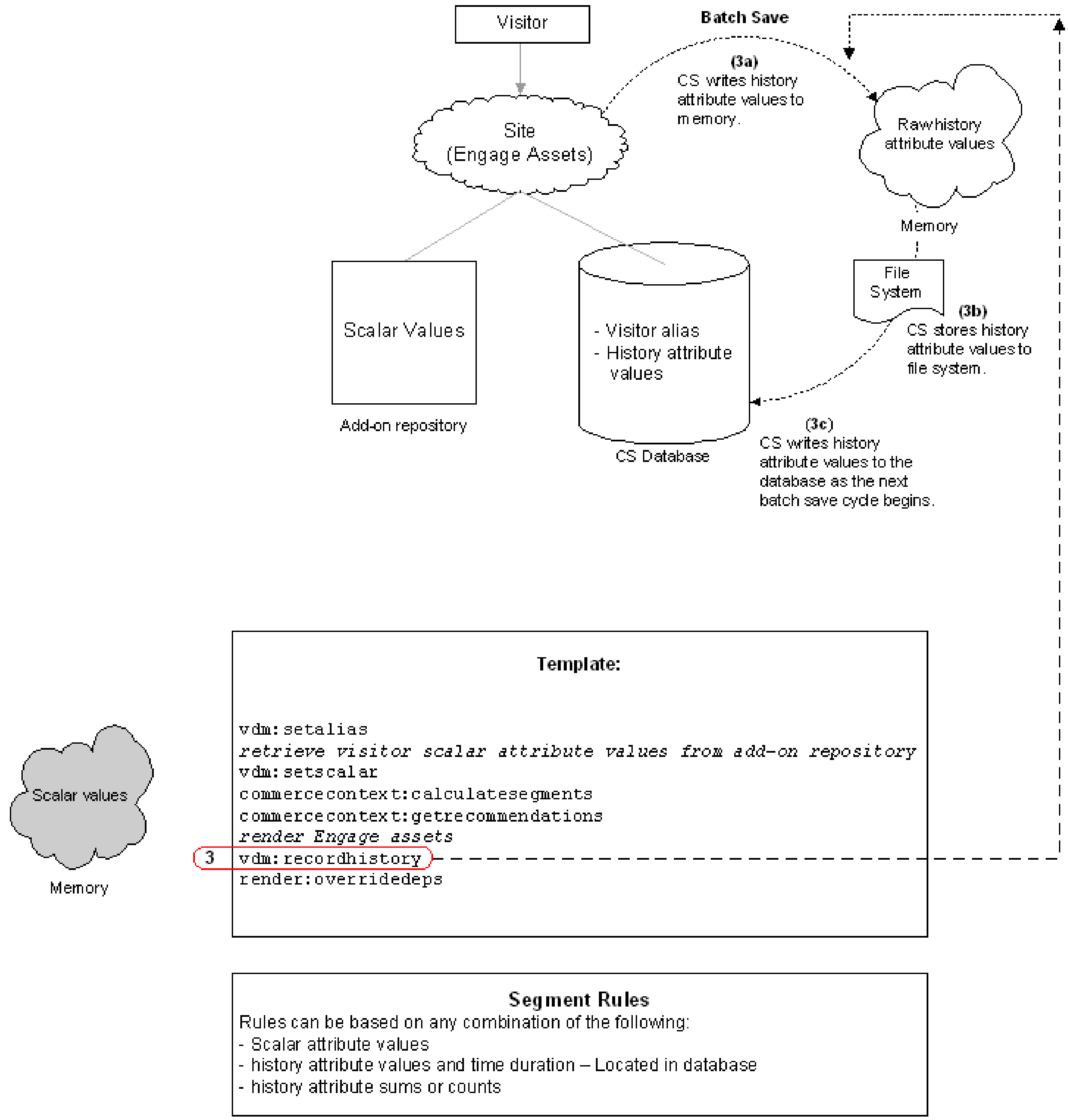
History attribute values are also required for computing a visitor's segments. During the session, history attribute values are collected cyclically:

1. The `vdm:recordhistory` tag is used to collect history attribute values into memory at one-minute intervals.
2. The history attribute values are saved to the file system and then written to WebCenter Sites's database in batches. The memory is cleared and the cycle starts again.

   Visitor history attribute values are collected first in memory to avoid the need for acquiring database connections at each call. They are saved to the file system to enable batch saving to the database, which minimizes memory and database connections usage, and increases the availability of history attribute values.

When history attribute values are committed to the database, computation of sums and counts begins, as shown in Section 41.3.4, "Computing Sums and Counts."

Figure 41–3 History Attribute Value Flow


Visitor
Batch Save
(3a)
CS writes history attribute values to memory.
Site (Engage Assets)
Raw history attribute values
Memory
File System
(3b)
CS stores history attribute values to file system.
Scalar Values
Add-on repository
- Visitor alias
- History attribute values
CS Database
(3c)
CS writes history attribute values to the database as the next batch save cycle begins.
Scalar values
Memory
3

Template:

```
vdm:setalias
retrieve visitor scalar attribute values from add-on repository
vdm:setscalar
commercecontext:calculatesegments
commercecontext:getrecommendations
render Engage assets
vdm:recordhistory
render:overridedeps
```

Segment Rules

Rules can be based on any combination of the following:
- Scalar attribute values
- history attribute values and time duration – Located in database
- history attribute sums or counts

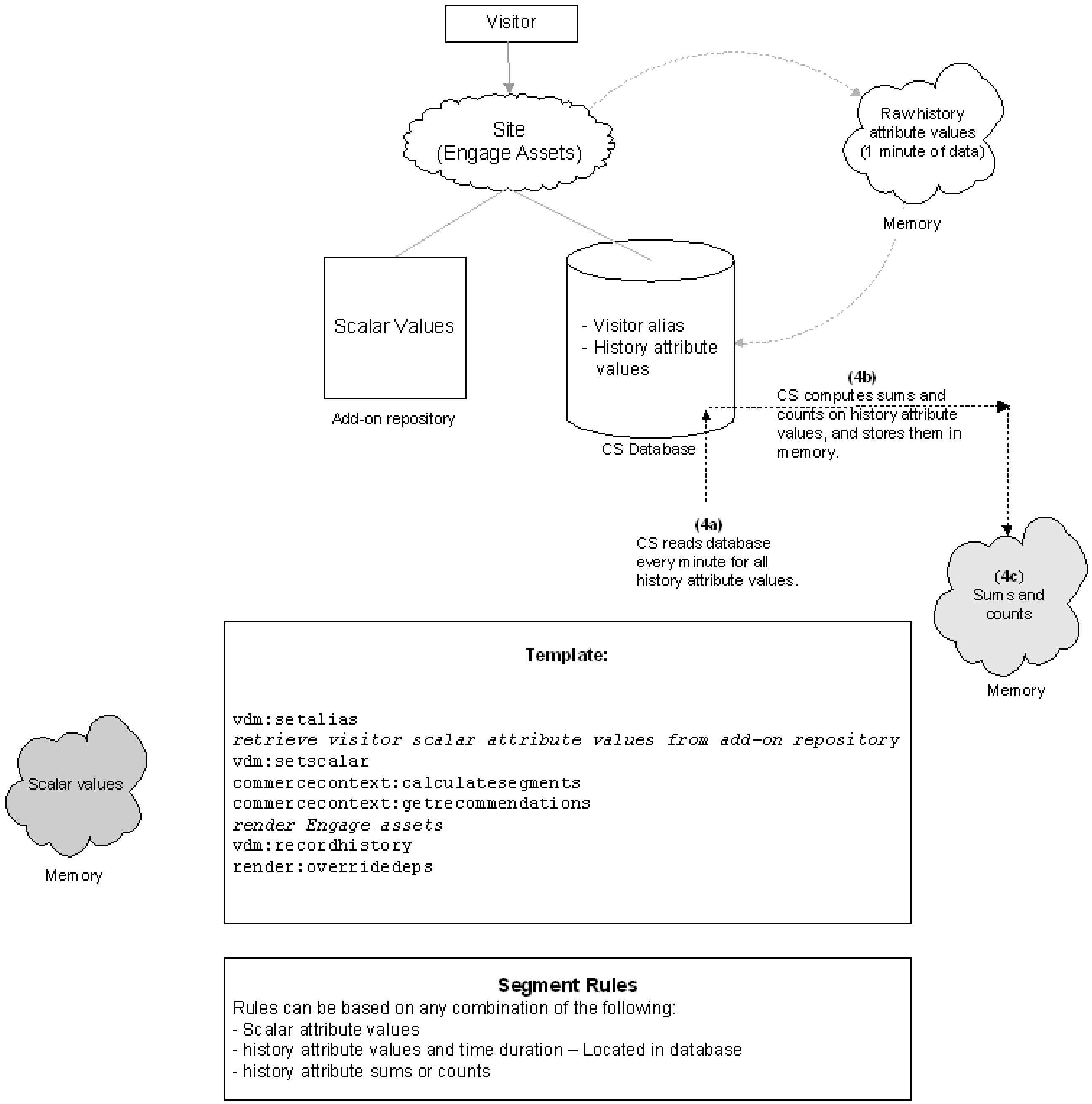
41.3.4 Computing Sums and Counts

After the first set of history attribute values is collected and written to the database, WebCenter Sites:

1. Reads the database at 1-minute intervals, computes the sums and counts, and Stores the results in memory.

Caching of sums and counts significantly reduces database queries, while the 1-minute interval makes the computation of sums and counts accurate to within 1 minute.

Figure 41–4 Computing Sums and Counts Flow


Visitor
Site (Engage Assets)
Raw history attribute values (1 minute of data)
Memory
Scalar Values
Add-on repository
- Visitor alias
- History attribute values
CS Database
(4b)
CS computes sums and counts on history attribute values, and stores them in memory.
(4a)
CS reads database every minute for all history attribute values.
(4c)
Sums and counts
Memory
Scalar values
Memory

Template:

```
vdm:setalias
retrieve visitor scalar attribute values from add-on repository
vdm:setscalar
commercecontext:calculatesegments
commercecontext:getrecommendations
render Engage assets
vdm:recordhistory
render:overridedeps
```

Segment Rules

Rules can be based on any combination of the following:
- Scalar attribute values
- history attribute values and time duration – Located in database
- history attribute sums or counts

41.3.5 Computing Segments

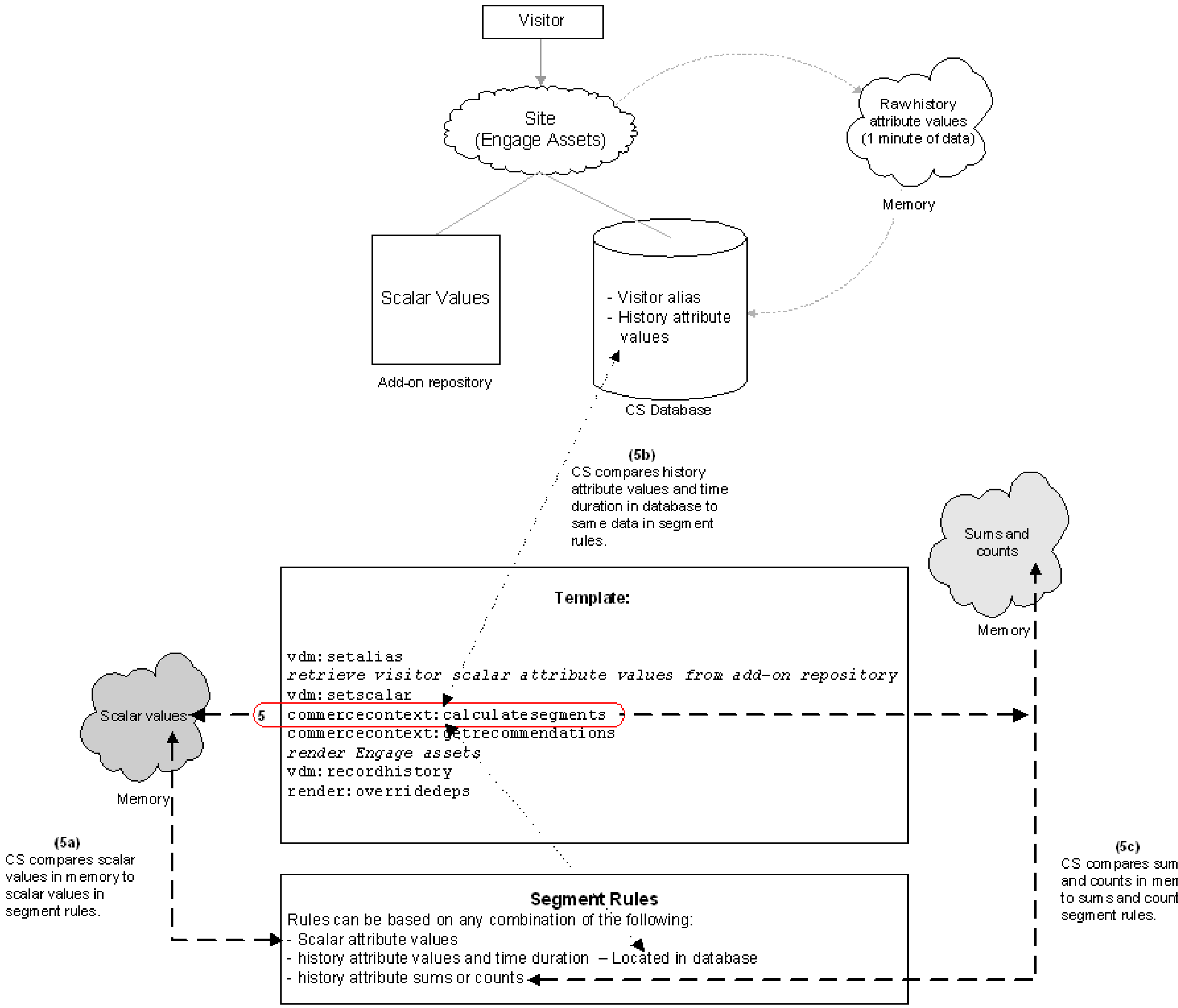
At this point, WebCenter Sites has enough information to compute the visitor's segments. Using the `commercecontext:calculatesegments` tag, WebCenter Sites does the following:

1. Compares the visitor's scalar values in memory to those in the segment rules.
2. Compares the history attribute values in the database to those in the segment rules.

3. Compares the sums and counts in memory to the sums or counts in the segment rules

From the comparisons, WebCenter Sites determines the visitor's segments.

Figure 41–5 Computing Segments Flow



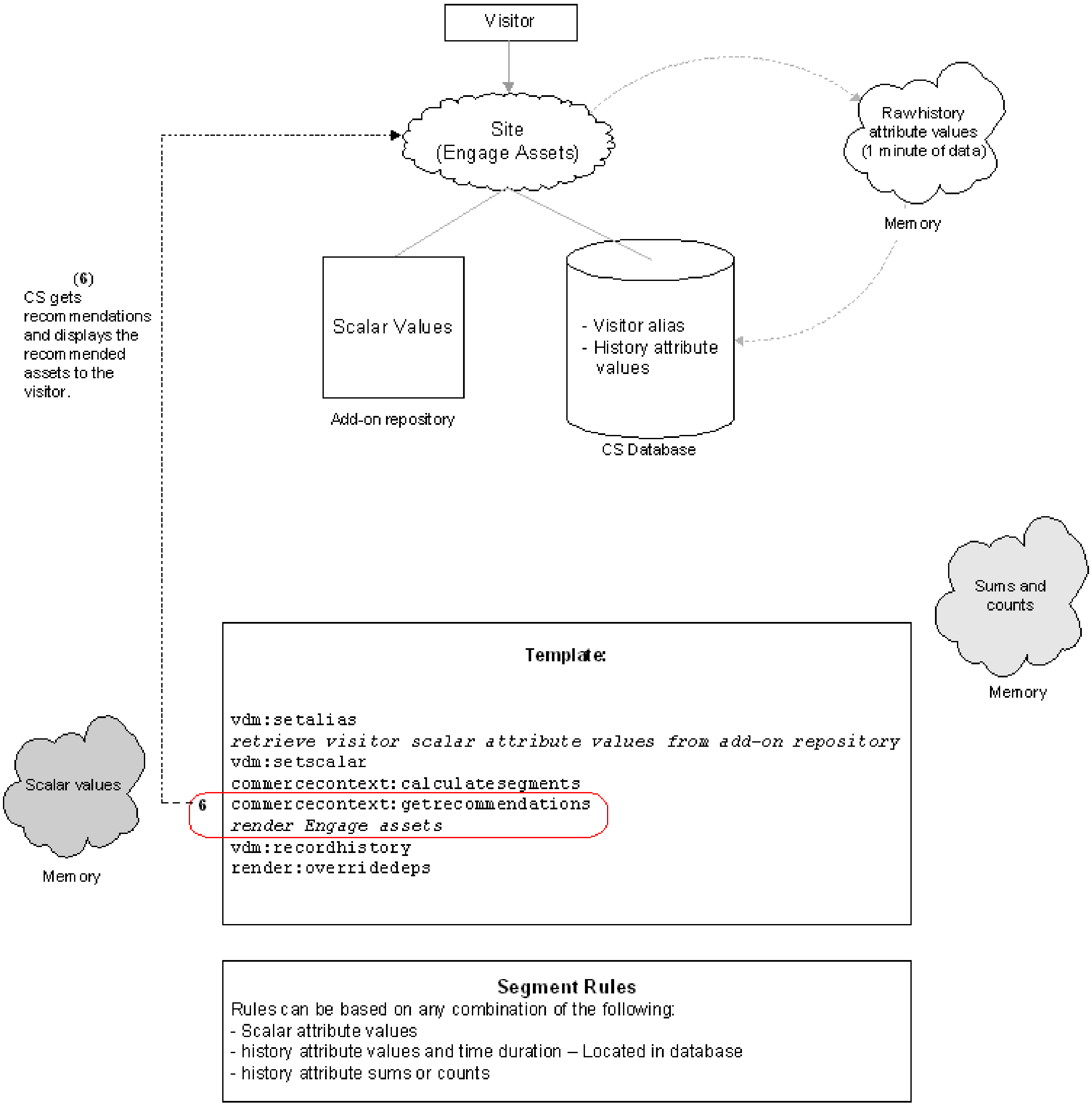
```
vdm:setalias
retrieve visitor scalar attribute values from add-on repository
vdm:setscalar
commercecontext:calculatesegments
commercecontext:getrecommendations
render Engage assets
vdm:recordhistory
render:overridedeps
```

41.3.6 Determining and Displaying Recommended Assets

Having determined the visitor's segments, WebCenter Sites uses the `commercecontext:getrecommendations` tag to determine which Engage assets are of most interest to the visitor. Templates for Engage assets display the recommended assets to the visitor.

Further performance gains are achieved by use of the `render:overridedeps` tag, as explained in Section 41.3.7, "Logging Dependencies."

Figure 41–6 Displaying Recommended Assets Flow



Template:

```
vdm:setalias
retrieve visitor scalar attribute values from add-on repository
vdm:setscalar
commercecontext:calculatesegments
commercecontext:getrecommendations
render Engage assets
vdm:recordhistory
render:overridedeps
```

Segment Rules

Rules can be based on any combination of the following:

- Scalar attribute values
- history attribute values and time duration – Located in database
- history attribute sums or counts

41.3.7 Logging Dependencies

The `render:overridedeps` tag is used to reduce the number of `SystemItemCache` entries logged. The tag must be inserted at the end of the template (or CSElement), just before the `</cs:ftcs>` tag, to remove all the existing dependencies logged on the page and to log the dependencies that are specified by the `render:overridedeps` tag. The possible dependencies are:

- Unknown dependencies
- Dependencies on a single specific asset
- Unknown dependencies for a specific asset type

The render:overridedeps tag takes the following parameters:

- cid: ID of the asset of type c. When combined with c, it works like asset, a dependency will be logged against the asset determined by c and cid.
- c: Type of asset.
- deptype: Dependency type.

Which parameters are used and how they are set determines the dependency that is logged:

- When deptype alone is specified and set to unknowndeps, render:overridedeps logs an unknown dependency.
- When c and cid are both specified, render:overridedeps logs a dependency on the asset specified by c and cid.
- When c and deptype='unknowndeps' are both specified, render:overridedeps logs an unknown dependency on the asset type specified by c.

Note: Use render:overridedeps carefully. The only way to flush page caches is to edit the asset for which dependencies are logged. For example, if you are logging dependencies for a single asset (or asset type), you can flush the cache by editing the asset for which dependencies are logged. Users must know which asset(s) to edit.

42

Asset API Tutorial

The purpose of this tutorial is to introduce the Asset API to the developer. It is meant to be a quick reference and not a substitute for the *Oracle Fusion Middleware WebCenter Sites Java API Reference*. Code samples throughout this tutorial illustrate how the Asset API is used.

This chapter contains the following sections:

42.1 Understanding the Asset API

The Asset API is a Java API used to access and manipulate WebCenter Sites assets. Its main purpose is to broaden the context in which assets can be handled. Its major features are the following:

- The Asset API supports WebCenter Sites in a non-servlet context.

  Prior to the Asset API, WebCenter Sites exposed primary interfaces for programming in the form of tags (both in XML and JSP), used as the means for creating web pages and applications. In this sense, WebCenter Sites was tightly built around the servlet model and was not usable in other contexts, such as standalone Java programs (EJB, for example). Therefore, there was a need to create an API that could be used anywhere, regardless of the servlet framework.

- The Asset API unifies the retrieval, creation, and modification of the two asset families: basic and flex.

  - The Asset API represents both asset families with `AssetData` and `AttributeData`.
  - The Asset API uses generic `Condition` and `Query` objects for both flex and basic assets.

- The Asset API supports the creation and editing of basic assets, flex assets, and flex parent assets in the form of Java objects.

Note: The Asset API does not log any dependencies other than the asset that is being loaded.

42.2 Primary Interfaces

The Asset API provides access to data as well as definitions for WebCenter Sites assets:

- Package `com.fatwire.assetapi.data` contains classes that are useful in reading data.
- Classes under `com.fatwire.assetapi.def` are for asset definitions.
- Package `com.fatwire.assetapi.query` contains constructs necessary for building a Query. For further details, see the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.

The Asset API defines the following primary interfaces:

- **Session**: The primary entry point into WebCenter Sites from the API. One has to obtain a session to be able to do anything at all in the Asset API.
- **AssetDataManager**: A manager for reading asset data. Developers can query for information here, as well as look up asset associations and other information.
- **AssetTypeDefManager**: A manager for reading an asset type's definition. 'Definition' is a loaded term in WebCenter Sites where flex assets are concerned. Here it is used in the generic sense, as something that defines the structure of an asset type. As a result, basic asset types also have a definition.
- **AssetData**: An asset's data; basically a collection of `AttributeData` instances and other information about the asset itself.
- **AssetId**: The asset type-id combination.
- **DimensionableAssetManager**: A manager that supports multilingual assets by retrieving translations of any given asset.

42.3 Getting Started

- FirstSiteII (FSII) is required to run the examples in this tutorial. The `tools.jar` file (available in JDK 1.5 and above) must be in the classpath.

Note: The Asset API can be used from a standalone Java program. Doing so requires some configuration. For details, see Section 42.7, "Optional: Setting Up to Use the Asset API from Standalone Java Programs."

- Working through the examples in this chapter requires a knowledge of `jsp` elements and how they are created in the WebCenter Sites environment. For information and procedures, see Chapter 23, "Creating Template, CSElement, and SiteEntry Assets."

42.4 Asset API Read

With this brief background, let's start writing some code based on FSII data.

This section contains the following topics:

42.4.1 A Simple Example: Reading Field Values

Let's try to read all the values of the **FSIIHeadline** field in FSII Articles. These are the steps to follow:

1. Get a session.
2. Get a handle to `AssetDataManager`.
3. Build a query.
4. Perform 'read' and print the results.

Here is the code that implements these steps (in a `jsp` element):

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr =(AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
Query q = new SimpleQuery("Content_C", "FSII Article", null,
Collections.singletonList("FSIIHeadline") );
for( AssetData data : mgr.read( q ) )
{
out.println( data.getAttributeData("FSIIHeadline").getData() );
out.println( "<br/>" );
}
%>
</cs:ftcs>
```

1. `SessionFactory.newSession()` builds a session for a given user. From that point on, all data that is read using this session instance is based on the user's ACL permissions. You could also simply call `newSession( null, null )` and get a Session that belongs to `DefaultReader`, an assumed user. CS-powered web applications generally run as this user at runtime. However, if the user name and password are specified and happen to be incorrect, you will get an exception.
2. Using the session, get a handle to `AssetDataManager.getManager`. (The `AssetDataManager.class.getName()` method does this.)

3. A Query represents results that are based on the user's search criteria. In this example, we are using a simple version of Query, where we specify the asset type (Content_C) subtype (FSII Article) and the list of attributes to be returned (just FSIIHeadline in this case). We want all assets; therefore the third parameter (which takes Condition instance) is null. We will see how to use Conditions later on in Section 42.4.5, "Complex Query."

4. The read() method of AssetDataManager returns an Iterable over AssetData. Each piece of asset data contains an instance of AttributeData against an attribute name. AttributeData.getData() returns the real data of the attribute itself.

42.4.2 Reading AssetId

What if we wanted to know the id's of all these assets? AssetData.getAssetId() returns an AssetId instance.

The code in Section 42.4.1, "A Simple Example: Reading Field Values" can be modified, as shown below, to print AssetId:

```
for( AssetData data : mgr.read( q ) )
{
AssetId id = data.getAssetId();
out.println( data.getAttributeData("FSIIHeadline").getData() + " id=" + id );
out.println( "<br/>" );
}
```

The code above prints the following lines (note that AssetId is a composite, it contains the id number as well as type. AssetId.getId() and AssetId.getType() return id and type separately:

```
AudioCo. America Announces H300 series id=Content_C:1114083739888
AudioCo. New Portable Media Player Offers Full Video Experience id=Content_
C:1114083739926
AudioCo.'s First Under Water MP3 Player id=Content_C:1114083739951
...
```

42.4.3 Reading Attributes Given the Asset ID

What if we already have an asset id, by some means? (Either passed into a template or acquired programmatically.) How can we read attributes of that? Let's consider an AssetId (Content_C:1114083739888, for example, as shown in the code used in Section 42.4.2, "Reading AssetId" and attempt to print the name, description, and FSIIBody. The following code does this:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<%@ page import="com.openmarket.xcelerate.asset.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr =(AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
AssetId id = new AssetIdImpl( "Content_C", 1114083739888L );
List attrNames = new ArrayList();
attrNames.add( "name" );
attrNames.add( "description" );
attrNames.add( "FSIIBody" );
```

```
AssetData data = mgr.readAttributes( id, attrNames );
AttributeData attrDataName = data.getAttributeData( "name" );
AttributeData attrDataDescr = data.getAttributeData( "description" );
AttributeData attrDataBody = data.getAttributeData("FSIIBody");

out.println( "name:" + attrDataName.getData() );
out.println( "<br/>" );
out.println( "description:" + attrDataDescr.getData() );
out.println( "<br/>" );
out.println( "FSII Body:" + attrDataBody.getData() );
out.println( "<br/>" );
%>
</cs:ftcs>
```

Here, we are indicating which attributes to read for a given `id`. As you can see, you can specify basic fields (such as name, description) as well as flex attributes; both are treated as attributes.

Alternatively, you can load all attributes for a given `id` by using `AssetDataManager.read( List<AssetId> ids)`. The following code demonstrates how this is done for a single `AssetId`:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<%@ page import="com.openmarket.xcelerate.asset.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr =(AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
AssetId id = new AssetIdImpl( "Content_C", 1114083739888L );

Iterable<AssetData> dataItr = mgr.read( Collections.singletonList( id ) );

for( AssetData data : dataItr )
{
for(AttributeData atrData : data.getAttributeData() )
{
out.println( "<br/>" );
out.println( "attribute name:" + atrData.getAttributeName() );
out.println( "data: " + atrData.getData() );
}
}
%>
</cs:ftcs>
```

42.4.4 Query

A `Query` specifies the criteria based on which data is looked up. What if we want to look up a product whose SKU is `iAC-008`? The code below does this and prints the name and `id`:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
```

```
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
Condition c = ConditionFactory.createCondition( "FSIISKU", OpTypeEnum.EQUALS,
"iAC-008" );
Query query = new SimpleQuery( "Product_C", "FSII Product", c,
Collections.singletonList( "name" ) );

for( AssetData data : mgr.read( query ) )
{
AttributeData attrData = data.getAttributeData( "name" );
out.println( "name:" + attrData.getData() );
out.println( "<br/>" );
out.println( "id:" + data.getAssetId() );
}
%>
</cs:ftcs>
```

Query consists of a Condition, set of Attributes to be returned, and a SortOrder. The example above uses a condition that is built on the FSIISKU attribute value being EQUAL to iAC-008. Just as we did in the previous example, we pass in the list of attribute names we want to be returned in the resulting collection of AssetData.

There are some considerations as to what types of attributes can be queried and how. See Section 42.6.2, "Query Types" for a complete discussion of different query algorithms. In short, some types of queries are possible with one algorithm, but not with the other. Note that this is exactly the behavior we have in the Asset family of tags and AssetSet family of tags. To illustrate this point, say we want to read all products whose price (FSIIPrice) is greater than 179.

FSIIPrice is of type MONEY. Consulting Table 42–2 in Section 42.6.3, "Data Types and Valid Query Operations," we see that the GREATER_THAN operation is allowed for this data type only in the basic/generic algorithm. The following code uses that algorithm to get all products whose price is greater than 179. The choice of query algorithm is made by the highlighted line in the code below:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>

<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
        AssetDataManager mgr = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
Condition c = ConditionFactory.createCondition( "FSIIPrice", OpTypeEnum.GREATER_
THAN, 179.0f );
Query query = new SimpleQuery( "Product_C", "FSII Product", c, Arrays.asList(
"name", "FSIIPrice" ) );
query.getProperties().setIsBasicSearch( true );

for( AssetData data : mgr.read( query ) )
{
AttributeData name = data.getAttributeData( "name" );
AttributeData price = data.getAttributeData( "FSIIPrice" );

out.println( "name:" + name.getData() );
out.println( "id:" + data.getAssetId() );
out.println( "price:" + price.getData() );

out.println( "<br/>" );
}
%>
</cs:ftcs>
```

42.4.5 Complex Query

A complex query can be achieved through nested `Conditions`. According to the choice of query algorithm, we are subjected to the constraints listed in Section 42.6.2, "Query Types."

The following code retrieves all the `Product_C` assets whose `FSIIPrice` attribute is greater than `179.0` and/or whose names are like "`FSII`":

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
Condition c1 = ConditionFactory.createCondition( "FSIIPrice", OpTypeEnum.GREATER_
THAN, 179.0f );
Condition c2 = ConditionFactory.createCondition( "name", OpTypeEnum.LIKE, "FSII"
);
Condition c = c1.and( c2 ); // c1.or( c2 );

Query query = new SimpleQuery( "Product_C", "FSII Product", c, Arrays.asList(
"name", "FSIIPrice" ) );
query.getProperties().setIsBasicSearch( true );

for( AssetData data : mgr.read( query ) )
{
AttributeData name = data.getAttributeData( "name" );
AttributeData price = data.getAttributeData( "FSIIPrice" );

out.println( "name:" + name.getData() );
out.println( "id:" + data.getAssetId() );
out.println( "price:" + price.getData() );

out.println( "<br/>" );
}
%>
</cs:ftcs>
```

42.4.6 Sorting

What if we wanted to retrieve the results sorted by price? The following code does that, by specifying a `SortOrder`, ascending in this example. To reverse the sort order, change `true` to `false` in the highlighted line of the code below:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
        AssetDataManager mgr = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );

SortOrder so = new SortOrder( "FSIIPrice", true );
Query query = new SimpleQuery( "Product_C", "FSII Product",
null, Collections.singletonList( "FSIIPrice" ), Collections.singletonList( so ) );

for( AssetData data : mgr.read( query ) )
{
AttributeData price = data.getAttributeData( "FSIIPrice" );

out.println( "id:" + data.getAssetId() );
out.println( "price:" + price.getData() );

out.println( "<br/>" );
}
%>
</cs:ftcs>

The above code sorts and prints asset ids and price in the ascending order of
FSIIPrice,

id:Product_C:1114083739851 price:89.99
id:Product_C:1114083739757 price:99.95
id:Product_C:1114083739696 price:129.99
id:Product_C:1114083739301 price:129.99
id:Product_C:1114083739471 price:179.95
id:Product_C:1114083739350 price:189.95
id:Product_C:1114083739225 price:399.99
id:Product_C:1114083739596 price:899.95
id:Product_C:1114083739804 price:1399.99
id:Product_C:1114083739549 price:3799.95
id:Product_C:1114083739663 price:6999.99
```

42.4.7 Reading BlobObject

The Asset API defines a special class to represent file type of data, data that is stored as a binary file. For example, the `FSIIDocumentFile` attribute in FirstSiteII is of type blob. The following code reads `FSIIDocumentFile` from all `Document_C` instances and prints the asset `id`, file name, and file size.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );

Query query = new SimpleQuery( "Document_C", "FSII Document", null,
Collections.singletonList( "FSIIDocumentFile" ) );

for( AssetData data : mgr.read( query ) )
```

```
{
AttributeData docAttr = data.getAttributeData("FSIIDocumentFile");
BlobObject fileObj = (BlobObject)docAttr.getData();
byte [] d = new byte[fileObj.getBinaryStream().available()];
fileObj.getBinaryStream().read(d);

out.println( "id:" + data.getAssetId() );
out.println( "file name:" + fileObj.getFilename() );
out.println( "file size:" + d.length );

out.println( "<br/>" );
}
%>
</cs:ftcs>
```

42.4.8 Retrieving Multi-Valued Attributes

The Asset API supports multi-valued attributes in the same way it supports single-valued attributes. `AttributeData` contains a companion method, `getDataAsList()` to retrieve multiple values. The following code attempts to print data contained in `FSIIKeyword`, a multi-valued attribute.

Note: Because sample data that ships with FirstSiteII does not have data for the `FSIIKeyword` attribute, this sample code does not print any keywords. Before running this code, edit some of the `FSIIDocument` instances to add data for this attribute.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );

Query query = new SimpleQuery( "Document_C", "FSII Document",
null, Collections.singletonList( "FSIIKeyword" ) );

for( AssetData data : mgr.read( query ) )
{
AttributeData attrData = data.getAttributeData( "FSIIKeyword" );
List retData = attrData.getDataAsList();
out.println( "id:" + data.getAssetId() );

for( Object o : retData )
{
out.println( "data:" + o );
}

out.println( "<br/>" );
}
%>
</cs:ftcs>
```

42.4.9 Multilingual Assets: Retrieving Translations

The Asset API also provides interfaces and methods to deal with multilingual assets. Basically, you have methods in DimensionableAssetManager to handle multilingual assets. They deal with getting all the locales for a given asset and specific translation of an asset.

The following example first retrieves the translations for asset Page: 1118867611403 by using the getRelatives method in DimensionableAssetManager with group set to Locale, and then it uses the getRelative method to get the fr_FR translation of the asset.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.mda.DimensionableAssetManager"%>
<%@ page import="com.openmarket.xcelerate.asset.*"%>
<%@ page import="java.util.*"%>

<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
DimensionableAssetManager mgr =
(DimensionableAssetManager)ses.getManager(DimensionableAssetManager.class.getName(
));

AssetId page_asset = new AssetIdImpl("Page", 1118867611403L);

for( AssetId id : mgr.getRelatives( page_asset, null, "Locale" ))
{
out.println( id );
}
out.println( "<br/>" );

AssetId fr_translation = mgr.getRelative( page_asset, "fr_FR" );
out.println( fr_translation );

%>
</cs:ftcs>
```

42.4.10 Reading Asset and Attribute Definitions

In addition to asset data, APIs also provide access to their definitions. Information such as the attributes that make up an asset definition, type of each attribute, etc. can be obtained through a manager called AssetTypeDefManager. The following example attempts to read all definition information from Document_C and print it to the browser.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.def.*"%>
<%@ page import="java.util.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
        AssetTypeDefManager mgr = (AssetTypeDefManager) ses.getManager(
AssetTypeDefManager.class.getName() );

AssetTypeDef defMgr = mgr.findByName( "Document_C", "FSII Document" );

out.println( "Asset type description: " + defMgr.getDescription() );
out.println( "<br/>" );

for( AttributeDef attrDef : defMgr.getAttributeDefs() )
{
out.println( "Attribute name: " + attrDef.getName() );
out.println( "Attribute description: " + attrDef.getDescription() );
out.println( "is required: " + attrDef.isDataMandatory() );
out.println( "Attribute type: " + attrDef.getType() );
out.println( "<br/>" );
}

%>
</cs:ftcs>
```

42.4.11 Reading Key-Value Mappings

The Asset API provides access to a given CSElement or Template's key-value mapping pairs. The following example reads all mapping pairs from a CSElement:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr =
(AssetDataManager)(ses.getManager(AssetDataManager.class.getName()));
Condition c = ConditionFactory.createCondition("name", OpTypeEnum.LIKE,
"FSIICommon/Nav/LocaleForm");
Query query = new SimpleQuery("CSElement", null, c, null);
query.getProperties().setReadAll(true);
for (AssetData data : mgr.read(query))
{
   List<AttributeData>  mappingArray =
(List<AttributeData>)data.getAttributeData("Mapping"). getData();
   for (int i=0; i<mappingArray.size(); i++)
   {
      HashMap mappingMap = (HashMap)mappingArray.get(i).getData();
      String key = (String)((AttributeData)mappingMap.get("key")).getData();
   String type =(String)((AttributeData)mappingMap.get("type")).getData();
      String value = (String)((AttributeData)mappingMap.get("value")).getData();
    String siteid = (String)((AttributeData)mappingMap.get("siteid")).getData();

      out.println("Mapping Entry #"+String.valueOf(i+1));
      out.println("<br/>");
      out.println("Key: "+key);
      out.println("Type: "+type);
   out.println("Value: "+value);

      out.println("Siteid: "+siteid);
      out.println("<br/>");
   }
}
%>
</cs:ftcs>
```

42.5 Asset API Write

The Asset API supports write operations on basic assets and selected types of flex assets. The supported write operations are asset creation, modification, and deletion. The supported assets are basic assets, flex assets, and flex parents. All other asset types are not currently supported: Flex Parent Definition, Flex Asset Definition, Flex Filter, and Flex Attribute. An `UnsupportedOperationException` will be thrown if any write operation (insert or update) is attempted on those asset types.

This section contains the following topics:

- Section 42.5.1, "Creating New Assets"
- Section 42.5.2, "Updating Existing Assets"
- Section 42.5.3, "Deleting Existing Assets"
- Section 42.5.4, "Multilingual Assets"

42.5.1 Creating New Assets

Asset API uses the `AssetDataManager.insert( List<AssetData> data )` method to create a new asset in WebCenter Sites. The method takes in a list of `AssetData` and uses these `AssetData` to create new assets in WebCenter Sites. If successful, the method will populate the passed in `AssetData` with the IDs of the newly created assets.

Creating a New Flex Asset

Asset API is able to create a new flex asset by combining the `AssetDataManager.newAssetData` method and `insert` method. The `newAssetData` method will return an empty `AssetData` with all the `AttributeData` objects populated with `null` or empty List/Collection. The `flextemplateid` (for flex assets) or `flexgroupid` (for flex parents) will be automatically populated if the correct subtype is specified.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="com.openmarket.xcelerate.asset.AssetIdImpl"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager adm = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
        MutableAssetData d = adm.newAssetData( "Content_C", "FSII Article" );
        d.getAttributeData( "name" ).setData( New Content" );
        d.getAttributeData( "FSIIHeadline" ).setData( "headline" );
        d.getAttributeData( "FSIIAbstract" ).setData( "abstract" );
        d.getAttributeData( "FSIIBody" ).setData( "body" );
        d.getAttributeData( "Publist" ).setData( Arrays.asList( "FirstSiteII" ) );
        d.setParents( Arrays.<AssetId>asList( new AssetIdImpl( "Content_P",
1112192431478L)) );
        adm.insert( Arrays.<AssetData>asList( d ));
        out.println( d.getAssetId() );
%>
</cs:ftcs>
```

Create a New Basic Asset

Asset API is also capable of creating new basic assets.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager adm = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
        MutableAssetData d = adm.newAssetData( "HelloArticle", "" );
        d.getAttributeData( "name" ).setData( New Hello Article" );
        d.getAttributeData( "headline" ).setData( "headline" );
        d.getAttributeData( "byline" ).setData( "abstract" );
        d.getAttributeData( "category" ).setData( "g" );
        BlobObject b = new BlobObjectImpl( "filename.txt", null, "body".getBytes()
);
        d.getAttributeData( "urlbody" ).setData( b );
        d.getAttributeData( "Publist" ).setData( Arrays.asList( "HelloAssetWorld"
) );
        adm.insert( Arrays.<AssetData>asList( d ));
        out.println( d.getAssetId() );
%>
</cs:ftcs>
```

42.5.2 Updating Existing Assets

Another operation that Asset API supports, in addition to insert, is update. Similar to insert, update will save the data from AssetData into WebCenter Sites, but to an existing asset. If the asset does not exist, update will throw an exception.

```
<%@ page import="com.openmarket.xcelerate.asset.AssetIdImpl"%>
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager adm = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
Iterable<AssetData> assets = adm.read( Arrays.<AssetId>asList( new AssetIdImpl(
"HelloArticle", 1238171255471L), new AssetIdImpl( "Content_C", 1238171254486L)) );
        List<AssetData> sAssets = new ArrayList<AssetData>();
        for ( AssetData a : assets )
        {
            sAssets.add( a );
            a.getAttributeData( "name" ).setData( "Changed Name" );
        }
        adm.update( sAssets );
%>
</cs:ftcs>
```

42.5.3 Deleting Existing Assets

AssetAPI also supports deletion of assets from WebCenter Sites. In the following example, after removing all the references to the two assets, adm.delete will be able to delete both assets from WebCenter Sites. An exception will be thrown if an asset is referenced by other assets, or if the asset is invalid. The delete process will stop when an exception is thrown.

```
<%@ page import="com.openmarket.xcelerate.asset.AssetIdImpl"%>
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
 Session ses = SessionFactory.getSession();
 AssetDataManager adm = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
 adm.delete( Arrays.<AssetId>asList( new AssetIdImpl( "HelloArticle",
1238171255471L), new AssetIdImpl( "Content_C", 1238171254486L)) );
%>
</cs:ftcs>
```

42.5.4 Multilingual Assets

Asset API supports the creation of multilingual assets. Creation of the assets requires a two-step process. First, the asset is created and saved. Next, locale information is added. The following example creates a new `Content_C asset` and sets the locale to `en_US`.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.mda.*"%>
<%@ page import="com.fatwire.mda.DimensionableAssetInstance.
                DimensionParentRelationship"%>
<%@ page import="java.util.*"%>
<%@ page import="com.openmarket.xcelerate.asset.*"%>
<%@ page import="com.openmarket.xcelerate.common.*"%>
<cs:ftcs>

<%
Session ses = SessionFactory.getSession();

AssetDataManager adm = (AssetDataManager) ses.getManager(
AssetDataManager.class.getName() );
MutableAssetData d = adm.newAssetData( "Content_C", "FSII Article" );
d.getAttributeData( "name" ).setData( New Content" );
d.getAttributeData( "FSIIHeadline" ).setData( "headline" );
d.getAttributeData( "FSIIAbstract" ).setData( "abstract" );
d.getAttributeData( "FSIIBody" ).setData( "body" );
d.getAttributeData( "Publist" ).setData( Arrays.asList( "FirstSiteII" ) );
d.setParents( Arrays.<AssetId>asList( new AssetIdImpl( "Content_P",
1112192431478L)) );
adm.insert( Arrays.<AssetData>asList( d ));
DimensionManager dam =
(DimensionManager)ses.getManager(DimensionManager.class.getName());
Dimension dim = dam.loadDimension("en_US");
d.getAttributeData("Dimension") .setData(Arrays.asList(new Dimension[]{dim}));
DimensionParentRelationship dpr = new DimParentRelationshipImpl("Locale",
d.getAssetId());
d.getAttributeData("Dimension-parent").setData(Arrays.asList(new
DimensionParentRelationship[]{dpr}));
adm.update( Arrays.<AssetData>asList( d ));
%>
```

```
</cs:ftcs>
```

42.6 Development Strategies

This section discusses data types and attribute data (maps WebCenter Sites data types to Java types) as well as query types (compares types of queries, their usage, and supported operations).

This section contains the following topics:

- Section 42.6.1, "Data Types and Attribute Data"
- Section 42.6.2, "Query Types"
- Section 42.6.3, "Data Types and Valid Query Operations"

42.6.1 Data Types and Attribute Data

`AttributeData` contains information about the data type (WebCenter Sites specific type) and the actual data. The types are defined by `AttributeTypeEnum`. `AttributeData.getData()` and `AttributeData.getDataAsList()` return data objects of a specific Java type. Here is a map of WebCenter Sites types and their corresponding Java types.

Table 42–1 WebCenter Sites Data Types and Java Types

WebCenter Sites Data Type	Java Type
INT	Integer
FLOAT	Double
STRING	String
DATE	Date
MONEY	Double
LONG	Long
LARGE_TEXT	String
ASSET	AssetId
BLOB	BlobObject

42.6.2 Query Types

Using the Asset API, you can perform two kinds of queries: generic/basic and flex.

There are two different algorithms, one using the generic asset infrastructure (generic/basic query), and the other using AssetSets and Search States (flex query). Note that it is possible to use the generic/basic query for flex assets as well as basic assets; flex query, however, works only for flex assets.

Each of these algorithms has its advantages and disadvantages. The Asset API seeks to unify the querying mechanism and eventually let the API user not be concerned about the choice of algorithm. However, at the present time as there is no equivalence between these algorithms, the user needs to specify if she wants to use a specific feature, offered by one of the two.

QueryProperties.setIsBasicSearch(true) sets the query algorithm to generic/basic search for this query. It is set to false by default. For basic assets, the setting does not matter.

Which Type of Query to Choose? Very simply put, if you want to look for basic attributes of a flex asset, use the basic. Otherwise use the default. This will work for most queries one generally encounters. Things are a bit more subtle than that. Given below are other considerations against each type of query.

Basic/Generic Query

- Cannot search on flex attributes if you do not specify subtype.
- Cannot search on a flex attribute that is not in the flex definition.
- Cannot sort on a flex attribute.
- Case sensitivity is not guaranteed (depends on the database).
- Only the AND operation is allowed between different fields (name=name1 AND description=descr1 is allowed, but name=name1 AND name=name2 is not).
- Only OR is allowed for two conditions involving the same field name. The OR condition does not work on flex attributes.

Flex Query

- Cannot have basic attributes in the condition (id, name, description, etc.).
- Cannot sort by basic attributes.
- Flex query works without a subtype being specified. The search applies to data of all subtypes.
- Can use only the following operands in the condition; LIKE, EQUALS, BETWEEN, and RICHTEXT.

42.6.3 Data Types and Valid Query Operations

Depending on the type of query being performed, there are further restrictions on what type of operation is allowed for a given data type.

In general, a flex type query (which is the default for flex assets) allows only the following OpTypeEnums; LIKE, EQUALS, BETWEEN, and RICHTEXT. Note that these are the same operations available from AssetSet/SearchState tags.

If you want to use other OpTypeEnums, you have to use basic/generic query (by setting QueryProperties.setIsBasicSearch(true)). Such a query, of course, has to adhere to the basic query rules above.

Here is the allowed set of operations per data type (single-valued or multi-valued) for a basic/generic query.

Table 42–2 Allowed Set of Operations

Data Type	EQUALS	NOT_ EQUALS	LIKE	GREATE R	LESSTHA N	BETWEE N	RICHTEX T
INT	Y	Y	N	Y	Y	–	N
FLOAT	Y	Y	N	Y	Y	–	N
STRING	Y	Y	Y	Y	Y	–	N
DATE	Y	Y	N	Y	Y	–	N

Table 42–2 (Cont.) Allowed Set of Operations

Data Type	EQUALS	NOT_ EQUALS	LIKE	GREATER	LESSTHAN	BETWEEN	RICHTEXT
MONEY	Y	Y	N	Y	Y	–	N
LONG	Y	Y	N	Y	Y	–	N
LARGE_ TEXT	N	N	Y	N	N	–	N
ASSET	Y	Y	N	N	N	–	N
BLOB	N	N	N	N	N	–	N

And here is the allowed set of operations per data type (single-valued or multi-valued) for the flex type query.

Table 42–3 Allowed Set of Operations per Data Type

Data Type	EQUALS	NOT_ EQUALS	LIKE	GREATER	LESS THAN	BETWEEN	RICHTEXT
INT	Y	–	N	–	–	Y	N
FLOAT	Y	–	N	–	–	Y	N
STRING	Y	–	Y	–	–	Y	N
DATE	Y	–	N	–	–	Y	N
MONEY	Y	–	N	–	–	Y	N
LONG	Y	–	N	–	–	Y	N
LARGE_ TEXT	N	–	Y	–	–	N	Y
ASSET	Y	–	N	–	–	Y	N
BLOB	N	–	N	–	–	N	Y

42.7 Optional: Setting Up to Use the Asset API from Standalone Java Programs

Although this tutorial shows usage of the Asset API from JSP templates, it is possible to use the API from a standalone Java program, as well. In order to do this, you can set up a single database connection or a connection pool (outside WebCenter Sites).

> **Note:** In the steps below, we assume that all components are local to your WebCenter Sites installation.

Before setting up a single database connection or pool

1. Make sure the following files are in the classpath of your Java program:
   - `javaee.jar` and `tools.jar` (both are available in JDK 1.5 and higher versions).
   - `ServletRequest.properties`. This file can be copied from the `WEB-INF/classes` folder.

- wem_sso.xml, given that your WebCenter Sites installation has WEM installed and single sign-on (in futuretense.ini) is set to true. The wem_sso.xml file can be copied from the WEB-INF/classes folder.
- All of WebCenter Sites's binary files (jar files in the WEB-INF/lib folder).

2. Continue with the steps in one of the following sections:
   - To set up a single database connection
   - To set up a database connection pool

To set up a single database connection

1. Set the following system properties for your Java program:

```
cs.dburl=<JDBC_URL_to_connect_to_DB>
cs.dbdriver=<driverClass>
cs.dbuid=<dbUserName>
cs.dbpwd=<dbPassword>
```

2. Locate the WebCenter Sites installation folder and pass its name as a JVM argument:

```
-Dcs.installDir=<install_dir>
```

To set up a database connection pool

1. Add the following jar files to the classpath: commons-dbcp.jar (which ships with WebCenter Sites) and commons-pool.jar (available from the Apache website).
2. Create the property file with the same name as your data source:

   Note the following:

   - The name of the data source is the value of the cs.dsn property (in futuretense.ini).
   - The value of cs.dbconnpicture (also in futuretense.ini) must refer to cs.dsn (the combination of cs.dbconnpicture and cs.dsn must yield a valid resource).

     For example, the combination of

     cs.dsn= csDataSource *and*

     cs.dbconnpicture= java\:comp/env/$dsn

     yields a valid resource:

     java:/csDataSource

     Therefore, you would name the property file csDataSource.properties (the value of cs.dsn).
   - When creating the property file, make sure to place it in the classpath of your Java program.
3. When the property file is created, add the following keys to the file:

```
driver=<driverClass>
url=<JDBC_URL_to_connect_to_DB>
maxconnections=<number_of_connections_to_pool>
user=<dbUserName>
password=<dbPassword>
```

4. Locate the WebCenter Sites installation folder and pass its name as a JVM argument:

   ```
   -Dcs.installDir=<install_dir>
   ```

43

Public Site Search

WebCenter Sites includes a new framework for managing search indices. This framework forms the basis for searches in both the editorial interface and on the live site. That is, the visitors' side. Hence the name Public Site Search.

This chapter introduces the search framework and discusses the usage of the API in building public site searches.

This chapter contains the following sections:

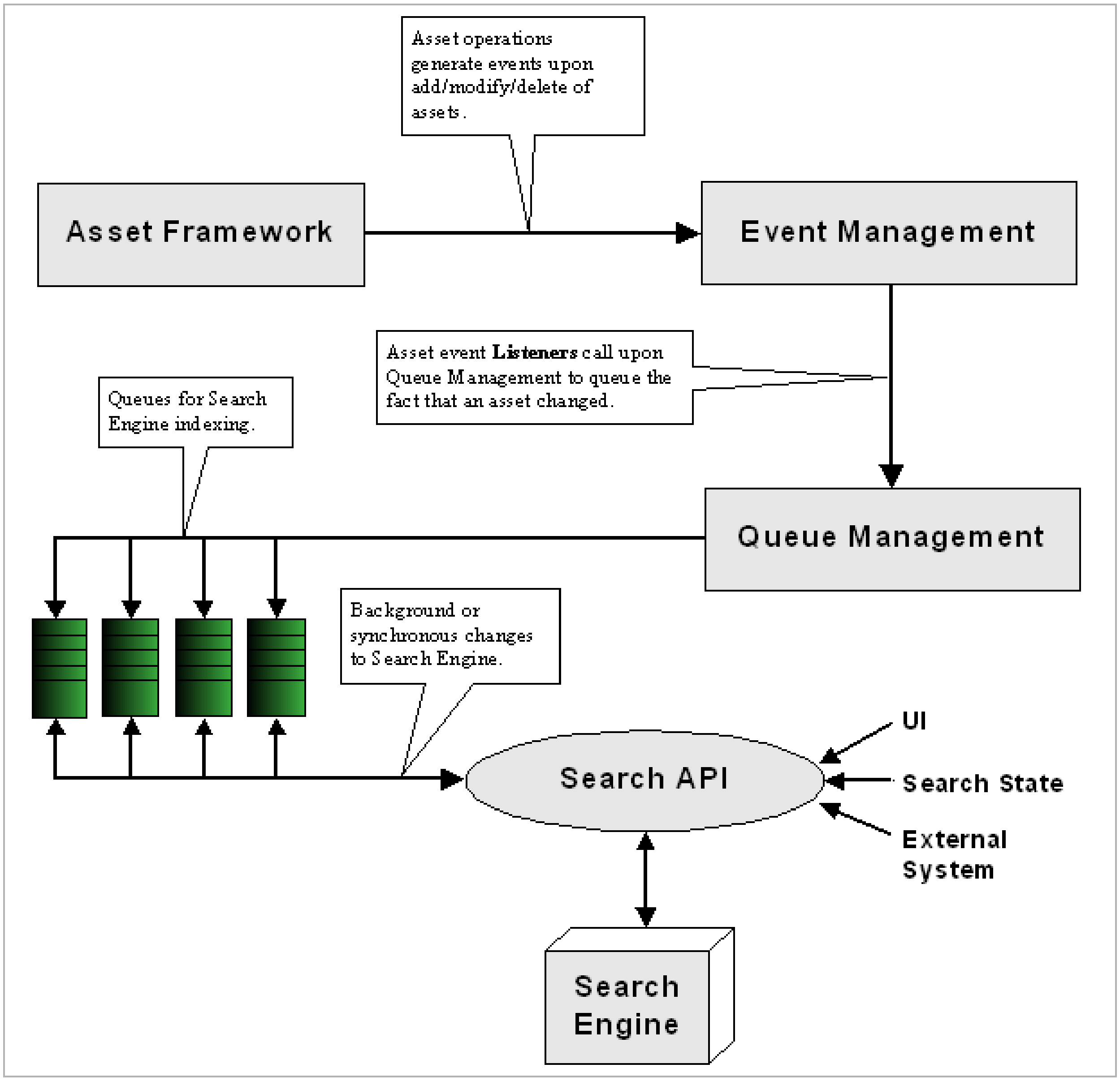
- Section 43.1, "Overview of the Search Framework"
- Section 43.2, "Index Types"
- Section 43.3, "Search API Overview"
- Section 43.4, "Advanced Configuration"

43.1 Overview of the Search Framework

> **Note:** You can skip this section if you are primarily interested in the usage of the search API.

Figure 43–1 shows how the search engine integration framework works with the rest of WebCenter Sites.

Figure 43–1 Search Engine Integration


Asset operations generate events upon add/modify/delete of assets.
Asset Framework
Event Management
Asset event **Listeners** call upon Queue Management to queue the fact that an asset changed.
Queues for Search Engine indexing.
Queue Management
Background or synchronous changes to Search Engine.
UI
Search API
Search State
External System
Search Engine

The search framework consists of the Search API, special asset event listeners, and a polling system for queues. This framework is used in coordination with the Event Management and Queue Management frameworks. This document focuses primarily on the search framework.

1. Asset framework detects changes/additions to assets and fires off events.
2. Registered listeners queue the changes, using a persistent queue implementation. A given event can be queued into one or many persistent queues. Each queue can be thought of as the source of data for a search index.
3. Once asset events are queued, a background process empties the queue contents and routes them to the Search API.
4. The Search API chooses the appropriate (configurable) search engine vendor implementation to start the indexing process.

43.2 Index Types

Two types of indices are created in WebCenter Sites: Global index and AssetType index. Global index is the index of all data (all asset types enabled for Global index). If you want to search for a phrase or expression in multiple asset types (such as attempting to build a Google-like search interface), Global index is more appropriate.

While Global index contains data for all fields of the index, it does not store the data in a form that is suitable for parametric searches. An AssetType index contains indexed information for a given asset type in a manner that can be searched parametrically. The WebCenter Sites Admin interface supports the configuration of Asset Type searches, which includes attribute-based searches for the indexing-enabled asset types. More information about the search configuration options is available in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

This section contains the following topics:

- Section 43.2.1, "Global Index"
- Section 43.2.2, "Asset Type Index"

43.2.1 Global Index

Global index is used by the Contributor interface to build a global search UI. An instance of the index also exists on the delivery server. The index on the delivery server can be used to build public site searches.

Note that only those assets that are published to the live site *after search is configured* are available for searches. It is during publishing that the data gets indexed. All assets that may exist on the live site before search is configured will not be reflected in the Global index (until the assets are re-indexed on the live site).

A search index functions roughly similar to a database table. Global index consists of the following fields:

Note: The field names are case-sensitive.

Table 43–1 Fields in the Global Index

Field name	Description
defaultSearchField	This field contains all the data of the indexed asset. This is the field you would search for in full-text searching. The defaultSearchField contains index data for all attributes of the asset, including any binary field data. Data for all the attributes is merged into this single field and indexed. The index itself does not 'store' data for this field, but does allow full-text searching.
id	This field contains the asset id.
AssetType	This field contains the asset type (Content_C/Product_P).
locale	This field contains the locale string (for example, en_US).
name	This field contains the name of the asset.
description	Description associated with the asset.
subtype	Name of the subtype (flex definition name).
subtypeid	ID of the subtype (flex def id).

Table 43–1 (Cont.) Fields in the Global Index

Field name	Description
updateddate	Last updated date as found at the time of indexing.
siteid	IDs of all sites in which this asset is available.
startdate	Start date field in the asset table.
enddate	End date field in the asset table.

43.2.2 Asset Type Index

An asset type index is created when it is enabled from the Admin interface by selecting **Admin tab**, then **Search**, and then **Configure Asset Type Search**. Once an asset type is enabled, an index will be created under `/shared/lucene/<Asset type name>`. This index contains all attributes of the given type as fields in the index. For example, the index for `Content_C` contains the following fields:

Note: The field names are case-sensitive.

Table 43–2 Fields in the Asset Type Index

Field Name	Description
DefaultSearchField	This field contains all the data of the indexed asset. This is the field you would search for in full text searching. The `DefaultSearchField` contains index data for all attributes of the asset, including any binary field data. Data for all the attributes is merged into this single field and indexed. The index itself does not 'store' data for this field, but does allow full text searching.
id	This contains the asset id
AssetType	This contains the asset type (Content_C/Product_P)
locale	Contains the locale string (example en_US)
name	Name of the asset
description	Description associated with the asset
subtype	Name of the subtype (flex definition name)
subtypeid	Id of the subtype (flex def id)
updateddate	Last updated date as found at the time of indexing.
siteid	All site ids this asset is available in
startdate	Startdate field in the asset table
enddate	Enddate field in the asset table
Dimension	ID of the dimension
Dimension-parent	ID of the Dimension parent
createdby	User name that created this asset
createddate	Date the asset was created
Publist	List of site names this asset belongs to
Relationships	Asset IDs of related items (flex only)

Table 43–2 (Cont.) Fields in the Asset Type Index

Field Name	Description
`externaldoctype`	Not used
`filename`	File name used for static publishing
`flextemplateid`	ID of the flex definition (flex only)
`fw_uid`	Globally unique id of this asset
`path`	Path used for static publishing
`renderid`	Object ID of the Template asset assigned to a flex asset.
`ruleset`	XML document of the ruleset
`status`	Status associated with the asset
`subtype`	Subtype name
`subtypeid`	Subtype id (flex only)
`template`	Template name
`updatedby`	User name that last updated this asset
`updateddate`	Date of last update
`urlexternaldoc`	Not used
`urlexternaldocxml`	Not used
`FSIIAbstract`	Flex attribute
`FSIIBody`	Flex attribute
`FSIIByline`	Flex attribute
`FSIIDescriptionAttr`	Flex attribute
`FSIIHeadline`	Flex attribute
`FSIINameAttr`	Flex attribute
`FSIIPostDate`	Flex attribute
`FSIISubheadline`	Flex attribute
`FSIITemplateAttr`	Flex attribute

To visualize which fields are available in a given index, use the tool Luke, available at:

`http://www.getopt.org/luke/`

Once you launch the tool, use the tool's browse function to load the index by simply locating the folder that contains the index (for example: `../shared/lucene/Content_C`).

43.3 Search API Overview

This section contains the following topics:

- Section 43.3.1, "SearchEngine"
- Section 43.3.2, "QueryExpression"
- Section 43.3.3, "Configuration"

43.3.1 SearchEngine

The SearchEngine interface defines the key functions of a search engine implementation; indexing and searching. More information about SearchEngine is available in the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.

- The source of index data is given to SearchEngine. SearchEngine, in response to the indexing request, creates the search index, if it is not already created and updates the contents based on IndexSource's accessors.
- index() works off of a given IndexSource instance. Depending on the search engine's implementation details, it operates on new, modified and deleted data coming from the IndexSource (in most search engines, all that is modified must be deleted first and then re-indexed).

  index() also invokes index lifecycle methods (startIndexing() and endIndexing()) on the given instance of IndexSource.
- search() operates on a QueryExpression against one or many indexes (IndexSources), resulting in a single set of results, sorted by their relevance (or SortOrder, if specified and usable across indices.
- A configuration lookup interface (IndexSourceConfig) is supplied to the SearchEngine instance which it can look up IndexSource properties, if needed.
- A QueryConverter interface is supplied to SearchEngine. This interface converts a given QueryExpression to its native form (recognizable by the specific search engine). This makes it possible to control the query language that the search engine uses externally.
- SearchResult is an abstraction over what is returned from the search engine. SearchResult is an iterator over ResultRow, a sub class of IndexRow, that contains relevance information. The getRelavence() method returns a double; the higher the value, the higher is the relevance of this ResultRow for the given query.

43.3.2 QueryExpression

The QueryExpression interface is a generic interface for defining search criteria. All search engines support native formats for building queries. The native form contains definitions of wildcards, relevance hints, etc. These tend to be very specific to each search engine.

Search engines also provide a basic query construct, which can be programmatically built (AND & OR over field matches). These can be thought of in terms of generalized programmable interfaces, although limited in power.

QueryExpression encapsulates four distinct characteristics of search engine queries:

- Native text search format: Most search engines support a very sophisticated native format for search, including wild cards, special hints etc. This is available through getStringFormat().
- Conjunction and disjunction: ANDs and ORs of conditions using and() and or() methods.
- Pagination: Using getStartIndex() and getMaxResults() methods.
- Sorting: using getSortOrder().

43.3.3 Configuration

1. Make sure that search indexing is enabled:

In the `SystemEvents` table, verify that the `SearchIndexEvent` is enabled (enabled field =1). This is configured to run in the background constantly (`*:*:* */*/*`); in practice it runs about every 30 seconds.

2. Make sure Asset listener is registered:

   Assets are queued for processing by the search framework, using asset events. The asset events are registered in the `AssetListener_reg` table. Make sure the following entry is found in this table (if it does not exist, add it).

Table 43–3 AssetListener_reg Table

ID	Listener	Blocking
1153937286234	`com.openmarket.basic.event.SearchAssetIdEventListener`	Y

`IndexSourceMetaDataConfig`: table that stores configuration information for `IndexSource`.

This describes the structure and nature of the index itself. This should have a row for `Global` by default. Any asset type enabled for Asset type index will have an additional row in this table.

`SearchEngineMetaDataConfig`: stores the search engine configuration. This table should have a row for Lucene, by default.

These are configured correctly by the installer and managed by the Admin interface.

Defaults here should suffice; the precise definition and meaning of "configuration" will be covered in some depth at a later point in this tutorial (see Section 43.4, "Advanced Configuration").

43.4 Advanced Configuration

The following sections explain aspects of advanced configuration that may be necessary in certain circumstances. However, in most cases, the defaults should be sufficient.

This section contains the following topics:

- Section 43.4.1, "Configuring Lucene Parameters"
- Section 43.4.2, "Configuring the Custom AnalyzerFactory"

43.4.1 Configuring Lucene Parameters

Note: Some of these parameters can cause significant changes in the way the index performs at run time. Refer to the Lucene documentation and rely on experimentation to determine the best settings for your site. If you do not have a compelling reason to change the defaults, it is highly advised that you keep the defaults.

In the Lucene search engine, an index can be created with a certain set of parameters that determine how the index is created and how it performs. While the Lucene default parameters are reasonable, WebCenter Sites provides administrators with a way to change them.

The IndexSourceMetaDataConfig table contains one row per index you would find in the system. Each row has a field named "properties" whose contents are used to configure Lucene parameters. Parameter-value pairs are separated by a semicolon (';') as shown below:

```
param1=value1;param2=value2
```

Parameters supported by WebCenter Sites are listed in Table 43–4.

Table 43–4 Parameters Supported by WebCenter Sites

Parameter	Type	Description
mergeFactor	Integer	Determines how often segment indices are merged. With smaller values, less RAM is used while indexing, and searches on unoptimized indices are faster, but indexing speed is slower. With larger values, more RAM is used during indexing, and while searches on unoptimized indices are slower, indexing is faster. Thus larger values (> 10) are best for batch index creation, and smaller values (< 10) for indices that are interactively maintained. This must never be less than 2. The default value is 10.
maxMergeDocs	Integer	Determines the largest number of documents ever merged. Small values (e.g., less than 10,000) are best for interactive indexing, as this limits the length of pauses while indexing to a few seconds. Larger values are best for batched indexing and speedier searches. Defaults to max integer value (231-1).
maxBufferedDocs	Integer	Determines the minimal number of documents required before the buffered in-memory documents are merging and a new Segment is created. Defaults to 10.
optimizeInterval	Integer	Determines the time interval (in seconds) between optimize() calls. The default value is 30 seconds, which is the recommended value for most systems. If you have a large amount of data changes, set this parameter to any value within the range of 300 to 600 seconds.

Table 43–4 (Cont.) Parameters Supported by WebCenter Sites

Parameter	Type	Description
`commitLockTimeout`	Long	Sets the maximum time to wait for a commit lock (in milliseconds). Defaults to `10000`.
`maxFieldLength`	Integer	Maximum number of terms that will be indexed for a single field in a document. This limits the amount of memory required for indexing, so that collections with very large files will not crash the indexing process by running out of memory. Note that this effectively truncates large documents, excluding from the index terms that occur further in the document. If you know your source documents are large, be sure to set this value high enough to accommodate the expected size. If you set it to max value of Integer (231-1), then the only limit is memory, but you should anticipate an OutOfMemoryError. By default, no more than 10,000 terms will be indexed for a field.

Table 43–4 (Cont.) Parameters Supported by WebCenter Sites

Parameter	Type	Description
`termIndexInterval`	Integer	Sets the interval between indexed terms. Large values cause less memory to be used by IndexReader, but slow random-access to terms. Small values cause more memory to be used by an IndexReader, and speed random-access to terms. This parameter determines the amount of computation required per query term, regardless of the number of documents that contain that term. In particular, it is the maximum number of other terms that must be scanned before a term is located and its frequency and position information may be processed. In a large index with user-entered query terms, query processing time is likely to be dominated not by term lookup but rather by the processing of frequency and positional data. In a small index or when many uncommon query terms are generated (e.g., by wildcard queries) term lookup may become a dominant cost. In particular, numUniqueTerms/interval terms are read into memory by an IndexReader, and, on average, interval/2terms must be scanned for each random term access. Default value is `128`.
`useCompoundFile`	String (must be `yes` or `no`)	Setting to turn on usage of a compound file. When on, multiple files for each segment are merged into a single file once the segment creation is finished.
`writeLockTimeout`	Long	Sets the maximum time to wait for a write lock. Default value is `1000`.

43.4.2 Configuring the Custom AnalyzerFactory

In Lucene, an Analyzer represents a policy for extracting index terms from text. Analyzers are used at the time of indexing as well as searching for various tasks such as removing stop words, removing white spaces, etc.

Different Analyzers exist in the Lucene repository for handling various locales. Often Analyzers are used for injecting synonyms or addressing accented characters gracefully. You can also build your own Analyzer by using any of the Lucene standard analyzers as a basis.

WebCenter Sites's Lucene implementation uses `StandadAnalyzer`, a general purpose analyzer for the English language. However, WebCenter Sites supports custom Analyzers via a plugin interface, `AnalyzerFactory`. The configured `AnalyzerFactory` is used to look up the analyzer, when required, in the process of indexing or searching. The `AnalyzerFactory` looks up the analyzer in the following instances:

- When building the index as a whole
- When parsing a query
- When indexing an individual row

To plug in a custom `AnalyzerFactory`, you need to implement and register the `AnalyzerFactory` interface. Registration is done by modifying a row in `SearchEngineMetaDataConfig` table. Add the following to the `properties` field of the row whose **Name** field is set to `Lucene`.

```
AnalyzerFactory=<fully qualified class name of your custom AnalyzerFactory>
```

44

Creating a Hierarchical Flex Family

This chapter provides a tutorial to help you create a flex family, a simple three-level hierarchy that is based on single-valued definitions (and for the time being, ignores attributes and their inheritance). When you complete the tutorial, you will have a basic understanding of the flex asset model and how it is used to create hierarchical content in WebCenter Sites. You will also have a model from which to create similar hierarchies.

Section 44.1, "Hierarchical Organization" and Section 44.2, "Flex Family Specifications" describe the flex family that you will be creating. Section 44.3, "Procedures" is a tutorial, where you will be creating a small flex family with generic names for its family members.

This chapter contains the following sections:

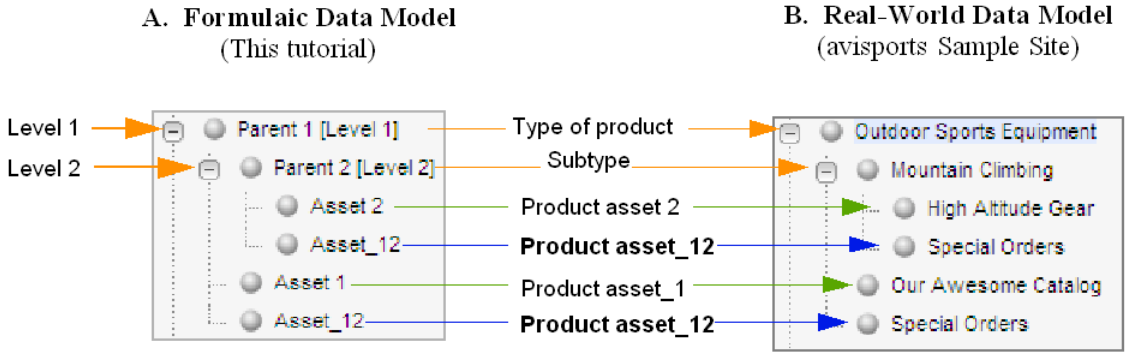
- Section 44.1, "Hierarchical Organization"
- Section 44.2, "Flex Family Specifications"
- Section 44.3, "Procedures"
- Section 44.4, "Next Steps"

44.1 Hierarchical Organization

The flex family that you will be creating in this tutorial consists of three levels:

- A top-level parent (named Parent 1 [Level 1] in our example).
- A second-level parent (named Parent 2 [Level 2] in our example).
- Assets, at the third level. One asset is placed directly under its level 1 parent; another asset is placed directly under its level 2 parent; the third asset is placed under both the level 1 parent and the level 2 parent.

In the WebCenter Sites interface, the hierarchy looks exactly as shown in Figure A of this graphic. The representation is formulaic; Figure B of this graphic shows how it translates to a real-world model, represented by the avisports sample site.



On the sample site (Figure B):

- Parent 1 [Level 1] is named Outdoor Sports Equipment.
- Parent 2 [Level 2] is named Mountain Climbing a subtype of Outdoor Sports Equipment.
- Asset 1 is named Our Awesome Catalog, an asset of the type Outdoor Sports Equipment.
- Asset_12 is named Special Offers. It appears under both Outdoor Sports Equipment and Mountain Climbing.

44.2 Flex Family Specifications

Table 44–1 lists the flex family members that you will be creating in this tutorial. Note that flex filters are optional components; they are not included in this tutorial.

Table 44–1 Flex Family Members

Flex Family Member	Name	Instances	Based on Parent Definition	Based on Flex Definition
Flex Attribute Type	My Attribute	Attribute_1[1] Attribute_2	n/a	n/a
Flex Parent Definition Type	My Parent Definition	Level 1 Def Level 2 Def	n/a Level 1 Def	n/a
Flex Definition Type	My Flex Definition	Flex Def 1 Flex Def 2 Flex Def_12	Level 1 Def Level 2 Def Level 1 Def *and* Level 2 Def	n/a
Flex Parent Type	My Parent	Parent 1 [Level 1] Parent 2 [Level 2]	Level 1 Def Level 2 Def	n/a
Flex Asset Type	My Asset	Asset 1 Asset 2 Asset_12	n/a	Flex Def 1 Flex Def 1 Flex Def_12
Flex Filter Type	n/a	n/a	n/a	n/a

[1] Suffixes 1, 2 and _12 refer to levels of the hierarchy (_12 denotes both levels 1 and 2). For example, Asset 1 denotes an asset that is placed under level 1. Asset_12 denotes an asset that is placed under both levels 1 and 2.

44.3 Procedures

In this tutorial, you will be creating a small flex family with generic names for its family members. This approach will help you visualize the formula for building hierarchies.

At the end of this tutorial, you will change the names of selected family members to real-world names to understand how a formulaic data model translates to a business-related data model. You will also add more parents and assets to the hierarchy, giving them real-world names as you create them.

This section contains the following topics:

44.3.1 Step 1: Create a Flex Family

In this step, you will create a flex family by naming its required members.

To create the flex family

1. Launch the WebCenter Sites Admin interface.
2. On the **Admin** tab, expand **Flex Family Maker** and double-click **Add New Family**.
3. In the Flex Family Maker form, fill in the fields exactly as shown below:

Table 44–2 Flex Family Maker Form

Field Name	Enter	Comments
Flex Attribute	MyAttribute	In this step, you are naming the database tables that WebCenter Sites will create for the flex family. The names must not contain spaces.
Flex Parent Definition	MyParentDefinition	n/a
Flex Definition	MyFlexDefinition	n/a
Flex Parent	MyParent	n/a
Flex Asset	MyAsset	n/a
Flex Filter	n/a	n/a

4. Click **Continue**.
5. In the next form, fill in the fields for each new member of the family as follows:

   a. Click in the **Description** field and enter the same name as in step 3, but separate the words in the name with spaces.

      The name that you enter will be used throughout the WebCenter Sites interface to identify the asset type.

   b. Click in the **Plural** field and enter the plural form of the name used in the preceding step.

   c. Click **Add New Flex Family**.

6. Wait for WebCenter Sites to create the flex family and return the message indicating that the flex family members (asset types) were successfully installed.

7. Go to the next step.

44.3.2 Step 2: Enable the New Flex Asset Types

In this step, you will enable the flex family members for the avisports sample site. You will also create start menu items for the members, so that you can create and search for their instances in subsequent steps.

> **Note:** If the avisports sample site is not installed on your system, you can enable the flex family for a site of your choice.

To enable the new flex asset types

1. On the **Admin** tab, expand the **Sites** node and complete the following steps:

   a. Expand **avisports** (the sample site where the flex family will be enabled), or a site of your choice.

   b. Under that site, expand **Asset Types** and double-click **Enable**.

      a. From the list, select the asset types that you just created (MyAsset, MyAttribute, MyFlexDefinition, MyParent, MyParentDefinition).

      b. Click **Enable Asset Types**.

   c. In the Enable Asset Types form:

      a. Make sure that all Start Menu options are selected (so that, later, you can create and search for instances of the family members.)

      b. Click **Enable Asset Types**.

2. Wait for WebCenter Sites to display the message indicating that the asset types have been enabled for the site.

3. Go to the next step.

44.3.3 Step 3: Add a Flex Family Tab to WebCenter Sites's Tree

In this step, you will add a tab that tracks the creation of your flex family. You will set up this tab to display selected members of the flex family as you finish creating them.

To add the tree tab

1. On the **Admin** tab, double-click the **Tree** node.

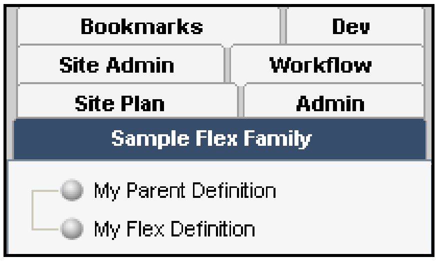
2. In the Tree Tabs form, scroll to the bottom and click **Add New Tree Tab**.

3. In the Add New Tree Tab form, fill in the fields as follows:

Table 44–3 Add New Tree Tab Form

Field Name	Enter or Select	Comments
Title	Sample Flex Family	Name of the tab.
Tooltip	Sample Flex Family	Description of the tab.
Sites	avisports (or the site you chose in Section 44.3.2, "Step 2: Enable the New Flex Asset Types")	Site on which to enable the flex family.
Required Roles	Any	n/a
Tab Contents	My Parent Definition My Parent My Flex DefinitionMy Asset **Note:** Click **Add Selected Items** and use the **Display Order** arrow to arrange the members in the order shown above.	n/a

4. Click **Save**.
5. Refresh the screen.
6. Click the Sample Flex Family tab and make sure its contents are identical to the display below:



7. Go to the next step.

44.3.4 Step 5: Create Parent Definition Assets

In this step, you will create two parent definitions. The first parent definition establishes the top level of the hierarchy; the second parent definition establishes the second level.

To create the first parent definition asset

1. In the menu bar, click **New**.
2. From the list of options that appears, select **New My Parent Definition**.
3. In the next form:
   a. Fill in the fields as follows:

Table 44–4 New My Parent Definition Form

Field Name	Enter or Select	Comments
Name	Level 1 Def	This is our name for the definition of the first level of the hierarchy.
Description	Level 1 Def	n/a
Parent Select Style	Select Box	n/a
My Parent Definitions	n/a	No parent definitions are selected (or available) in this field. Therefore, this parent definition establishes the first level of the hierarchy.

   b. Click the **Save icon**.

To create the second parent definition asset

1. In the menu bar, click **New**.
2. From the list of options that appears, select **New My Parent Definition**.
3. In the next form:

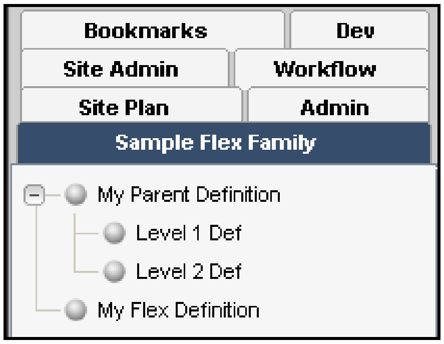
   a. Fill in the fields as follows:

Table 44–5 New My Parent Definition Form

Field Name	Enter or Select	Comments
Name	Level 2 Def	This is our name for the definition of the second level of the hierarchy.
Description	Level 2 Def	n/a
Parent Select Style	Select Box	n/a
My Parent Definitions	Level 1 Def **Note**: Under Single Value, click the **Required** arrow to move your selection to the Selected list.	Choosing **Level 1 Def** subordinates the current parent definition to **Level 1 Def**. Chaining definitions in this manner establishes Level 2 Def as the second level. When parents are created and based on the current parent definition (Level 2 Def), they are subordinated to parents that are based on Level 1 Def.

   b. Click the **Save icon**.

4. Refresh the screen.
5. Click the **Sample Flex Family** tab and expand its contents to make sure they are identical to the display below:



6. Go to the next step.

44.3.5 Step 6: Create Flex Parent Assets

In this step, you will create two flex parent assets, and base them on the flex parent definitions that you created in the previous step. The first parent asset will occupy the top level of the hierarchy. The second parent asset will occupy the second level of the hierarchy.

To create the top-level parent of the hierarchy

1. Switch to the Contributor application by clicking the **Contributor** icon on the top.
2. From the **Content** menu, choose **New**, then **New My Parent**.
3. In the form that appears, fill in the fields as follows:

Table 44–6 New My Parent Form

Field Name	Enter or Select	Comments
Name	Parent 1 [Level 1]	This is our name for a level 1 parent in the hierarchy (in our example, the name is genericized simply to help you identify the level). **Note:** At the end of this tutorial, you will change the name to a business-specific name (Outdoor Sports Equipment, in our example, which describes the inventory of a company dealing with sports gear.)
My Parent Definition	Level 1 Def	Choosing **Level 1 Def** places the parent you are creating at the top level of the hierarchy.

4. On top of the form, click the **Save** icon.

To create the second-level parent of the hierarchy

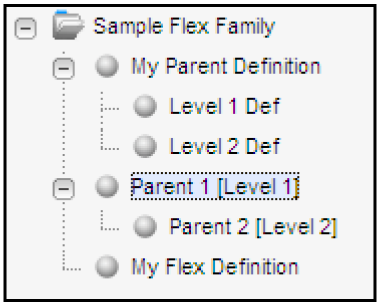
1. From the **Content** menu, choose **New**, then **New My Parent**.
2. In the form that appears, fill in the fields as follows:

Table 44–7 New My Parent Form

Field Name	Enter or Select	Comments
Name	Parent 2 [Level 2]	This is our name for a level 2 parent in the hierarchy (in our example, the name is genericized simply to help you identify the level). **Note:** At the end of this tutorial, you will change the name to a business-specific name, Mountain Climbing in our example (an appropriate name given that Parent 1 [Level 1] is Sports Equipment).
My Parent Definition	Level 2 Def	Selecting **Level 2 Def** places the parent you are creating at the second level of the hierarchy.
Level 1 Def	Parent 1 [Level 1] is selected by default.	n/a

3. Click the **Save** icon.

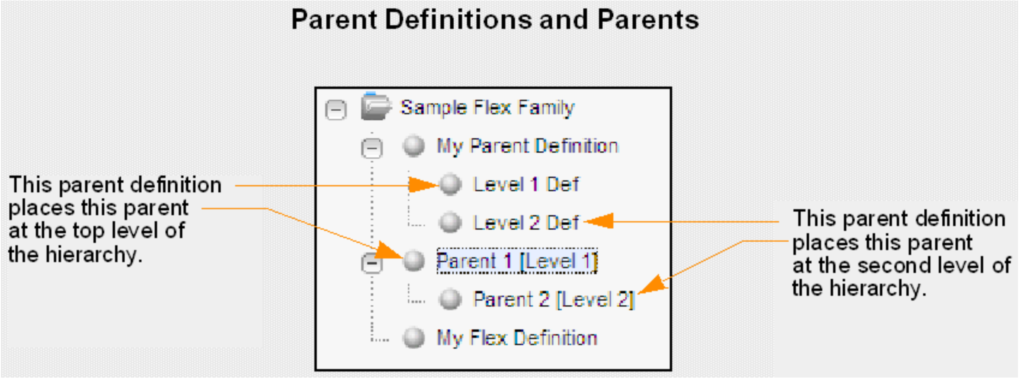
4. On the Content Tree, expand **Sample Flex Family** to make sure its contents are identical to the display below:



Sample Flex Family
- My Parent Definition
  - Level 1 Def
  - Level 2 Def
- Parent 1 [Level 1]
  - Parent 2 [Level 2]
- My Flex Definition

Note: Before going to the next step, review Figure 44–1. The figure summarizes how the objects that you created, parent definitions and parents based on the definitions, relate to each other.

Figure 44–1 Parents and Parent Definitions


Parent Definitions and Parents
Sample Flex Family
My Parent Definition
Level 1 Def
Level 2 Def
Parent 1 [Level 1]
Parent 2 [Level 2]
My Flex Definition
This parent definition places this parent at the top level of the hierarchy.
This parent definition places this parent at the second level of the hierarchy.

44.3.6 Step 7: Create Flex Definition Assets

In this step, you will create three flex definition assets:

- The first flex definition asset will be used to place assets under Parent 1 [Level 1].
- The second flex definition asset will be used to place assets under Parent 2 [Level 2].
- The third flex definition asset will be used to place the asset under both levels of the hierarchy.

To create the first flex definition asset

1. Switch to the Admin interface by clicking the **Admin** icon, located in the applications bar.
2. In the menu bar, click **New**.
3. From the list of options that appears, select **New My Flex Definition**.
4. In the form that appears, fill in the fields as follows:

Table 44–8 New My Flex Definition Form

Field Name	Enter or Select	Comments
Name	Flex Def 1	n/a
My Parent Definitions	Level 1 Def **Note:** Under Single Value, click the **Required** arrow.	Choosing **Level 1 Def** and **Single Value** means that when you use the current flex definition to create flex assets, the assets can be placed under *only one* parent that use Level 1 Def as its parent definition. In our example, the asset will be placed under Parent 1 [Level 1]. **Note**: Selecting a Multiple Values option would allow you to place the asset under *any* and *all* parents that use Level 1 Def as their parent definition.

5. Click the **Save** icon.

To create the second flex definition asset

1. From the button bar, click **New**.
2. From the list of options that appears, select **New My Flex Definition**.
3. In the form that appears, fill in the fields as follows:

Table 44–9 New My Flex Definition Form

Field Name	Enter or Select	Comments
Name	Flex Def 2	n/a
My Parent Definitions	Level 2 Def **Note**: Under Single Value, click the **Required** arrow.	Choosing **Level 2 Def** and **Single Value** means that when you use the current flex definition to create flex assets, the assets can be placed under *only one* parent that uses Level 2 Def as its parent definition. In our example, the asset will be placed under Parent 2 [Level 2]. **Note**: Selecting a Multiple Values option would allow you to place the asset under *any* and *all* parents that use Level 2 Def as their parent definition.

4. Click the **Save** icon.

To create the third flex definition asset

1. From the button bar, click **New**.
2. From the list of options that appears, select **New My Flex Definition**.
3. In the form that appears, fill in the fields as follows:

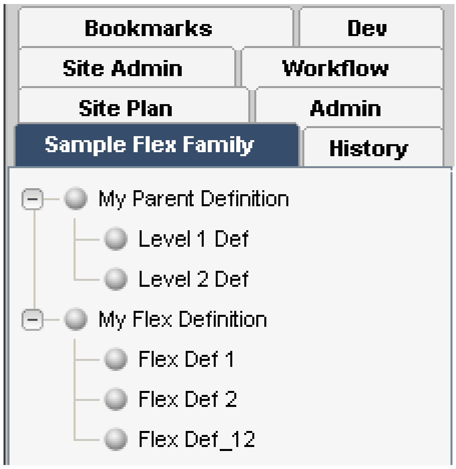
Table 44–10 New My Flex Definition Form

Field Name	Enter or Select	Comments
Name	Flex Def_12	n/a
Parent Definitions	Level 1 Def Level 2 Def **Note:** Under Single Value, click the **Required** arrow.	Choosing **Level 1 Def** *and* **Level 2 Def** and **Single Value** means that when you use the current flex definition to create flex assets, the assets will be placed under *only one* parent that uses Level 1 Def *and* under *only one* parent that uses Level 2 Def as parent definitions. In our example, the assets will be placed under Parent 1 [Level 1] *and* Parent 2 [Level 2]).

Table 44–10 (Cont.) New My Flex Definition Form

Field Name	Enter or Select	Comments
Parent Definitions (*continued*)	n/a	**Note**: Selecting a Multiple Values option would allow you to place the asset under *any* and *all* parents that use Level 1 Def and Level 2 Def as their parent definitions.

4. Click the **Save** icon.
5. Refresh the screen.
6. Click the Sample Flex Family tab and expand its contents to make sure they are identical to the display below:



> **Note:** Before going to the next step, review Figure 44–2, which depicts the Sample Flex Family structure as it appears in the Contributor application. This figure summarizes how the objects that you created, flex definitions, relate to the parent definitions they are based upon.

44.3.7 Step 8: Create Flex Assets

In this step, you will complete the flex family by adding the third level of the hierarchy; the flex assets. You will create three assets:

- The first asset you will place under Parent 1 [Level 1].
- The second asset you will place under Parent 2 [Level 2].
- The third asset you will place under both Parent 1 [Level 1] and Parent 2 [Level 2].

To create the first flex asset

1. Switch to the Contributor application by clicking the **Contributor** icon on the top.
2. From the Content menu, choose **New**, then **New My Asset**.
3. In the form that appears, fill in the fields as follows:

Table 44–11 New My Asset Form

Field Name	Enter or Select	Comments
Name	Asset 1	n/a
My Flex Definition	Flex Def 1	Choosing Flex Def 1 will place the asset you are creating under Parent 1 (Level 1].

4. Click the **Save** icon.

To create the second flex asset

1. Switch to the Contributor application by clicking the **Contributor** icon on the top.
2. From the Content menu, choose **New**, then **New My Asset**.
3. In the form that appears, fill in the fields as follows:

Table 44–12 New My Asset Form

Field Name	Enter or Select	Comments
Name	Asset 2	n/a
My Flex Definition	Flex Def 2	Choosing Flex Def 2 will place the asset you are creating under Parent [Level 2].

4. In the next form, click **Save**.

To create the third flex asset

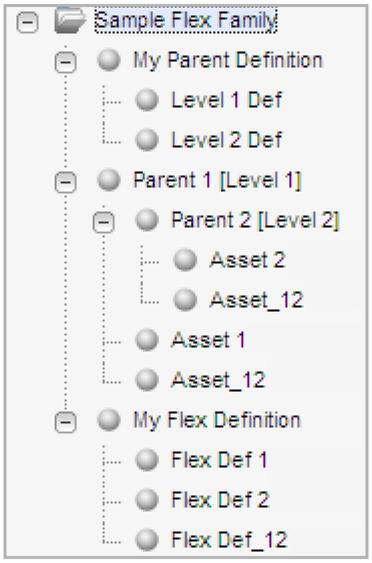
1. Switch to the Contributor application by clicking the **Contributor** icon on the top.
2. From the Content menu, choose **New**, then **New My Asset**.
3. In the form that appears, fill in the fields as follows:

Table 44–13 New My Asset Form

Field Name	Enter or Select	Comments
Name	Asset_12	n/a
Flex Definition	Flex Def_12	Choosing Flex Def_12 will place the asset you are creating under both Parent [Level 1] *and* Parent [Level 2].

4. In the next form, click the **Save** icon.
5. Refresh the screen.
6. Click the **Sample Flex Family** tab and expand its contents to make sure they are identical to the display below:

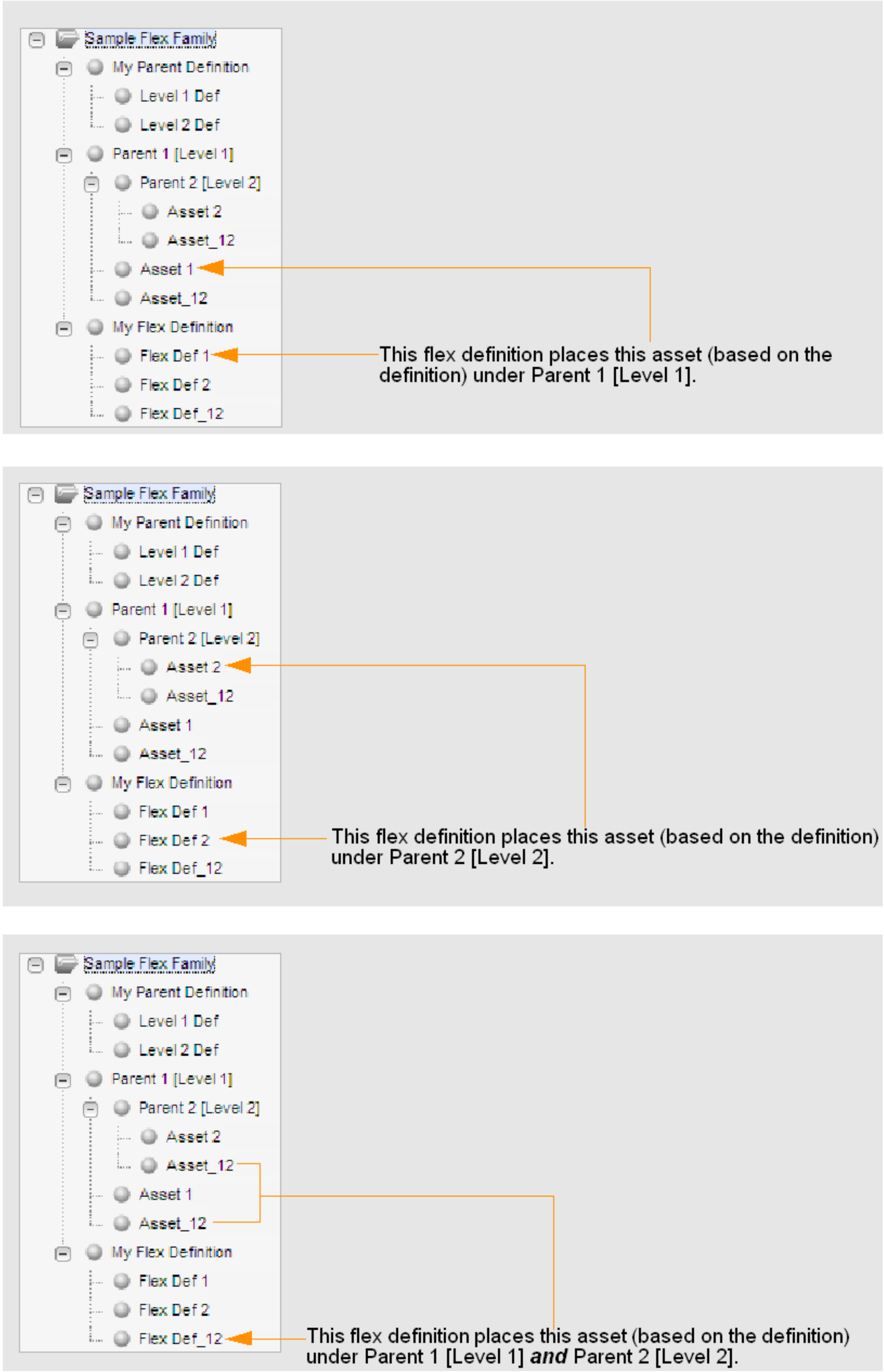
Figure 44–2 Sample Flex Family Structure



- Sample Flex Family
  - My Parent Definition
    - Level 1 Def
    - Level 2 Def
  - Parent 1 [Level 1]
    - Parent 2 [Level 2]
      - Asset 2
      - Asset_12
    - Asset 1
    - Asset_12
  - My Flex Definition
    - Flex Def 1
    - Flex Def 2
    - Flex Def_12

Note: The next figure summarizes how the objects that you created, assets, relate to the flex definitions they are based upon.

Flex Definitions and Assets



- Sample Flex Family
  - My Parent Definition
    - Level 1 Def
    - Level 2 Def
  - Parent 1 [Level 1]
    - Parent 2 [Level 2]
      - Asset 2
      - Asset_12
    - Asset 1
    - Asset_12
  - My Flex Definition
    - Flex Def 1
    - Flex Def 2
    - Flex Def_12

This flex definition places this asset (based on the definition) under Parent 1 [Level 1].

- Sample Flex Family
  - My Parent Definition
    - Level 1 Def
    - Level 2 Def
  - Parent 1 [Level 1]
    - Parent 2 [Level 2]
      - Asset 2
      - Asset_12
    - Asset 1
    - Asset_12
  - My Flex Definition
    - Flex Def 1
    - Flex Def 2
    - Flex Def_12

This flex definition places this asset (based on the definition) under Parent 2 [Level 2].

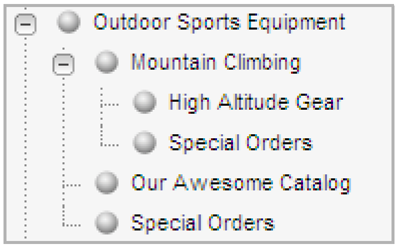
- Sample Flex Family
  - My Parent Definition
    - Level 1 Def
    - Level 2 Def
  - Parent 1 [Level 1]
    - Parent 2 [Level 2]
      - Asset 2
      - Asset_12
    - Asset 1
    - Asset_12
  - My Flex Definition
    - Flex Def 1
    - Flex Def 2
    - Flex Def_12

This flex definition places this asset (based on the definition) under Parent 1 [Level 1] ***and*** Parent 2 [Level 2].

44.3.8 Step 9: Translate the Formulaic Data Model into a Real-World Data Model

In this step, you will rename the parent definitions, parents, and assets in order to translate the formulaic data model you just created into a real-world model. (In practice, instead of renaming flex family members, you would name them directly in their business context, as you create them.)

In our example, the real-world model describes a business that deals with sports gear. The flex parents and assets, after you finish renaming them, will be listed in your Content Tree tab as shown in the figures below:



```
⊖ ● Outdoor Sports Equipment
   ⊖ ● Mountain Climbing
        ● High Altitude Gear
        ● Special Orders
     ● Our Awesome Catalog
     ● Special Orders
```

To create the real-world model

1. Rename the parent definitions as follows:

   a. Open the Admin application.

   b. In the Sample Flex Family Tree, right-click **Level 1 Def** and select **Edit** from the context menu.

   c. Replace the parent's name with **Level 1 Def (Type of Sports Equipment)**, and click the **Save** icon.

   d. In the same manner, replace the name of **Level 2 Def** with **Level 2 Def (Sport)**.

2. Rename the parents as follows:

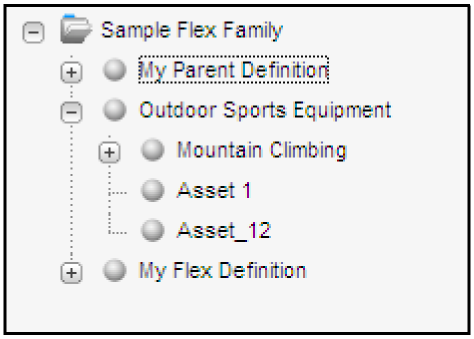
   a. Switch to the Contributor application.

   b. Click **Content Tree** to display its contents.

   c. In the Sample Flex Family Tree, right-click **Parent 1 [Level 1]** and select **Edit** from the context menu.

   d. Replace the parent's name with **Outdoor Sports Equipment**, and click the **Save i**con.

   e. In the same manner, replace the name of **Parent 2 [Level 2]** with **Mountain Climbing**. The Sample Flex Family Tree should look like this:



```
⊖ Sample Flex Family
   ⊕ ● My Parent Definition
   ⊖ ● Outdoor Sports Equipment
      ⊕ ● Mountain Climbing
        ● Asset 1
        ● Asset_12
   ⊕ ● My Flex Definition
```

3. Rename the assets as follows:

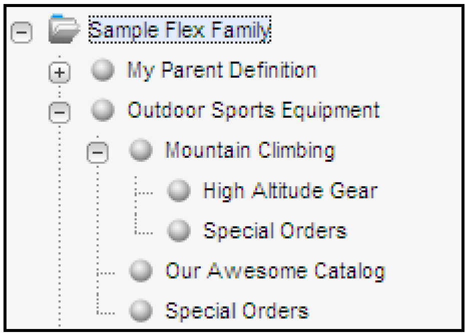
   a. In the Sample Flex Family Tree, expand **Outdoor Sports Equipment**.

   b. Right-click **Asset 1** and select **Edit**.

   c. Replace the asset's name with **Our Awesome Catalog**, then click the **Save** icon.

d. In the same manner, expand **Mountain Climbing** and replace the name of **Asset 2** with **High Altitude Gear**.

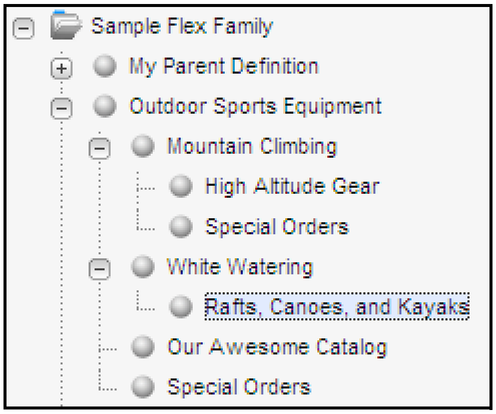
e. Replace the name of **Asset_12** with **Special Orders**. The assets you renamed should look like this:



44.3.9 Step 10: Develop Your Real-World Model

In this step, you will develop your data model by creating a new parent and its asset, giving each a real-world name. You will do the following:

- Create a second level-2 parent and name it **White Watering**.
- Create the parent's asset (a catalog) and name it **Rafts, Canoes, and Kayaks**. Expand White Watering to display its asset.



Note: At the conclusion of this procedure are suggestions for further developing the real-world model, using advanced techniques. Our example illustrates the creation of n:1 parent-child relationships through the use of multi-valued parent and flex definitions. Guidelines, rather than step-by-step instructions are provided to help you through the development process and let you test your understanding of the process.

If you need instructions for developing your model, follow the steps below:

To create the level-2 parent:

1. Switch to the Contributor application by clicking the **Contributor** icon on the top.
2. From the Content menu, choose **New**, then **New My Parent**.
3. In the form that appears, fill in the fields as follows:

Table 44–14 New My Parent Form

Field Name	Value
Name	White Watering
My Parent Definition	Level 2 Def (Sports)

4. Click the **Save icon**.

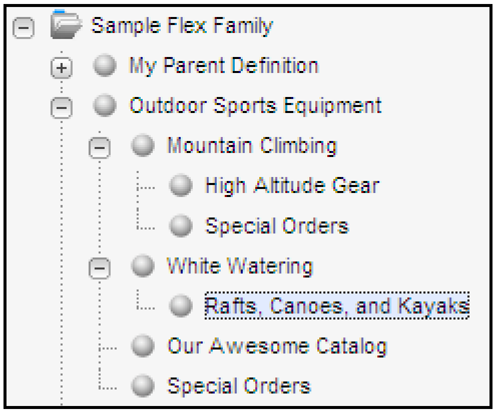
To create the parent's asset:

1. Switch to the Contributor application by clicking the **Contributor** icon on the top.
2. From the Content menu, choose **New**, then **New My Asset**.
3. In the form that appears, fill in the fields as follows:

Table 44–15 New My Asset Form

Field Name	Value
Name	Rafts, Canoes, and Kayaks
My Flex Definition	Flex Def 2

4. From the Level 2 Def drop-down, select **White Watering**.
5. Click the **Save** icon.
6. Refresh the screen and display the Sample Flex Family tree tab. Its content should be identical to the display below:



44.4 Next Steps

At this point you have created a rudimentary data model. From here, you can expand the data model by creating additional parents to occupy a given level, creating additional levels (by chaining flex parent definitions), and so on. You can also create associations among assets, write template code to format the assets, and test the publishing options that display the assets to site visitors. For information on these operations, see Chapter 16, "Designing Flex Asset Types."

45

WebCenter Sites URL Assemblers

This chapter explains WebCenter Sites URL assemblers, which manage URL assembly and disassembly, and provide an interface that you can use to define the appearance of URLs.

This chapter contains the following sections:

- Section 45.1, "Overview of WebCenter Sites URL Assemblers"
- Section 45.2, "Assemblers Installed with WebCenter Sites"
- Section 45.3, "Working with Assemblers"
- Section 45.4, "Generating Vanity URL Links in a Web Page"

45.1 Overview of WebCenter Sites URL Assemblers

URL assemblers, in conjunction with URL generation tags, are used to generate WebCenter Sites URLs and to disassemble the URLs they generate.

This section contains the following topics:

- Section 45.1.1, "URL Assembly"
- Section 45.1.2, "Assembler Discovery and Disassembly"
- Section 45.1.3, "URL Assembly and Disassembly Using GET and POST Requests"

45.1.1 URL Assembly

WebCenter Sites URL generation tags (`<satellite.link>`, `<satellite.blob>`, `<render.getpageurl>`, `<render.getbloburl>`, `<render.satelliteblob>`, `<rendergettemplateurl>`) are used to construct a link to a WebCenter Sites resource, such as a page or a blob. The data, such as tag attributes and nested argument tags which you specify when using the tag, is converted into an abstract object called a **URL definition**. The URL definition is passed into the **URL assembler**. The URL assembler then converts the definition into a string URL that is returned.

Two assemblers are installed with WebCenter Sites, but you have the option of creating your own assemblers in order to directly control the appearance of your URLs. Before you can use the assemblers you create, you must first register them with WebCenter Sites.

The assembler that is configured as the default is used to create all WebCenter Sites URLs. You can change the default assembler. You can also override the use of this default assembler in individual link tags.

45.1.2 Assembler Discovery and Disassembly

Because an assembler can create a URL in any form that the assembler's author dictates, it may be impossible for the URL to be decoded into parameters by an application server when an assembled link is requested. For decoding to take place, the assembler must be able to disassemble the string URL into its definition. Assemblers are therefore reversible, that is, capable of disassembling any URLs that they assembled.

If a URL has been created using an assembler other than the default assembler, then the default assembler cannot disassemble the URL. At that point, the next highest ranked assembler attempts to disassemble the URL. If it succeeds in creating a definition, then the assembler engine is said to have discovered its assembler, and the definition is converted into parameters for processing. If the next highest ranked assembler fails to disassemble the URL, the third highest ranked assembler is called upon to disassemble it. This process continues until the URL is successfully disassembled. Note that this process requires an assembler to be able to recognize the URLs it assembled as its own, and all other URLs as foreign.

See Section 45.3.1, "Creating Assemblers" and Section 45.3.2, "Registering and Ranking Assemblers" for more information about creating, registering, and ranking assemblers.

45.1.3 URL Assembly and Disassembly Using GET and POST Requests

URL assemblers are only invoked on `GET` requests. They are not invoked on POST requests. For example, when accessing a page with a `GET` request, the URL assembler is invoked to disassemble the URL. It then provides the appropriate parameters that WebCenter Sites requires to open that page (such as `c`, `cid`, and `pagename`) by adding them to the definition (if they do not already exist in the definition). However, when a request is POSTed, such as a form with `method=post`, the URL assembler is not invoked to disassemble the URL. Therefore, the parameters WebCenter Sites requires to open the page must be part of the post request itself.

This can be accomplished by encoding the following tag into the page's template or element (replacing the sample values with the parameters WebCenter Sites requires for the page's URL):

```
<satellite:form method="post" id="assetid">
<render:gettemplateurlparameters list="args" ... /><!-- add all parameters that
are normally part of a URL -->
        <ics:listloop listname="args">
               <input type="hidden" name="<string:stream list="args"
column="name"/>" value="<string:stream list="args" column="value"/>"/>
        </ics:listloop>
```

45.2 Assemblers Installed with WebCenter Sites

The two assemblers that are installed with WebCenter Sites are Query Assembler and QueryAsPathInfo Assembler.

This section contains the following topics:

- Section 45.2.1, "Query Assembler"
- Section 45.2.2, "QueryAsPathInfo Assembler"

45.2.1 Query Assembler

The Query Assembler creates URLs with query strings. It is the default assembler, and it is automatically registered in WebCenter Sites. Therefore, until you make any modifications (such as changing the default assembler or overriding the default in link tags), Query Assembler will be used to generate all URLs.

45.2.2 QueryAsPathInfo Assembler

The QueryAsPathInfo Assembler does not use query strings. Instead, the QueryAsPathInfo Assembler encodes the query string and appends it to the end of the servlet name. The benefit of this assembler is that it creates URLs that can be indexed by search engines. The QueryAsPathInfo Assembler is not automatically registered with WebCenter Sites.

Note: A sample URL assembler implementation ships with the FirstSiteII sample site. The source code is located on the WebCenter Sites CD:

```
/ContentServer/FirstSiteII/PrettyURL/src/com/fatwire/firstsi
te/uri/FSIIAssembler.java
```

Additional information is available in the *Oracle Fusion Middleware WebCenter Sites Java API Reference*. The classes of interest are:

- `Definiton`
- `Definiton.AppType`
- `Definiton.ContainerType`
- `AbstractAssembler`
- `AbstractAssembler.AssemblyContext`
- `AbstractAssembler.DisAssemblyContext`
- `Assembler`
- `DisAssembler`

It may also be beneficial to set up a debug environment with `FSIIAssembler.java` so that you can step through the code.

45.3 Working with Assemblers

This section explains how to create and register your own assemblers. This section also explains how to modify link tags to override the use of the default assembler.

This section contains the following topics:

- Section 45.3.1, "Creating Assemblers"
- Section 45.3.2, "Registering and Ranking Assemblers"
- Section 45.3.3, "Modifying Link Tags"

45.3.1 Creating Assemblers

The WebCenter Sites URL Assembly module enables you to create your own assemblers. This option gives you direct control of the appearance of your URLs.

To create an assembler

1. Write a java class that implements the `com.fatwire.cs.core.uri.Assembler` interface. For information about this class, see the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.
2. Compile the class into a `.jar` file.
3. Deploy your class into the WebCenter Sites web application and the web application for each remote Satellite Server you have installed.

   This usually means copying the `.jar` file you just created into your web application's `WEB-INF/lib` folder. For remote Satellite Servers, this means copying it to the `resin/webapp/ROOT/WEB-INF/lib` folder.
4. Register your new assembler in the `ServletRequest.properties` file on both WebCenter Sites and all of your remote Satellite Servers (see Section 45.3.2, "Registering and Ranking Assemblers" if you need instructions).
5. Restart WebCenter Sites and all of your remote Satellite Servers.

45.3.2 Registering and Ranking Assemblers

Before an assembler can be used to create URLs, it must first be registered with WebCenter Sites. The registration is done by listing assembler class names with corresponding short forms in a property file. The registration also includes a ranking that indicates in which order the assemblers should be used.

To register an assembler

1. Invoke the Property Editor.
2. Open the `ServletRequest.properties` file.
3. Click the **URI Assembler** tab to access the assembler properties.
4. Specify the `classname` and `shortform` of the assembler you want to register.

   The third element in the property name indicates the ranking of the assembler. The assembler with the ranking of `0` is the highest ranked (and default) assembler, the assembler with the ranking of `1` is the next highest ranked, and so on.

   If you want to configure the new assembler to be the default assembler, enter the `classname` and `shortform` values in the properties that have 1 as their ranking.

   For example, the syntax to register the QueryAsPathInfo assembler as the default assembler would be as follows:

Table 45–1 Assembler Properties

Property Name	Property Value
`uri.assembler.0.classname`	`com.fatwire.cs.core.uri.QueryAsPathInfoAssembler`
`uri.assembler.0.shortform`	`pathinfo`
`uri.assembler.1.classname`	`com.fatwire.cs.core.uri.QueryAssembler`
`uri.assembler.1.shortform`	query

> **Note:** Make sure that the Query Assembler is always registered, even if you have lowered its ranking. The Query Assembler must be registered with the `shortform` value of query.

1. Depending on the ranking of the new assembler, you may need to adjust the rankings of the other assemblers. Verify that all of the assemblers are configured and ranked correctly in the property file. If they are not, make any necessary changes.
2. Choose **File**, then **Save** to save your changes and close the Property Editor.
3. Repeat steps 1 through 2 for each remote Satellite Server you have installed.
4. Restart WebCenter Sites and all of your remote Satellite Servers.

45.3.3 Modifying Link Tags

WebCenter Sites link tags can be modified to use an assembler other than the default assembler. The link tags accept an attribute, `assembler`, which take an assembler short form as a value.

For example, to override the default assembler with the QueryAsPathInfo assembler in an individual link tag, the syntax would be as follows:

```
<satellite:link pagename=example assembler=pathinfo />
```

45.4 Generating Vanity URL Links in a Web Page

If an asset has a vanity URL for the template passed in the name argument, then the tag returns a vanity URL. If the asset does not have a vanity URL, then the tag returns the Sites URL. The `<render:gettemplateurl>` tag can be used to generate a vanity URL link. Thus any redirected URL will always return the vanity URL if present. That way if an existing link is accessed (for example, from a bookmark), all subsequent URLs will be vanity URLs.

To generate a vanity URL for a specific WebRoot, an additional optional parameter called `hostparam` should be passed to `render: gettemplateurl`. The value of the `hostparam` should match the `hostname` attribute of WebRoot. This can be used to generate a link to another website or subdomain within the same website.

```
<render:gettemplateurl outstr="pageURL" hostparam=<hostname of the WebRoot asset>
tname='<%=ics.GetVar("template")%>' args="c,cid" />
```

46

White Space and Compression

When WebCenter Sites streams a text page, the page may contain a significant amount of white space (spaces, carriage returns and tabs) that have no effect on the data that is consumed by the client. The white space is visible when the source code is viewed by the consumer, and this is not desirable. Furthermore, excessive white space needlessly increases the size of the response, which ultimately increases bandwidth use. Consequently, it is beneficial to eliminate white space whenever possible.

This chapter contains the following sections:

- Section 46.1, "White Space and JSP"
- Section 46.2, "White Space and XML"
- Section 46.3, "Compression"
- Section 46.4, "JSP Design"

46.1 White Space and JSP

The JSP specification requires that all white space be preserved. Thus, a page that looks like this:

```
<%@ page import="my class name"%>
<%@ page import="my class 2"%>
<cs:ftcs>
<p>Hello world!</p>
</cs:ftcs>
```

will have three carriage returns and a tab preceding the `<p>` because the text is displayed on third line after the JSP has been interpreted. With more complicated pages, the problem is compounded.

46.2 White Space and XML

WebCenter Sites's XML processing language, being a proprietary set of xml-compliant tags, does not adhere to the white space preserving rules of JSP. As such, a WebCenter Sites XML page like this:

```
<? XML version 1.0 ?>
<FTCS>
<p>Hello World!</p>
</FTCS>
```

will display <p> as the first characters of output, because our xml parser will strip all of the white space (unless xml debug is enabled, in which case all the white space is preserved).

46.3 Compression

Because white space is an artifact of writing well-formatted code, its presence is an unfortunate side effect of programming practices that benefit the developer. The impact on the consumer and the customer is minimal except for bandwidth. To address bandwidth, the output of all text-based pages can be compressed. Compressing the output is done on the server-side, and decompression is done by the consumer's user-agent (browser). The compression/decompression is completely transparent to the end user. This sort of compression can yield up to an 80% reduction in bandwidth use. One commonly-used compression mechanism is the mod-gzip extension to the Apache web server. This module will automatically gzip all output to the user agent provided that it can decompress it. Configuration is minimal and its effectiveness is quite high. It can be obtained from SourceForge (http://sourceforge.net/projects/mod-gzip/). Similar tools are available for other common web servers such as IIS.

Another possibility is to do the compression at the application server layer, and leave the web server alone. This is best done by connecting a standard servlet filter to Satellite Server (or to WebCenter Sites if Satellite Server is not being used). The servlet filter is invoked in a prescribed order prior to and/or after the invocation of the specified servlet, and during invocation it can compress the output prior to sending it to compatible user-agents, exactly the same way mod-gzip works. One such compression filter can be found at SourceForge (http://sourceforge.net/projects/pjl-comp-filter/).

If you are interested in compression but need assistance, contact Oracle Consulting Services.

46.4 JSP Design

If compression is not an option, consider altering your JSP pages so that they do not require compression to address the white space problem. This can be done by changing the code above to this:

```
<%@ page import="my class name"
%><%@ page import="my class 2"
%><cs:ftcs><p>Hello world!</p></cs:ftcs>
```

While this is not as elegant (or readable), it will result in page output without any white space whatsoever prior to the <p> tag. An intermediate solution may be something like this:

```
<%@ page import="my class name"
%><%@ page import="my class 2"
%><cs:ftcs>
<p>Hello world!</p>
</cs:ftcs>
```

For extensive examples of how to address white space issues in JSP, refer to our WebServices elements in the ElementCatalog. They are included with WebCenter Sites.

47

Asset and Publish Events in WebCenter Sites

WebCenter Sites's event framework enables developers to write custom business logic to execute upon certain specific events in WebCenter Sites. The custom business logic needs to be implemented in the form of Java classes in the classpath that implements a pre-defined interface. This tutorial takes you through the process of creating listeners.

This chapter takes you through the process of creating event listeners and contains the following sections:

- Section 47.1, "Types of Events"
- Section 47.2, "Asset Events"
- Section 47.3, "Publishing Events"

47.1 Types of Events

Two types of events are currently supported by WebCenter Sites: Asset events and Publishing events.

- Asset events are the events that happen when assets are added, modified or deleted, either by a user or programmatically. Upon these events, the event framework looks up and executes the set of configured events.
- Publishing events are events that the RealTime publishing framework generates at each step of the publishing process. This feature is intended to facilitate system monitoring (such as SNMP) and other housekeeping processes.

47.2 Asset Events

This section contains the following topics:

- Section 47.2.1, "Writing an Asset Event Listener"
- Section 47.2.2, "Registering an Asset Event Listener"

47.2.1 Writing an Asset Event Listener

Asset listeners need to extend `AssetEventListener` in order to be notified of asset changes. A convenient base class `AbstractAssetEventListener` comes with WebCenter Sites. Extending from this class makes it easy to recognize the specific type of action that led to the event (add/modify/delete).

The following sample code implements a custom asset listener. All it does is print the asset IDs; however, it is easy to see how one could plug in custom business logic.

```
package com.mycompany
public final class CustomAssetEventListener extends AbstractAssetEventListener
{
    public void assetAdded(AssetId id)
    {
        System.out.println("Asset " + id + " added");
    }
    public void assetUpdated(AssetId id)
    {
        System.out.println("Asset " + id + " added");
    }
    public void assetDeleted(AssetId id)
    {
        System.out.println("Asset " + id + " added");
    }
}
```

Asset event listeners are invoked after the asset operation has taken place, but before committing the data.

47.2.2 Registering an Asset Event Listener

> **Note:** WebCenter Sites ships with a standard listener that is used for search indexing. Do not alter or delete it.

Asset event listeners are registered in the `AssetListener_reg` database table. Here is the structure of the table:

Table 47–1 AssetListener_reg Database Table

ID (integer)	Unique Identifier for the Row
`listener(String)`	Fully qualified class name that implements `AssetEventListener`. For example: `com.mycompany.CustomAssetEventListener`
`blocking(Y or N)`	'`Y`' indicates that the listener is blocking (runs synchronously with the thread that generated the event). '`N`' indicates that the listener is non blocking (runs in a separate thread).

47.3 Publishing Events

This section contains the following topics:

- Section 47.3.1, "Writing a Publishing Event Listener"
- Section 47.3.2, "Registering a Publishing Event Listener"

47.3.1 Writing a Publishing Event Listener

Publishing listeners need to implement `PublishingEventListener`. A `PublishingEvent` passed into the listener indicates the specific event that caused the invocation.

The following example shows a custom implementation that prints the `pubsession` ID to the console

```
package com.mycompany;
```

```
public class CustomPublishingEventListener implements PublishingEventListener
{
    public void onEvent( PublishingEvent event ) throws EventException
    {
        System.out.println( "Publishing event fired for     pubsession: " +
event.getPubSessionId());
        System.out.println( "Publishing task : " + event.getTaskName());
        System.out.println( "Status of the task :  " + event.getStatus());
        System.out.println( "Message associated with the task : " +
event.getMessage() );
    }
}
```

Publishing consists of multiple tasks (data gathering, packaging, transport, and so on), and each of them generates events. The `PublishingEvent` class represents an event in the publishing task. An implementation can query the task and its status from the event as shown above.

Each task generates events when the following states are reached:

- STARTED
- DONE
- CANCELLED
- SUBTASK_FINISHED
- FAILED

47.3.2 Registering a Publishing Event Listener

Publish event listeners are registered in the `FW_PublishingEventRegistry` database table. Here is the structure of the table:

Table 47–2 FW_PublishingEventRegistry Database Table

ID (integer)	Unique Identifier for the Row
`listener(String)`	Fully qualified class name that implements `AssetEventListener`. For example `com.mycompany.CustomPublishingEventListener`
`blocking(Y or N)`	'`Y`' indicates that the listener is blocking (runs synchronously with the thread that generated the event). '`N`' indicates that the listener is non blocking (runs in a separate thread).

48

Proxy Assets: Integrating Third-Party Content Sources

This chapter describes the integration of external web content into sites delivered by WebCenter Sites using the proxy asset type framework.

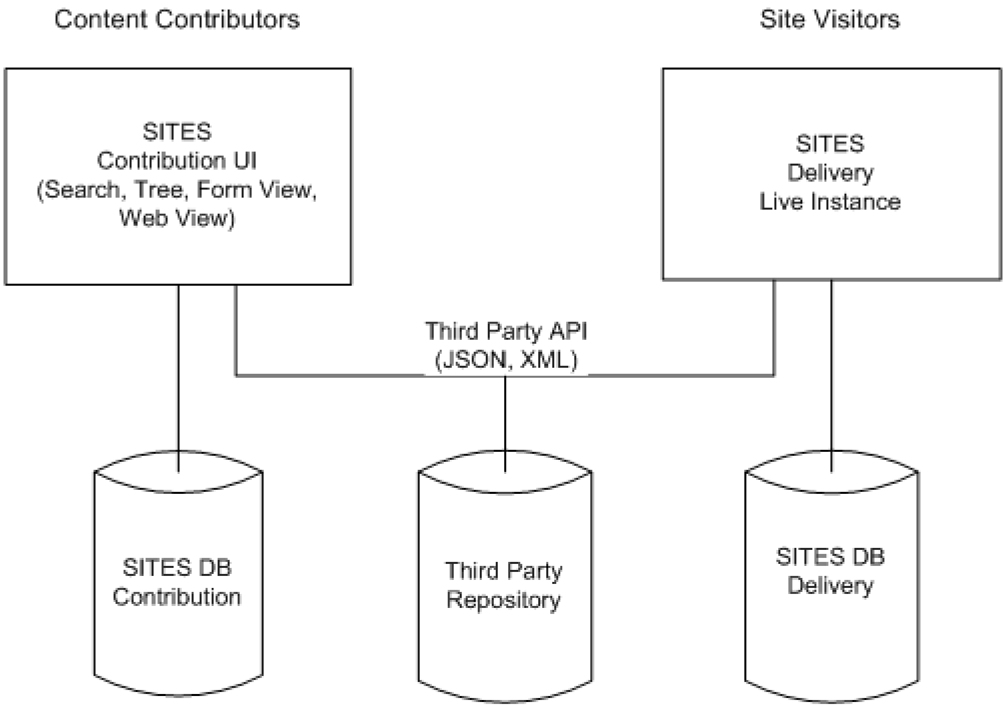
This chapter contains the following sections:

48.1 Introduction

Principle

A proxy asset is an asset representing content stored and managed in a remote location. This is illustrated in Figure 48–1.

Figure 48–1 Proxy Asset Architecture



In this architecture:

- A third-party content repository is assumed, accessible through a public read API from both the editorial and delivery instances.
- The Contributor interface is customized to access the third-party repository in order to make external content visible in the UI.
- A proxy asset is created on the fly for every external content rendered in the UI. A proxy asset does not store any metadata, which is assumed to be accessible through a public read API provided by the third-party service.
- On the live instance, templates written for proxy assets are accessing the same repository and API to render external content on the live site.

Note: Only those proxy assets that are used are permanently stored in the Sites database. For example, proxy assets created while rendering search results are later purged by a dedicated cleaning event.

Contributor Interface

Once a new proxy asset type is registered in the WebCenter Sites database, developers need to provide the appropriate UI customizations before contributors can start interacting with external content.

The nature of the UI customizations being implemented depend on the contributor's specific requirements. For example, it is expected that the Contributor interface search functionality will be hooked to the external repository's own search service in most cases.

Once this is done, and external content are surfaced in the Contributor interface in the form of proxy assets, they behave mostly like standard assets. That is, contributors are able to:

- Search the external content repository, using the same asset search tab, showing results in list or thumbnail view, docked or undocked
- Use drag and drop from search, or tree
- Associate external content to other assets, whether in form view or web view
- Preview external content, using WebCenter Sites templates
- Bookmark, tag, set in workflow, approve and publish

A default inspection screen is provided for all proxy types.

Note: The following restrictions apply:

- The external content lifecycle is assumed to be entirely managed in a third-party UI; that is, WebCenter Sites only needs read access to the external repository. Consequently, UI actions such as editing, versioning, and so forth, are disabled by default for all proxy asset types.
- Proxy assets cannot have associated assets or subtypes.
- The external content repository must be accessible from both the contribution and delivery WebCenter Sites instances.

48.2 User Interface Customizations

To make the use of WebCenter Sites a better experience, the User Interface can be customized many ways. This section details customizing the portions of the UI to sync with third-party software.

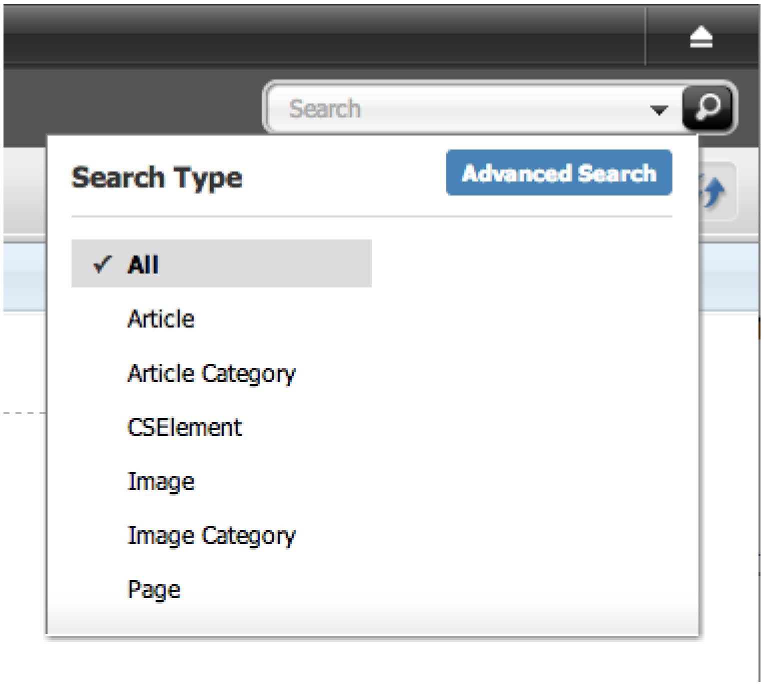
This section contains the following topics:

- Section 48.2.1, "Customizing the Search Start Menu"
- Section 48.2.2, "Customizing the Content Tree"

48.2.1 Customizing the Search Start Menu

In the WebCenter Sites UI, users run searches restricted to a specific asset type by selecting a search start menu, as shown in Figure 48–2.

Figure 48–2 Search Type Start Menu Selection



In order to customize search for a given asset type, override the controller element `UI/Data/Search/Search` by creating the following elements:

```
CustomElements/<AssetType>/UI/Data/Search/SearchAction
CustomElements/<AssetType>/UI/Data/Search/SearchJson
```

For more details about controller elements, and element overrides, refer to Part III, "Customizing the Contributor Interface."

`UI/Data/Search/Search` runs the search code and generates the appropriate JSON data for consumption by the grid widget (whether in list or thumbnail view, docked or undocked).

The JSON data must be a valid dojo datastore. Section 48.3.4, "Customizing Search." Section 48.2.2, "Customizing the Content Tree" gives an actual implementation example.

48.2.2 Customizing the Content Tree

The content tree can be customized by defining a custom tree section, contained in a new or existing tree tab: For more details about defining custom tree tab sections, see the *WebCenter Sites Administrator's Guide*.

A custom tree tab section points to a JSP element generating data based on some rendering logic, eventually consumed by the tree widget. Tree data is generated using the following utility elements:

- `OpenMarket/Gator/UIFramework/BuildTreeNodeId:` generates a tree node ID
- `OpenMarket/Gator/UIFramework/BuildTreeNode`: generates properly formatted tree node data

The following examples show how to build a tree node, whether it represents an asset or not (that is, an asset node compared to an *adhoc* node). For more information, see Chapter 34, "Customizing the WebCenter Sites Admin Interface."

Example 1: Build a Tree Node Representing an Asset

This example shows how to generate a tree node representing a single asset node. The node will execute the `inspect` action when double-clicked (see `executeFunction` parameter), that is, the asset default inspect view will be rendered in a new tab (or focused if the asset is already opened in a tab).

```
<%--
- Generates a tree node id.
- The element expects the asset id and type to be passed in parameters
- called respectively "ID" and "AssetType".
- The generated id is stored in a Sites variable called "TreeNodeID"
--%>
<ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNodeID">
     <ics:argument name="AssetType" value="Article" />
     <ics:argument name="ID" value="1234567890" />
</ics:callelement>

<%--
- Generates a tree node for this asset.
- Note that this element implicitly consumes variables
- currently present in the ICS scope, including "TreeNodeID"
--%>
<ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNode">
     <ics:argument name="Label" value="Node Label, for example asset name or other
                  readable string" />
     <ics:argument name="Description" value="Some optional tooltip text" />
     <ics:argument name="executeFunction" value="inspect" />
</ics:callelement>
```

Example 2: Build an *adhoc* Tree Node

Tree nodes do not necessarily represent assets, in which case they are called *adhoc* nodes. The example below shows how to generate an *adhoc* node representing a parent node, a node which has children and can be expanded and collapsed.

```
<%--
- Generates a tree node id.
- For adhoc nodes, the expected parameter is called "AdHoc",
- and can be any arbitrary string, which must be unique across tree nodes
--%>
<ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNodeID" >
  <ics:argument name="AdHoc" value="SomeUniqueString" />
</ics:callelement>

<%--
- Generates a "LoadURL", that is the URL to be called when a tree node
- is expanded. In this case, we're calling the default utility SiteCatalog entry
- (OpenMarket/Gator/UIFramework/LoadTab) which, in turn, will invoke
- a custom element ("Some/Other/Element" in our example), in charge
- of generating the child nodes, based on some custom logic.
--%>
<satellite:link
       assembler="query"
       pagename="OpenMarket/Gator/UIFramework/LoadTab"
       outstring="LoadURL">
  <satellite:argument name="populate" value="Some/Other/Element"/>
  <satellite:argument name="op" value="load"/>
</satellite:link>

<%--
```

```
- Generates the corresponding tree node data
- Note that this element consumes the "LoadURL" variable previously generated.
- The presence of LoadURL indicates whether a node should be marked as expandable
or not.
--%>
<ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNode">
  <ics:argument name="Label" value="Some meaningful node label" />
</ics:callelement>
```

48.3 Integrating External Content in the WebCenter Sites Contributor Interface

Before contributors can work with external content, a developer has to implement the required UI integration code.

In this section, we'll use a dummy content repository as a case study, and explain the steps to integrate it with:

- **search**: to use the repository search service instead of the standard Sites search
- **content tree**: to allow contributors to browse the repository content in a custom content tree

This section contains the following topics:

48.3.1 Case Study: The Dummy Repository

Setting Up Sample Data

We're using a set of static JSON files deployed directly in the WebCenter Sites web application to emulate a dummy content repository. The dummy content is assumed to be media content (images).

Those JSON files simulate the following services:

- Search the repository for a given term (searching on all, "ski" or "surfing" will return actual results):
  `http://localhost:7001/sites/samples/search/<searchterm>.json`
- Get all content categories ("Ski" and "Surfing"):
  `http://localhost:7001/sites/samples/browse/categories.json`
- Get all dummy content for a given category:
  `http://localhost:7001/sites/samples/browse/<category>.json`
- Get metadata for a given content id:
  `http://localhost:7001/sites/samples/content/<id>.json`

For example, `/samples/search/ski.json` returns the following example content:

```
{
  "items": [
    {
      "id": "1001",
      "title": "Yellow Skier",
      "foo": "bar1",
      "lastModified": "1354735336444",
      "thumbnail": "samples/images/image7_thumbnail.png"
    },
    {
      "id": "1002",
      "title": "Female Skier",
      "foo": "bar2",
      "lastModified": "1354735336444",
      "thumbnail": "samples/images/image8_thumbnail.png"
    },
    {
      "id": "1003",
      "title": "Ski Jump",
      "foo": "bar3",
      "lastModified": "1354735336444",
      "thumbnail": "samples/images/image9_thumbnail.png"
    }
  ]
}
```

Later in this guide an example of JSP element which consumes this type of JSON data is shown.

> **Note:** Sample code is stored in the folder `/misc/Samples/`. This folder is located in WebCenter Sites. Sample code specific to proxy assets can be found in `/misc/Samples/SampleProxy/proxy_sample.zip`.

Retrieving Data from the Dummy Repository

In order to avoid duplication of code, the logic needed to query the external content source is encapsulated in a dedicated element, named `Dummy/GetData`. In this example, data is returned in JSON format. Therefore, this example uses the jersey (`http://jersey.java.net/`) and jettison (`http://jettison.codehaus.org/`) libraries, which are already deployed in the WebCenter Sites web application, in order to retrieve and deserialize incoming JSON data.

The element receives a query, for example, `/search/ski.json`, in an ICS variable named `serviceURL` and returns a `JSONArray` object, stored in the ICS scope using `ics.SetObj(String, Object)`:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ page import="com.sun.jersey.api.client.*" %>
<%@ page import="com.sun.jersey.api.client.config.*" %>
<%@ page import="org.codehaus.jettison.json.*"%>
<cs:ftcs>
<%
//
// Gets data from dummy repository
// This element expects an ICS variable "serviceURL"
//
Client client = Client.create();
//
// Host, port and webapp prefix are hardcoded for simplicity, modify as
// appropriate.
// In a real-world scenario, the base service URL would probably be stored in
// a configuration file.
//
WebResource res = client.resource("http://localhost:7001/sites/samples" +
                  ics.GetVar("serviceURL"));
ClientResponse resp = res.accept("application/json").get(ClientResponse.class);
JSONArray list;
if (resp.getStatus() != 200) {
         list = new JSONArray(); // empty array
}
else {
       JSONObject json = new JSONObject(resp.getEntity(String.class));
       list = json.getJSONArray("items");
}

// store the object in the ICS scope
ics.SetObj("items", list);
%>
</cs:ftcs>
```

48.3.2 Registering a New Proxy Asset Type

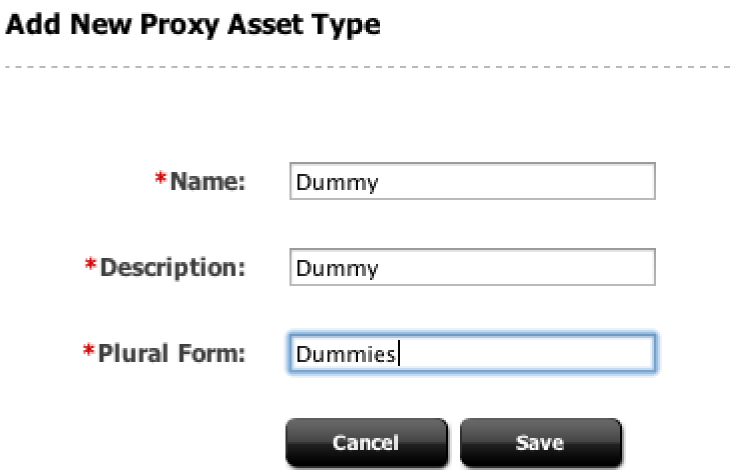
In order to represent our dummy content in the WebCenter Sites repository, we need to define a new proxy asset type.

To create a proxy asset type, complete the following steps:

1. In the **Admin** tab, expand **Proxy Asset Manager**. Double-click **Add New**.

   The Add New Proxy Asset Type page is displayed.

Figure 48–3 Add New Proxy Asset Type Page



Add New Proxy Asset Type

*Name: Dummy

*Description: Dummy

*Plural Form: Dummies

2. In the **Name** field, enter a name for the proxy asset type. Similarly, add a description in the **Description** field, and a plural form of the name in the **Plural Form** field.

Note: Creating or editing proxy assets through the Contributor interface is not available. Consequently, in this step, only a Search start menu should be enabled.

In this example, and to use with the examples continuing through the rest of the document, enter `Dummy` in the **Name** field, `Dummy` in the **Description** field, and `Dummies` in the **Plural Form** field.

3. Click **Save**.

Note: Proxy Asset Maker will register the new asset type and create a single table with the same name. A proxy asset table has only a subset of standard asset metadata, and defines only one specific column: `externalid`. This field is meant to store the identifier of the external content in the external repository.

48.3.3 Implementing UI Integration Code

Assets can appear in many places in the Contributor interface. Some examples of these places include:

- Asset forms, for fields including asset references
- Search results
- Content tree panel

In each case, the actual integration code will vary based on the requirements and available customization hooks. However, in all cases, the following principle must be followed: All external content presented in the Contributor interface must be registered as a proxy asset.

In practice, this means creating (or reusing) a proxy asset which `externalid` field refers to the content identifier in the external content source. This can be done using the usual Asset API classes and methods. However, for simplicity, the proxy JSP tag library is provided.

It contains utility tags to register a given external content as a proxy asset type:

- `<proxy:register />`

It also contains utility tags to generate a JSON datastore for data grid widgets (such as search):

- `<proxy:createstore />`: initializes a new store
- `<proxy:addstoreitem />`: adds an item to a given store
- `<proxy:tojson />`: serializes a store to JSON

More details are provided in the *Oracle Fusion Middleware WebCenter Sites Tag Reference*, and in the code examples in Section 48.3.4, "Customizing Search."

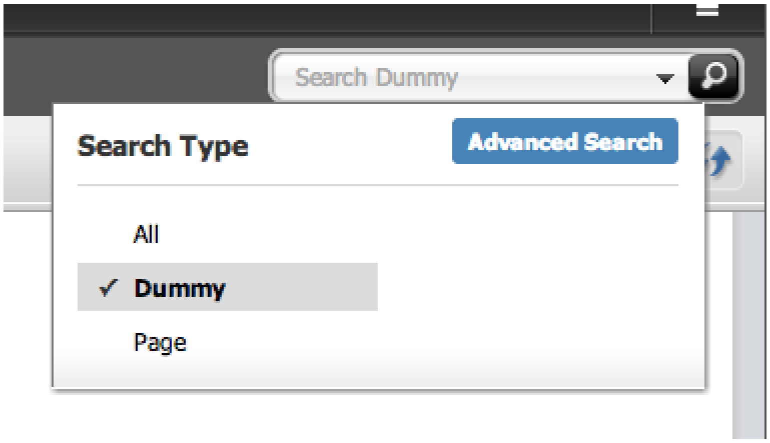
48.3.4 Customizing Search

As explained in Section 48.2.1, "Customizing the Search Start Menu," an override of the controller element `UI/Data/Search/Search` for the Dummy asset type by creating the following elements:

```
CustomElements/Dummy/UI/Data/Search/SearchAction
```

```
CustomElements/Dummy/UI/Data/Search/SearchJson
```

Figure 48–4 Search Drop-Down Showing Dummy Asset



As illustrated, Dummy appears in the search drop-down. Selecting **Dummy** (that is, the name of the proxy asset type already created) will run the custom search code, instead of the Lucene-based search.

To implement a full custom search, complete the remaining topics in this section.

This section contains the following topics:

- Section 48.3.4.1, "Get Search Results Using the Provided Third-Party API"
- Section 48.3.4.2, "Turn Search Results into Proxy Assets, Filter Incoming Search Results, Register External Content, and Gather Data for Search Grid Widget"
- Section 48.3.4.3, "Build a Data Store for the Grid Widget"
- Section 48.3.4.4, "Testing Custom Search"
- Section 48.3.4.5, "Additional Customizations"

48.3.4.1 Get Search Results Using the Provided Third-Party API

In `CustomElements/Dummy/UI/Data/Search/SearchAction`, JSON data is retrieved using the element written in Section 48.3.1, "Case Study: The Dummy Repository."

This code example is used in the retrieval:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="proxy" uri="futuretense_cs/proxy.tld"%>
<%@ page import="org.codehaus.jettison.json.*"%>
<%@ page import="COM.FutureTense.Interfaces.Utilities"%>
<cs:ftcs>

<%
// in this dummy example, only the empty search string, 'ski' or 'surfing' will
// return results String searchTerm = ics.GetVar("searchText");
if (!Utilities.goodString(searchTerm)) searchTerm = "all";
%>
<ics:setvar name="serviceURL" value='<%="/search/" + searchTerm + ".json" %>' />
<ics:callelement element="Dummy/GetData" />
<%
JSONArray list = (JSONArray)ics.GetObj("items");
```

```
//
// ...to be continued in the next section
//
%>

</cs:ftcs>
```

At this point, `list` contains the JSON data received from the external content source.

48.3.4.2 Turn Search Results into Proxy Assets, Filter Incoming Search Results, Register External Content, and Gather Data for Search Grid Widget

This step builds the code through a series of steps.

First, a new data store is built:

```
<proxy:createstore store="<storeName>" />
```

where <*storeName*> is an arbitrary string designating the data store.

Then, for each incoming external content asset, the current asset is registered as a proxy asset:

```
<proxy:register
       externalid="<external_content_identifier>"
       type="<proxy_asset_type>"
       name="<proxy_asset_name>"
       varname="<variable_name>" />
```

where:

- *<external_content_identifier>* is the identifier of the current external content item in the third-party repository. In the case of this example dummy repository, this identifier is returned in the `id` field of incoming JSON data.
- *<proxy_asset_type>* is the proxy asset type name. In this example, "Dummy."
- *<proxy_asset_name>* is the readable string to be used as name throughout the Contributor interface. In this example, the title field returned in the incoming JSON data.
- *<variable_name>* is the name of the WebCenter Sites variables which will be contain the proxy asset id corresponding to the current external content item.

Then, for each incoming registered proxy asset, add the asset to the data store:

```
<proxy:addstoreitem
       store="<storeName>"
       id="<proxy_asset_id>"
       type="<proxy_asset_type>" />
```

where:

- *<storeName>* is the data store, as defined by `<proxy:createstore />`.
- *<id>* is the proxy asset identifier.
- *<type>* is the proxy asset type.

The full code for the Dummy proxy asset type is then:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="proxy" uri="futuretense_cs/proxy.tld"%>
<%@ page import="org.codehaus.jettison.json.*"%>
<%@ page import="COM.FutureTense.Interfaces.Utilities"%>
<cs:ftcs>

<%
// in this dummy example, only the empty search string, 'ski' or 'surfing' will
// return results
String searchTerm = ics.GetVar("searchText");
if (!Utilities.goodString(searchTerm)) searchTerm = "all";
%>
<ics:setvar name="serviceURL" value='<%="/search/" + searchTerm + ".json" %>' />
<ics:callelement element="Dummy/GetData" />
<%
JSONArray list = (JSONArray)ics.GetObj("items");
%>

<%-- create a new data store --%>
<proxy:createstore store="assets" />
<%
// go through each incoming item
for (int i = 0; i < list.length(); i++) {
     JSONObject item = (JSONObject)list.get(i);%>

     <%-- Register the current external content item as a proxy asset --%>
     <proxy:register externalid='<%=item.getString("id") %>'
     type="Dummy"
     name='<%=item.getString("title") %>'
     varname="internalid" />

<%-- Add the proxy asset to the datastore --%>
<proxy:addstoreitem store="assets"
                    id='<%=ics.GetVar("internalid") %>'
                    type="Dummy" />
<%
}
// put store name in request scope
request.setAttribute("store", "assets");
request.setAttribute("total", Integer.valueOf(list.length()));
%>
</cs:ftcs>
```

48.3.4.3 Build a Data Store for the Grid Widget

This step consists in rendering the JSON to be sent back to the UI widget. This step is done inside `CustomElements/Dummy/UI/Data/Search/SearchJson`.

The `proxy:tojson` utility tag does this, as shown in the following example:

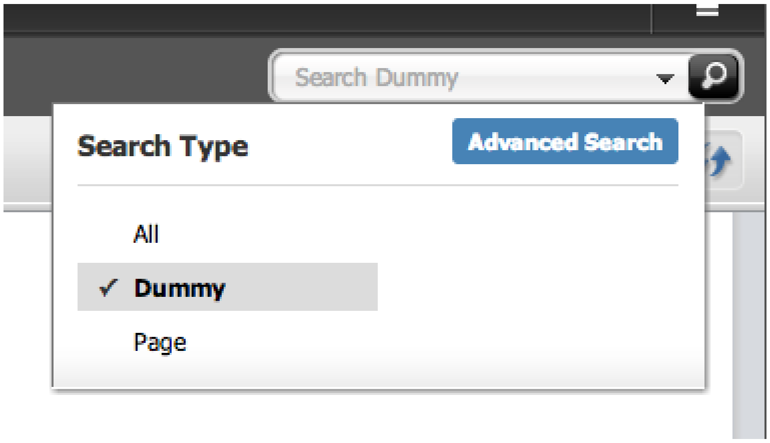
```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="proxy" uri="futuretense_cs/proxy.tld"%>
<cs:ftcs>
<proxy:tojson store="${store}" total="${total}" />
</cs:ftcs>
```

48.3.4.4 Testing Custom Search

Once the JSON is scripted, test the search function to ensure everything works properly. To test the search, complete the following steps:

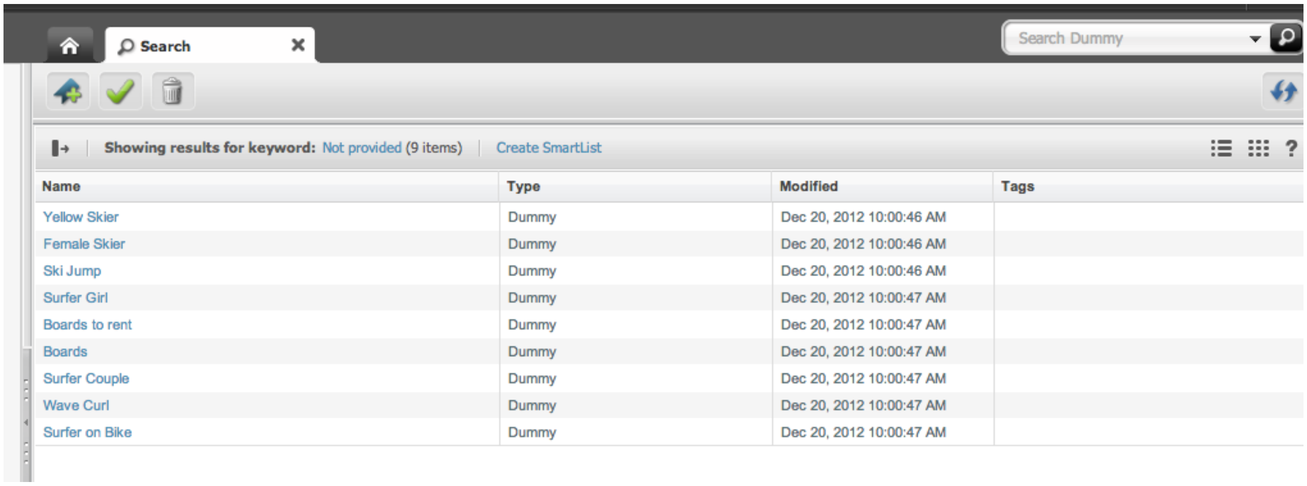
1. In the Contributor interface, from the search box select **Dummy** as the search type. Click the search icon.

Figure 48–5 Search Drop-Down Showing Dummy Asset



2. The **Search** tab will open displaying the associated data.

Figure 48–6 Search Results

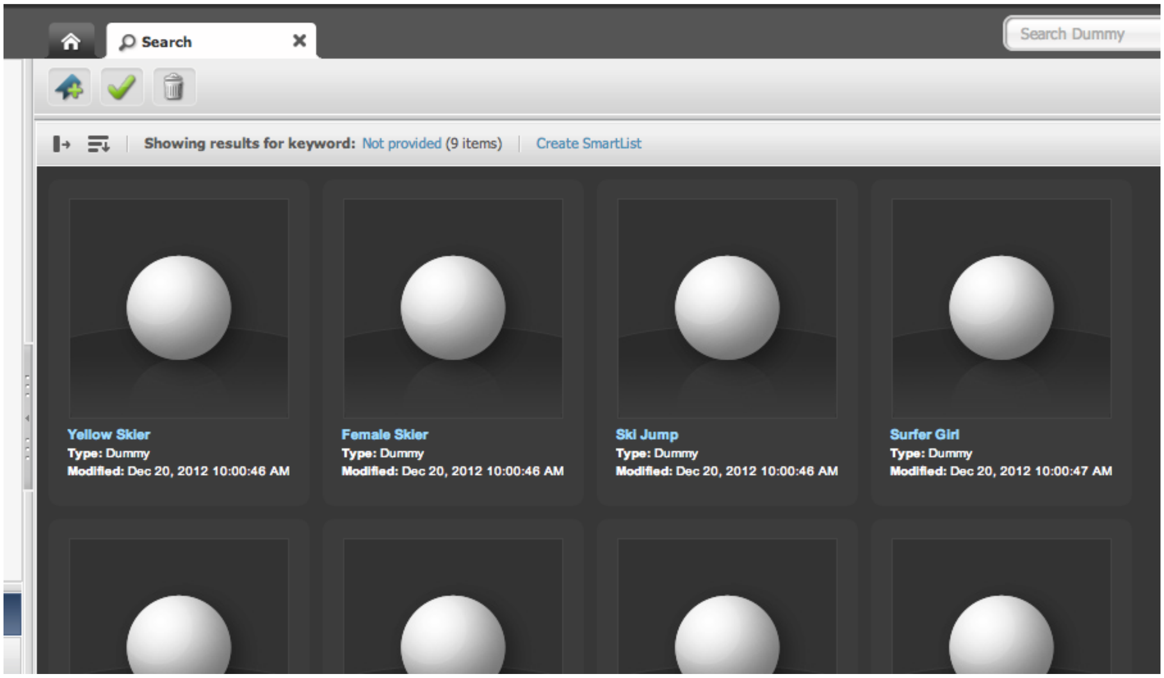


Showing results for keyword: Not provided (9 items) | Create SmartList

Name	Type	Modified	Tags
Yellow Skier	Dummy	Dec 20, 2012 10:00:46 AM	
Female Skier	Dummy	Dec 20, 2012 10:00:46 AM	
Ski Jump	Dummy	Dec 20, 2012 10:00:46 AM	
Surfer Girl	Dummy	Dec 20, 2012 10:00:47 AM	
Boards to rent	Dummy	Dec 20, 2012 10:00:47 AM	
Boards	Dummy	Dec 20, 2012 10:00:47 AM	
Surfer Couple	Dummy	Dec 20, 2012 10:00:47 AM	
Wave Curl	Dummy	Dec 20, 2012 10:00:47 AM	
Surfer on Bike	Dummy	Dec 20, 2012 10:00:47 AM	

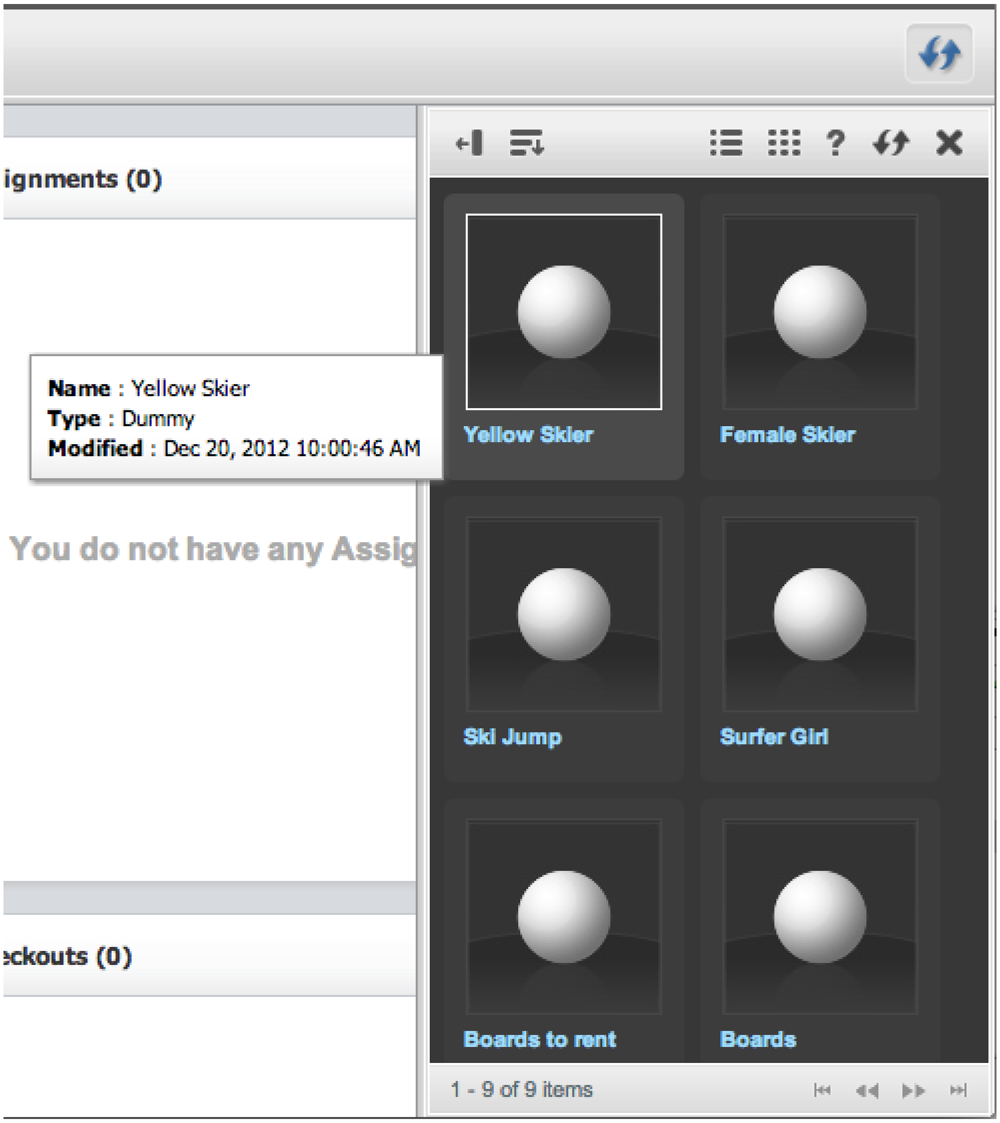
3. Switching to thumbnail view shows the following:

Figure 48–7 Search Results in Thumbnail View


Search
Search Dummy
Showing results for keyword: Not provided (9 items) | Create SmartList
Yellow Skier
Type: Dummy
Modified: Dec 20, 2012 10:00:46 AM
Female Skier
Type: Dummy
Modified: Dec 20, 2012 10:00:46 AM
Ski Jump
Type: Dummy
Modified: Dec 20, 2012 10:00:46 AM
Surfer Girl
Type: Dummy
Modified: Dec 20, 2012 10:00:47 AM

The search tab is also functional in docked mode. Note that the tooltip is filled in with default data:

Figure 48–8 Search Results in Docked Panel



48.3.4.5 Additional Customizations

This section deals with customizations that can be implemented beyond what has already been covered. It is not necessary to use any of these customizations to have implemented a custom search.

Rendering a Thumbnail

In the example, the Dummy repository returns a URL to a thumbnail image for each content item.

We will modify it to retrieve the thumbnail URL (sent by our Dummy repository) in order to render it. For that to happen, we must customize the grid in thumbnail view (whether docked and undocked), that is, overriding the configuration elements for:

```
UI/Layout/CenterPane/Search/View/ThumbnailView
UI/Layout/CenterPane/Search/View/DockedThumbnailView
```

Configuration files for each must be created:

```
CustomElements/Dummy/UI/Layout/CenterPane/Search/View/ThumbnailViewConfig
CustomElements/Dummy/UI/Layout/CenterPane/Search/View/DockedThumbnailViewConfig
```

The `ThumbnailViewConfig` configuration file will have the following XML configuration:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="xlat" uri="futuretense_cs/xlat.tld" %>
<cs:ftcs>
  <thumbnailviewconfig>
    <formatter>fw.ui.GridFormatter.simpleThumbnailFormatter</formatter>
  </thumbnailviewconfig>
</cs:ftcs>
```

You can see that this overrides the formatter setting for the Dummy thumbnail view (any other setting contained in the global configuration element is maintained). This formatter renders a default view for each search result showing a thumbnail image, and the name field ("inspect" link).

Similarly, docked thumbnail view settings should be overwritten by creating `DockedThumbnailViewConfig` in `CustomElements/Dummy/UI/Layout/CenterPane/Search/View/` with the following XML:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<cs:ftcs>
<ics:callelement element=
"[CustomElements/Dummy/UI/Layout/CenterPane/Search/View/ThumbnailViewConfig]" />
</cs:ftcs>
```

Note: For more details about search configuration, refer to Part III, "Customizing the Contributor Interface."

The brackets around the element name indicates that the `ics:callelement` tag should not search for a customized version of the given element name.

Configuring the Grid Context Menu

Not all asset operations apply to proxy assets. For this reason, we need to override the default grid context menu setup. Override the element `UI/Config/GlobalHtml` by creating a new element named `MyConfig` in `UI/Config`. For more information on creating MyConfig, see Section 65.5.3, "Customizing Context Menus."

This ensures that none of the global settings will be merged with the overridden settings.

Sort Fields

Depending on the external content source API capabilities, sorting might not be available for all fields. In this case, the sort drop-down menu can be customized by overriding the default context menu configuration file. This is handled by creating the configuration file:

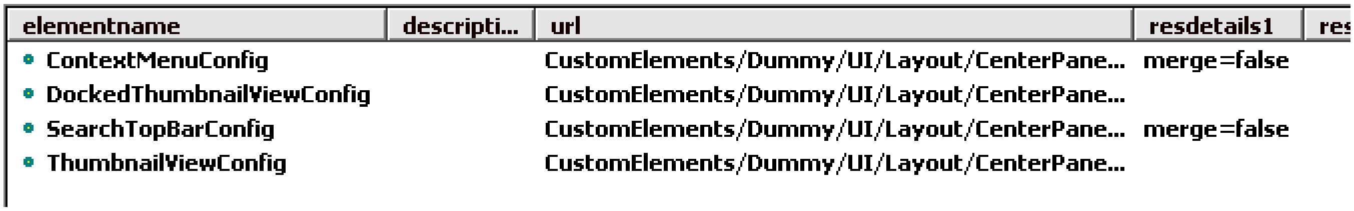
```
CustomElements/Dummy/UI/Layout/CenterPane/Search/View/SearchTopBarConfig
```

and use the following XML code (in this case, assuming that only sorting by name is available):

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="xlat" uri="futuretense_cs/xlat.tld"%>
<cs:ftcs>
<sortconfig>
  <sortfields>
    <sortfield id="name_asc">
      <fieldname>name</fieldname>
      <displayname>
          <xlat:stream key="UI/UC1/Layout/NameSort1" escape="true"/>
      </displayname>
      <sortorder>ascending</sortorder>
    </sortfield>
    <sortfield id="name_desc">
      <fieldname>name</fieldname>
      <displayname>
          <xlat:stream key="UI/UC1/Layout/NameSort2" escape="true"/>
      </displayname>
      <sortorder>descending</sortorder>
    </sortfield>
  </sortfields>
</sortconfig>
</cs:ftcs>
```

Be aware that both `SearchTopBarConfig` and `ContextMenuConfig` elements must have `merge=false` set in the element `resdetails1` (or `resdetails2`) field, as shown in Figure 48–9.

Figure 48–9 Configuration File Element Details Value Field

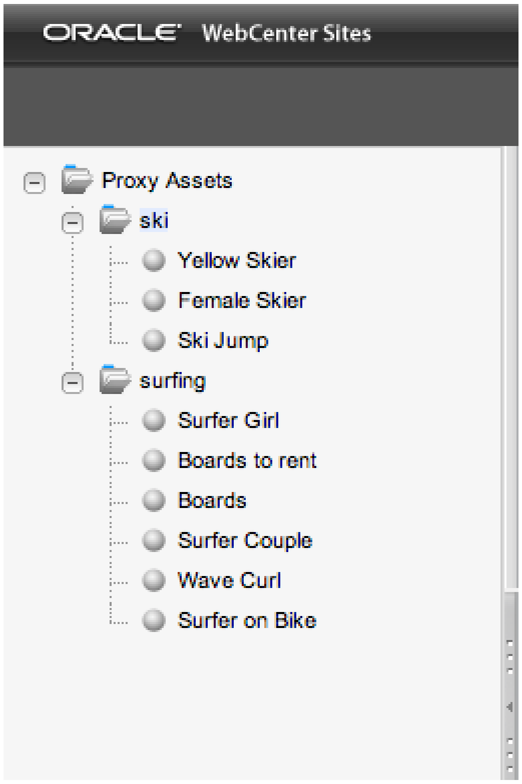


elementname	descripti...	url	resdetails1	res
ContextMenuConfig		CustomElements/Dummy/UI/Layout/CenterPane...	merge=false	
DockedThumbnailViewConfig		CustomElements/Dummy/UI/Layout/CenterPane...		
SearchTopBarConfig		CustomElements/Dummy/UI/Layout/CenterPane...	merge=false	
ThumbnailViewConfig		CustomElements/Dummy/UI/Layout/CenterPane...		

48.3.5 Implementing a Custom Tree

A custom tree is useful to allow users to browse external content by category, as shown in Figure 48–10.

Figure 48–10 Custom Tree



Two elements are created to render the tree:

- `Dummy/Tree/Root`: renders the tree root nodes, that is, the content categories.
- `Dummy/Tree/Load`: renders all content under a given category.

To implement the custom tree, you must first register the custom tree tab, and then implement the custom tree code.

This section contains the following topics:

- Section 48.3.5.1, "Registering the Custom Tree Tab"
- Section 48.3.5.2, "Implementing the Tree Code"

48.3.5.1 Registering the Custom Tree Tab

The custom tree tab is defined in the Add New Tree Tab page. With the tab created in this example, a new content tree called Proxy Assets will be created, containing a single custom section (Dummy) pointing at `Dummy/Tree/Root`.

To register the custom tree tab for this example, complete the following:

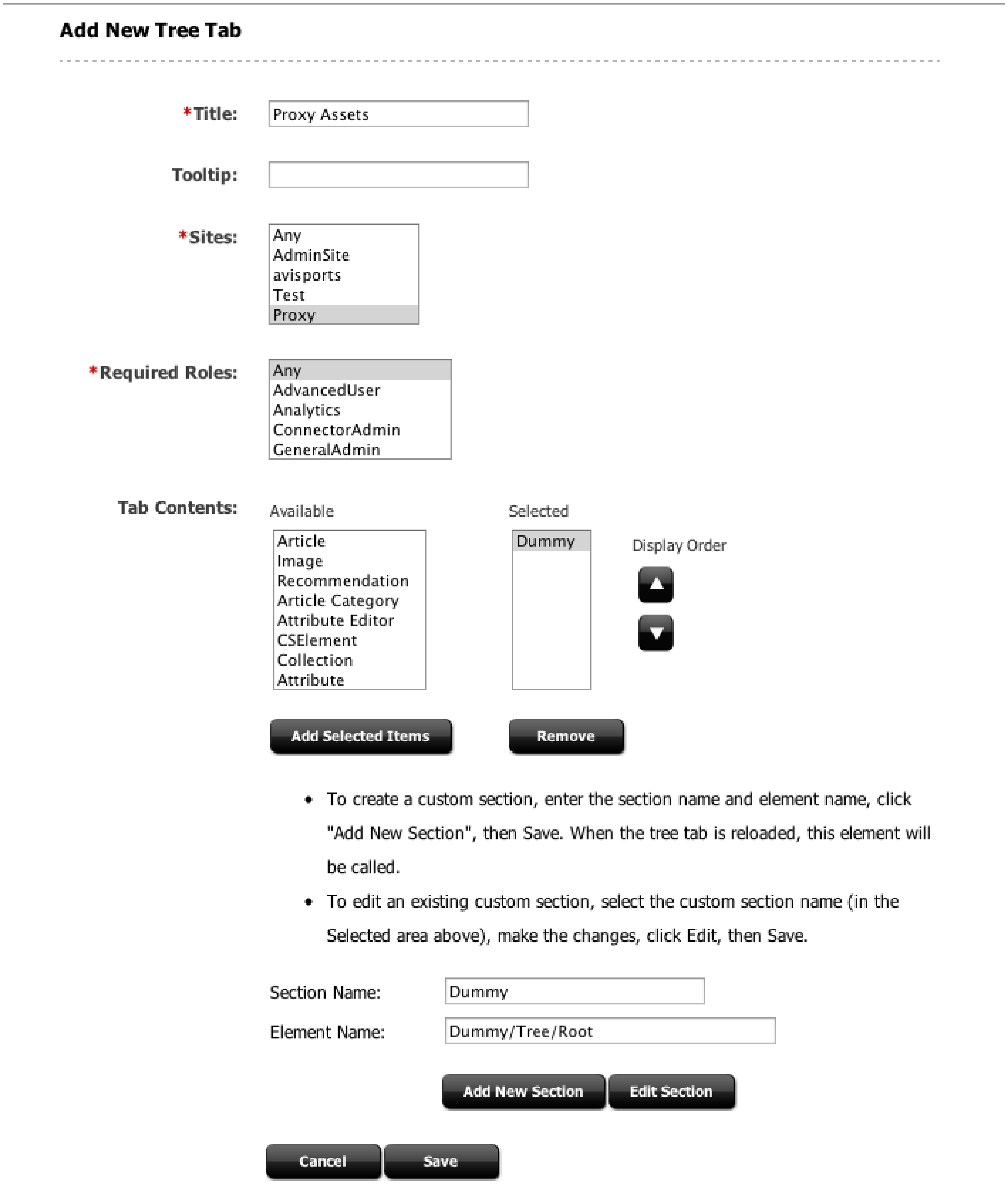
1. In the Admin interface, select the **Admin** tab, then double-click **Tree**.

   The Tree Tabs page opens.

2. At the bottom of the Tree Tabs page, click **Add New Tree Tab**.

   The Add New Tree Tab page opens (Figure 48–11).

Figure 48–11 Add New Tree Tab Page



Add New Tree Tab

*Title: Proxy Assets

Tooltip:

*Sites: Any, AdminSite, avisports, Test, Proxy (selected: Proxy)

*Required Roles: Any, AdvancedUser, Analytics, ConnectorAdmin, GeneralAdmin (selected: Any)

Tab Contents:

Available: Article, Image, Recommendation, Article Category, Attribute Editor, CSElement, Collection, Attribute

Selected: Dummy

- To create a custom section, enter the section name and element name, click "Add New Section", then Save. When the tree tab is reloaded, this element will be called.
- To edit an existing custom section, select the custom section name (in the Selected area above), make the changes, click Edit, then Save.

Section Name: Dummy

Element Name: Dummy/Tree/Root

3. Fill in the fields as needed. For this specific example, fill in the fields in this way:
   - **Title**: Enter `Proxy Assets` for the title of the tab.
   - **Sites**: Select **Proxy** from the list.
   - **Required Roles**: Select **Any** from the list.
   - **Tab Contents**: Select **Dummy** and click **Add Selected Items** to move it to the Selected column. **Dummy** should be the only item in the selected column.

- **Section Name**: Enter `Dummy` in the field.
- **Element Name**: Enter `Dummy/Tree/Root` in the field.

4. Once the fields have been properly entered, click **Save** to save the asset.

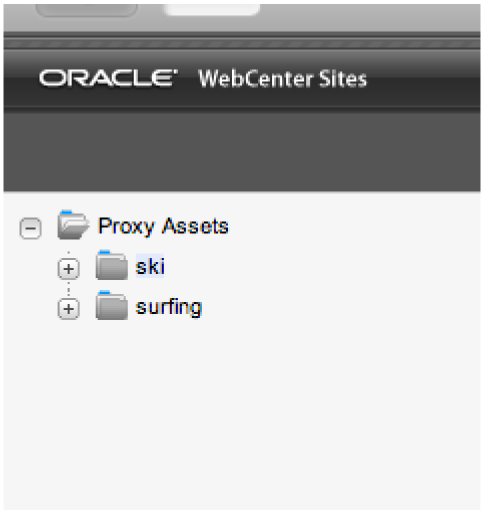
48.3.5.2 Implementing the Tree Code

In `Dummy/Tree/Root`, the external repository is queried to get the tree root nodes., In this case, the list of categories. Tree node data is then generated for each category:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ page import="org.codehaus.jettison.json.*"%>
<cs:ftcs>

<%-- Retrieve all categories from dummy repository --%>
<ics:setvar name="serviceURL" value="/browse/categories.json" />
<ics:callelement element="Dummy/GetData" />

<%
JSONArray list = (JSONArray)ics.GetObj("items");
for (int i = 0; i < list.length(); i++) {
     String category = (String)list.get(i);
     // 1) build a tree node id - needs to be unique
     // Note: we need to make sure that AssetType is not present in the scope
     // (BuildTreeNodeID is otherwise assuming the tree node to be an asset node)
     %>
     <ics:removevar name="AssetType" />
     <ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNodeID" >
          <ics:argument name="AdHoc" value='<%=category %>' />
     </ics:callelement>
     <%
     // 2) build the 'LoadURL' i.e. the URL to call to load this node's children
     // OpenMarket/Gator/UIFramework/LoadTab is a standard element.
     // Required input is:
     // - populate: the element rendering tree nodes to execute
     // - op: must be set to 'load'
     //
     // We're also passing 'category' which is required by our custom logic in
     // Dummy/Tree/Load
     //
     %>
     <satellite:link assembler="query"
                     pagename="OpenMarket/Gator/UIFramework/LoadTab"
                     outstring="LoadURL">
     <satellite:argument name="populate" value="Dummy/Tree/Load"/>
     <satellite:argument name="op" value="load"/>
     <satellite:argument name="category" value='<%=category %>' />
     </satellite:link>

     <ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNode">
          <ics:argument name="Label" value='<%=category %>' />
     </ics:callelement>
<%}%>
</cs:ftcs>
```

This code should generate the following tree:

Figure 48–12 Generated Custom Tree



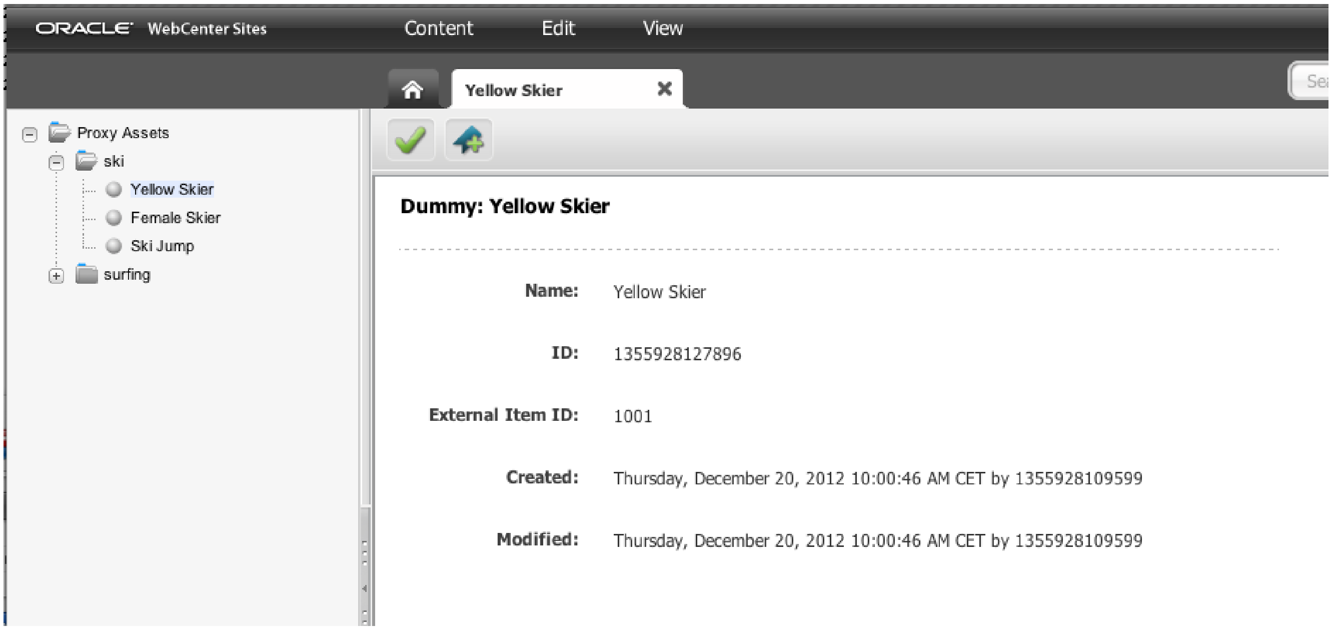
Note: The elements `OpenMarket/Gator/UIFramework/BuildTreeNodeID` and `OpenMarket/Gator/UIFramework/BuildTreeNode` both consume variables in the ICS scope. Since the `<ics:callelement />` tag has a global scope, it is, in some cases, necessary to explicitly remove (permanently or temporarily) certain variables from the ICS scope, using `<ics:removevar />`.

In `Dummy/Tree/Load`, we query the external service to retrieve all content in a given category, then render a node for each content item:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="proxy" uri="futuretense_cs/proxy.tld"%>
<%@ page import="org.codehaus.jettison.json.*"%>
<cs:ftcs>

<%-- Retrieve all content for a given category --%>
<ics:setvar name="serviceURL" value='<%="/browse/" + ics.GetVar("category") +
   ".json"%>' />
<ics:callelement element="Dummy/GetData" />

<%
JSONArray list = (JSONArray)ics.GetObj("items");

for (int i = 0; i < list.length(); i++) {
     JSONObject item = (JSONObject)list.get(i);%>

     <proxy:register externalid='<%=item.getString("id") %>'
                     type="Dummy"
                     name='<%=item.getString("title") %>'
                     varname="internalid" />

     <ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNodeID">
        <ics:argument name="AssetType" value="Dummy" />
        <ics:argument name="ID" value='<%=ics.GetVar("internalid") %>' />
     </ics:callelement>
     <ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNode">
        <ics:argument name="Label" value='<%=item.getString("title") %>' />
        <ics:argument name="Description" value='<%=item.getString("title") %>' />
        <ics:argument name="executeFunction" value="inspect" />
     </ics:callelement>
     <%
}
%>
</cs:ftcs>
```

You should then be able to browse each category. You should then be able to browse each category, as shown in Figure 48–13.

Figure 48–13 Browsing Custom Tree Objects



Dummy: Yellow Skier

Name: Yellow Skier

ID: 1355928127896

External Item ID: 1001

Created: Thursday, December 20, 2012 10:00:46 AM CET by 1355928109599

Modified: Thursday, December 20, 2012 10:00:46 AM CET by 1355928109599

Double-clicking a node will open the default proxy asset inspection page for that node.

48.4 Embedding Proxy Assets in Web Pages

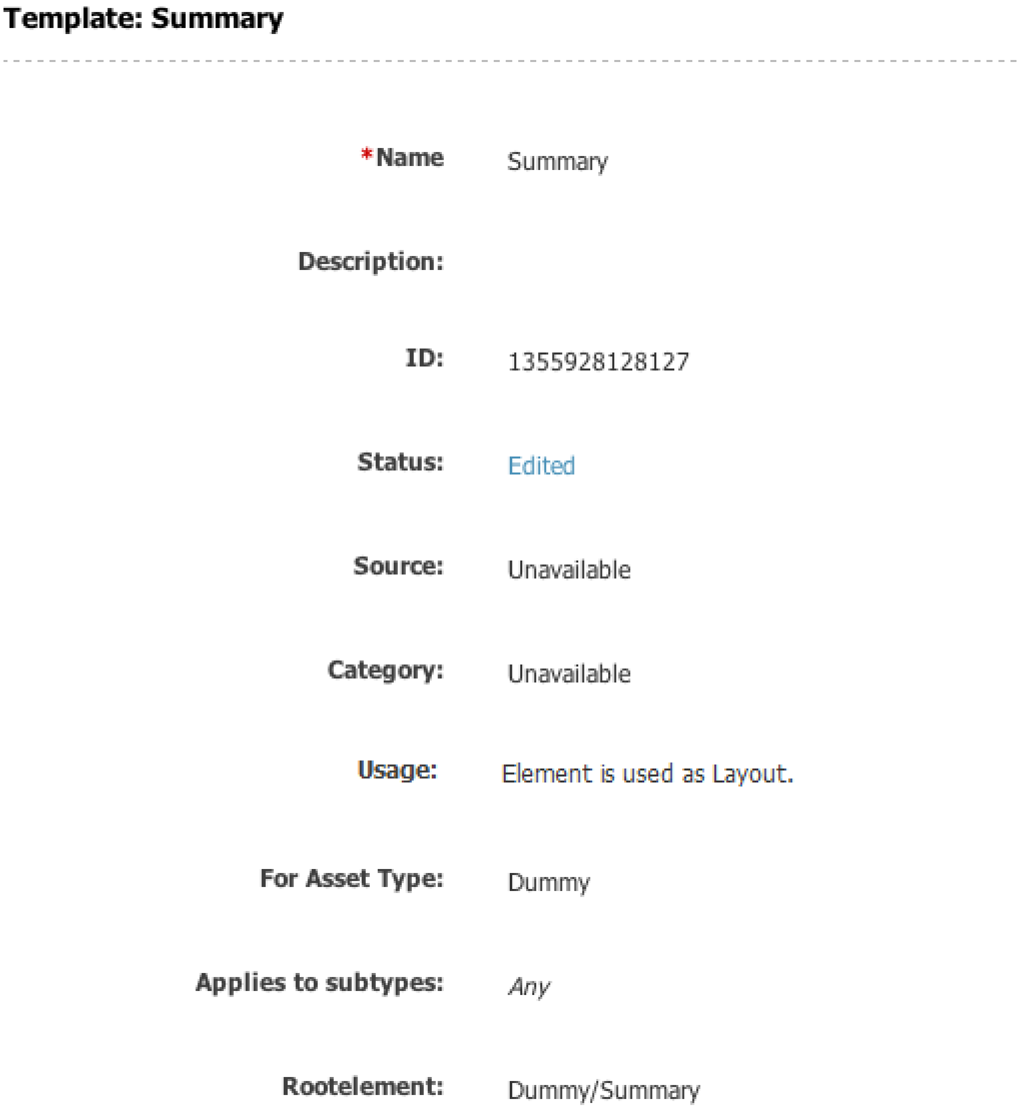
This section contains the following topics:

- Section 48.4.1, "Writing Template for Proxy Assets"
- Section 48.4.2, "Using Proxy Assets in Slots"
- Section 48.4.3, "Caching Proxy Assets"

48.4.1 Writing Template for Proxy Assets

Templates can be defined for proxy assets, just as templates can be defined for any other asset type. A Dummy asset, as has been used in the preceding examples, can have a template defined in the manner described in Figure 48–14.

Figure 48–14 Defined Summary Template



Template: Summary

*Name	Summary
Description:	
ID:	1355928128127
Status:	Edited
Source:	Unavailable
Category:	Unavailable
Usage:	Element is used as Layout.
For Asset Type:	Dummy
Applies to subtypes:	*Any*
Rootelement:	Dummy/Summary

For all Templates, the usage should be listed as **Element is used as Layout**. If **Element is used within an HTML Page** is selected, the Contributor UI will not display the created templates while creating a Page asset.

Writing templates for proxy assets usually involves two distinct actions. First, given an (internal) asset ID, an external ID is retrieved. Second, the third-party repository API is used to retrieve content metadata.

For the specific example of the Dummy asset, the following code would be implemented:

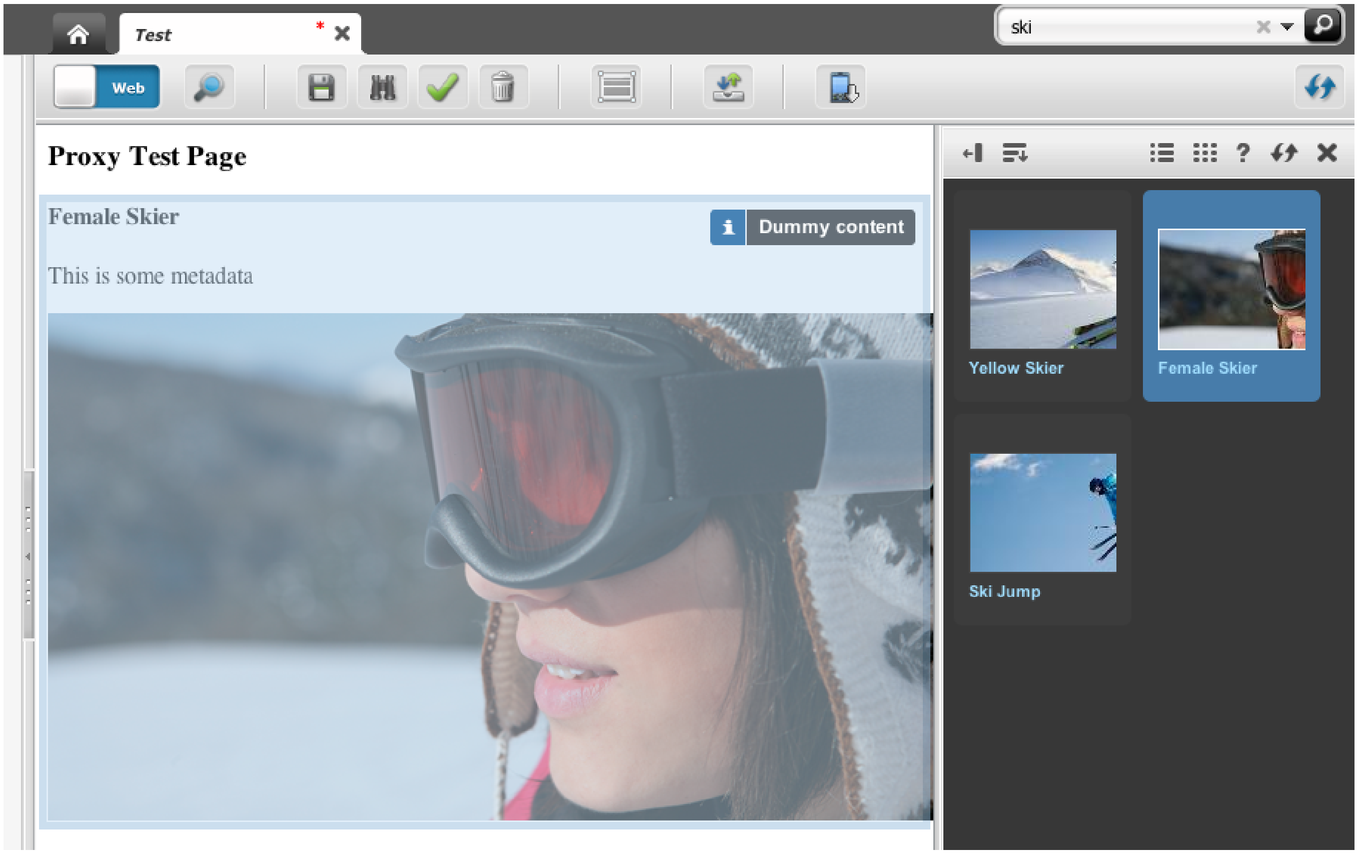
```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ page import="com.sun.jersey.api.client.*"%>
<%@ page import="com.sun.jersey.api.client.config.*"%>
<%@ page import="org.codehaus.jettison.json.*"%>
<cs:ftcs>
<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
```

```
<%-- first, retrieve the external id --%>
<asset:load name="asset" type='<%=ics.GetVar("c") %>'
       objectid='<%=ics.GetVar("cid") %>' />
<asset:get name="asset" field="name" />
<asset:get name="asset" field="externalid" />

<%
// given the external content id, invoke the  third-party repository API
// to retrieve content metadata
Client client = Client.create();
WebResource res = client.resource("http://localhost:7001/sites/samples/content/" +
     ics.GetVar("externalid") + ".json");
ClientResponse resp = res.accept("application/json").get(ClientResponse.class);
JSONObject data = new JSONObject(resp.getEntity(String.class));
%>
<%-- finally render content metadata --%>
<h4><%=data.getString("title") %></h4>
<p><%=data.getString("foo") %></p>
<img src="<%=data.getString("image") %>" alt="<%=data.getString("title") %>" />
</cs:ftcs>
```

48.4.2 Using Proxy Assets in Slots

Proxy assets can be embedded inside pages using the exact same tags and principles which apply to standard assets. This is illustrated in Figure 48–15:

Figure 48–15 Example of an Embedded Proxy Asset



In the code sample below, we assume that a Page asset has a page attribute called `dummyAttribute`, accepting `Dummy` assets:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<cs:ftcs><render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
<html>
<head>
     <title>Proxy test page</title>
</head>
<body>
     <assetset:setasset name="page" id='<%=ics.GetVar("cid") %>'
                       type='<%=ics.GetVar("c") %>' />
     <assetset:getattributevalues attribute="dummyAttribute" listvarname="dummy"
                       name="page" typename="PageAttribute" />
     <ics:listget listname="dummy" fieldname="value" output="id" />
     <h3>Proxy Test Page</h3>
     <insite:calltemplate field="dummyAttribute" c='Dummy'
                       cid='<%=ics.GetVar("id") %>' tname="Summary" />
</body>
</html>
</cs:ftcs>
```

48.4.3 Caching Proxy Assets

Because the rendered content is stored and managed in a separate repository, WebCenter Sites has no way to know when any content is modified, or even if content is modified.

Because of this, it is recommended to set cache expiration to a limited period of time on templates rendering proxy assets (such as the Summary template defined in Section 48.4.1, "Writing Template for Proxy Assets."

49

RealTime Publishing Customization Hooks

This chapter provides an overview of RealTime Publishing and includes information on how to write your own transporter, implementation details, helper methods, example transporter implementation, full code listing, edge-case scenarios, and information on intercepting asset publishing events on the management instance.

This chapter contains the following sections:

- Section 49.1, "Overview of RealTime Publishing"
- Section 49.2, "Writing a Custom Transporter"

49.1 Overview of RealTime Publishing

RealTime Publishing is a pipeline consisting of several jobs. Some jobs run on the management instance while others on the target instance. Following is a brief description of each:

- **Gatherer**: Creates the list of publishable assets and decorates it with additional resources (asset types, table rows) that together make up the canonical set of data to be published.
- **Packager**: Given the resource listing assembled by Gatherer, Packager creates serialized renditions of each resource and save it in the local `fw_PubDataStore` table.
- **Transport**: Takes the serialized data in `fw_PubDataStore` created by Packager and copy it to the target-side `fw_PubDataStore` table.
- **Unpacker**: Takes the serialized data in the target-side `fw_PubDataStore` table and deserialize/save it to the target database.
- **CacheUpdater**: Given the list of assets that were successfully saved by Unpacker, CacheUpdater flushes and optionally regenerates relevant parts of the page caches.

RealTime Publishing uses asynchronous messaging to track the status of each job. It is not necessary to know the details of the messaging framework, but note that communication with the target system is facilitated through the Transporter. This also includes messages issued by Unpacker to inform the management system that an asset has been saved, prompting the management logic to mark that asset published.

49.2 Writing a Custom Transporter

Transporter offers several customization options, including:

- Replacing the OOTB (out-of-the-box or ready to use) transport based on HTTP(s) with one using a different protocol.
- Publishing to multiple targets within the same publishing session.

This section contains the following topics:

- Section 49.2.1, "To Write Your Own Transporter"
- Section 49.2.2, "Implementation Details"
- Section 49.2.3, "Helper Methods"
- Section 49.2.4, "Example of a Transporter Implementation"
- Section 49.2.5, "Full Code Listing"
- Section 49.2.6, "Edge-Case Scenarios"
- Section 49.2.7, "Intercepting Asset Publishing Events on the Management Instance"
- Section 49.2.8, "Finishing Touches"

49.2.1 To Write Your Own Transporter

Follow these steps to write your own transporter:

1. Subclass the `com.fatwire.realtime.AbstractTransporter` class.
2. Override the methods `ping`, `sendBatch`, `listTransports`, `toString`, and `remoteExecute`.
3. Install the transporter by editing the `classes/AdvPub.xml` file on the management side.

   replace the line:

   ```
   <bean id="DataTransporter"
     class="com.fatwire.realtime.MirrorBasedTransporterImpl"
     singleton="false">
   ```

   with:

   ```
   <bean id="DataTransporter"
     class="[your transporter class]"
     singleton="false">
   ```

49.2.2 Implementation Details

When you override the `AbstractTransporter` methods, keep the following in mind:

- `ping()` contains the logic that checks whether the target is up or down. Its most prominent use is to power the green/red diagnostic indicator in the publishing console. It is not necessary for ping to be successful in order to launch a publishing session, but this can be a handy tool for diagnosing connection problems.

  If you are using http(s) to connect to your target, you may be able to use the default implementation rather than override and implement your own.

- `sendBatch()` is responsible for uploading data to the remote `fw_PubDataStore`. It is invoked multiple times with small batches of data from the local `fw_PubDataStore` that comes in the form of an IList. Batching helps keep memory usage down and is already done behind the scenes for you.

- `remoteExecute()` is responsible for communicating with the remote system. The communication is two-way - management sends commands to dispatch remote jobs and cancellation requests, while the target sends back messages that indicate its status. The contents of these messages are immaterial to `remoteExecute`, all it needs to do is send those requests and return the responses.
- `listTransports()` is a listing of the underlying transports, in case there are multiple targets. If there is only a single target, this method can just return a `toString()` rendition of the current transport.
- `toString()` is a human-friendly descriptor of this transport. For example, a typical value would be `http://mytarget:8081/cs/`. However, any other string is acceptable, including `targetDataCenter-Virginia`, `serverOn8080`, etc.

49.2.3 Helper Methods

A few helper methods are available in `AbstractTransporter`:

- `protected void writeLog(String msg)` write message to the publish log
- `protected AbstractTransporter getStandardTransporterInstance()` get a new instance of the standard HTTP-based transporter. This can be useful to implement a transport to multiple targets.
- `protected String getParam(String param)` obtain the value of a publishing parameter, as configured in the publishing console.

49.2.4 Example of a Transporter Implementation

Following is an example of a transporter implementation that works with multiple targets. The target is configured as follows:

1. In the Destination Address, specify comma-separated destination URLs.

   For example:

   ```
   http://tgt1:9030/cs/
   http://virginia:9040/cs/
   ```

2. In the More Arguments, specify ampersand-separated username, password, and optional proxy information for the additional servers, suffixed with indexes starting at 1.

   For example, with one additional target:

   ```
   REMOTEUSER1=fwadmin&REMOTEPASS1=xceladmin&PROXYSERVER1=proxy.com&PROXYPORT1=909
   0&PROXYUSER1=pxuser&PROXYPASSWORD1=pxpass
   ```

3. In `AdvPub.xml`, replace the `DataTransporter` bean entry with:

```
<bean id="DataTransporter"
  class="my.sample.MultiTransporter"
  singleton="false">
```

49.2.5 Full Code Listing

```
package my.sample;
import COM.FutureTense.Interfaces.*;
import com.fatwire.cs.core.realtime.TransporterReply;
import java.net.URL;
import java.util.*;
```

```
/**
 * RealTime Publishing transporter to multiple targets.
 */
public class MultiTransporter extends AbstractTransporter
{
  private boolean initialized = false;
  List<AbstractTransporter>  transporters  = new ArrayList();
  /**
   * Ping each underlying target and return true if all of them are up.
   */
  @Override
  public boolean ping(StringBuilder sbOut)
  {
  init();
  boolean ret = true;
  for(AbstractTransporter  t : transporters)
  {
  boolean thisret = t.ping(sbOut);
  sbOut.append(t.getRemoteUrl() + (thisret ? " OK" : " Not reachable"));
  sbOut.append(" ||| ");
  ret &= thisret;
  }
  return ret;
  }
  /**
   * Send the batch to each underliyng transport.
   */
  @Override
  protected int sendBatch(ICS ics, IList iList, StringBuffer outputMsg)
  {
  init();
  for(AbstractTransporter  t : transporters)
  {
  int res = t.sendBatch(ics, iList, outputMsg);
  if(res != 0)
  {
  // Just log the error for now, but this is an
  // indication that the target may be down
  // and other notifications may also be appropriate.
  writeLog("Transporter " + t + " failed with " + res   + " " + outputMsg);
  }
  }
  return 0;
  }
  /**
   * Execute the remote command on each transporter and
   * accumulate their responses.
   */
  @Override
  protected List<TransporterReply> remoteExecute(ICS ics, String s,
  Map<String, String> stringStringMap)
  {
  init();
  List<TransporterReply> res = new ArrayList<TransporterReply>();
  for(AbstractTransporter  t : transporters)
  {
  List<TransporterReply> tres = t.remoteExecute(ics, s, stringStringMap);
  res.addAll(tres);
  }
  return res;
```

```
}
/**
 * Do some initialization by parsing out the configuration
 * settings and instantiating a standard http transport
 * to each target.
 */
private void init()
{
if(!initialized)
{
String remoteURLs = getRemoteUrl();
int count = 0;
for(String remoteUrl : remoteURLs.split(","))
  {
  String suffix = (count == 0) ? "" : String.valueOf(count);
  AbstractTransporter t1 =
  AbstractTransporter.getStandardTransporterInstance();
  URL url;
  try
  {
    url = new  URL(remoteUrl);
  }
  catch(Exception e)
  {
    throw new RuntimeException(e);
  }
  t1.setRemoteUrl(remoteUrl);
  t1.setHost(url.getHost());
  t1.setUsername(getParam("REMOTEUSER" + suffix));
  t1.setPassword(getParam("REMOTEPASS" + suffix));
  t1.setUseHttps("https".equalsIgnoreCase(url.getProtocol()));
  t1.setContextPath(url.getPath());
  t1.setPort(url.getPort());
  t1.setProxyserver(getProxyserver());
  t1.setProxyport(getProxyport());
  t1.setProxyuser(getProxyuser());
  t1.setProxypassword(getProxypassword());
  t1.setHttpVersion(getHttpVersion());
  t1.setTargetIniFile(getTargetIniFile()); transporters.add(t1);
  ++count;
  }
initialized = true;
writeLog("Initialized transporters: " + toString());
}
}
/**
 * Provide a full listing of all underlying transports. This is
 * can be used by other components to determine
 * whether they need to perform special actions depending on
 * the number of targets. For example, asset publishing
 * status processing may need to buffer responses until they're
 * received from all targets before marking assets published.
 * @return
 */
@Override
public List<String> listTransports()
{
init();
List<String> list = new ArrayList();
for(AbstractTransporter  t : transporters)
```

```
  {
  list.add(t.toString());
  }
   return list;
  }
  /**
   * Just a human-friendly description of the transport. This may  show
   * up in the  logs, so make it descriptive enough.
   */
  @Override
  public String toString()
  {
  List<String> transs = listTransports();
  StringBuilder sb = new StringBuilder();
  for(String t : transs)
  sb.append(t + " ");
  return sb.toString();
  }
}
```

49.2.6 Edge-Case Scenarios

While the example in Section 49.2.5, "Full Code Listing" will work in the optimistic case where all targets are running, there will be times when one target has stopped for a shorter or longer period of time. If you only publish to one target but still mark assets as published, the target that stopped will not be synchronized. You can handle such scenarios in the following ways:

- If a target stops for a short period of time, you should not mark assets as published, but continue publishing to the target that is running. When the other target is restarted, you will have all earlier assets still queued for publishing. Those assets will be redundantly published to the first target as well, but over short periods of time this is a negligible overhead.
- If a target stays down for a long period of time, it may be best to remove it from the list of targets in the destination configuration (in this example, remove the second target from the Destination Address in the publishing configuration). That way, assets will continue to be marked as published even though you have only one active target. When the second target is restarted, first perform a database and file system sync, and then add it back to the list of destination addresses.

49.2.7 Intercepting Asset Publishing Events on the Management Instance

In the first case above, you need to only mark assets as published once they are saved on all targets. To do so, implement custom notification logic as follows:

1. Extend `com.fatwire.realtime.messaging.AssetPublishCallback`.
2. Override the `notify()` and optionally `progressUpdate()` method.

   For a detailed sample implementation, see "Sample Implementation for Steps 1 and 2" below.
3. Enable the callback in `AdvPub.xml` on the management side:
   - Add `AssetCallback` bean.
   - Register the bean with `PubsessionMonitor`.

   For the code, see "Enabling the Callback Bean for Step 3" below.

Sample Implementation for Steps 1 and 2

```
package my.sample;
import com.fatwire.assetapi.data.AssetId;
import java.util.HashMap;
import java.util.Map;
/**
 * Buffer asset save notifications until we've received one
 * from each target. Then mark asset published.
 */
public class AssetPublishCallbackMulti extends AssetPublishCallback
{
  Map<String, Integer> saveEventsCount  = new HashMap<String, Integer>();
  /**
   * Receive notifications about the asset status.
   * Currently the only available status is SAVED.
   */
  @Override
  public void notify(AssetId assetId, String status, String from)
  {
  String assetIdStr = String.valueOf(assetId);
  writeLog("Got " + status + " notification from "
  + from + " for " + assetIdStr);
  if("SAVED".equals(status))
    {
    Integer numNotifications;
    if((numNotifications = saveEventsCount.get(assetIdStr)) == null)
      {
      numNotifications = 0;
      }
    numNotifications = numNotifications + 1;
    saveEventsCount.put(assetIdStr, numNotifications);
    if(numNotifications == this.getTargets().size())
      {
      super.notify(assetId, status, from);
      writeLog("Marked " + assetIdStr  + " published");
      }
    }
  }
  /**
   * Intercept progress update messages. Can be used for
   * monitoring the health of the system but is not required.
   */
  @Override
  public void progressUpdate(String sessionId, String job,
  String where, String progress, String lastAction, char status)
  {
  super.progressUpdate(sessionId, job, where, progress, lastAction, status);
  }
}
```

Enabling the Callback Bean for Step 3

To add the AssetCallback bean:

```
<bean id="AssetCallback"
  class="my.sample.AssetPublishCallbackMulti"
  singleton="false"/>
```

To register the bean with PubsessionMonitor:

```
<bean id="PubsessionMonitor"
  class="com.fatwire.realtime.messaging.PubsessionMonitor"
  singleton="false">

  <constructor-arg index="0">
    <ref local="DataTransporter" />
  </constructor-arg>

  <constructor-arg index="1">
    <ref local="AssetCallback" />
  </constructor-arg>

  <property name="pollFreqMillis" value="5000" />
  <property name="timeoutMillis" value="100000" />
</bean>
```

49.2.8 Finishing Touches

When publishing to multiple destinations, it is useful to distinguish between their respective Unpackers and CacheUpdaters. This comes in handy when looking at the progress bars in the RT publishing console and looking at logs. To make that distinction, simply edit the `AdvPub.xml` file on the target side, and change the id values of the `DataUnpacker` and `PageCacheUpdater` beans.

For example:

```
<bean id="DataUnpacker"
  class="com.fatwire.realtime.ParallelUnpacker"
  singleton="false">
  <property name="id" value="Unpacker-Virginia2"/>
  ...
</bean>

<bean id="PageCacheUpdater"
  class="com.fatwire.realtime.regen.ParallelRegeneratorEh"
  singleton="false">
  <property name="id" value="CacheFlusher-Virginia2"/>
  ...
</bean>
```

50

Coding the Crawler Configuration File

This chapter contains information about the BaseConfigurator class, about implementing its methods and interfaces to control a crawler's site capture process, and about sample code that is available in the Site Capture installation for the FirstSiteII crawler.

This chapter contains the following topics:

50.1 Overview of Controlling a Crawler

Controlling a crawler requires coding its CrawlerConfigurator.groovy file with at least the following information: the starting URI and link extraction logic. Both pieces of information are supplied through the getStartUri() and createLinkExtractor() methods. Additional code can be added as necessary to specify, for example, the number of links to be crawled, the crawl depth, and the invocation of a post-crawl event such as copying statically downloaded files to a web server's doc base.

The methods and interfaces you will use are provided in the BaseConfigurator class. The default implementations can be overridden to customize and control a crawl process in a way that agrees with the structure of the target site and the data you need to collect.

This chapter begins with the BaseConfigurator methods (see Section 50.2, "BaseConfigurator Methods") and a simple CrawlerConfigurator.groovy file (see

Section 50.2.2, "createLinkExtractor.") to demonstrate usage of the required methods. Crawler customization methods are then discussed and followed by information about Site Capture's Java interfaces, including their default and custom implementations.

50.2 BaseConfigurator Methods

The `CrawlerConfigurator.groovy` file contains the code of the `CrawlerConfigurator` class. This class must extend `BaseConfigurator`, which is an abstract class that provides default implementations for the crawler. Methods and interfaces of the `BaseConfigurator` class are listed in Table 50–1 and described in the sections that follow. A basic sample `CrawlerConfigurator.groovy` file is shown in Section 50.2.2, "createLinkExtractor."

Table 50–1 Methods in the `BaseConfigurator` Class

Method Type	Method	Notes
Required	Section 50.2.1, "getStartUri."	N/A
Required	Section 50.2.2, "createLinkExtractor."	Factory method in the Section 50.12.1, "LinkExtractor." interface.[1],[2]
Crawler Customization	Section 50.3.1, "getMaxLinks."	N/A
Crawler Customization	Section 50.3.2, "getMaxCrawlDepth."	N/A
Crawler Customization	Section 50.3.3, "getConnectionTimeout."	N/A
Crawler Customization	Section 50.4, "getSocketTimeout."	N/A
Crawler Customization	Section 50.5, "getPostExecutionCommand."	N/A
Crawler Customization	Section 50.6, "getNumWorkers."	N/A
Crawler Customization	Section 50.7, "getUserAgent."	N/A
Crawler Customization	Section 50.8, "createResourceRewriter."	Factory method in the Section 50.12.2, "ResourceRewriter." interface.a,b

[1] The listed interfaces have default implementations, described in this chapter.

[2] Site Capture provides a sample link extractor and resource rewriter, both used by the FirstSiteII sample crawler. See Section 50.12.1.3, "Writing and Deploying a Custom Link Extractor." and Section 50.12.2.3, "Writing a Custom ResourceRewriter."

This section contains the following topics:

Two abstract methods in `BaseConfigurator` must be overridden in `CrawlerConfigurator`. They are `getStartUri()` and `createLinkExtractor()`.

- Section 50.2.1, "getStartUri"
- Section 50.2.2, "createLinkExtractor"

50.2.1 getStartUri

This method is used to inject the crawler's start URI. You can configure one or more start URIs for the crawl, as long as the URIs belong to the same site. If you specify multiple starting points, the crawls will start in parallel.

Example

To provide the start URI for the `www.example.com` site:

```
/**
 * The method is used to configure the site url which needs to be crawled.
 */
public String[] getStartUri()
{
return ["http://www.example.com/home"]; //Groovy uses brackets for an array.
}
```

Example

To provide multiple start URIs for the site, enter a comma-separated array:

```
/**
 * The method is used to configure the site url which needs to be crawled.
 */
public String[] getStartUri()
{
return ["http://www.example.com/product","http://www.example.com/support"];
//Groovy uses brackets for an array.
}
```

50.2.2 createLinkExtractor

This method is used to configure the logic for extracting links from the crawled pages. The extracted links are then traversed. The `createLinkExtractor` method is a factory method in the `LinkExtractor` interface:

- You can implement the `LinkExtractor` interface to create your own link extraction algorithm, for example, using an HTML parser to parse the pages and extract links for the crawler to consume.

- You can also use the default implementation, `PatternLinkExtractor`, which uses regular expressions for extracting links. For example, `PatternLinkExtractor` can be used to extract links of the format `/home/products` from expressions such as `<a href="/home/product">Products</a>`, as shown in the sample code in "Basic Configuration File" on page 50-4.

  Example

  To use a regular expression for extracting links from `<a href="/home/product">Products</a>` on the `www.example.com` site:

  ```
  /**
   * The method is used to define the link extraction
   * algorithm from the crawled pages.
   * PatternLinkExtractor is a regex based extractor
   * which parses the links on the web page
   * based on the pattern configured inside the constructor.
   */
  public LinkExtractor createLinkExtractor()
  {
  return new PatternLinkExtractor("['\"\\(](/[^\\s<'\"\\)]*)",1);
  }
  ```

- For more information about regular expressions and `PatternLinkExtractor`, see Section 50.12.1.2, "Using the Default Implementation of LinkExtractor."

- For more information about implementing the `LinkExtractor` interface, see Section 50.12.1.3, "Writing and Deploying a Custom Link Extractor."

Basic Configuration File

Below is an example of a simple `CrawlerConfigurator.groovy` file in which the required methods, `getStartUri()` and `createLinkExtractor()`, are overridden.

> **Note:** In the sample below, we override an additional method `getMaxLinks()`, which is discussed on page 50-5. In the sample, it is set to return `150` so that the test run can be completed quickly.
>
> The file named `CrawlerConfigurator.groovy` is used to inject dependency. Hence, its name must not be changed.

```
package com.fatwire.crawler.sample

import java.text.DateFormat;
import java.text.SimpleDateFormat;

import java.util.regex.Pattern;

import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;

import com.fatwire.crawler.*;
import com.fatwire.crawler.remote.*;
import com.fatwire.crawler.remote.di.*;
import com.fatwire.crawler.impl.*;
import com.fatwire.crawler.util.FileBuilder;

import org.apache.commons.lang.SystemUtils;
import org.apache.http.HttpHost;
import org.apache.http.auth.*;
import org.apache.http.client.*;
import org.apache.http.impl.client.*;
/**
 * Configurator for the crawler.
 * This is used to inject the dependency inside the crawler
 * to control the crawling process
 */

public class CrawlerConfigurator extends BaseConfigurator {

public CrawlerConfigurator(GlobalConfigurator delegate){
super(delegate);
}

/**
 * The method is used to configure the site url which needs to be crawled.
 */
public String[] getStartUri() {
return ["http://www.fatwire.com/home"]; //Groovy uses brackets for an array.
}

/**
 * The method is used to define the link extraction algorithm
 * from the crawled pages.
 * PatternLinkExtractor is a regex based extractor which parses
 * the links on the web page
 * based on the pattern configured inside the constructor.
 */
```

```
public LinkExtractor createLinkExtractor() {
return new PatternLinkExtractor("['\"\\(](/[^\\s<'\"\\)]*)",1);
}

/**
 * The method is used to control the maximum number of links
 * to be crawled as part of this crawl session.
 */
public int getMaxLinks()
{
150;
}
```

50.3 Crawler Customization Methods

In addition to the required methods, the `BaseConfigurator` class has methods with default implementations that you may want to override to customize the crawl process in a way that agrees with the structure of the target site and the data you need to collect.

This section contains the following topics:

- Section 50.3.1, "getMaxLinks"
- Section 50.3.2, "getMaxCrawlDepth"
- Section 50.3.3, "getConnectionTimeout"

50.3.1 getMaxLinks

This method is used to control the number of links to be crawled. The number of links should be a positive integer. Otherwise, the crawl will scan all the links in the same domain that are reachable from the start URI(s).

Example

To specify crawling 500 links:

```
/**
 * default: -1; crawler will crawl over all the links reachable from the start URI
 * @return the maximum number of links to download.
 */
public int getMaxLinks()
{
return 500;
}
```

50.3.2 getMaxCrawlDepth

This method is used to control the maximum depth to which a site will be crawled. Links beyond the specified depth are ignored. The depth of the starting page is `0`.

```
/**
 * default: -1. Indicates infinite depth for a site.
 * @return the maximum depth to which we need to crawl the links.
 */
public int getMaxCrawlDepth()
{
return 4;
}
```

50.3.3 getConnectionTimeout

This method determines how long the crawler will wait to establish a connection to its target site. If a connection is not established within the specified time, the crawler will ignore the link and continue to the next link.

Example

To set a connection timeout of 50,000 milliseconds:

```
/**
 * default: 30000 ms
 * @return Connection timeout in milliseconds.
 */
public int getConnectionTimeout()
{
return 50000; // in milliseconds
}
```

50.4 getSocketTimeout

This method controls the socket timeout of the request that is made by the crawler for the link to be crawled.

Example

To provide a socket timeout of 30,000 milliseconds:

```
/**
 * default: 20000 ms
 * @return Socket timeout in milliseconds.
 */
public int getSocketTimeout()
{
return 30000; // in milliseconds
}
```

50.5 getPostExecutionCommand

This method is used to inject custom post-crawl logic. This method is invoked when the crawler finishes its crawl session. The absolute path of the script or command and parameters (if any) must be returned by this method.

For example, the `getPostExecutionCommand()` can be used to automate deployment to a web server's doc base by invoking a batch or shell script to copy statically captured files after the crawl session ends.

Note:

- The script or command should be present in the same location on all servers hosting Site Capture.
- Avoid downloading large archive files (exceeding 250MB) from the Site Capture interface. Use `getPostExecutionCommand` to copy the files from the Site Capture file system to your preferred location. Archive size can be obtained from the crawler report, on the Job Details form.

Example

To execute a batch script named `copy.bat` on the Site Capture server:

```
/**
 * default: null.
 * @return the command string for post execution.
 * Null if there is no such command.
 */
public String getPostExecutionCommand()
{
// The file is supposed to be at the path C:\\commands folder
// on the computer where the site capture server is running
return "C:\\commands\\copy.bat";
}
```

50.6 getNumWorkers

This method controls the number of worker threads used for the crawl process. The ideal number of parallel threads to be spawned for the crawl session depends on the architecture of the computer on which Site Capture is hosted.

Example

To start 10 worker threads for a crawl process:

```
/**
 * default: 4.
 * @return the number of workers to start.
 * Workers will concurrently downloads resources.
 */
public int getNumWorkers()
{
// Start 10 worker threads which is involved in the crawl process.
return 10;
}
```

50.7 getUserAgent

This method is used to configure the user agent that will be utilized by the crawler when it traverses the site. This method should be used when the site is rendered differently from the way it is rendered in a browser (for example, when the site is rendered on a mobile device).

Example

To configure the FireFox 3.6.17 user agent:

```
/**
 * default: publish-crawler/1.1 (http://www.fatwire.com)
 * @return the user agent identifier
 */
public String getUserAgent()
{
return "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US;rv:1.9.2.17) Gecko/20110420
Firefox/3.6.17 ";
}
```

50.8 createResourceRewriter

This method is used to rewrite URLs inside the html pages that are crawled. For example, you may want to rewrite the URLs to enable static delivery of a dynamic WebCenter Sites website.

The `createResourceRewriter` method is a factory method in the `ResourceRewriter` interface:

- You can implement the `ResourceRewriter` interface to convert dynamic URLs to static URLs, convert absolute URLs to relative URLs, and so on.
- You can also use the following default implementations:
  - `NullResourceRewriter:` Does not rewrite any of the URLs.
  - `PatternResourceRewriter:` Searches for a regular pattern and rewrites as specified.

Example

To use `PatternResourceRewriter` to rewrite URLs such as `http://www.site.com/home.html` to `/home.html`:

```
/**
 * Factory method for a ResourceRewriter.
 * default: new NullResourceRewriter();
 * @return the rewritten resource modifies the html before it is saved to disk.
 */
public ResourceRewriter createResourceRewriter()
{
new PatternResourceRewriter("http://www.site.com/([^\\s'\"]*)", '/$1');
}
```

- For more information about the default implementations, see Section 50.12.2.2, "Using the Default Implementations of ResourceRewriter."
- For more information about implementing the `ResourceRewriter` interface, see Section 50.12.2.3, "Writing a Custom ResourceRewriter.".

50.9 createMailer

This method provides the implementation for sending email at the end of the crawl. The `createMailer` method is a factory method in the `Mailer` interface:

- Site Capture comes with an SMTP over TLS implementation, which emails the crawler report when a static or archive capture session ends (the crawler report is the `report.txt` file, described in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*).

- If you are using a mail server other than SMTP-TLS (such as SMTP without authentication, or POP3), you will have to provide your own implementation.

Example

To send no email:

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email at the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
return new NullMailer();
}
```

- For more information about the default implementation, see Section 50.12.3.2, "Using the Default Implementation of Mailer."
- For more information about implementing the `ResourceRewriter` interface, see Section 50.12.3.3, "Writing a Custom Mailer."

50.10 getProxyHost

This method must be overridden if the site being crawled is behind a proxy server. You can configure the proxy server in this method.

> **Note:** If you use `getProxyHost`, also use `getProxyCredentials`, described on Section 50.11, "getProxyCredentials."

Example

To configure a proxy server:

```
/**
 * default: null.
 * @return the host for the proxy,
 * null when there is no proxy needed
 */
public HttpHost getProxyHost()
{
//using the HTTPClient library return a HTTPHost
return new HttpHost("www.myproxyserver.com", 883);
}
```

50.11 getProxyCredentials

This method is used to inject credentials for the proxy server that is configured in the `getProxyHost` method (see Section 50.10, "getProxyHost").

Example

To authenticate a proxy server user named `sampleuser`:

```
/**
 * default: null.
 * example: new UsernamePasswordCredentials(username, password);
 * @return user credentials for the proxy.
 */
public Credentials getProxyCredentials()
{
return new UsernamePasswordCredentials("sampleuser", "samplepassword");
//using the HTTPClient library return credentials
}
```

50.12 Interfaces

Site Capture provides the following interfaces with default implementations:

- Section 50.12.1, "LinkExtractor"
- Section 50.12.2, "ResourceRewriter"
- Section 50.12.3, "Mailer"

50.12.1 LinkExtractor

A link extractor is used to specify which links will be traversed by Site Capture in a crawl session. The implementation is injected through the `CrawlerConfigurator.groovy` file. The implementation is called by the Site Capture framework during the crawl session to extract links from the markup that is downloaded as part of the crawl session.

Site Capture comes with one implementation of `LinkExtractor`. You can also write and deploy your own custom link extraction logic. For more information, see the following sections:

- Section 50.12.1.1, "LinkExtractor Interface"
- Section 50.12.1.2, "Using the Default Implementation of LinkExtractor"
- Section 50.12.1.3, "Writing and Deploying a Custom Link Extractor"

50.12.1.1 LinkExtractor Interface

This interface has only one method (`extract)` that needs to be implemented to provide the algorithm for extracting links from downloaded markup.

```
package com.fatwire.crawler;
import java.util.List;
import com.fatwire.crawler.url.ResourceURL;

/**
 * Extracts the links out of a WebResource.
 */

public interface LinkExtractor
{
/**
 * Parses the WebResource and finds a list of links (if possible).
 * @param resource the WebResource to inspect.
 * @return a list of links found inside the WebResource.
 */
List<ResourceURL> extract(final WebResource resource);
}
```

50.12.1.2 Using the Default Implementation of LinkExtractor

PatternLinkExtractor is the default implementation for the LinkExtractor interface. PatternLinkExtractor extracts links on the basis of a regular expression. It takes a regular expression as input and returns only links matching that regular expression.

Common usage scenarios are described below. They are: Using PatternLinkExtractor for sites with dynamic URLs, and using PatternLinkExtractor for sites with static URLs.

- Using PatternLinkExtractor for sites with dynamic URLs:

  For example, on www.example.com, the links have a pattern of /home, /support, and /cs/Satellite/. To extract and traverse such kinds of links, we use PatternLinkExtractor in the following way:

  ```
  /**
   * The method is used to define the link extraction algorithm
   * from the crawled pages.
   * PatternLinkExtractor is a regex based extractor which parses
   * the links on the web page
   * based on the pattern configured inside the constructor.
   */
  public LinkExtractor createLinkExtractor()
  {
  return new PatternLinkExtractor("['\"\\(](/[^\\s<'\"\\)]*)",1);
  }
  ```

  The pattern **[**'\"\\(**]** **(**/[^\\s<'\"\\)]***)** is used to extract links

  - that start with any one of the following characters:
    - single quote (')
    - double quotes (")
    - left parenthesis (
  - continue with a forward slash (/) ,
  - and end with any one of the following characters:
    - spaces (\s)
    - less-than symbol (<)
    - single quote (')
    - double quote (")
    - right parenthesis)

  Let's consider the URL inside the following markup:

  <a href='/home'>Click Me</a>

  We are interested only in extracting the /home link. This link matches the regular expression pattern because it starts with a single quote (') and ends with a single quote ('). The grouping of 1 will return the result as /home.

- Using PatternLinkExtractor for sites with static URLs:

  For example, the markup for www.example.com has links such as:

  <a href="http://www.example.com/home/index.html">Click Me</a>

  To extract and traverse such types of links, we can use PatternLinkExtractor in the following way

```
/**
 * The method is used to define the link extraction algorithm
 * from the crawled pages.
 * PatternLinkExtractor is a regex based extractor which parses
 * the links on the web page
 * based on the pattern configured inside the constructor.
 */
public LinkExtractor createLinkExtractor()
{
return new
PatternLinkExtractor(Pattern.compile("http://www.example.com/[^\\s<'\"]*"));
}
```

The above example instructs the crawler to extract links that start with `http://www.example.com` and end with any one of the following characters: spaces (`\s`), less-than symbol (`<`), single quote (`'`), or double quotes (`"`).

Note: For more details on groups and patterns, refer to the Java documentation for the `Pattern` and `Matcher` classes.

50.12.1.3 Writing and Deploying a Custom Link Extractor

Note: Site Capture provides a sample link extractor (and resource rewriter) used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. For more information, see the source code for the `FSIILinkExtractor` class in the following folder:

`<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

To write a custom link extractor

1. Create a project in your favorite Java IDE.
2. Copy the file `fw-crawler-core.jar` from the `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder to your project's build path.
3. Implement the `LinkExtractor` interface to provide the implementation for the `extract()` method. (The `LinkExtractor` interface is shown on Section 50.12.1.1, "LinkExtractor Interface.")

   Below is pseudo-code showing a custom implementation:

   ```
   package com.custom.crawler;
   import java.util.List;
   import com.fatwire.crawler.url.ResourceURL;
   import com.fatwire.crawler.LinkExtractor;
   /**
    * Extracts the links out of a WebResource.
    */
   public class CustomLinkExtractor implements LinkExtractor
   {
   /**
    * A sample constructor for CustomLinkExtractor
    */
   public CustomLinkExtractor(String .......)
   ```

```
  {
  // Initialize if there are private members.
  // User's custom logic
  }

/**
 * Parses the WebResource and finds a list of links (if possible).
 * @param resource the WebResource to inspect.
 * @return a list of links found inside the WebResource.
 */
List<ResourceURL> extract(final WebResource resource)
  {
  // Your custom code for extraction Algorithm.
  }
}
```

4. Create a `jar` file for your custom implementation and copy it to the following folder:

   `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib`

5. Restart the Site Capture application server.

6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom link extractor class (`CustomLinkExtractor`, in this example):

```
/**
 * User's custom link extractor mechanism to extract the links from the
 * web resource downloaded as part of the crawl session.
 * The code below is only a pseudo code for an example.
 * User is free to implement their own custom constructor
 * as shown in the next example.
 */
public LinkExtractor createLinkExtractor()
{
return new CustomLinkExtractor("Custom Logic For Your Constructor");
}
```

50.12.2 ResourceRewriter

A resource rewriter is used to rewrite URLs inside the markup that is downloaded during the crawl session. The implementation must be injected through the `CrawlerConfigurator.groovy` file.

Some use cases that will require a resource rewriter are:

- Crawling a dynamic site and creating a static copy. For example, the FirstSiteII sample site has dynamic links. Converting FirstSiteII into a static site requires rewriting the URLs inside the downloaded markup.
- Converting absolute URLs to relative URLs. For example, if the markup has URLs such as `http://www.example.com/abc.html`, the crawler should remove `http://www.example.com` from the URL, thus allowing resources to be served from the host on which the downloaded files are stored.

Site Capture comes with the two implementations of `ResourceRewriter`. You can also create custom implementations. For more information, see the following sections:

- Section 50.12.2.1, "ResourceRewriter Interface"
- Section 50.12.2.2, "Using the Default Implementations of ResourceRewriter"

- Section 50.12.2.3, "Writing a Custom ResourceRewriter"

50.12.2.1 ResourceRewriter Interface

The `rewrite` method is used to rewrite URLs inside the markup that is downloaded during the crawl session.

```
package com.fatwire.crawler;
import java.io.IOException;

/**
 * Service for rewriting a resource. The crawler will use the implementation for
 * rewrite method to rewrite the resources that are downloaded as part of crawl
 * session.
 */
public interface ResourceRewriter
{
/**
 * @param resource
 * @return the bytes after the rewrite.
 * @throws IOException
 */
byte[] rewrite(WebResource resource) throws IOException;

}
```

50.12.2.2 Using the Default Implementations of ResourceRewriter

Site Capture comes with the following implementations of `ResourceRewriter:`

- `NullResourceRewriter`, configured by default to skip the rewriting of links. If `ResourceRewriter` is not configured in the `CrawlerConfigurator.groovy` file, `NullResourceRewriter` is injected by default.
- `PatternResourceRewriter`, used to rewrite URLs based on the regular expression. `PatternResourceRewriter` takes as input a regular expression to match the links inside the markup and replaces those links with the string that is provided inside the constructor.

  Example

  To rewrite an absolute URL as a relative URL:

  From:

  ```
  <a href="http://www.example.com/about/index.html">Click Me</a>
  ```

  To:

  ```
  <a href="/about/index.html">Click Me</a>
  ```

  ```
  /**
   * Factory method for a ResourceRewriter.
   * default: new NullResourceRewriter();
   * @return the rewritten resource modifies the html before it is saved to disk.
   */
  public ResourceRewriter createResourceRewriter()
  {
  new PatternResourceRewriter("http://www.example.com/([^\\s'\"]*)", '/$1');
  }
  ```

  `PatternResourceRewriter` has only one constructor that takes a regular expression and a string replacement:

```
PatternResourceRewriter(final String regex, final String replacement)
```

50.12.2.3 Writing a Custom ResourceRewriter

> **Note:** Site Capture provides a sample resource rewriter (and link extractor) used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. For more information, see the source code for the `FSIILinkExtractor` class in the following folder:
>
> `<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

To write a custom resource rewriter

1. Create a project inside your favorite IDE.
2. Copy the files `fw-crawler-core.jar` from `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder to your project's build path.
3. Implement the `ResourceRewriter` interface to provide the implementation for the `rewrite` method. (The `ResourceRewriter` interface is shown on on page 50-14.)

   Below is pseudo-code showing a custom implementation:

   ```
   package com.custom.crawler;
   import com.fatwire.crawler.WebResource;
   import com.fatwire.crawler.ResourceRewriter;

   /**
    * Rewrite the links inside the markup downloaded as part of
    * crawl session.
    */
   public class CustomResourceRewriter implements ResourceRewriter
   {
   /**
    * A sample constructor for CustomResourceRewriter
    */
   public CustomResourceRewriter(String .......)
     {
     // Initialize if there are private members.
     // User's custom logic
     }

   /**
    * @param resource
    * @return the bytes after the rewrite.
    * @throws IOException
    */
   byte[] rewrite(WebResource resource) throws IOException
     {
     // Your custom code for re-writing Algorithm.
     }
   }
   ```

4. Create a `jar` file for your custom implementation and copy it to the following folder: `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib`
5. Restart your Site Capture application server.

6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom `resource rewriter` class (`CustomResourceRewriter`, in this example):

```
/*
 * User's custom resource rewriting mechanism to rewrite the links from the
 * web resource downloaded as part of the crawl session.
 *
 * The code below is only a pseudo code for an example.
 * User is free to implement their own custom constructor
 * as shown in the next example.
 */
public ResourceRewriter createResourceRewriter()
  {
  new CustomResourceRewriter("User's custom logic to initialize the things");
  }
```

50.12.3 Mailer

A mailer is used to send email after the crawl ends. The implementation must be injected through the `CrawlerConfigurator.groovy` file.

Site Capture provides an `SMTPTlsMailer` implementation, which can be used to send the crawler report from the SMTP-TLS mail server. You can also implement the `Mailer` interface to provide custom logic for sending emails from a server other than SMTP-TLS (such as SMTP without authentication, or POP3). Your custom logic can also specify the email to be an object other than the crawler report. If `Mailer` is not configured in the `CrawlerConfigurator.groovy` file, `NullMailer` is injected by default. For more information, see the following topics:

- Section 50.12.3.1, "Mailer Interface"
- Section 50.12.3.2, "Using the Default Implementation of Mailer"
- Section 50.12.3.3, "Writing a Custom Mailer"

50.12.3.1 Mailer Interface

The `sendMail` method is automatically called if the `Mailer` is configured in the `CrawlerConfigurator.groovy` file.

```
package com.fatwire.crawler;
import java.io.IOException;
import javax.mail.MessagingException;

/**
 * Service to send an email.
 */
public interface Mailer
{
/**
 * Sends the mail.
 *
 * @param subject
 * @param report
 * @throws MessagingException
 * @throws IOException
 */
void sendMail(String subject, String report)
throws MessagingException, IOException;
}
```

50.12.3.2 Using the Default Implementation of Mailer

Site Capture provides an SMTP-TLS server-based email implementation that sends out the crawler report when a static or archive crawl session ends (the crawler report is a (`report.txt` file, described in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*). You can use the default mailer by injecting it through the `CrawlerConfigurator.groovy` file as shown below:

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email
 * at the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
 try
  {
  // Creating a SmtpTlsMailer Object
  SmtpTlsMailer mailer = new SmtpTlsMailer();

  InternetAddress from;
  // Creating an internet address from whom the mail
  // should be sent from = new InternetAddress("example@example.com");

  // Setting the mail address inside the mailer object mailer.setFrom(from);

  // Setting the email address of the recipient inside
  // mailer.mailer.setTo(InternetAddress.parse("example@example.com"));

  // Setting the email server host for to be used for email.
  // The email server should be SMTP-TLS enabled.
  mailer.setHost("smtp.gmail.com", 587);

  // Setting the credentials of the mail account
  // mailer.setCredentials("example@example.com", "examplepassword");

  return mailer;
  }
catch (AddressException e)
  {
  log.error(e.getMessage());
  }
}
```

50.12.3.3 Writing a Custom Mailer

To write a custom mailer

1. Create a project inside your favorite IDE.
2. Copy the files `fw-crawler-core.jar` from the `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder in your project's build path.
3. Implement the `Mailer` interface with the `sendMail` method.

   Below is pseudo-code showing a custom implementation:

```
package com.custom.crawler;
import java.io.IOException;
import javax.mail.MessagingException;
import com.fatwire.crawler.Mailer;

/**
 * Implements an interface to implement the logic for sending emails
 * when the crawl session has been completed.
 */
public class CustomMailer implements Mailer
{
/**
 * A sample constructor for CustomMailer
 */
public CustomMailer()
  {
  // Initialize if there are private members.
  // User's custom logic
  }

/**
 * Sends the mail.
 *
 * @param subject
 * @param report
 * @throws MessagingException
 * @throws IOException
 */
void sendMail(String subject, String report)
throws MessagingException, IOException
  {
  // User's custom logic to send the emails.
  }
}
```

4. Create a `jar` file for your custom implementation and copy it to the following folder: `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib`

5. Restart your Site Capture application server.

6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom mailer class (`CustomMailer`, in this example):

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email
 * at the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
CustomMailer mailer = new CustomMailer();
// Do some of the initilization stuffs
return mailer;
}
package com.custom.crawler;
import java.io.IOException;
import javax.mail.MessagingException;
import com.fatwire.crawler.Mailer;
```

```
/**
 * Implements an interface to implement the logic for sending emails
 * when the crawl session has been completed.
 */
public class CustomMailer implements Mailer
{
  /**
   * A sample constructor for CustomMailer
   */
  public CustomMailer()
    {
    // Initialize if there are private members.
    // User's custom logic
    }
  /**
   * Sends the mail.
   *
   * @param subject
   * @param report
   * @throws MessagingException
   * @throws IOException
   */
  void sendMail(String subject, String report)
  throws MessagingException, IOException
    {
    // User's custom logic to send the emails.
    }
}
```

The implementation above will email the crawler report (`report.txt` file, described in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*), given that the `String report` argument in the `sendMail` method names the crawler report, by default. You can customize the logic for emailing objects other than the crawler report.

50.13 Summary of Methods and Interfaces

This chapter discussed methods and interfaces in Site Capture's `BaseConfigurator` class for controlling a crawler's site capture process. This section summarizes the methods as well as the interfaces and their default implementations.

This section contains the following topics:

- Section 50.13.1, "Methods"
- Section 50.13.2, "Interfaces"

50.13.1 Methods

The following is the list of methods used in Site Capture's `BaseConfigurator`:

- Section 50.2.1, "getStartUri"
- Section 50.2.2, "createLinkExtractor"
- Section 50.3.1, "getMaxLinks"
- Section 50.3.2, "getMaxCrawlDepth"
- Section 50.3.3, "getConnectionTimeout"
- Section 50.4, "getSocketTimeout"

- Section 50.5, "getPostExecutionCommand"
- Section 50.6, "getNumWorkers"
- Section 50.7, "getUserAgent"
- Section 50.8, "createResourceRewriter"
- Section 50.9, "createMailer"
- Section 50.10, "getProxyHost"
- Section 50.11, "getProxyCredentials"

In the preceding list, the factory methods are in the following interfaces:

- Section 50.2.2, "createLinkExtractor" is in the Section 50.12.1, "LinkExtractor" interface.
- Section 50.8, "createResourceRewriter" is in the Section 50.12.2, "ResourceRewriter" interface.
- Section 50.9, "createMailer" is in the Section 50.12.3, "Mailer" interface.

50.13.2 Interfaces

The following interfaces are used in Site Capture's `BaseConfigurator:`

- Section 50.12.1, "LinkExtractor"

  Its default implementation is `PatternLinkExtractor`, which extracts links on the basis of a regular expression.

  Site Capture also provides a sample link extractor (and a sample resource rewriter), used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. Source code is available in the following folder: `<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

  You can write and deploy your own custom link extraction logic.

- Section 50.12.2, "ResourceRewriter"

  Its default implementations are `NullResourceRewriter`, which skips the rewriting of links; and `PatternResourceRewriter`, which rewrites URLs based on the regular expression.

  Site Capture provides a sample resource rewriter (and a sample link extractor), used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. Source code is available in the following folder:`<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/ FirstSiteII/src`

  You can write and deploy your own logic for rewriting URLs.

- Section 50.12.3, "Mailer"

  Its default implementation is `SMTPTlsMailer`, which sends the crawler report from the SMTP-TLS mail server. You can customize the logic for emailing other types of objects from other types of servers.

Part II

Working with the Developer Tools

This part describes Oracle WebCenter Sites: Developer Tools, a toolkit used to integrate Oracle WebCenter Sites with the Eclipse Integrated Development Environment (IDE). The Developer Tools kit enables developers to work in a distributed environment using the Eclipse IDE and version control systems (VCS).

This part contains the following chapters:

51

About Oracle WebCenter Sites: Developer Tools

This chapter provides an overview about developing WebCenter Sites content management sites using Oracle WebCenter Sites: Developer Tools.

This chapter contains the following sections:

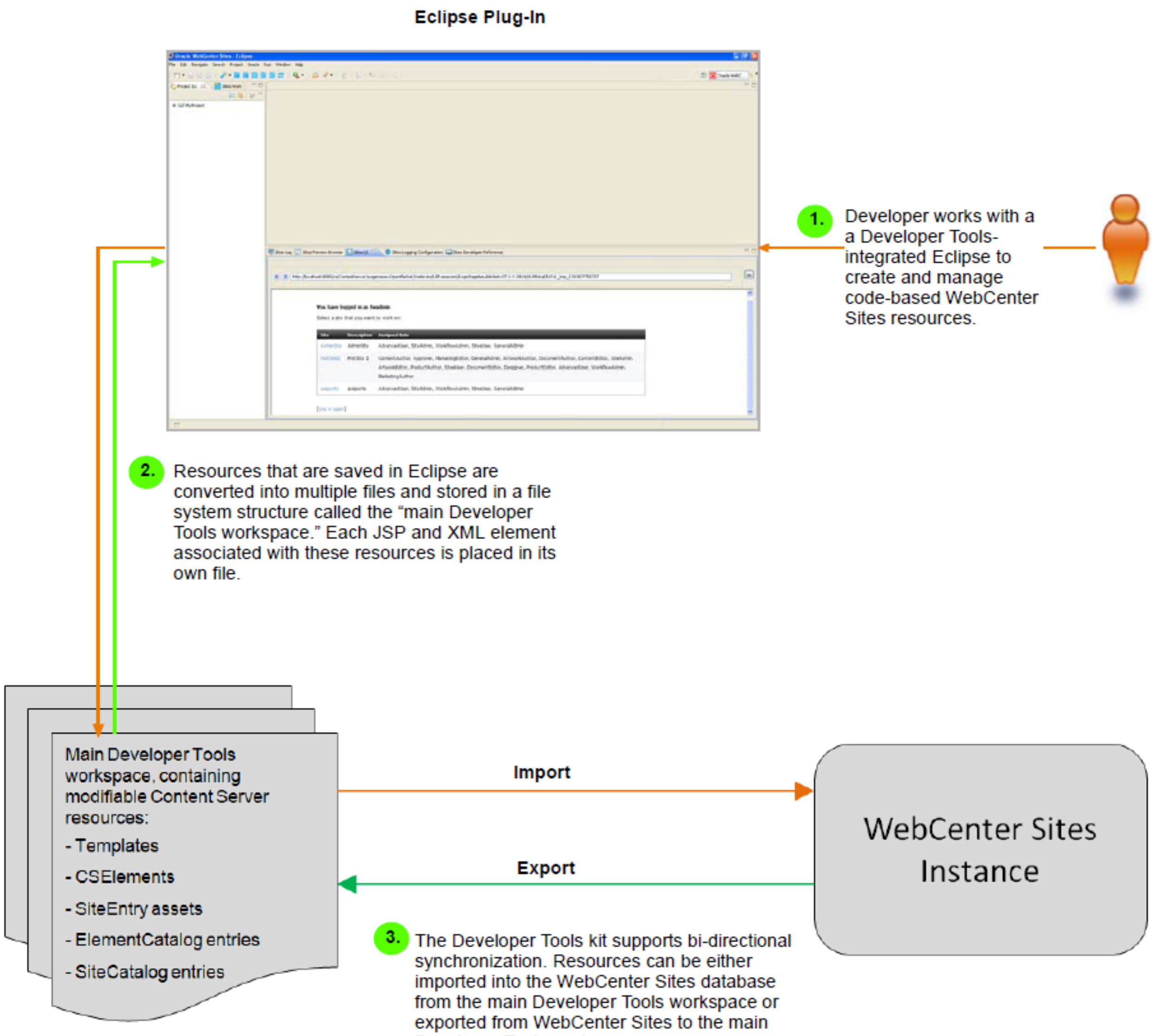
- Section 51.1, "Developer Tools Architecture"
- Section 51.2, "IDE Integration"
- Section 51.3, "The Developer Tools Workspace"
- Section 51.4, "Synchronization"
- Section 51.5, "JSP Management"
- Section 51.6, "Command-line Tool"
- Section 51.7, "Using a Version Control System"
- Section 51.8, "Next Steps"

51.1 Developer Tools Architecture

On any computer running Oracle WebCenter Sites, Developer Tools can be used to integrate WebCenter Sites with the Eclipse IDE to create a personal and flexible developer's environment. The developer interacts with Developer Tools (and therefore WebCenter Sites), primarily through Eclipse, which provides a rich set of functions for managing WebCenter Sites resources. Whether managed through Eclipse or in WebCenter Sites, the resources can be either automatically or manually synchronized by the Developer Tools kit.

For example, Eclipse-managed resources are stored as files in a file system, giving developers the option to integrate with a version control system of their choice. If the resources are modified and WebCenter Sites is running, the resources are automatically synchronized, that is, imported into WebCenter Sites, in its native database representation. Manual synchronization can be performed in both directions. A summary of Developer Tools can be found in Figure 51–1 and in the rest of this chapter.

Figure 51–1 Developer Tools Process Flow



Developer works with a a Developer Tools-integrated Eclipse to create and manage code-based WebCenter Sites resources.

Resources that are saved in Eclipse are converted into multiple files and stored in a file system structure called the "main Developer Tools workspace." Each JSP and XML element associated with these resources is placed in its own file.

The Developer Tools kit supports bi-directional synchronization. Resources can be either imported into the WebCenter Sites database from the main Developer Tools workspace or exported from WebCenter Sites to the main Developer Tools workspace.

51.2 IDE Integration

Using the Eclipse integration, developers can:

- Create, edit, and delete CSElement, Template, and SiteEntry assets, as well as SiteCatalog and ElementCatalog entries
- Develop JSP elements with standard Eclipse features such as tag completion, syntax highlighting, and debugging
- Export and import assets, asset types, flex families, sites, roles, tree tabs, and start menu items
- Preview WebCenter Sites pages within the Eclipse IDE using an embedded preview browser
- View the WebCenter Sites log file in a dynamically refreshing panel
- Integrate with version control systems

Note: Integrating Developer Tools with Eclipse embeds the WebCenter Sites Admin and Contributor interfaces in Eclipse to make it easily accessible to developers. For information about working in the Admin interface, see Part I, "Developing Oracle WebCenter Sites." For informationn about working in the Contributor interface, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.

51.3 The Developer Tools Workspace

WebCenter Sites resources that are managed in Eclipse are stored as files in a file system structure called the *main Developer Tools workspace*. This structure enables resources to be easily managed and exchanged with WebCenter Sites instances. The main Developer Tools workspace is the only workspace accessible from Eclipse.

Note: Creating a custom workspace is optional, and in most distributed environments the only necessary workspace is the main Developer Tools workspace. Custom workspaces are not accessible from Eclipse. Creating a custom workspace is described in Chapter 57, "Developer Tools: Workspaces." The rest of the Developer Tools chapters discuss the main Developer Tools workspace.

51.4 Synchronization

The Developer Tools kit enables you to synchronize resources in Eclipse with resources in WebCenter Sites, ensuring that the Developer Tools workspace and WebCenter Sites database contain the same content. Manual synchronization is bi-directional, that is, you can import resources into the WebCenter Sites database and export resources to the Developer Tools workspace.

Exporting resources to Eclipse converts the resources into files. Importing a resource into WebCenter Sites converts the resource to native WebCenter Sites format (database representation).

Note: Automatic synchronization occurs when WebCenter Sites resources are edited, created, or deleted in Eclipse, but only if the WebCenter Sites instance is running. This synchronization includes transparent flushing of page and resultset caches in WebCenter Sites.

51.5 JSP Management

The Developer Tools kit exposes WebCenter Sites JSPs as individual files in the Developer Tools workspace. Developers can then manage (create, edit, and debug) the JSPs as any other JSP file.

51.6 Command-line Tool

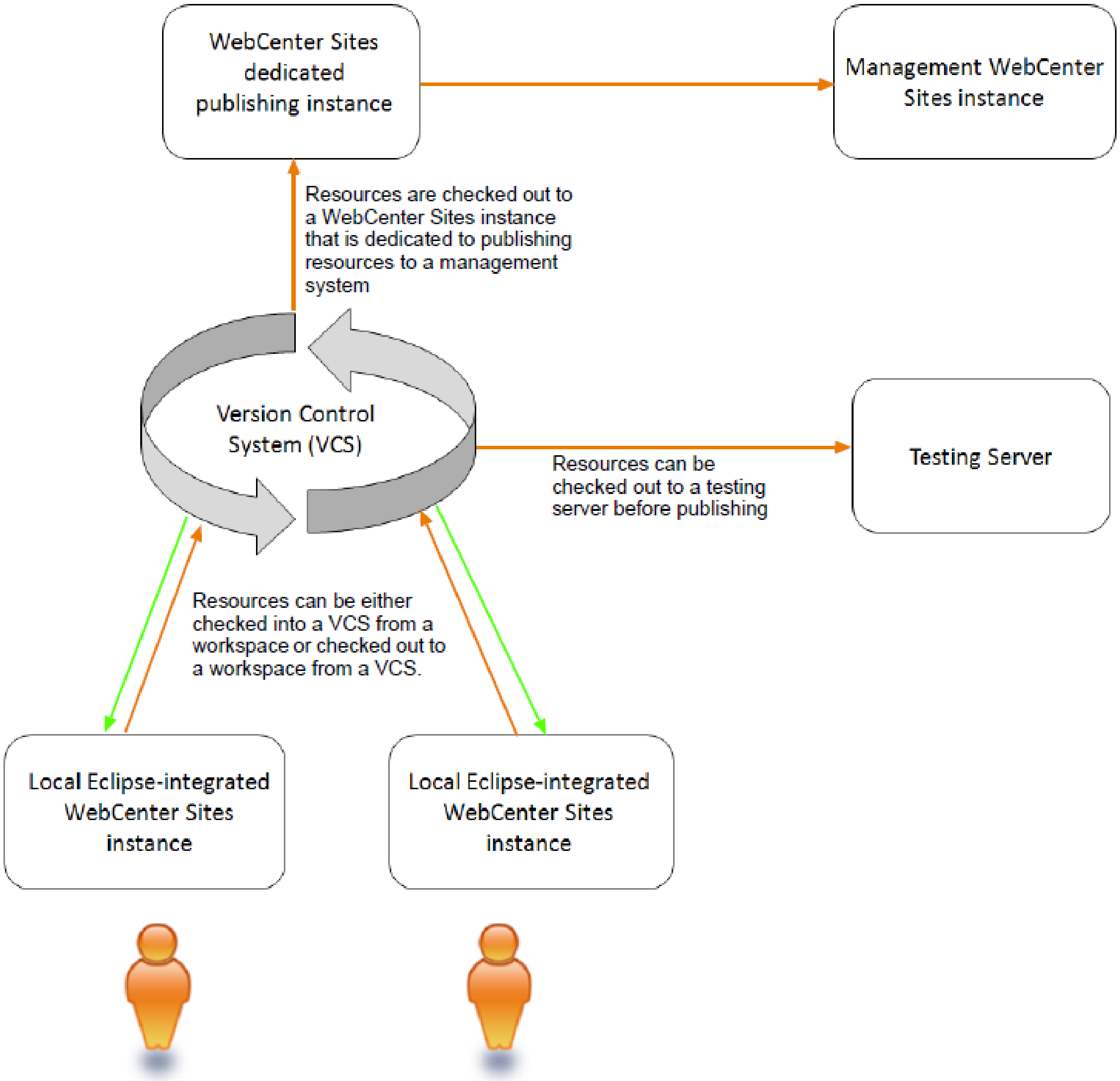
The Developer Tools kit provides a command-line utility which can be used for automating import and export tasks, large-scale resource movement, creating custom workspaces, and synchronizing resources with any workspace (unlike the Eclipse integration which lets you work only with resources stored in the main Developer Tools workspace).

51.7 Using a Version Control System

The Developer Tools kit supports the implementation of a version control system. Checking in resources from your workspace to a version control system enables the exchange of resources between WebCenter Sites instances and developers. You can also update your workspace with the resources checked in to the version control system by other developers. A version control system lets you check out resources to any target system, including testing servers, Management WebCenter Sites systems, or another developer's WebCenter Sites instance, to make use of their functions (such as publishing).

Figure 51–2 shows an example of using Developer Tools with a version control system. This example uses a dedicated WebCenter Sites instance to publish resources to a Management WebCenter Sites instance. Therefore, the Approval/Publishing feature provided by WebCenter Sites can be used to publish resources that were checked out from the version control system. This example is the recommended way to use a version control system with Developer Tools, but it is not required.

Figure 51–2 Using Developer Tools with a version control system


WebCenter Sites dedicated publishing instance
Management WebCenter Sites instance
Resources are checked out to a WebCenter Sites instance that is dedicated to publishing resources to a management system
Version Control System (VCS)
Resources can be checked out to a testing server before publishing
Testing Server
Resources can be either checked into a VCS from a workspace or checked out to a workspace from a VCS.
Local Eclipse-integrated WebCenter Sites instance
Local Eclipse-integrated WebCenter Sites instance

51.8 Next Steps

The next chapter provides instructions for setting up Developer Tools and for integrating a WebCenter Sites instance with Eclipse. It also provides information to help you get started creating and managing resources in an Eclipse-integrated WebCenter Sites instance. See Chapter 52, "Developer Tools: Installing and Configuring."

52 Developer Tools: Installing and Configuring

This chapter provides instructions for setting up Developer Tools, updating Developer Tools to the latest version, and integrating a WebCenter Sites instance with Eclipse. This chapter also provides a brief overview for managing WebCenter Sites resources in Eclipse. The last section in this chapter provides instructions for uninstalling Developer Tools.

This chapter contains the following sections:

- Section 52.1, "Prerequisites"
- Section 52.2, "Setting Up Developer Tools"
- Section 52.3, "Updating Developer Tools"
- Section 52.4, "Managing WebCenter Sites Resources in Eclipse"
- Section 52.5, "Uninstalling WebCenter Sites Developer Tools"

52.1 Prerequisites

Before setting up Developer Tools, keep in mind the following:

- After you have integrated Eclipse with WebCenter Sites, you must log in to WebCenter Sites with general administrator credentials (for example, `fwadmin/xceladmin`). This user must be a part of the `RestAdmin` group.
- To use the command-line tool feature, you must have an advanced knowledge of Developer Tools. Information and instructions about running and using the command-line tool are provided in Chapter 58, "Developer Tools: Command-Line Tool."

52.2 Setting Up Developer Tools

1. Download the Eclipse package that suits your development needs. The Eclipse IDE for Java EE Developers is recommended for development on WebCenter Sites. You can download Eclipse from the following URL:

   `http://www.eclipse.org/downloads/`

2. Unzip the `csdt.zip` archive, located in the WebCenter Sites distribution package. This archive contains the following folders:

   – `csdt-client`: command-line client

   – `csdt-eclipse`: Eclipse plug-in, which contains the archived Developer Tools plug-in (`com.fatwire.EclipseCSDT_11.1.1.v8_0_rxxxxxx.zip`).

3. In the `csdt-eclipse` folder, extract the Developer Tools plug-in (`com.fatwire.EclipseCSDT_11.1.1.v8_0_rxxxxxx.zip`) into a folder on your file system (for example, `${USER_HOME}`).

4. Install the Developer Tools plug-in:

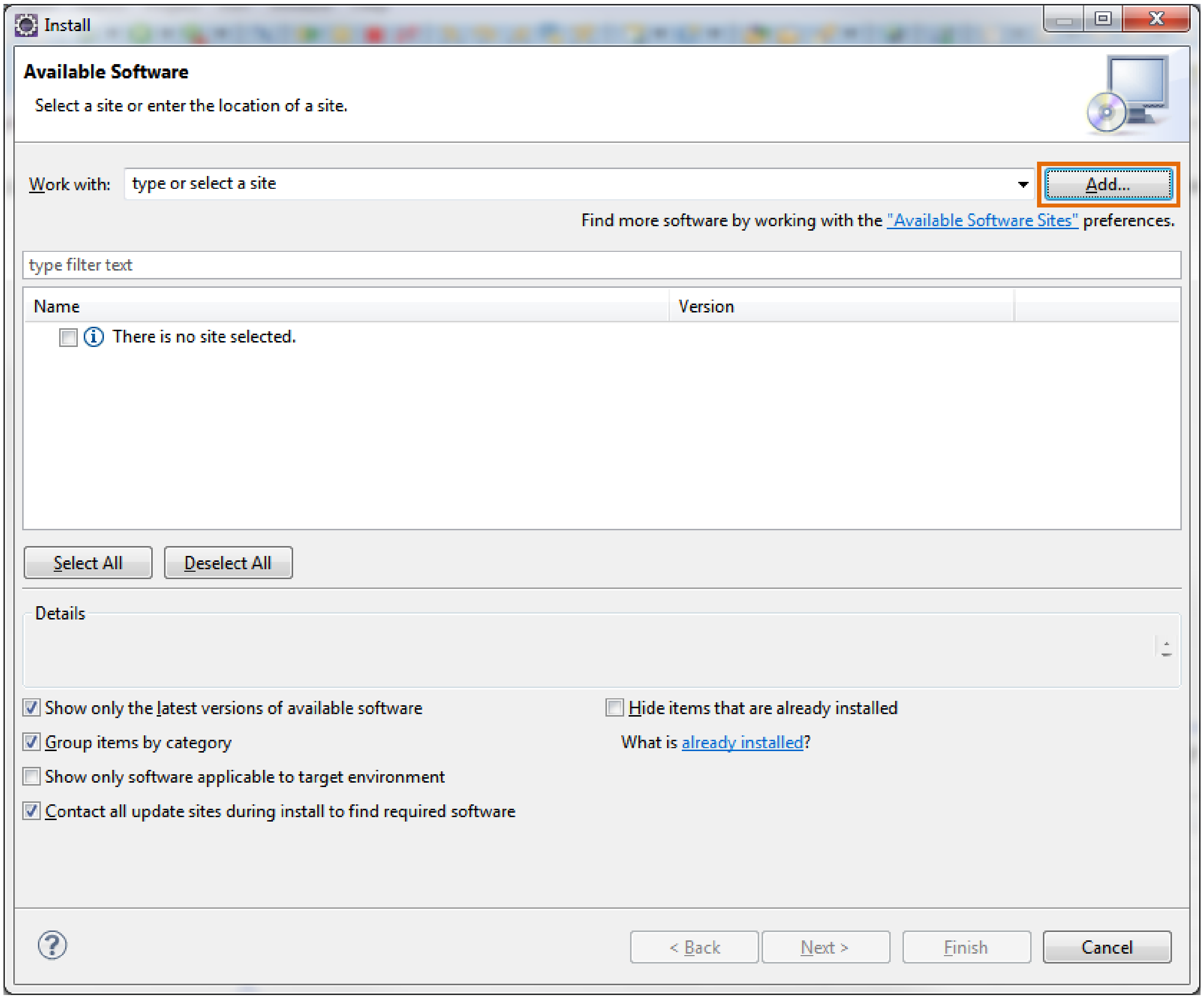
   Note: Eclipse installs plug-ins from local file system (or archive) locations and remote locations. The location of a plug-in is referred to as a "software site." For the Developer Tools plug-in, the "software site" is the path to the archived Developer Tools plug-in on the file system (which was extracted in step 3).

   a. Start Eclipse (execute `eclipse.exe`).

   b. In the Eclipse menu, select **Help**, then select **Install New Software**.

   The installation wizard opens and displays the "Available Software" screen (shown in Figure 52–1).

Figure 52–1 Installation Wizard: 'Available Software' Screen



Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
□ There is no site selected.	

Details

☑ Show only the latest versions of available software

☑ Group items by category

□ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

□ Hide items that are already installed

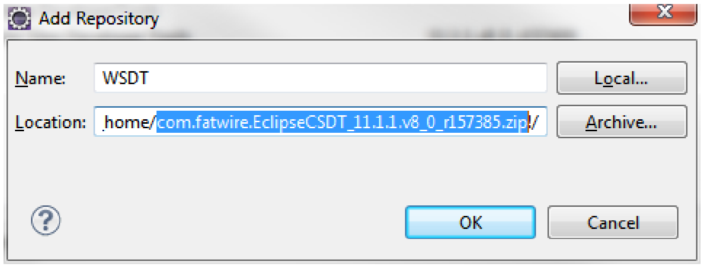
What is already installed?

   c. In the "Available Software" screen, click **Add** (located to the right of the "Work with" field).

d. In the "Add Repository" dialog box (shown in Figure 52–2), fill in the following fields and then click **OK**.

- In the **Name** field, enter a unique name for the Developer Tools plug-in.
- In the **Location** field, click **Archive...** to specify the location (software site) of the Developer Tools plug-in archive (extracted in step 3).

Figure 52–2 'Add Repository' Dialog Box



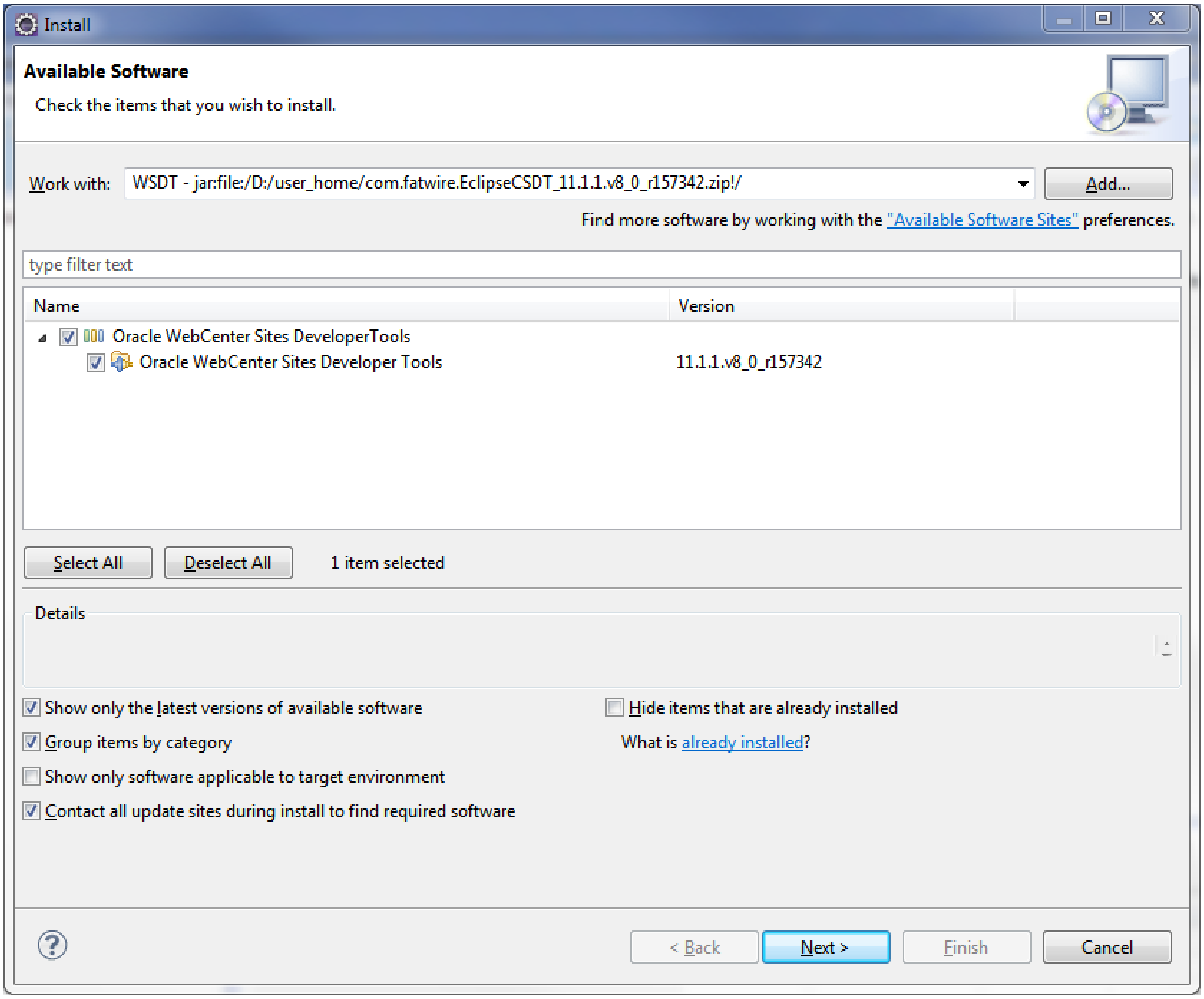
Name: WSDT

Location: home/com.fatwire.EclipseCSDT_11.1.1.v8_0_r157385.zip!/

The installation wizard picks up the Developer Tools plug-in from the specified location (software site), and lists the name of the plug-in in the "Available Software" screen.

e. In the "Available Software" screen, select **Oracle WebCenter Sites Developer Tools** (as shown in Figure 52–3), then click **Next**.

Figure 52–3 Installation Wizard: 'Available Software' Screen Listing the Developer Tools Plug-In



Install
Available Software
Check the items that you wish to install.

Work with: WSDT - jar:file:/D:/user_home/com.fatwire.EclipseCSDT_11.1.1.v8_0_r157342.zip!/

Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text

Name	Version
☑ Oracle WebCenter Sites DeveloperTools	
☑ Oracle WebCenter Sites Developer Tools	11.1.1.v8_0_r157342

Select All
Deselect All
1 item selected
Details

☑ Show only the latest versions of available software

☐ Hide items that are already installed

☑ Group items by category

What is already installed?

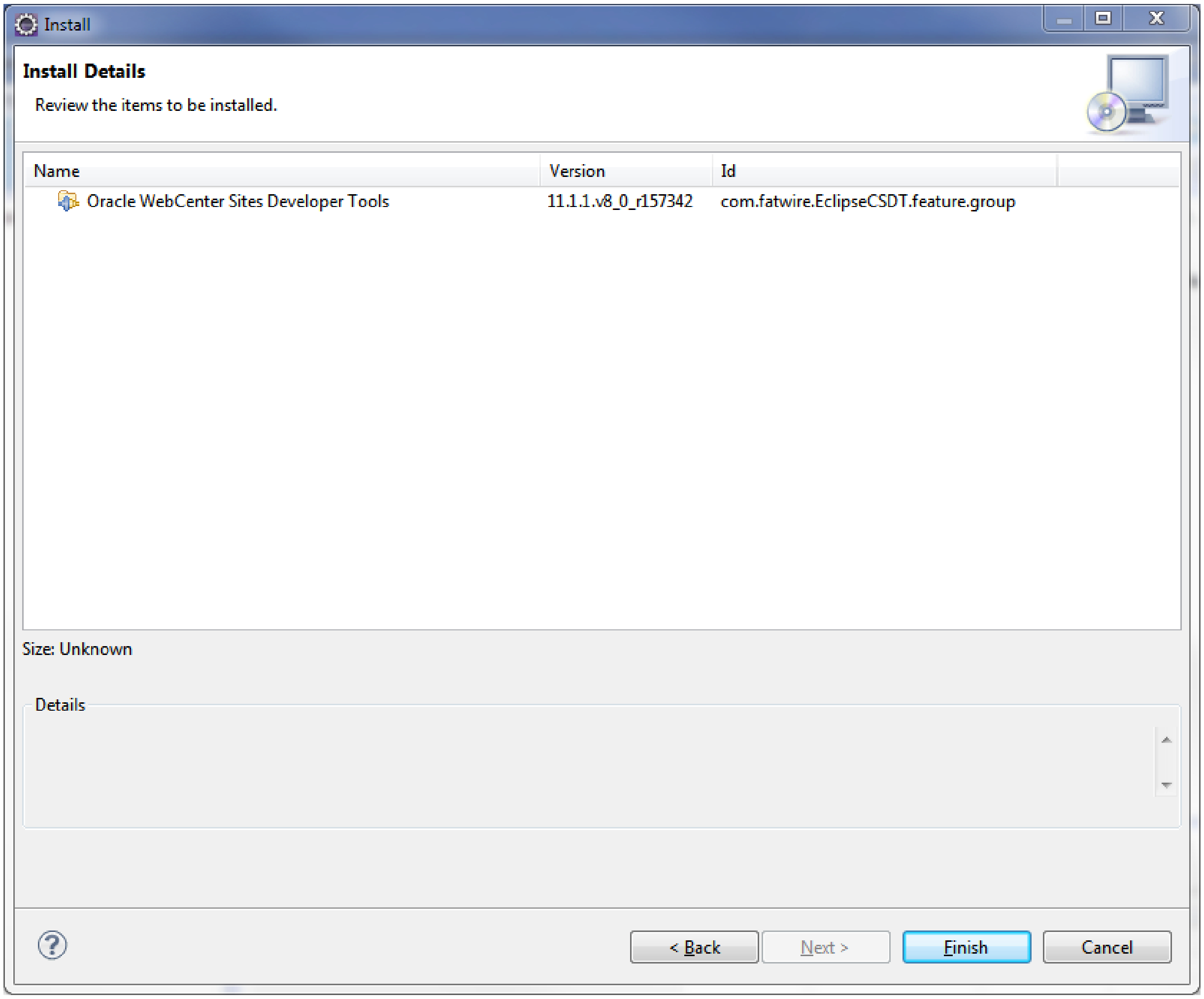
☐ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

The "Install Details" screen opens.

f. In the "Install Details" screen (shown in Figure 52–4), click **Finish**.

Figure 52–4 Installation Wizard: 'Install Details' Window



Install

Install Details

Review the items to be installed.

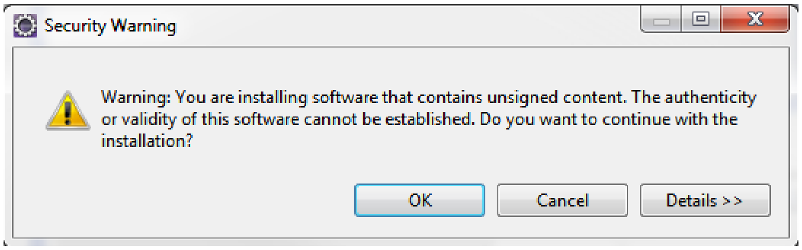
Name	Version	Id
Oracle WebCenter Sites Developer Tools	11.1.1.v8_0_r157342	com.fatwire.EclipseCSDT.feature.group

Size: Unknown

Details

g. During the installation process, a "Security Warning" dialog box opens (as shown in Figure 52–5). Click **OK** to continue with the installation.

Figure 52–5 'Security Warning' Dialog Box



Security Warning

Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?

h. Once the installation completes successfully, the "Software Updates" dialog box opens, as shown in Figure 52–6. Do the following:

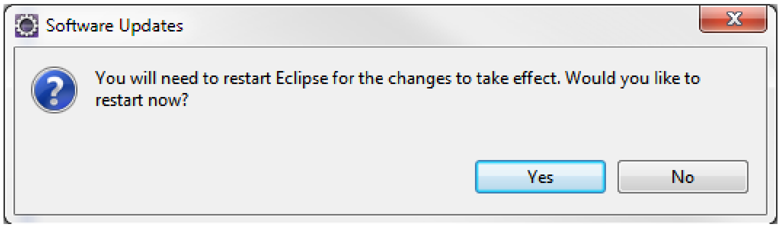
a. Start your local WebCenter Sites instance.

To ensure WebCenter Sites is running, access `${base_url}/HelloCS`, where `${base_url}` is the base URL for your WebCenter Sites instance. For example, if your base URL is `http://example:8080/cs`, then `http://example:8080/cs/HelloCS` is the verification URL for your WebCenter Sites instance.

b. Restart Eclipse by clicking **Yes** in the "Software Updates" dialog box.

Restarting Eclipse refreshes the plug-in cache and makes the Developer Tools plug-in available in Eclipse.

Figure 52–6 'Software Updates' Dialog Box

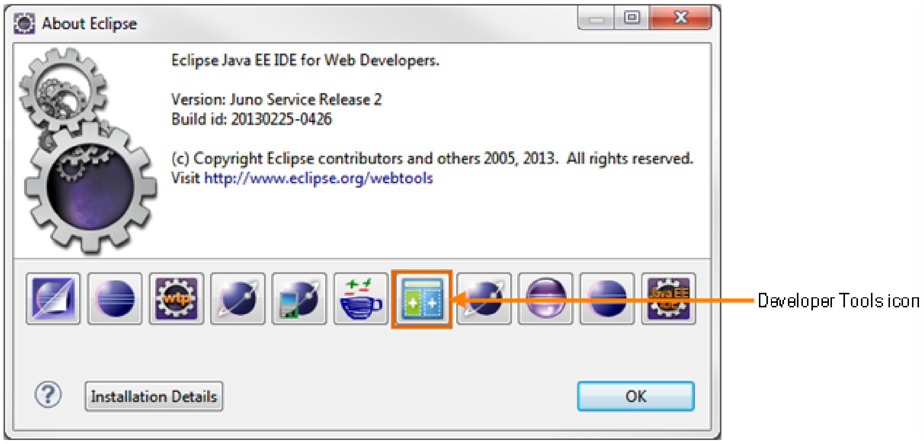


You will need to restart Eclipse for the changes to take effect. Would you like to restart now?

i. Once Eclipse restarts, verify that the Developer Tools plug-in is installed successfully. In the Eclipse menu, select **Help**, then select **About Eclipse**.

The "About Eclipse" dialog box opens and displays the **Developer Tools** icon in the list of installed plug-ins, as shown in Figure 52–7.

Figure 52–7 'About Eclipse' Dialog Box



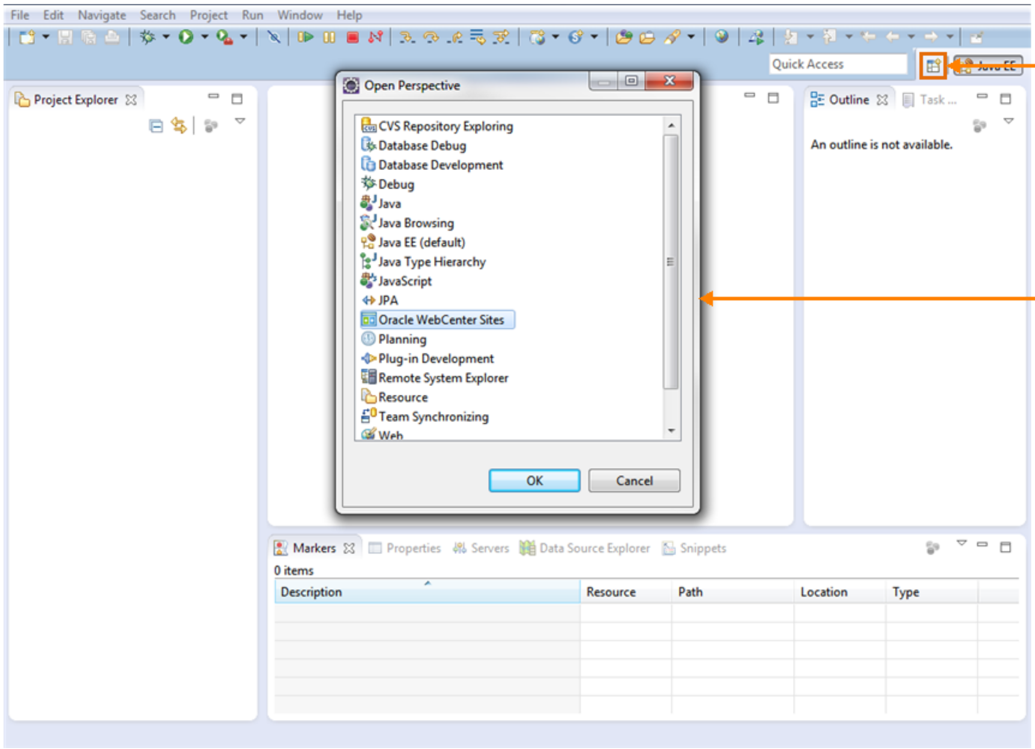
Eclipse Java EE IDE for Web Developers.

Version: Juno Service Release 2
Build id: 20130225-0426

(c) Copyright Eclipse contributors and others 2005, 2013. All rights reserved.
Visit http://www.eclipse.org/webtools

5. Open the "Oracle WebCenter Sites" perspective by clicking the **Open Perspective** icon, located on the top right of the Eclipse IDE. In the "Open Perspective" dialog, select **Oracle WebCenter Sites** (as shown in Figure 52–8), then click **OK**.

Figure 52–8 Open Perspective: Oracle WebCenter Sites



Open Perspective icon

"Open Perspective" window

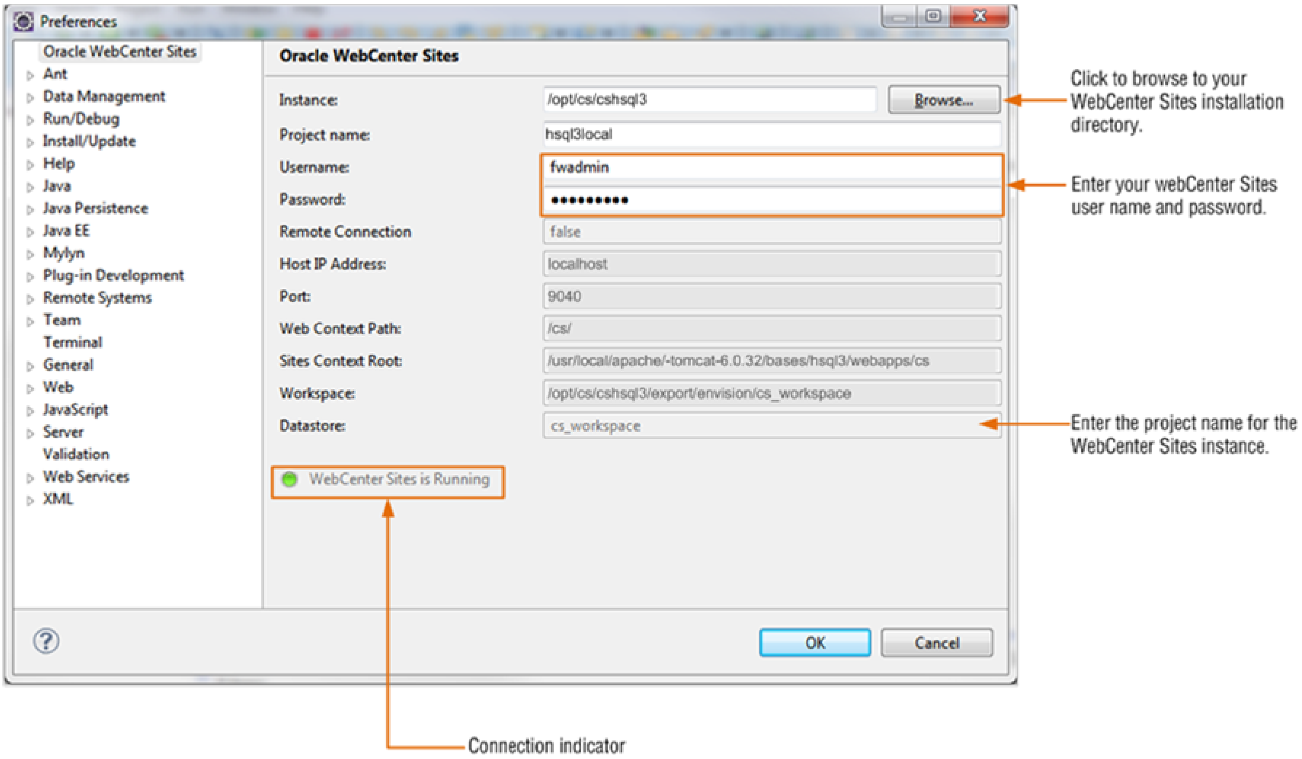
6. Integrate WebCenter Sites with the Eclipse IDE:

   – If you are setting up Developer Tools for the first time, the configuration form is automatically displayed

   – If you have already connected to a WebCenter Sites instance and wish to connect to a different instance, navigate to the WebCenter Sites toolbar and click the **Select Configuration** wizard to open the configuration form.

   In the configuration form (shown in Figure 52–9), fill in the following fields with the information for the WebCenter Sites instance you wish to connect to:

Figure 52–9 Configuration Form



Oracle WebCenter Sites

Instance: /opt/cs/cshsql3

Project name: hsql3local

Username: fwadmin

Password: •••••••••

Remote Connection: false

Host IP Address: localhost

Port: 9040

Web Context Path: /cs/

Sites Context Root: /usr/local/apache/-tomcat-6.0.32/bases/hsql3/webapps/cs

Workspace: /opt/cs/cshsql3/export/envision/cs_workspace

Datastore: cs_workspace

WebCenter Sites is Running

Click to browse to your WebCenter Sites installation directory.

Enter your webCenter Sites user name and password.

Enter the project name for the WebCenter Sites instance.

Connection indicator

a. In the "Instance" field, do one of the following, depending on whether you are connecting to a local or remote host:

Note: A remote connection supports the creation of multiple projects on the same remote host. Local connections support only one project at a time.

- If you are connecting to a local host, click **Browse** to select the directory containing the `futuretense.ini` file for the WebCenter Sites instance.
- If you are connecting to a remote host (for example, `example:8080/cs`), enter the path to the host in the format `[host]:[port]/[path]`.

Once you enter a value in the "Instance" field, the following fields are automatically populated. The values of these fields are based on whether you specified a local or remote host in the "Instance" field:

- **Remote Connection**: If you are connecting to a remote host, the value is set to `true`. If you are connecting to a local host, the value is set to `false`.
- **Host IP Address**: Specifies the IP address of the host to which you are connecting.
- **Port**: Specifies the port number of the host to which you are connecting.
- **Web Context Path**: Specifies the context path to the host.
- **Workspace**: Specifies the location of the Developer Tools workspace (Eclipse project).

  - **Sites Context Root**: If you are connecting to a local host, this field contains the path of the WebCenter Sites web application. If you are connecting to a remote host, this field is blank.

b. In the "Project name" field, enter a name for the project on which you will be working.

c. In the "Username" field, enter the user name of a general administrator. This user must be a member of the `RestAdmin` group.

d. In the "Password" field, enter the password for the user name you entered in step c.

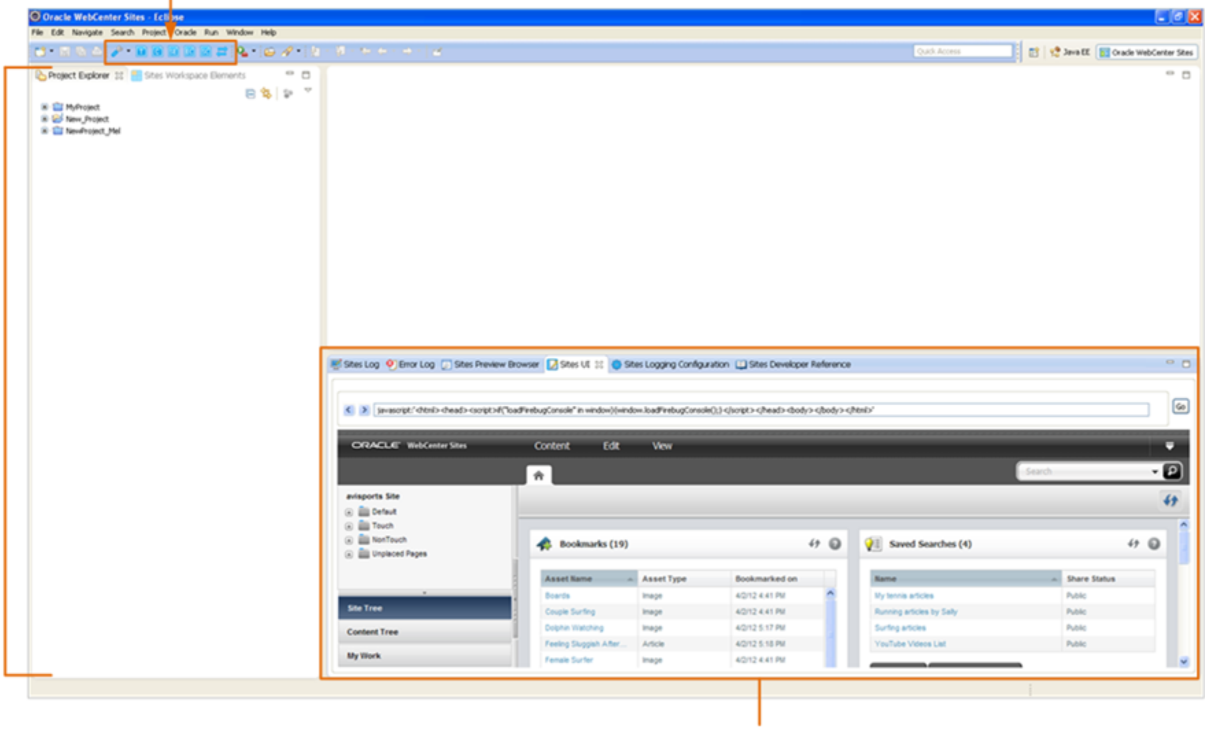
e. Click **OK**.

The "Oracle WebCenter Sites" perspective opens. If you are accessing the WebCenter Sites-integrated Eclipse for the first time, Figure 52–10 shows how the Oracle WebCenter Sites perspective will look.

Figure 52–10 Oracle WebCenter Sites Perspective

WebCenter Sites Toolbar, provides:

- Shortcut to the WebCenter Sites configuration screen
- Shortcuts to creating SiteEntry, CSElement, and Template assets
- Shortcuts for creating SiteCatalog and ElementCatalog entries
- Synchronization Tool



Left panel containing the **Project Explorer** and **Sites Workspace Elements**

Bottom panel views: **Sites Log**, **Sites Preview Browser**, **Sites UI**, **Sites Logging Configuration**, and **Sites Developer Reference**

If the Oracle perspective is rendered as shown in Figure 52–10, then you have successfully set up Developer Tools.

> **Note:** The panels containing the Eclipse and Developer Tools views are interchangeable. To move a view to a different panel, click the view's tab and drag it to the desired panel.

7. (Optional) Associate the *Oracle Fusion Middleware WebCenter Sites Tag Reference* and *Oracle Fusion Middleware WebCenter Sites Java API Reference* with Eclipse:

   For local hosts:

   a. Download the `TagReference.zip` and `javadoc.zip`.

   b. Create a folder named `developerdocs` under your server's context path. For example, if your server is on `myhost:8080/cs`, the path to the developerdocs folder should be `myhost:8080/cs/developerdocs`.

   c. Extract the `TagReference.zip` and `javadoc.zip` inside the `developerdocs` folder.

   d. In Eclipse, open the "Sites Developer Reference" view. In both the "Tag Reference" and "Javadoc" tabs, click **Home**. The *Oracle Fusion Middleware WebCenter Sites Tag Reference* and *Oracle Fusion Middleware WebCenter Sites Java API Reference* are displayed in their respective tabs.

   For remote hosts:

   a. Download the `TagReference.zip` and `javadoc.zip`.

   b. Create a folder named `developerdocs` under your server's context path. For example, if your server is on `example:8080/cs`, the path to the `developerdocs` folder should be `example:8080/cs/developerdocs`.

   c. Extract the `TagReference.zip` and `javadoc.zip` inside the `developerdocs` folder. The "Tag Reference" and "Javadoc" are now accessible to all users who connect to this server.

   d. In Eclipse, open the "Sites Developer Reference" view. In both the "Tag Reference" and "Javadoc" tabs, click **Home**. The *Oracle Fusion Middleware WebCenter Sites Tag Reference* and *Oracle Fusion Middleware WebCenter Sites Java API Reference* are displayed in their respective tabs.

8. (For remote hosts) Enable code completion, copy the `WEB-INF` directory from the remote computer into your Eclipse project. This enables Eclipse to perform tag completion and proper syntax highlighting. For more information, see Section 54.2, "Tag and Java API Completion."

   > **Note:** Code completion is enabled automatically for local instances.

9. If you upgraded your WebCenter Sites system, and wish to use Developer Tools to work with resources created prior to this release (existing resources), see Section 56.4.3, "Using Developer Tools with Pre-Existing Resources."

10. To quickly get started with managing WebCenter Sites resources in the Oracle WebCenter Sites perspective, continue to the next section. Start with step 4 in Section 52.4, "Managing WebCenter Sites Resources in Eclipse."

52.3 Updating Developer Tools

If you have upgraded your WebCenter Sites installation, follow the steps in this section to update your Developer Tools plug-in.

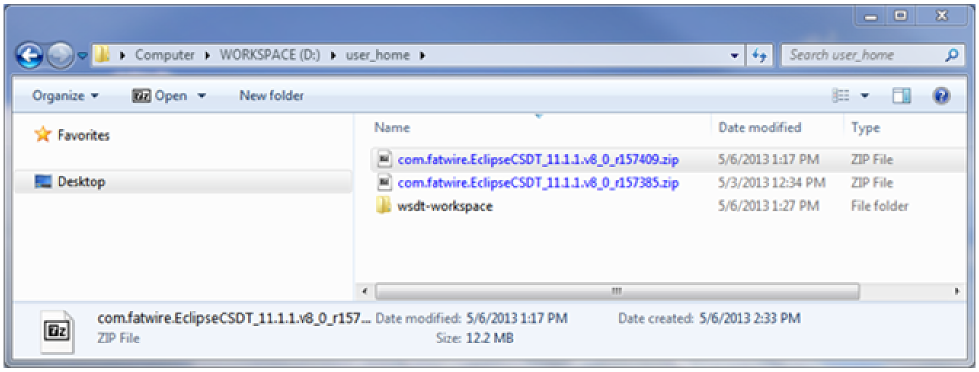
To update the Developer Tools plug-in

1. Extract the updated Developer Tools plug-in archive into a folder on your local file system.

   Figure 52–11, shows an example of the updated Developer Tools plug-in archive (`com.fatwire.EclipseCSDT_11.1.1.v8_0_r157409.zip`) extracted into the same folder as the currently installed Developer Tools plug-in archive (`com.fatwire.EclipseCSDT_11.1.1.v8_0_r157385.zip`).

   > **Note:** To ensure you have the latest version of Developer Tools, note the version number (*rxxxxxx*) appended to the archive's name.

   Figure 52–11 Updated Plug-In on File System



Name	Date modified	Type
com.fatwire.EclipseCSDT_11.1.1.v8_0_r157409.zip	5/6/2013 1:17 PM	ZIP File
com.fatwire.EclipseCSDT_11.1.1.v8_0_r157385.zip	5/3/2013 12:34 PM	ZIP File
wsdt-workspace	5/6/2013 1:27 PM	File folder

com.fatwire.EclipseCSDT_11.1.1.v8_0_r157... Date modified: 5/6/2013 1:17 PM Date created: 5/6/2013 2:33 PM
ZIP File Size: 12.2 MB

2. Start Eclipse (execute `eclipse.exe`).

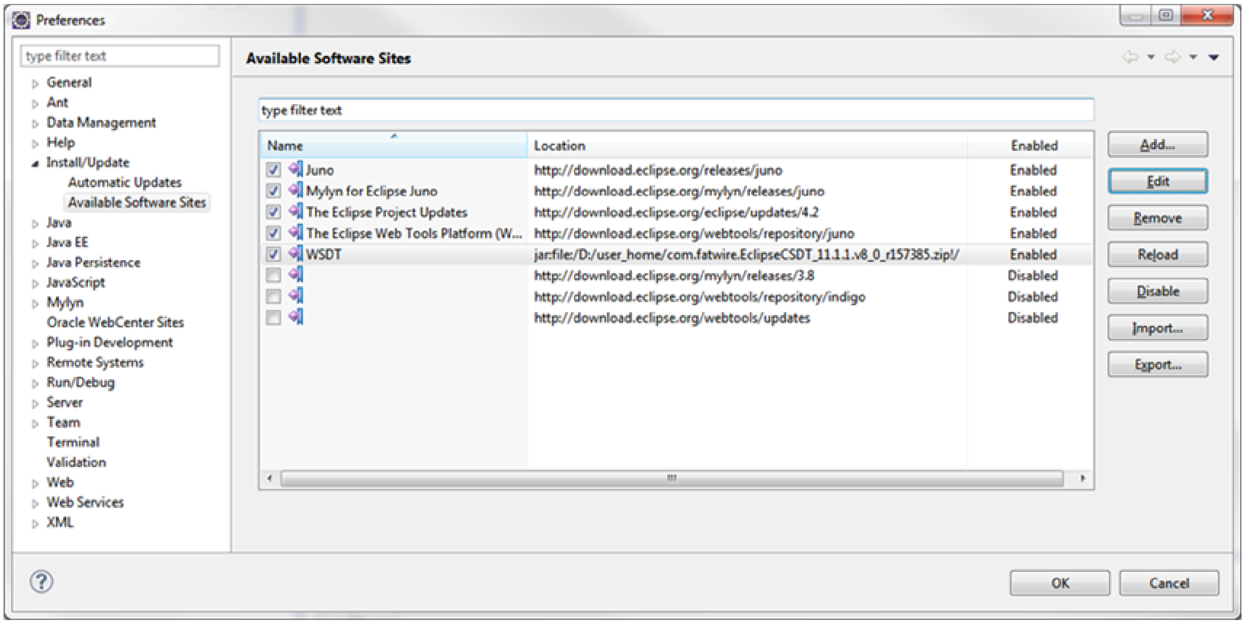
3. Update the location (software site) of the Developer Tools plug-in:

   a. In the Eclipse menu, select **Window**, then select **Preferences**.

      The "Preferences" window opens.

   b. In the "Preferences" window, select **Install/Update**, then select **Available Software Sites**. In the "Available Software Sites" list, select the Developer Tools entry (**WSDT** entry in Figure 52–12), and then click **Edit**.

Figure 52–12 Preferences Window: Edit the Developer Tools (WSDT) Entry

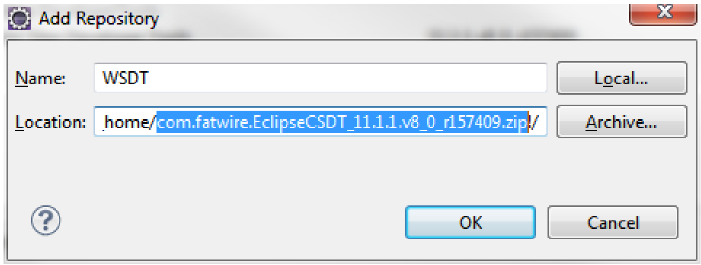


Name	Location	Enabled
☑ Juno	http://download.eclipse.org/releases/juno	Enabled
☑ Mylyn for Eclipse Juno	http://download.eclipse.org/mylyn/releases/juno	Enabled
☑ The Eclipse Project Updates	http://download.eclipse.org/eclipse/updates/4.2	Enabled
☑ The Eclipse Web Tools Platform (W...	http://download.eclipse.org/webtools/repository/juno	Enabled
☑ WSDT	jar:file:/D:/user_home/com.fatwire.EclipseCSDT_11.1.1.v8_0_r157385.zip!/	Enabled
☐	http://download.eclipse.org/mylyn/releases/3.8	Disabled
☐	http://download.eclipse.org/webtools/repository/indigo	Disabled
☐	http://download.eclipse.org/webtools/updates	Disabled

The "Add Repository" dialog box opens.

c. In the "Location" field, click **Archive...** to specify the location of the updated plug-in (extracted in step 1), then click **OK** (as shown in Figure 52–13).

Figure 52–13 'Add Repository' Dialog Box



Name: WSDT

Location: home/com.fatwire.EclipseCSDT_11.1.1.v8_0_r157409.zip!/

The "Add Repository" window closes and the "Available Software Sites" list in the "Preferences" window displays the updated path to the Developer Tools entry.

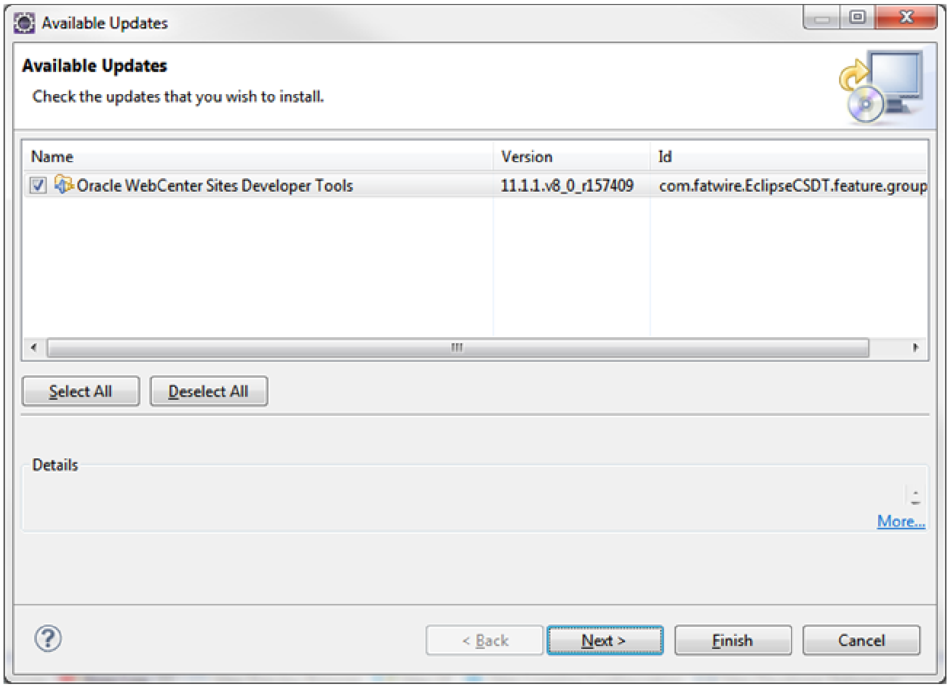
d. In the "Preferences" window, click **OK**.

4. Check for updates to existing plug-ins.

   a. In the Eclipse menu, select **Help**, then select **Check for Updates**.

   The "Available Updates" window opens, as shown in Figure 52–14.

Figure 52–14 'Available Updates' Window



Available Updates

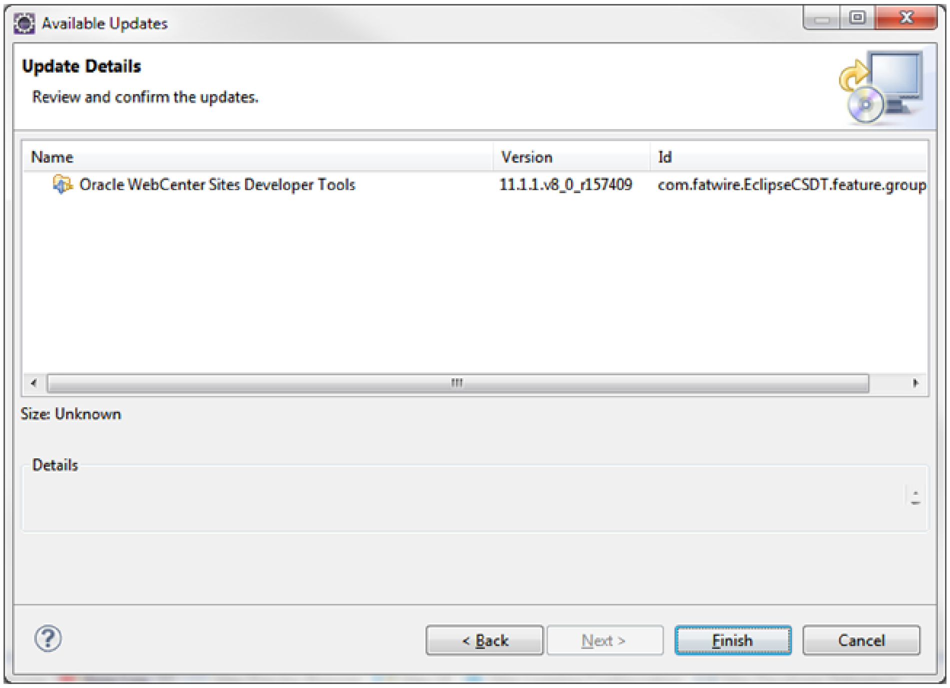
Check the updates that you wish to install.

Name	Version	Id
☑ Oracle WebCenter Sites Developer Tools	11.1.1.v8_0_r157409	com.fatwire.EclipseCSDT.feature.group

b. Select **Oracle WebCenter Sites Developer Tools**, then click **Next**.

The "Update Details" window opens, as shown in Figure 52–15.

Figure 52–15 'Update Details' Window



Update Details

Review and confirm the updates.

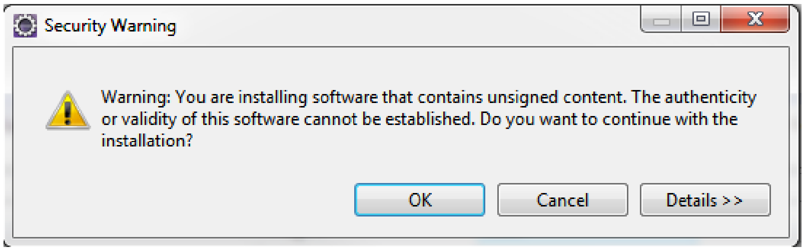
Name	Version	Id
Oracle WebCenter Sites Developer Tools	11.1.1.v8_0_r157409	com.fatwire.EclipseCSDT.feature.group

Size: Unknown

c. To proceed with the update, click **Finish**.

d. During the update process, a "Security Warning" dialog box opens, as shown in Figure 52–16. Click **OK** to continue the update.

Figure 52–16 'Security Warning' Dialog Box Displayed During the Update Process



Security Warning

Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?

e. Once the update completes successfully, the "Software Updates" dialog box opens. Do the following:

   a. Start your local WebCenter Sites instance.

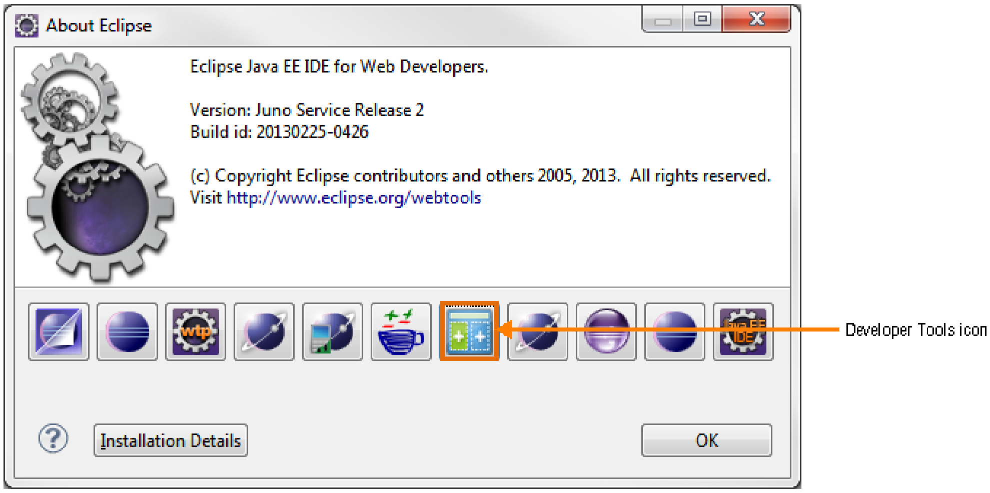
   b. Restart Eclipse by clicking **Yes** in the "Software Updates" dialog box.

f. Once Eclipse restarts, verify that the Developer Tools plug-in has been successfully updated to the latest version:

   a. In the Eclipse menu, select **Help**, then select **About Eclipse**.

      The "About Eclipse" dialog box opens and displays the Developer Tools icon in the list of installed plug-ins, as shown in Figure 52–17.

Figure 52–17 'About Eclipse' Dialog Box



About Eclipse

Eclipse Java EE IDE for Web Developers.

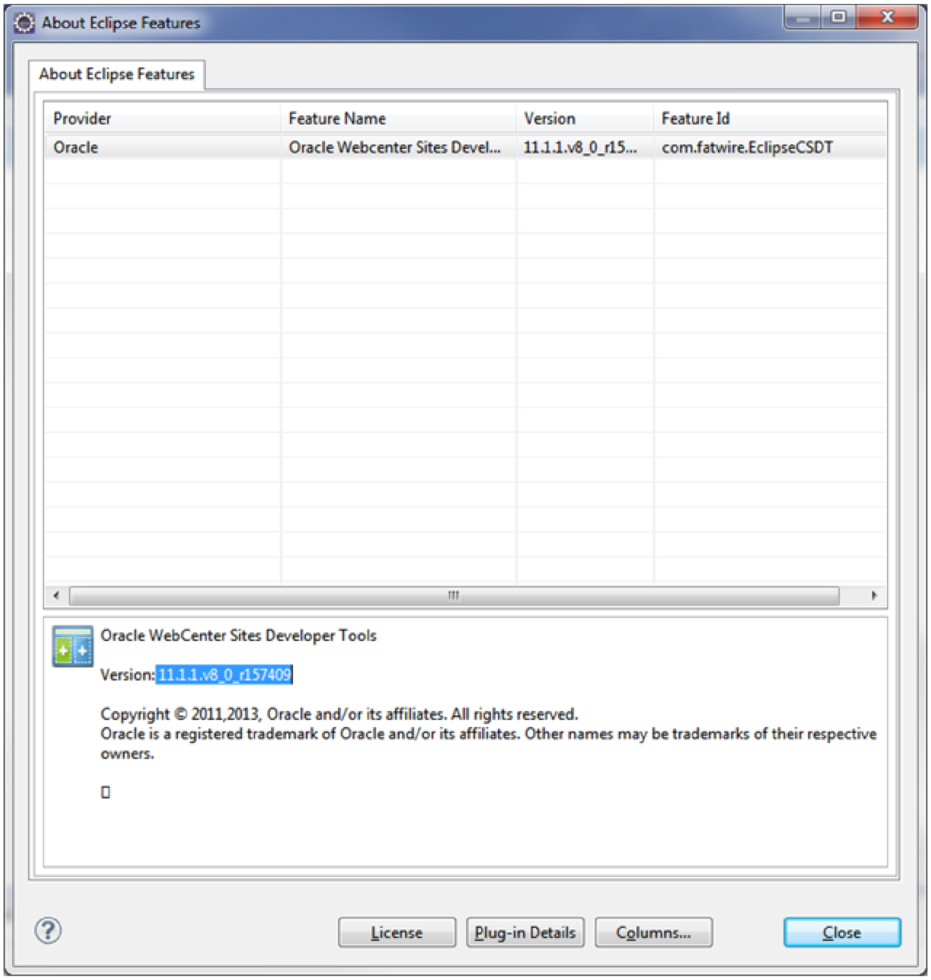
Version: Juno Service Release 2
Build id: 20130225-0426

(c) Copyright Eclipse contributors and others 2005, 2013. All rights reserved.
Visit http://www.eclipse.org/webtools

   b. In the "About Eclipse" dialog box, click the **Developer Tools** icon.

      The "About Eclipse Features" window opens, and displays the version details of the Developer Tools plug-in, as shown in Figure 52–18.

Figure 52–18 'About Eclipse Features' Window



Provider	Feature Name	Version	Feature Id
Oracle	Oracle Webcenter Sites Devel...	11.1.1.v8_0_r15...	com.fatwire.EclipseCSDT

Oracle WebCenter Sites Developer Tools

Version:11.1.1.v8_0_r157409

Copyright © 2011,2013, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

□

52.4 Managing WebCenter Sites Resources in Eclipse

This section takes you through the Eclipse Oracle WebCenter Sites perspective by summarizing the steps you would take to create, edit, and otherwise manage code-based WebCenter Sites resources:

- SiteEntry assets
- CSElement assets
- Template assets
- ElementCatalog Entries
- SiteCatalog Entries

To manage WebCenter Sites resources in Eclipse

1. Start your local WebCenter Sites.
2. Start Eclipse.
3. Open the "Oracle WebCenter Sites" perspective:

   In the Eclipse menu bar, select **Window**, then select **Open Perspective**, then **Other**, and then **Oracle WebCenter Sites**.

4. To create resources, do the following:

   – To create a SiteEntry asset, click the **New Site Entry** (**SE**) icon and fill in the forms.

   – To create a CSElement asset, click the **New CSElement** (**C**) icon and fill in the forms.

   – To create a Template asset, click the **New Template** (**T**) icon and fill in the forms.

   – To create an ElementCatalog entry, click the **New ElementCatalog Entry** (**EC**) icon and fill in the forms.

   – To create a SiteCatalog entry, click the **New SiteCatalog Entry** (**SC**) icon and fill in the forms.

   For field definitions, see Chapter 23, "Creating Template, CSElement, and SiteEntry Assets."

5. To manage the resources you create, edit, delete, or share with other sites use the "Sites Workshops Elements" view. Right-click the resource and select the desired option. For information about the available options, see Section 53.4.1, "Sites Work."

6. To display Developer Tools views in panels, do the following:

   a. Select **Window**, then **Show View**, and then **Other**.

   b. In the "Show View" dialog box, select the desired view (located under the **Oracle WebCenter Sites** folder):

      – **Sites UI** displays the embedded WebCenter Sites Admin and Contributor interfaces.

      – **Sites Log** displays the log file for WebCenter Sites.

      – **Sites Developer Reference** displays the *Oracle Fusion Middleware WebCenter Sites Tag Reference* and *Oracle Fusion Middleware WebCenter Sites Java API Reference*, only if you have associated the *Tag Reference* and *Javadoc* with your current WebCenter Sites instance. For instructions, see step 7 in Section 52.2, "Setting Up Developer Tools."

      – **Sites Workspace Elements** provides access to code-related resources. This view displays the resources in a tree and groups each resource according to its site affiliation.

      – **Sites Logging Configuration** displays a dynamically updating view of the log4j configuration. In this view you can set the log levels of each WebCenter Sites logger.

      – **Sites Preview Browser** displays an embedded preview browser.

   For more information about the Developer Tools views, see Section 53.4, "Developer Tools Views."

7. Synchronize WebCenter Sites resources by selecting the **Synchronize Data** icon. The synchronization tool enables you to either export data from WebCenter Sites to the Developer Tools workspace or import data to WebCenter Sites from your Developer Tools workspace.

   – For a quick overview of using the synchronization tool, see Section 53.5, "Data Synchronization (Export/Import) Tool."

– For detailed information about synchronizing resources, see Chapter 56, "Developer Tools: Synchronization and Data Exchange."

52.5 Uninstalling WebCenter Sites Developer Tools

This section contains instructions for uninstalling Developer Tools.

To uninstall Developer Tools

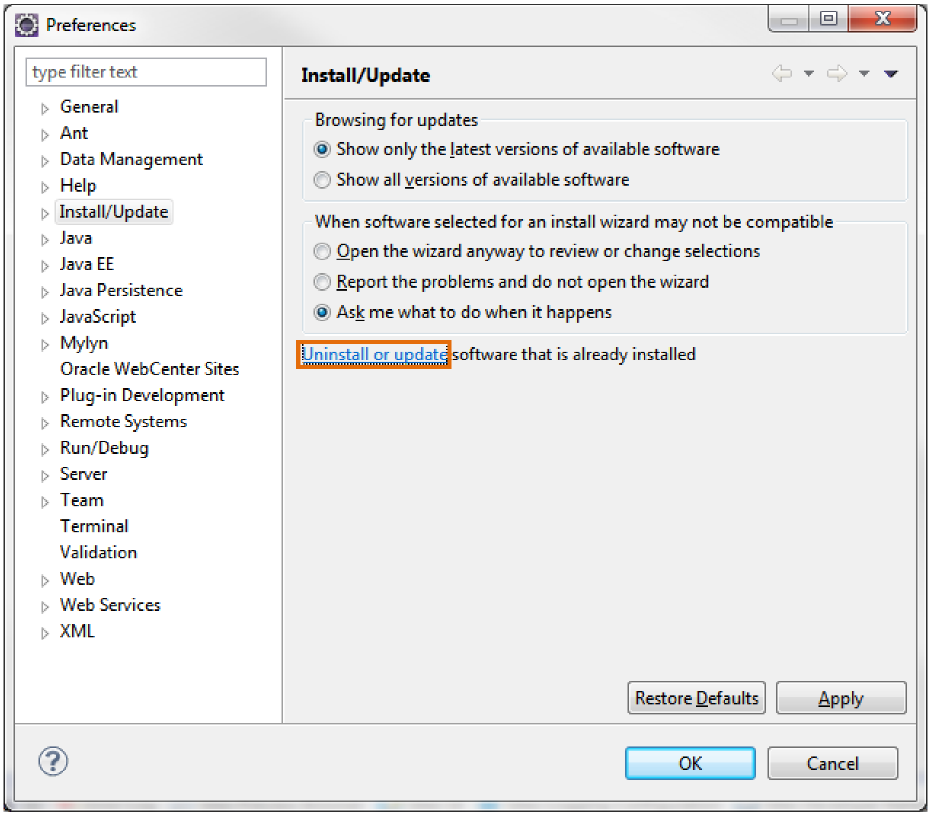
1. Close the "Oracle WebCenter Sites Perspective" (along with any views associated with the "Oracle WebCenter Sites" perspective).

2. In the Eclipse menu, select **Window**, then select **Preferences**.

   The "Preferences" window opens.

3. In the "Preferences" window, select **Install/Update**, then click the **Uninstall or update** link (as shown in Figure 52–19).

Figure 52–19 Preferences: 'Install/Update' Screen



Install/Update

Browsing for updates

● Show only the latest versions of available software

○ Show all versions of available software

When software selected for an install wizard may not be compatible

○ Open the wizard anyway to review or change selections

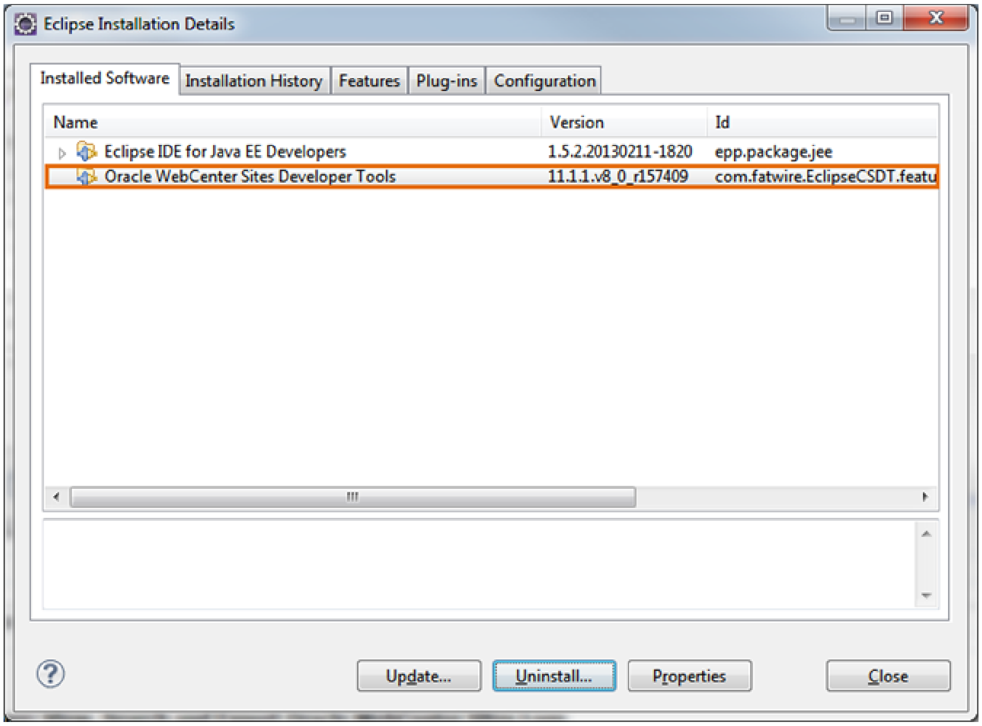
○ Report the problems and do not open the wizard

● Ask me what to do when it happens

Uninstall or update software that is already installed

4. In the "Eclipse Installation Details" screen, select the Developer Tools plug-in and then click **Uninstall...** (as shown in Figure 52–20).

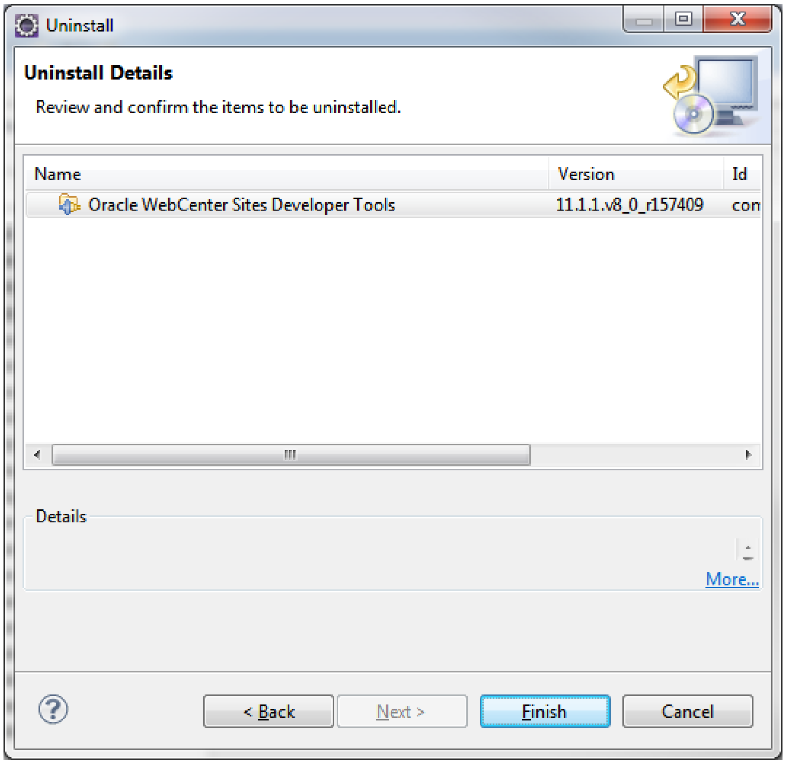
Figure 52–20 'Eclipse Installation Details' Screen



Name	Version	Id
Eclipse IDE for Java EE Developers	1.5.2.20130211-1820	epp.package.jee
Oracle WebCenter Sites Developer Tools	11.1.1.v8_0_r157409	com.fatwire.EclipseCSDT.featu

5. In the "Uninstall Details" screen (shown in Figure 52–21), click **Finish** to proceed with the process of uninstalling the Developer Tools plug-in.

Figure 52–21 'Uninstall Details' Screen



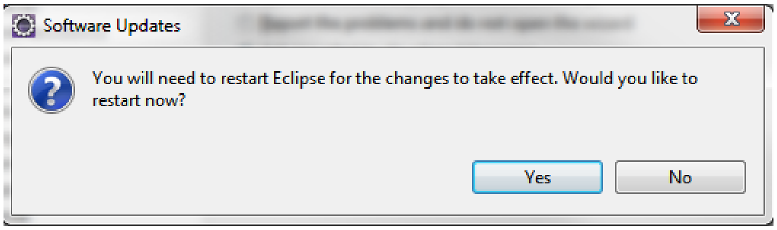
Uninstall Details

Review and confirm the items to be uninstalled.

Name	Version	Id
Oracle WebCenter Sites Developer Tools	11.1.1.v8_0_r157409	con

6. Once the update completes successfully, the "Software Updates" dialog box opens, as shown in Figure 52–22. Click **Yes** to restart Eclipse.

Figure 52–22 'Software Updates' Dialog Box



You will need to restart Eclipse for the changes to take effect. Would you like to restart now?

Restarting Eclipse refreshes the plug-in cache and completely removes the Developer Tools plug-in from Eclipse.

53

Developer Tools: WebCenter Sites Features in Eclipse

This chapter provides information about the Developer Tools functionality made available in the WebCenter Sites-integrated Eclipse IDE.

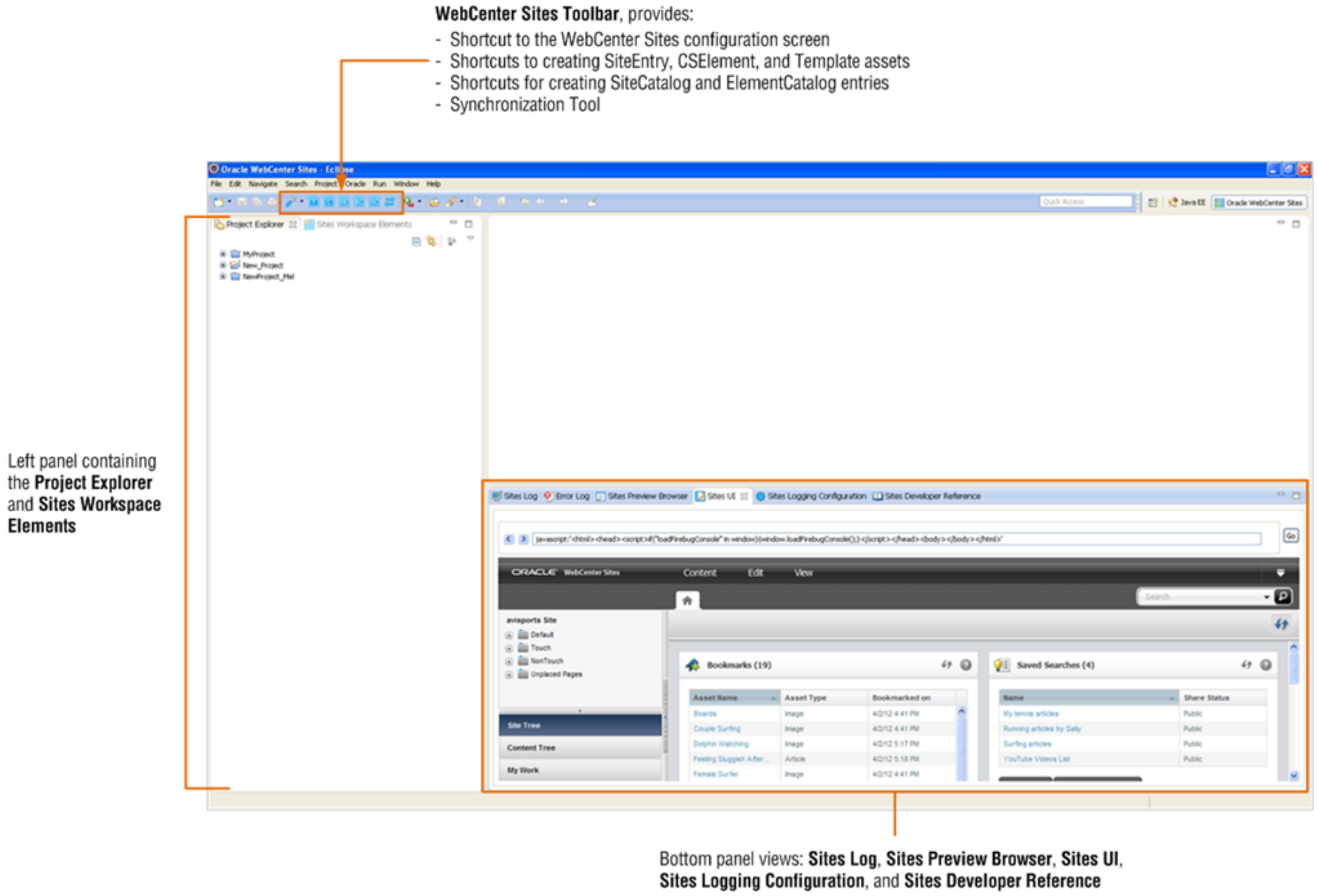
This chapter contains the following sections:

- Section 53.1, "Oracle WebCenter Sites Perspective"
- Section 53.2, "Configuration Screen"
- Section 53.3, "Project and Workspace in Eclipse"
- Section 53.4, "Developer Tools Views"
- Section 53.5, "Data Synchronization (Export/Import) Tool"
- Section 53.6, "Next Steps"

53.1 Oracle WebCenter Sites Perspective

All Developer Tools functionality in Eclipse is grouped under the Oracle WebCenter Sites perspective. Select **Window**, then select **Open Perspective**, then **Other**, and then **Oracle WebCenter Sites**.

Figure 53–1 Oracle WebCenter Sites Perspective



WebCenter Sites Toolbar, provides:

- Shortcut to the WebCenter Sites configuration screen
- Shortcuts to creating SiteEntry, CSElement, and Template assets
- Shortcuts for creating SiteCatalog and ElementCatalog entries
- Synchronization Tool

Left panel containing the **Project Explorer** and **Sites Workspace Elements**

Bottom panel views: **Sites Log**, **Sites Preview Browser**, **Sites UI**, **Sites Logging Configuration**, and **Sites Developer Reference**

javascript:'<html><head><script>if("loadFirebugConsole" in window){window.loadFirebugConsole();}</script></head><body></body></html>'

Bookmarks (19)

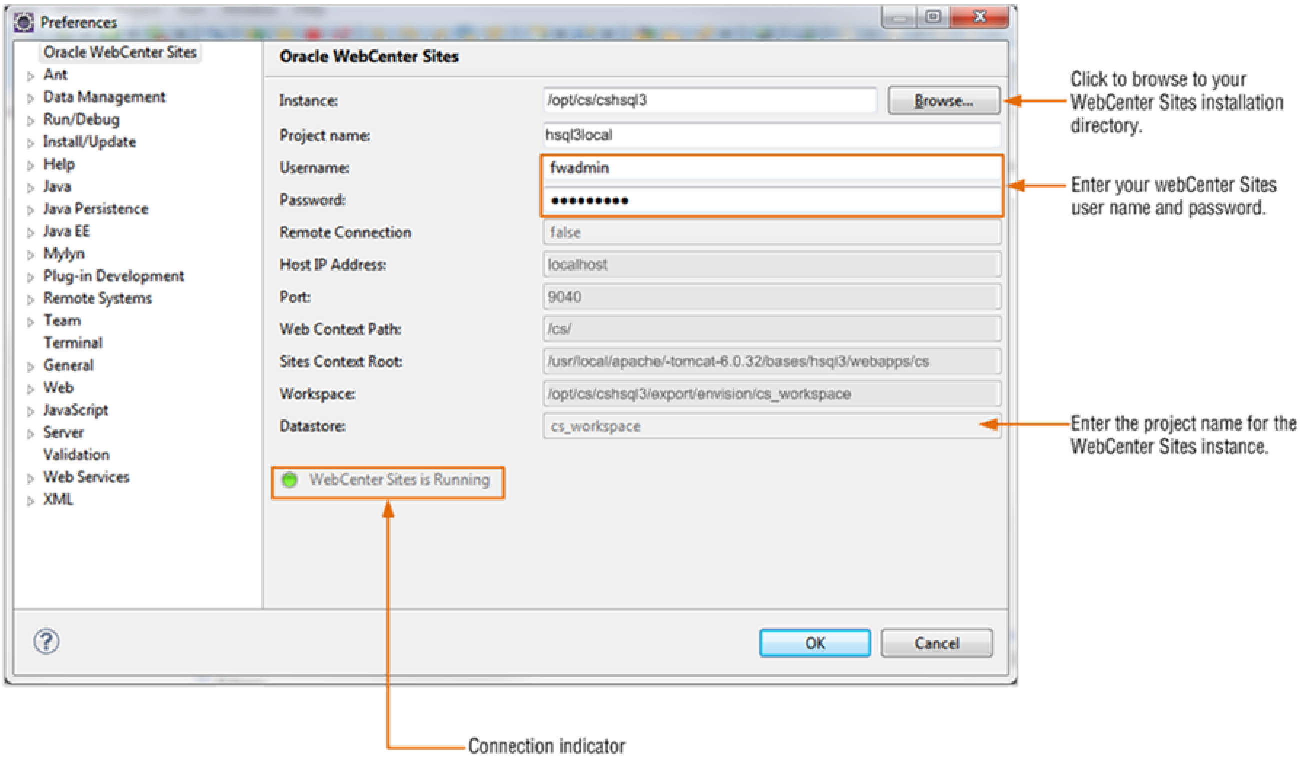
Asset Name	Asset Type	Bookmarked on
Boards	Image	4/2/12 4:41 PM
Couple Surfing	Image	4/2/12 4:41 PM
Dolphin Watching	Image	4/2/12 5:17 PM
Feeling Sluggish After...	Article	4/2/12 5:18 PM
Female Surfer	Image	4/2/12 4:41 PM

Saved Searches (4)

Name	Share Status
My tennis articles	Public
Running articles by Sally	Public
Surfing articles	Public
YouTube Videos List	Public

53.2 Configuration Screen

The configuration screen opens automatically on first access of WebCenter Sites-integrated Eclipse. On subsequent access the configuration screen can be opened by selecting the **Configuration** button on the WebCenter Sites toolbar. This screen enables you to specify the WebCenter Sites instance with which you wish to work.



Instance: /opt/cs/cshsql3

Project name: hsql3local

Username: fwadmin

Password: •••••••••

Remote Connection: false

Host IP Address: localhost

Port: 9040

Web Context Path: /cs/

Sites Context Root: /usr/local/apache/-tomcat-6.0.32/bases/hsql3/webapps/cs

Workspace: /opt/cs/cshsql3/export/envision/cs_workspace

Datastore: cs_workspace

● WebCenter Sites is Running

Click to browse to your WebCenter Sites installation directory.

Enter your webCenter Sites user name and password.

Enter the project name for the WebCenter Sites instance.

Connection indicator

The configuration screen requires the path to the WebCenter Sites installation directory, a WebCenter Sites user that is part of the `RestAdmin` group, and a project name for the WebCenter Sites instance. After you fill in all the required information, Developer Tools determines a number of other parameters for your WebCenter Sites instance and displays them for your information in read-only fields. In addition, the connection indicator shows whether Developer Tools is connected to the specified WebCenter Sites instance.

53.3 Project and Workspace in Eclipse

Each WebCenter Sites instance that is accessed through Eclipse is assigned an Eclipse project. The Eclipse project's folder is displayed in the "Project Explorer" view. The project is needed for tracking Developer Tools workspace items. Only one project is created by default for each WebCenter Sites instance and only one WebCenter Sites instance can be serviced by a project.

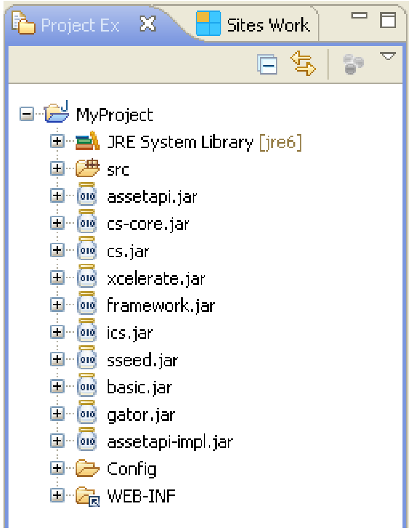
> **Note:** The main purpose of the project is to facilitate information tracking and process Eclipse events. Projects are managed by the Developer Tools plug-in. **Do not open, close, or modify the project.**

Each Eclipse project includes the following elements:

- `src`: The Developer Tools workspace folder for the current WebCenter Sites instance. This folder contains all the files for the resources stored in the Developer Tools workspace. The resources in this folder can be checked in to a version control system.
- `Config`: This folder contains links to common configuration files belonging to the current WebCenter Sites instance.

- `WEB-INF`: This folder contains links to the current WebCenter Sites instance's `WEB-INF` folder.

For example:



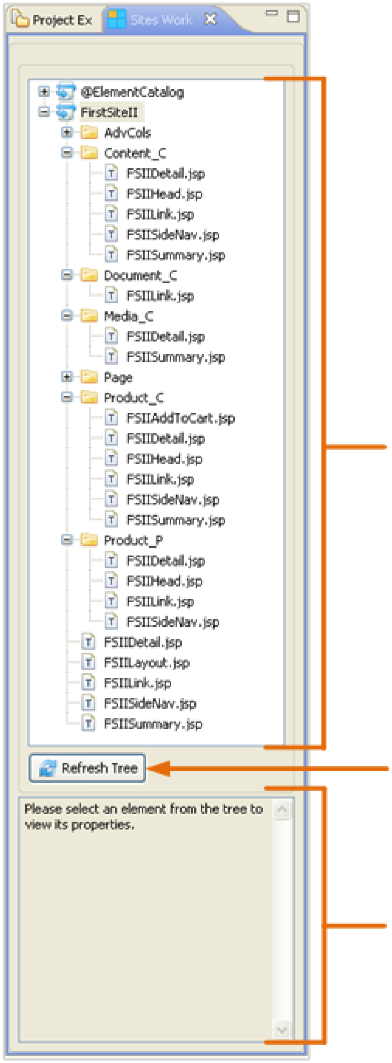
- MyProject
  - JRE System Library [jre6]
  - src
  - assetapi.jar
  - cs-core.jar
  - cs.jar
  - xcelerate.jar
  - framework.jar
  - ics.jar
  - sseed.jar
  - basic.jar
  - gator.jar
  - assetapi-impl.jar
  - Config
  - WEB-INF

53.4 Developer Tools Views

- Section 53.4.1, "Sites Work"
- Section 53.4.2, "Sites Log View"
- Section 53.4.3, "Sites Preview Browser View"
- Section 53.4.4, "Sites UI View"
- Section 53.4.5, "Sites Logging Configuration View"
- Section 53.4.6, "Sites Developer Reference View"
- Section 53.4.7, "Wizards"

53.4.1 Sites Work

Provides access to code-related resources. The resources are grouped according to their site affiliation. When you select a resource, a quick summary of that resource is shown in the text box at the bottom of the view.



Please select an element from the tree to view its properties.

Any resource you create or modify with Developer Tools is listed in this tree, along with its associated metadata, dependencies, and JSP and XML files.

Click this button to refresh the view.

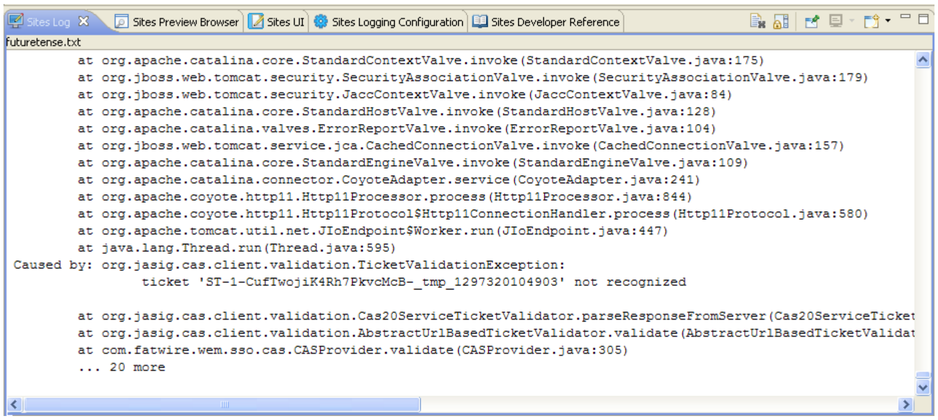
When you select an item from the tree, a summary of that item is displayed in this text box.

Right-click a resource in the tree to view the available management options. The options that are displayed to you depend on the resource you select:

- **Show Metadata**: Shortcut to the `.main.xml` file, which contains the metadata of the selected item.
- **Site Entry**: View the resource's site entry, share the site entry with other sites, create a new site entry, and delete a site entry.
- **Share**: Manage the sites with which this resource is associated.
- **Properties**: Manage properties of this resource, such as cache criteria and default arguments.
- **Delete**: Delete this resource.

53.4.2 Sites Log View

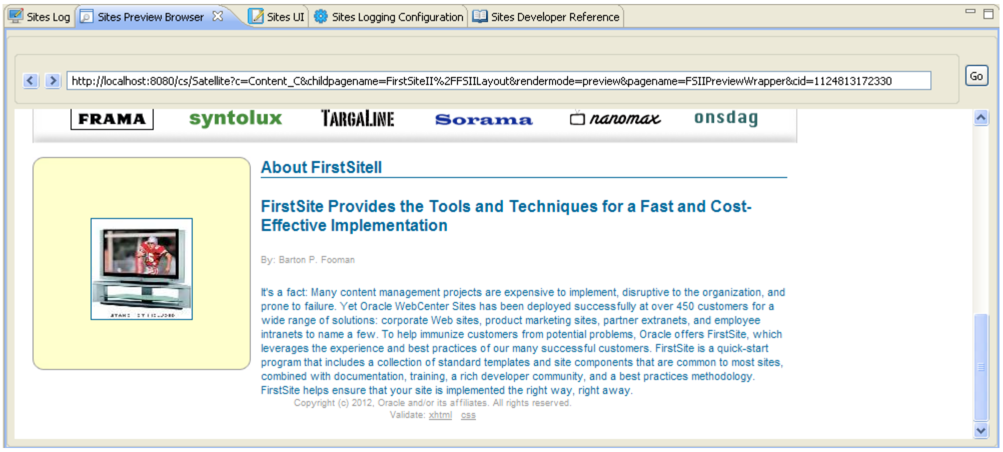
Shows a dynamically updating record of the WebCenter Sites log file. This view can be used to monitor the behavior of your Eclipse-integrated WebCenter Sites instance.



```
        at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:175)
        at org.jboss.web.tomcat.security.SecurityAssociationValve.invoke(SecurityAssociationValve.java:179)
        at org.jboss.web.tomcat.security.JaccContextValve.invoke(JaccContextValve.java:84)
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:128)
        at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:104)
        at org.jboss.web.tomcat.service.jca.CachedConnectionValve.invoke(CachedConnectionValve.java:157)
        at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:109)
        at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:241)
        at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:844)
        at org.apache.coyote.http11.Http11Protocol$Http11ConnectionHandler.process(Http11Protocol.java:580)
        at org.apache.tomcat.util.net.JIoEndpoint$Worker.run(JIoEndpoint.java:447)
        at java.lang.Thread.run(Thread.java:595)
Caused by: org.jasig.cas.client.validation.TicketValidationException:
                ticket 'ST-1-CufTwojiK4Rh7PkvcMcB-_tmp_1297320104903' not recognized

        at org.jasig.cas.client.validation.Cas20ServiceTicketValidator.parseResponseFromServer(Cas20ServiceTicket
        at org.jasig.cas.client.validation.AbstractUrlBasedTicketValidator.validate(AbstractUrlBasedTicketValidat
        at com.fatwire.wem.sso.cas.CASProvider.validate(CASProvider.java:305)
        ... 20 more
```

53.4.3 Sites Preview Browser View

Provides a quick way to preview pages. To preview a web page with this view, enter the URL of the page in the address bar and press **Enter** or click **Go**. To refresh the current page, use the **Ctrl + r** keyboard shortcut or click **Go**.



http://localhost:8080/cs/Satellite?c=Content_C&childpagename=FirstSiteII%2FFSIILayout&rendermode=preview&pagename=FSIIPreviewWrapper&cid=1124813172330

About FirstSiteII

FirstSite Provides the Tools and Techniques for a Fast and Cost-Effective Implementation

By: Barton P. Fooman

It's a fact: Many content management projects are expensive to implement, disruptive to the organization, and prone to failure. Yet Oracle WebCenter Sites has been deployed successfully at over 450 customers for a wide range of solutions: corporate Web sites, product marketing sites, partner extranets, and employee intranets to name a few. To help immunize customers from potential problems, Oracle offers FirstSite, which leverages the experience and best practices of our many successful customers. FirstSite is a quick-start program that includes a collection of standard templates and site components that are common to most sites, combined with documentation, training, a rich developer community, and a best practices methodology. FirstSite helps ensure that your site is implemented the right way, right away.

Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.

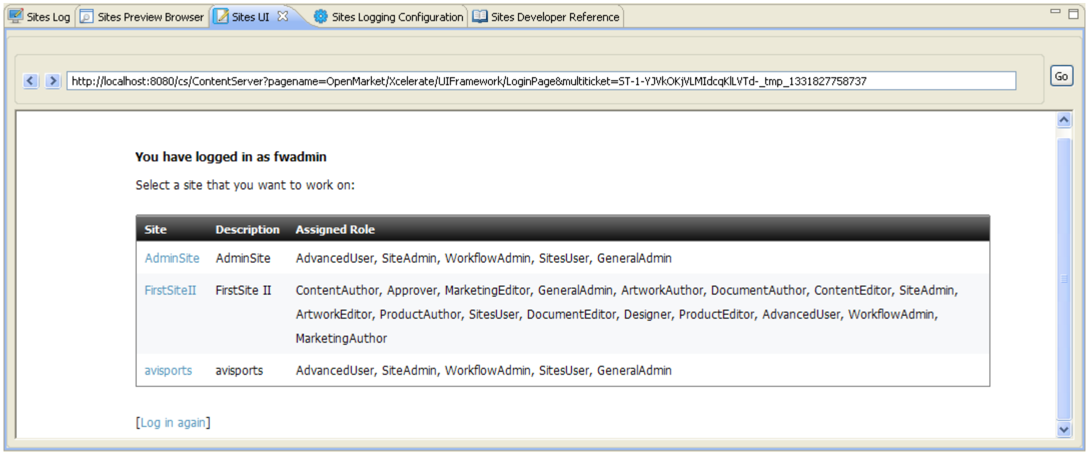
Validate: xhtml css

53.4.4 Sites UI View

Displays the WebCenter Sites Admin and Contributor interfaces in an embedded browser. This is equivalent to using either the Admin interface or Contributor interface in a standalone browser.

Note: The WebCenter Sites Admin interface utilizes a Java applet to display the left pane. For information about running applets in browser views, see the Eclipse FAQ, located at `http://www.eclipse.org/swt/faq.php#browserapplets`.

If you are unable to run the applet, use the applet-free Admin interface mode or use a standalone browser to work in the Admin interface.



http://localhost:8080/cs/ContentServer?pagename=OpenMarket/Xcelerate/UIFramework/LoginPage&multiticket=ST-1-YJVkOKjVLMIdcqKlLVTd-_tmp_1331827758737

You have logged in as fwadmin

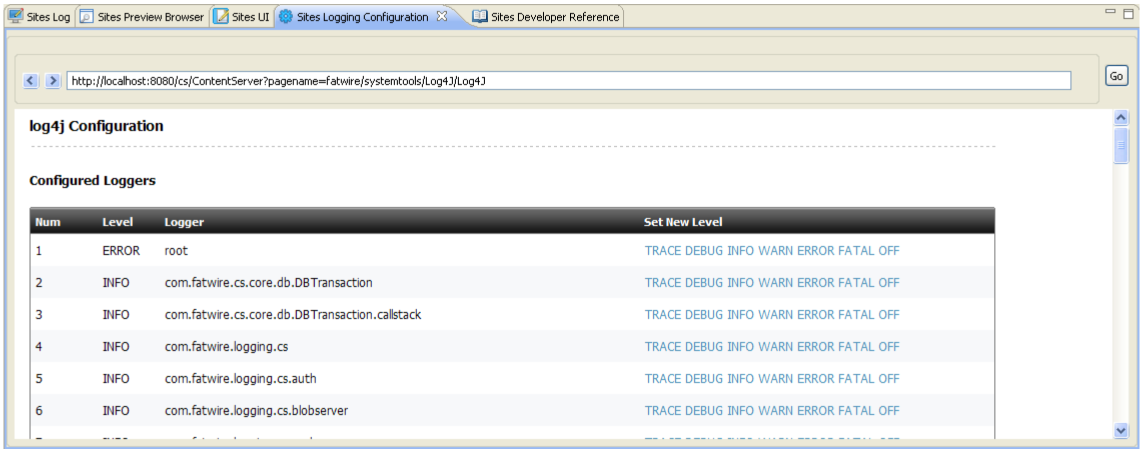
Select a site that you want to work on:

Site	Description	Assigned Role
AdminSite	AdminSite	AdvancedUser, SiteAdmin, WorkflowAdmin, SitesUser, GeneralAdmin
FirstSiteII	FirstSite II	ContentAuthor, Approver, MarketingEditor, GeneralAdmin, ArtworkAuthor, DocumentAuthor, ContentEditor, SiteAdmin, ArtworkEditor, ProductAuthor, SitesUser, DocumentEditor, Designer, ProductEditor, AdvancedUser, WorkflowAdmin, MarketingAuthor
avisports	avisports	AdvancedUser, SiteAdmin, WorkflowAdmin, SitesUser, GeneralAdmin

[Log in again]

53.4.5 Sites Logging Configuration View

If your WebCenter Sites system is using Apache log4j, this view displays a dynamically updating log4j configuration screen. The log4j configuration screen enables you to view current loggers, change logger levels, add new loggers, and search logs.



http://localhost:8080/cs/ContentServer?pagename=fatwire/systemtools/Log4J/Log4J

log4j Configuration

Configured Loggers

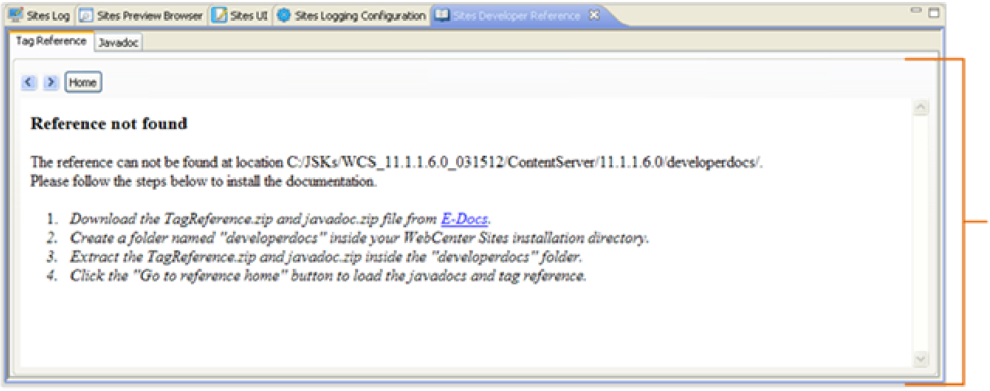
Num	Level	Logger	Set New Level
1	ERROR	root	TRACE DEBUG INFO WARN ERROR FATAL OFF
2	INFO	com.fatwire.cs.core.db.DBTransaction	TRACE DEBUG INFO WARN ERROR FATAL OFF
3	INFO	com.fatwire.cs.core.db.DBTransaction.callstack	TRACE DEBUG INFO WARN ERROR FATAL OFF
4	INFO	com.fatwire.logging.cs	TRACE DEBUG INFO WARN ERROR FATAL OFF
5	INFO	com.fatwire.logging.cs.auth	TRACE DEBUG INFO WARN ERROR FATAL OFF
6	INFO	com.fatwire.logging.cs.blobserver	TRACE DEBUG INFO WARN ERROR FATAL OFF

For information about using the log4j configuration screen, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

53.4.6 Sites Developer Reference View

This view contains two tabs, one of which displays the *Oracle Fusion Middleware WebCenter Sites Tag Reference* and the other displays the *Oracle Fusion Middleware WebCenter Sites Java API Reference*. This information is only displayed if you have associated the *Tag Reference* and *Javadoc* with Eclipse. Otherwise, the view displays instructions for associating the *Tag Reference* and *Javadoc* with Eclipse. For instructions, you can also refer to step 7 in Section 52.2, "Setting Up Developer Tools."

If the Tag Reference and Javadoc are not associated with Eclipse, the tabs display the following:



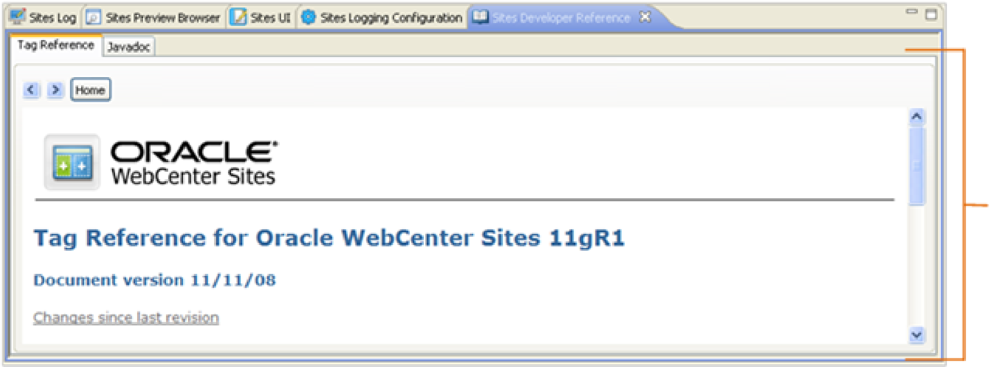
Reference not found

The reference can not be found at location C:/JSKs/WCS_11.1.1.6.0_031512/ContentServer/11.1.1.6.0/developerdocs/.
Please follow the steps below to install the documentation.

1. *Download the TagReference.zip and javadoc.zip file from E-Docs.*
2. *Create a folder named "developerdocs" inside your WebCenter Sites installation directory.*
3. *Extract the TagReference.zip and javadoc.zip inside the "developerdocs" folder.*
4. *Click the "Go to reference home" button to load the javadocs and tag reference.*

Instructions for downloading and installing the Tag Reference and Javadoc

If the Tag Reference and Javadoc are associated with Eclipse, the tabs display the following:

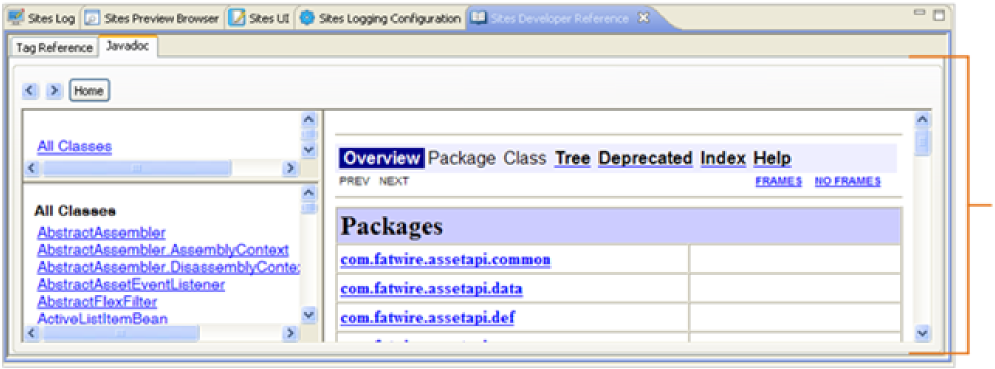


Tag Reference for Oracle WebCenter Sites 11gR1

Document version 11/11/08

Changes since last revision

Tag Reference tab



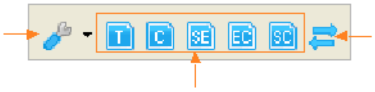
All Classes
AbstractAssembler
AbstractAssembler.AssemblyContext
AbstractAssembler.DisassemblyConte:
AbstractAssetEventListener
AbstractFlexFilter
ActiveListItemBean

Packages	
com.fatwire.assetapi.common	
com.fatwire.assetapi.data	
com.fatwire.assetapi.def	

Javadoc tab

53.4.7 Wizards

Wizards can be invoked from either the Oracle menu or the WebCenter Sites Toolbar, and enable you to create code-based WebCenter Sites resources. The available wizards are: SiteEntry, CSElement, Template, ElementCatalog, SiteCatalog, the configuration tool, and the synchronization tool.



Configuration tool

Synchronization tool

Template, CSElement,
SiteEntry, ElementCatalog,
and SiteCatalog wizards

53.5 Data Synchronization (Export/Import) Tool

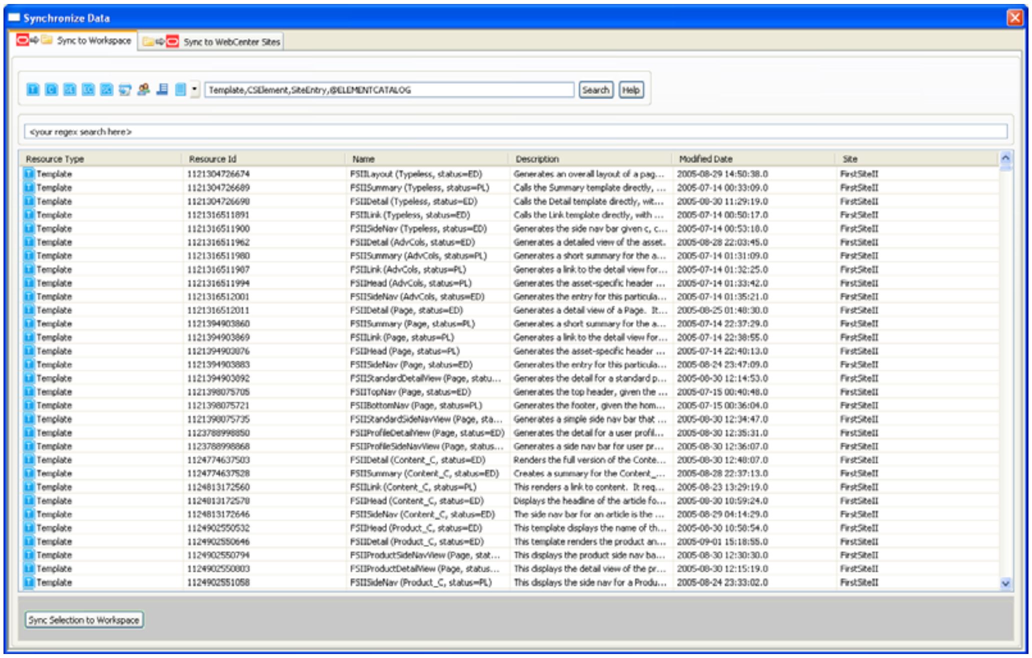
The synchronization tool provides you with two tabs:

- Section 53.5.1, "Sync to Workspace (Export from WebCenter Sites)"
- Section 53.5.2, "Sync to WebCenter Sites (Import into WebCenter Sites)"

53.5.1 Sync to Workspace (Export from WebCenter Sites)

Sync to Workspace is used to export data from the IDE-integrated WebCenter Sites to your Developer Tools workspace. In the process, Developer Tools serializes selected resources (transforms database representations into files) and copies the serialized representation to the Developer Tools workspace. You can then modify the resources in Eclipse.

Figure 53–2 Synchronization Icon / Sync to Workspace Tab



Template,CSElement,SiteEntry,@ELEMENTCATALOG

<your regex search here>

Resource Type	Resource Id	Name	Description	Modified Date	Site
Template	1121304726674	FSIILayout (Typeless, status=ED)	Generates an overall layout of a pag...	2005-08-29 14:50:38.0	FirstSiteII
Template	1121304726689	FSIISummary (Typeless, status=PL)	Calls the Summary template directly, ...	2005-07-14 00:33:09.0	FirstSiteII
Template	1121304726698	FSIIDetail (Typeless, status=ED)	Calls the Detail template directly, wit...	2005-08-30 11:29:19.0	FirstSiteII
Template	1121316511891	FSIILink (Typeless, status=ED)	Calls the Link template directly, with ...	2005-07-14 00:50:17.0	FirstSiteII
Template	1121316511900	FSIISideNav (Typeless, status=ED)	Generates the side nav bar given c, c...	2005-07-14 00:53:10.0	FirstSiteII
Template	1121316511962	FSIIDetail (AdvCols, status=ED)	Generates a detailed view of the asset.	2005-08-28 22:03:45.0	FirstSiteII
Template	1121316511980	FSIISummary (AdvCols, status=PL)	Generates a short summary for the a...	2005-07-14 01:31:09.0	FirstSiteII
Template	1121316511987	FSIILink (AdvCols, status=PL)	Generates a link to the detail view for...	2005-07-14 01:32:25.0	FirstSiteII
Template	1121316511994	FSIIHead (AdvCols, status=PL)	Generates the asset-specific header ...	2005-07-14 01:33:42.0	FirstSiteII
Template	1121316512001	FSIISideNav (AdvCols, status=ED)	Generates the entry for this particula...	2005-07-14 01:35:21.0	FirstSiteII
Template	1121316512011	FSIIDetail (Page, status=ED)	Generates a detail view of a Page. It...	2005-08-25 01:48:30.0	FirstSiteII
Template	1121394903860	FSIISummary (Page, status=PL)	Generates a short summary for the a...	2005-07-14 22:37:29.0	FirstSiteII
Template	1121394903869	FSIILink (Page, status=PL)	Generates a link to the detail view for...	2005-07-14 22:38:55.0	FirstSiteII
Template	1121394903876	FSIIHead (Page, status=PL)	Generates the asset-specific header ...	2005-07-14 22:40:13.0	FirstSiteII
Template	1121394903883	FSIISideNav (Page, status=ED)	Generates the entry for this particula...	2005-08-24 23:47:09.0	FirstSiteII
Template	1121394903892	FSIIStandardDetailView (Page, statu...	Generates the detail for a standard p...	2005-08-30 12:14:53.0	FirstSiteII
Template	1121398075705	FSIITopNav (Page, status=ED)	Generates the top header, given the ...	2005-07-15 00:40:48.0	FirstSiteII
Template	1121398075721	FSIIBottomNav (Page, status=PL)	Generates the footer, given the hom...	2005-07-15 00:36:04.0	FirstSiteII
Template	1121398075735	FSIIStandardSideNavView (Page, sta...	Generates a simple side nav bar that ...	2005-08-30 12:34:47.0	FirstSiteII
Template	1123788998850	FSIIProfileDetailView (Page, status=ED)	Generates the detail for a user profil...	2005-08-30 12:35:31.0	FirstSiteII
Template	1123788998868	FSIIProfileSideNavView (Page, status...	Generates a side nav bar for user pr...	2005-08-30 12:36:07.0	FirstSiteII
Template	1124774637503	FSIIDetail (Content_C, status=ED)	Renders the full version of the Conte...	2005-08-30 12:48:07.0	FirstSiteII
Template	1124774637528	FSIISummary (Content_C, status=ED)	Creates a summary for the Content_...	2005-08-28 22:37:13.0	FirstSiteII
Template	1124813172560	FSIILink (Content_C, status=PL)	This renders a link to content. It req...	2005-08-23 13:29:19.0	FirstSiteII
Template	1124813172570	FSIIHead (Content_C, status=ED)	Displays the headline of the article fo...	2005-08-30 10:59:24.0	FirstSiteII
Template	1124813172646	FSIISideNav (Content_C, status=ED)	The side nav bar for an article is the ...	2005-08-29 04:14:29.0	FirstSiteII
Template	1124902550532	FSIIHead (Product_C, status=ED)	This template displays the name of th...	2005-08-30 10:50:54.0	FirstSiteII
Template	1124902550646	FSIIDetail (Product_C, status=ED)	This template renders the product an...	2005-09-01 15:18:55.0	FirstSiteII
Template	1124902550794	FSIIProductSideNavView (Page, stat...	This displays the product side nav ba...	2005-08-30 12:30:30.0	FirstSiteII
Template	1124902550803	FSIIProductDetailView (Page, status...	This displays the detail view of the pr...	2005-08-30 12:15:19.0	FirstSiteII
Template	1124902551058	FSIISideNav (Product_C, status=PL)	This displays the side nav for a Produ...	2005-08-24 23:33:02.0	FirstSiteII

To export items from WebCenter Sites to your workspace

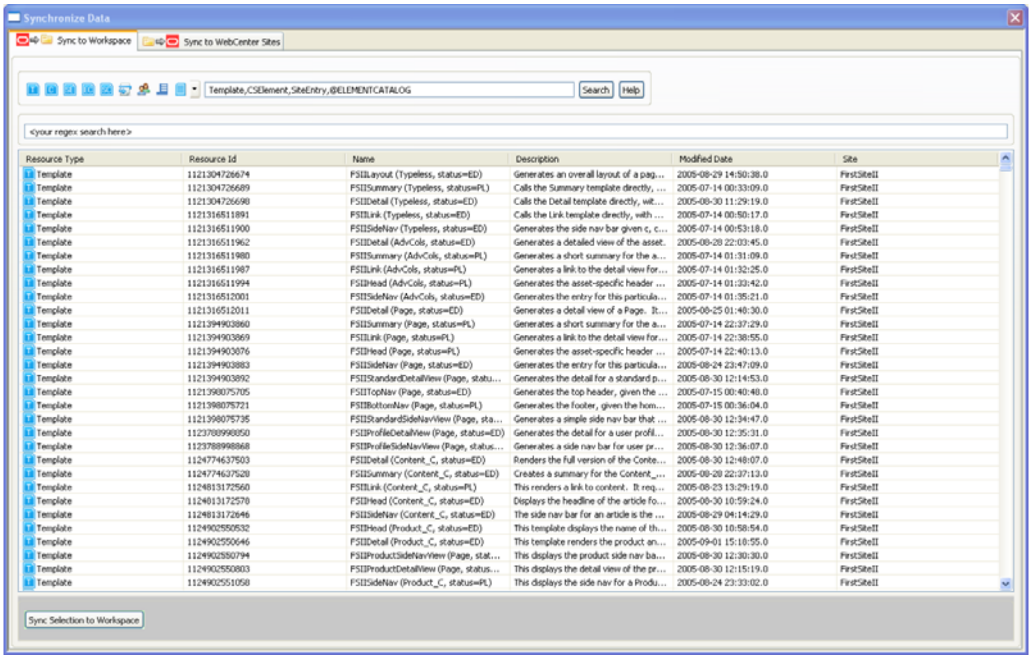
1. Select the items you wish to export. To narrow down the list of items, go to the "regex" search bar and enter the name of the asset type you are searching for. To search for multiple asset types, enter a comma-separated list.
2. Click **Sync Selection to Workspace**.

   The assets you exported to your Developer Tools workspace are now listed in the "Sites Work" tree tab.

53.5.2 Sync to WebCenter Sites (Import into WebCenter Sites)

Sync to WebCenter Sites is used to import resources from your Developer Tools workspace into the IDE-integrated WebCenter Sites. In the process, Developer Tools transforms the selected resource to its native WebCenter Sites representation and copies it to the WebCenter Sites database.

Figure 53–3 Synchronization Icon / Sync to WebCenter Sites Tab



Resource Type	Resource Id	Name	Description	Modified Date	Site
Template	1121304726674	FSIILayout (Typeless, status=ED)	Generates an overall layout of a pag...	2005-08-29 14:50:38.0	FirstSiteII
Template	1121304726689	FSIISummary (Typeless, status=PL)	Calls the Summary template directly, ...	2005-07-14 00:33:09.0	FirstSiteII
Template	1121304726698	FSIIDetail (Typeless, status=ED)	Calls the Detail template directly, wit...	2005-08-30 11:29:19.0	FirstSiteII
Template	1121316511891	FSIILink (Typeless, status=ED)	Calls the Link template directly, with ...	2005-07-14 00:50:17.0	FirstSiteII
Template	1121316511900	FSIISideNav (Typeless, status=ED)	Generates the side nav bar given c, c...	2005-07-14 00:53:18.0	FirstSiteII
Template	1121316511962	FSIIDetail (AdvCols, status=ED)	Generates a detailed view of the asset.	2005-08-28 22:03:45.0	FirstSiteII
Template	1121316511980	FSIISummary (AdvCols, status=PL)	Generates a short summary for the a...	2005-07-14 01:31:09.0	FirstSiteII
Template	1121316511987	FSIILink (AdvCols, status=PL)	Generates a link to the detail view for...	2005-07-14 01:32:25.0	FirstSiteII
Template	1121316511994	FSIIHead (AdvCols, status=PL)	Generates the asset-specific header ...	2005-07-14 01:33:42.0	FirstSiteII
Template	1121316512001	FSIISideNav (AdvCols, status=ED)	Generates the entry for this particula...	2005-07-14 01:35:21.0	FirstSiteII
Template	1121316512011	FSIIDetail (Page, status=ED)	Generates a detail view of a Page. It...	2005-08-25 01:40:30.0	FirstSiteII
Template	1121394903860	FSIISummary (Page, status=PL)	Generates a short summary for the a...	2005-07-14 22:37:29.0	FirstSiteII
Template	1121394903869	FSIILink (Page, status=PL)	Generates a link to the detail view for...	2005-07-14 22:38:55.0	FirstSiteII
Template	1121394903876	FSIIHead (Page, status=PL)	Generates the asset-specific header ...	2005-07-14 22:40:13.0	FirstSiteII
Template	1121394903883	FSIISideNav (Page, status=ED)	Generates the entry for this particula...	2005-08-24 23:47:09.0	FirstSiteII
Template	1121394903892	FSIIStandardDetailView (Page, statu...	Generates the detail for a standard p...	2005-08-30 12:14:53.0	FirstSiteII
Template	1121398075705	FSIITopNav (Page, status=ED)	Generates the top header, given the ...	2005-07-15 00:40:48.0	FirstSiteII
Template	1121398075721	FSIIBottomNav (Page, status=PL)	Generates the footer, given the hom...	2005-07-15 00:36:04.0	FirstSiteII
Template	1121398075735	FSIIStandardSideNavView (Page, sta...	Generates a simple side nav bar that ...	2005-08-30 12:34:47.0	FirstSiteII
Template	1123788998850	FSIIProfileDetailView (Page, status=ED)	Generates the detail for a user profil...	2005-08-30 12:35:31.0	FirstSiteII
Template	1123788998868	FSIIProfileSideNavView (Page, status...	Generates a side nav bar for user pr...	2005-08-30 12:36:07.0	FirstSiteII
Template	1124774637503	FSIIDetail (Content_C, status=ED)	Renders the full version of the Conte...	2005-08-30 12:48:07.0	FirstSiteII
Template	1124774637528	FSIISummary (Content_C, status=ED)	Creates a summary for the Content_...	2005-08-28 22:37:13.0	FirstSiteII
Template	1124813172560	FSIILink (Content_C, status=PL)	This renders a link to content. It req...	2005-08-23 13:29:19.0	FirstSiteII
Template	1124813172578	FSIIHead (Content_C, status=ED)	Displays the headline of the article fo...	2005-08-30 10:59:24.0	FirstSiteII
Template	1124813172646	FSIISideNav (Content_C, status=ED)	The side nav bar for an article is the ...	2005-08-29 04:14:29.0	FirstSiteII
Template	1124902550532	FSIIHead (Product_C, status=ED)	This template displays the name of th...	2005-08-30 10:58:54.0	FirstSiteII
Template	1124902550646	FSIIDetail (Product_C, status=ED)	This template renders the product an...	2005-09-01 15:10:55.0	FirstSiteII
Template	1124902550794	FSIIProductSideNavView (Page, stat...	This displays the product side nav ba...	2005-08-30 12:30:30.0	FirstSiteII
Template	1124902550803	FSIIProductDetailView (Page, status...	This displays the detail view of the pr...	2005-08-30 12:15:19.0	FirstSiteII
Template	1124902551058	FSIISideNav (Product_C, status=PL)	This displays the side nav for a Produ...	2005-08-24 23:33:02.0	FirstSiteII

To import items into WebCenter Sites from the workspace

1. Select the items you wish to import. To narrow down the list of items, go to the "regex" search bar and enter the name of the asset type you are searching for. To search for multiple asset types, enter a comma-separated list.
2. Click **Sync Selection to WebCenter Sites**.

53.6 Next Steps

The next chapters provide information about using the WebCenter Sites features, provided by Developer Tools, in the Eclipse IDE. Proceed to Chapter 54, "Developer Tools: Developing JSPs" for information about JSP development in Eclipse.

54

Developer Tools: Developing JSPs

This chapter contains information about developing WebCenter Sites JSPs with Developer Tools.

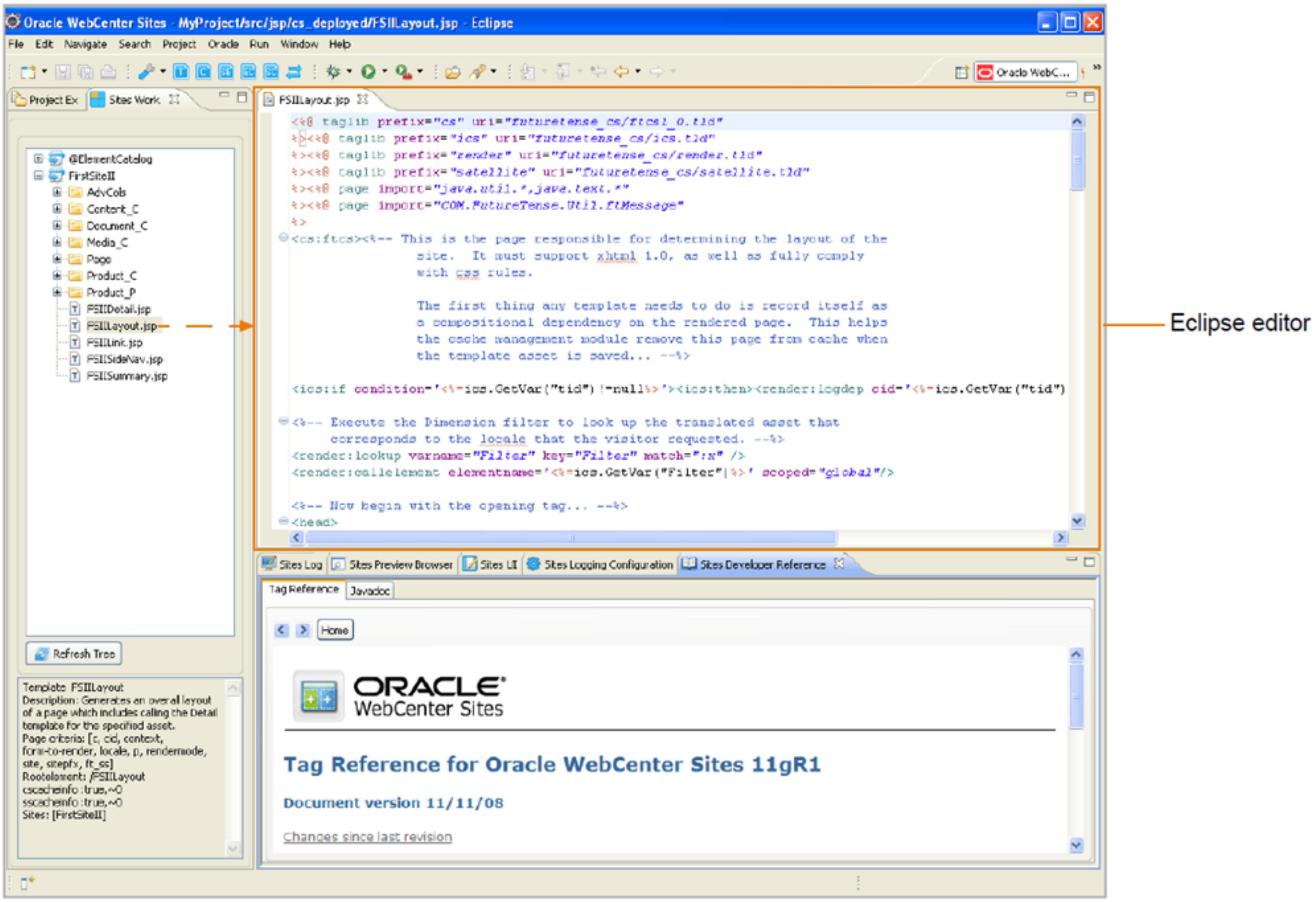
This chapter contains the following sections:

- Section 54.1, "JSP Development with Developer Tools"
- Section 54.2, "Tag and Java API Completion"
- Section 54.3, "Debugging"

54.1 JSP Development with Developer Tools

The Developer Tools kit supports the development of WebCenter Sites JSPs using the native Eclipse JSP editor. The Eclipse JSP editor includes support for WebCenter Sites tag and Java API completion, syntax highlighting, and debugging. Figure 54–1 shows an example of a WebCenter Sites JSP in the Eclipse editor.

Figure 54–1 Eclipse JSP editor



Template: FSIILayout
Description: Generates an overall layout of a page which includes calling the Detail template for the specified asset.
Page criteria: [c, cid, context, form-to-render, locale, p, rendermode, site, sitepfx, ft_ss]
Rootelement: /FSIILayout
cscacheinfo :true,~0
sscacheinfo :true,~0
Sites: [FirstSiteII]

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld"
%><%@ page import="java.util.*,java.text.*"
%><%@ page import="COM.FutureTense.Util.ftMessage"
%>
<cs:ftcs><%-- This is the page responsible for determining the layout of the
                 site.  It must support xhtml 1.0, as well as fully comply
                 with css rules.

                 The first thing any template needs to do is record itself as
                 a compositional dependency on the rendered page.  This helps
                 the cache management module remove this page from cache when
                 the template asset is saved... --%>

<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep cid='<%=ics.GetVar("tid")

<%-- Execute the Dimension filter to look up the translated asset that
     corresponds to the locale that the visitor requested. --%>
<render:lookup varname="Filter" key="Filter" match=":x" />
<render:callelement elementname='<%=ics.GetVar("Filter")%>' scoped="global"/>

<%-- Now begin with the opening tag... --%>
<head>
```

Tag Reference for Oracle WebCenter Sites 11gR1

Document version 11/11/08

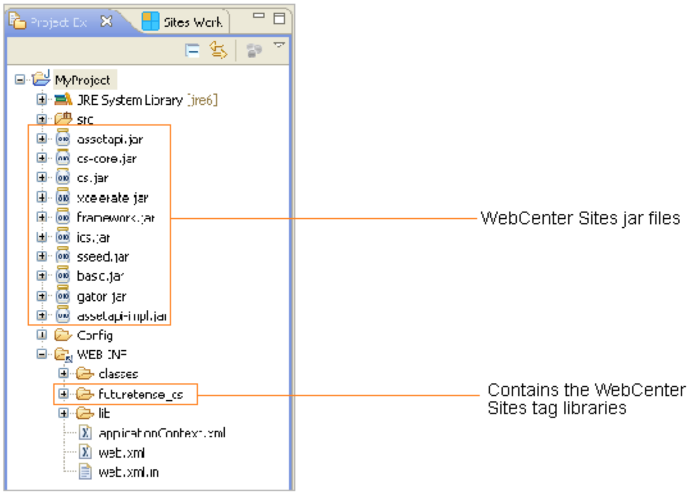
Changes since last revision

WebCenter Sites JSPs can include page caching, resultset caching, and associated metadata such as Template assets, CSElement assets, or ElementCatalog entries. The metadata of a JSP enables WebCenter Sites to track and manage it. Developer Tools handles a JSP's underlying WebCenter Sites processes transparently, including tracking the JSP and its corresponding metadata. If your WebCenter Site instance is running, and you save a JSP in Eclipse, the Developer Tools kit automatically synchronizes those changes with the WebCenter Sites instance. Any metadata associated with the JSP is also synchronized with WebCenter Sites. This enables you to view the changes in WebCenter Sites as soon as you save the JSP in Eclipse.

54.2 Tag and Java API Completion

Eclipse provides tag and Java API completion features. Eclipse uses the tag libraries and jar files belonging to the current WebCenter Sites instance to provide the appropriate code completion for WebCenter Sites related tags and Java APIs. For local hosts, the WebCenter Sites tag libraries and jar files are automatically linked to your Eclipse project, and contained within the Eclipse project folder (located in the "Project Explorer" view). For remote hosts, the WebCenter Sites tag libraries and jar files must be manually copied from the remote host to your Eclipse project. For instructions, see Section 52.2, "Setting Up Developer Tools."

- The tag libraries are contained in the `futuretense_cs` folder under the `WEB-INF` folder.
- The jar files are contained under the main Eclipse project folder.



WebCenter Sites jar files

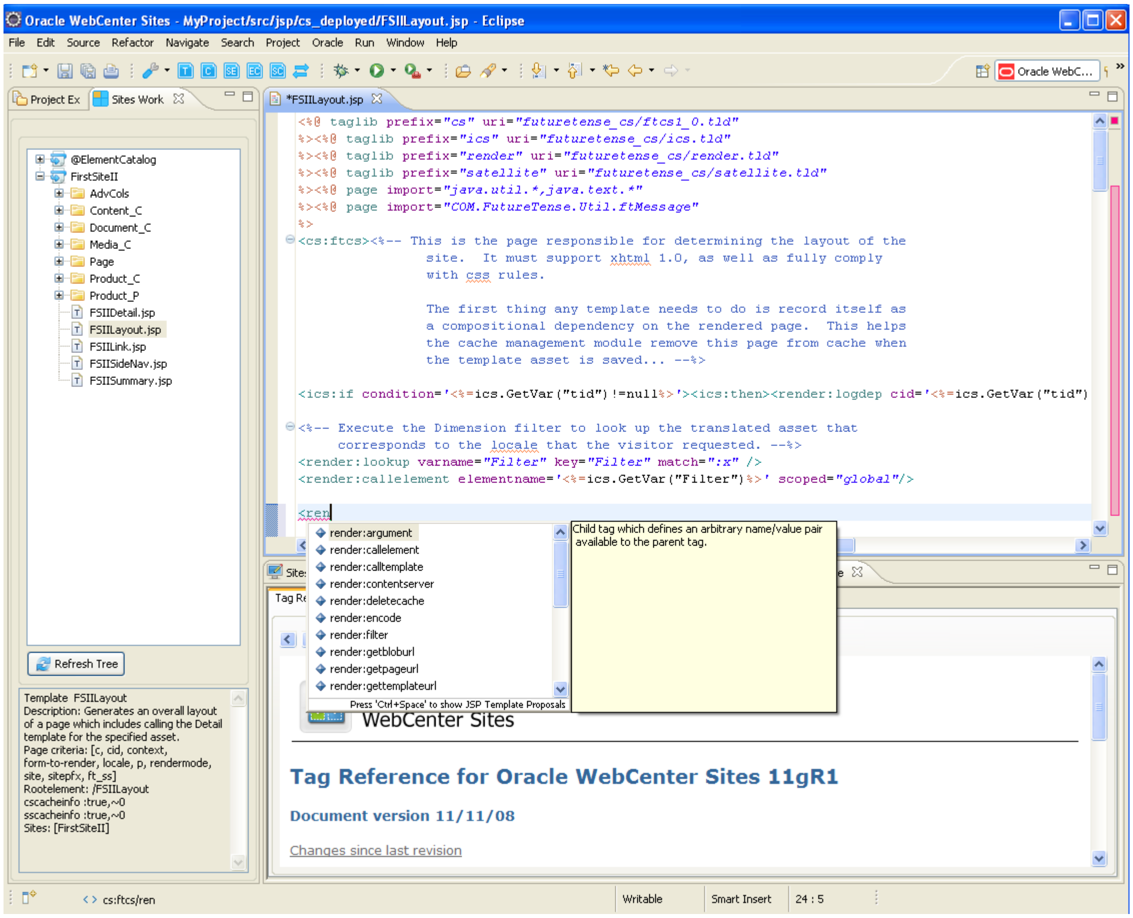
Contains the WebCenter Sites tag libraries

Note: When you use the tag and Java API completion feature, keep in mind the following:

- Make sure you follow strict JSP coding standards. This way your code can be deployed on any application server.
- Eclipse code completion displays all public Java methods contained within the WebCenter Sites jars, only use the APIs that are in the WebCenter Sites documentation. Using undocumented functionality is risky and unsupported.

Associating the *Oracle Fusion Middleware WebCenter Sites Java API Reference* and *Oracle Fusion Middleware WebCenter Sites Tag Reference* with Eclipse enables the tag and Java API completion features to display information about each tag and piece of Java code you use when managing a WebCenter Sites JSP. For example, when you are working with a WebCenter Sites JSP and you begin to type the name of a tag, a window opens listing code completion suggestions. If you associated the *Tag Reference* and *Javadoc* with Eclipse, a second dialog box is displayed containing information about each suggestion (see Figure 54–2).

Figure 54–2 Tag and Java API completion feature



```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld"
%><%@ page import="java.util.*,java.text.*"
%><%@ page import="COM.FutureTense.Util.ftMessage"
%>
<cs:ftcs><%-- This is the page responsible for determining the layout of the
                site.  It must support xhtml 1.0, as well as fully comply
                with css rules.

                The first thing any template needs to do is record itself as
                a compositional dependency on the rendered page.  This helps
                the cache management module remove this page from cache when
                the template asset is saved... --%>

<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep cid='<%=ics.GetVar("tid")

<%-- Execute the Dimension filter to look up the translated asset that
     corresponds to the locale that the visitor requested. --%>
<render:lookup varname="Filter" key="Filter" match=":x" />
<render:callelement elementname='<%=ics.GetVar("Filter")%>' scoped="global"/>

<ren
```

Child tag which defines an arbitrary name/value pair available to the parent tag.

Template FSIILayout
Description: Generates an overall layout of a page which includes calling the Detail template for the specified asset.
Page criteria: [c, cid, context, form-to-render, locale, p, rendermode, site, sitepfx, ft_ss]
Rootelement: /FSIILayout
cscacheinfo :true,~0
sscacheinfo :true,~0
Sites: [FirstSiteII]

Tag Reference for Oracle WebCenter Sites 11gR1

Document version 11/11/08

In addition to adding functionality to the tag and Java code completion features, the *Javadoc* and *Tag Reference* are both made accessible in the "Sites Developer Reference" view. For more information, see Section 53.4.6, "Sites Developer Reference View."

54.3 Debugging

To debug Java and JSP code in Developer Tools, you must first attach the debugger to the JVM process that runs WebCenter Sites. It is recommended to do so with remote debugging. To attach the WebCenter Sites JVM, follow the instructions provided by Eclipse at the following URL:

`http://www.ibm.com/developerworks/library/os-ecbug/`

Once the JVM is attached to the debugger, you can set breakpoints in your JSP and Java code, view variables, and so on.

55

Developer Tools: Creating Templates for Mobile Websites

Developer Tools enables you to create templates for mobile sites.

This chapter contains the following sections:

- Section 55.1, "Overview of Mobility Support in Developer Tools"
- Section 55.2, "Mobile Template Development with Developer Tools"

55.1 Overview of Mobility Support in Developer Tools

Developer Tools supports the WebCenter Sites: Mobility framework. The Mobility feature allows you to create websites for mobile devices. For detailed information about the Mobility feature, see Chapter 68, "Configuring Oracle WebCenter Sites: Mobility to Support Mobile Websites."

55.2 Mobile Template Development with Developer Tools

In the "Oracle WebCenter Sites" perspective, templates are displayed as nodes in the **Sites Workspace Elements** tab. The workspace displays the templates (default and mobile) you exported from your WebCenter Sites instance to Eclipse. You can recognize mobile templates by their developer-defined suffix. (For example, a template created for a touchscreen device would be `_Touch`.)

If you need to develop a mobile template you must create it from an existing template. The rest of this section provides steps for creating a mobile template.

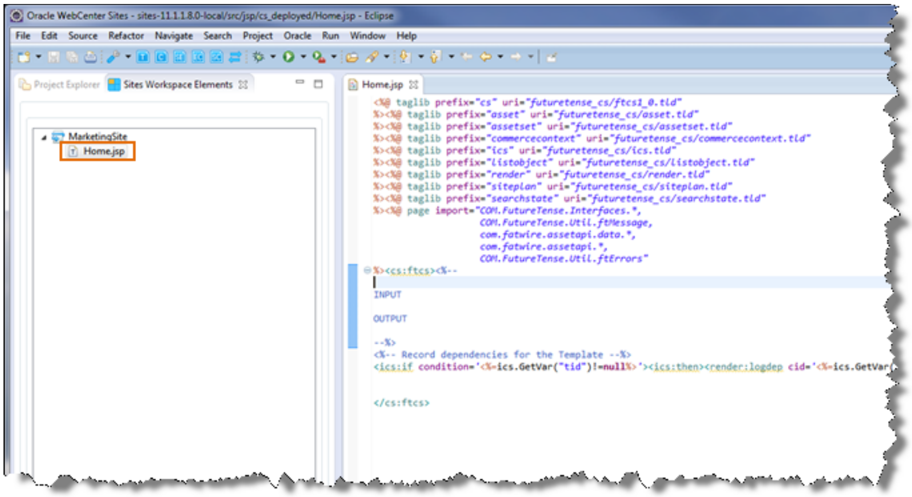
To create a template for mobile websites

1. In the **Sites Workspace Elements** tab (located in the left panel), expand the node of the site for which you want to create a mobile template.

   > **Note:** A mobile template is associated with one or more device groups (that is a group of devices with similar features) by a developer-defined suffix. For more information about device groups, see Chapter 68, "Configuring Oracle WebCenter Sites: Mobility to Support Mobile Websites."

   Figure 55–1 shows the expanded **MarketingSite** tree containing the `Home.jsp` template for the default device group (for desktop and laptop devices).

Figure 55–1 Home.jsp Template Element Displayed Under the 'MarketingSite' Node

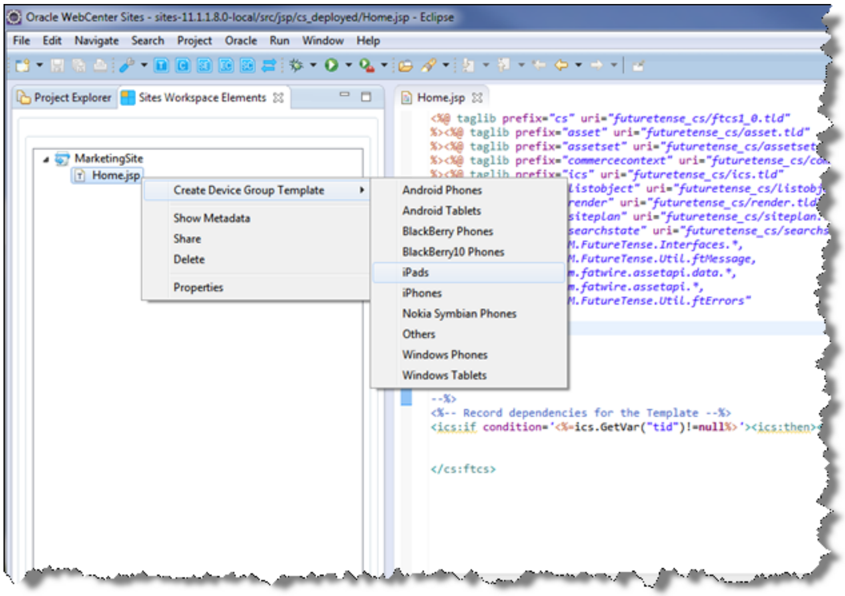

Oracle WebCenter Sites - sites-11.1.1.8.0-local/src/jsp/cs_deployed/Home.jsp - Eclipse
File Edit Source Refactor Navigate Search Project Oracle Run Window Help
Project Explorer
Sites Workspace Elements
MarketingSite
Home.jsp
Home.jsp

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/commercecontext.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="listobject" uri="futuretense_cs/listobject.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="siteplan" uri="futuretense_cs/siteplan.tld"
%><%@ taglib prefix="searchstate" uri="futuretense_cs/searchstate.tld"
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   com.fatwire.assetapi.data.*,
                   com.fatwire.assetapi.*,
                   COM.FutureTense.Util.ftErrors"
%><cs:ftcs><%--

INPUT

OUTPUT

--%>
<%-- Record dependencies for the Template --%>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep cid='<%=ics.GetVar(


</cs:ftcs>
```

2. Right-click the template off which you want to base the mobile template. In the context menu, select **Create Device Group Template** and then select the name of the device group for which you want to create the template.

Note: Multiple device groups can share the same suffix. If you do not see the desired device group in the list, a template defined by the same suffix as that device group may have already been created. For more information about developer-defined suffixes, see Chapter 68, "Configuring Oracle WebCenter Sites: Mobility to Support Mobile Websites"

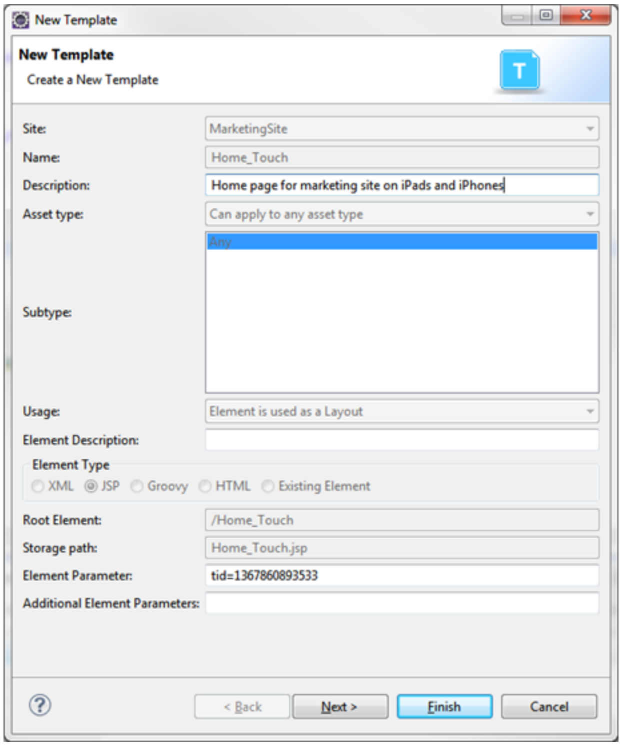
Figure 55–2 Create Device Group Template Context Menu



The "New Template" window opens (as shown in Figure 55–3) displaying fields with fixed and modifiable property values which the template wizard copied from the source template ("Home" template of the **MarketingSite** in this example).

3. In the "New Template" window, edit the properties in the modifiable fields according to your mobile template requirements, and then click **Finish**.

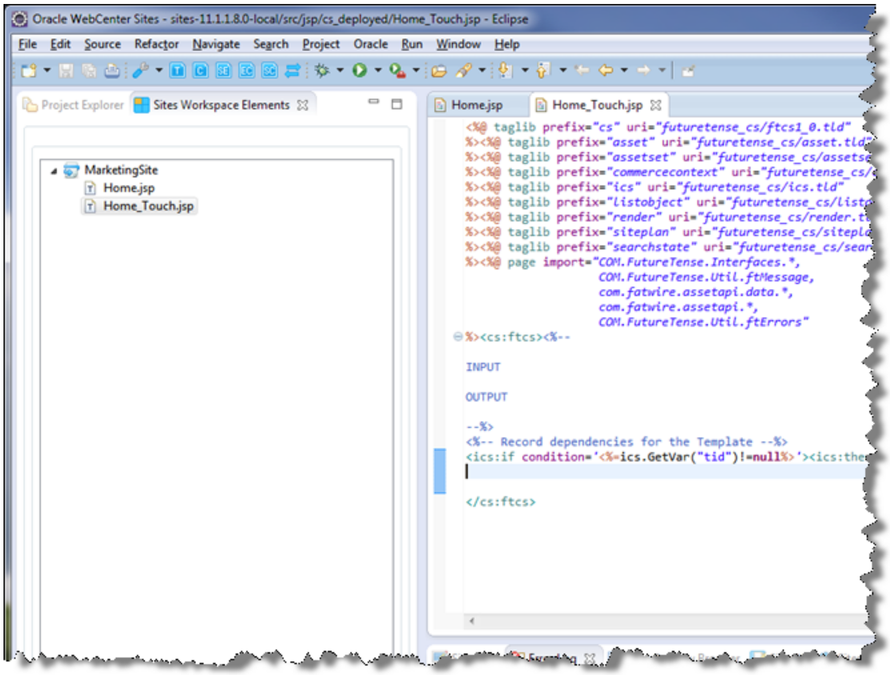
Figure 55–3 'New Template' Form for the 'Home_Touch' Mobile Template



Site: MarketingSite
Name: Home_Touch
Description: Home page for marketing site on iPads and iPhones
Asset type: Can apply to any asset type
Subtype: Any
Usage: Element is used as a Layout
Element Description:
Element Type: ○ XML ● JSP ○ Groovy ○ HTML ○ Existing Element
Root Element: /Home_Touch
Storage path: Home_Touch.jsp
Element Parameter: tid=1367860893533
Additional Element Parameters:

The file name (defined in the **Storage path** field in the "New Template" form) of the mobile template is listed in the **Sites Workspace Elements** tab, as shown in Figure 55–4.

Figure 55–4 Home_Touch.jsp Displayed in the 'Sites Workspace Elements' tab


Oracle WebCenter Sites - sites-11.1.1.8.0-local/src/jsp/cs_deployed/Home_Touch.jsp - Eclipse
File Edit Source Refactor Navigate Search Project Oracle Run Window Help
Project Explorer
Sites Workspace Elements
MarketingSite
Home.jsp
Home_Touch.jsp
Home.jsp
Home_Touch.jsp

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetse
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/c
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="listobject" uri="futuretense_cs/listo
%><%@ taglib prefix="render" uri="futuretense_cs/render.tl
%><%@ taglib prefix="siteplan" uri="futuretense_cs/sitepla
%><%@ taglib prefix="searchstate" uri="futuretense_cs/sear
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   com.fatwire.assetapi.data.*,
                   com.fatwire.assetapi.*,
                   COM.FutureTense.Util.ftErrors"
%><cs:ftcs><%--

INPUT

OUTPUT

--%>
<%-- Record dependencies for the Template --%>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:the


</cs:ftcs>
```

In Figure 55–4, the mobile template is named "Home_Touch," where "Home" is the name of the template and "_Touch" is the suffix defined for the device group for which this template was created.

4. Modify the code of your new mobile template in the native Eclipse JSP editor. (For information, see Chapter 54, "Developer Tools: Developing JSPs").

56

Developer Tools: Synchronization and Data Exchange

This chapter provides information about the export/import features supported by Developer Tools. This chapter also provides information about exchanging resources between WebCenter Sites instances, and the Developer Tools mappings processes.

The Developer Tools kit uses ID and site mapping processes to enable developers to exchange resources between WebCenter Sites instances and is discussed in the sections on ID mapping and site mappings.

This chapter contains the following sections:

56.1 Synchronization Using Developer Tools

Synchronization is the bi-directional flow of resources between a WebCenter Sites instance and its associated workspace. Using Developer Tools, you can perform the following synchronization operations:

- Export/import assets with built-in dependency resolution and ID mapping.
- Export/import asset types, such as flex families and AssetMaker asset types.
- Export/import site definitions, roles, start menu items, and tree tabs.
- Export/import SiteCatalog and ElementCatalog entries.
- (Command-line tool operation) Perform site re-mapping. For example, creating reusable modules which can be imported into any WebCenter Sites CM site.

Exporting or importing all resources of a given site enables you to track the entire site in a version control system. Advanced developers can use the command-line tool to re-map the resources of one site to another by creating reusable modules (custom workspaces).

56.2 Synchronization Scenarios

Depending on the scenario, resources are synchronized either automatically or manually.

If WebCenter Sites is running, resources between WebCenter Sites and Eclipse are automatically synchronized when the following actions are performed in Eclipse:

- Code-based resources (Templates, CSElements, SiteEntries, ElementCatalog entries, and SiteCatalog entries) are created with the Developer Tools wizards in Eclipse.
- Code-based resources (Templates, CSElements, and ElementCatalog entries) stored in the Developer Tools workspace are edited in Eclipse. This includes edits to JSP files, XML files, metadata, and other files associated with the resource.

  For example, if you edit a resource's associated JSP file in the Eclipse editor, the Developer Tools kit automatically synchronizes the changes into the WebCenter Sites instance. Using the Eclipse editor, advanced developers can also edit metadata files (`.main.xml`) of flex definitions and the Developer Tools kit will automatically synchronize the changes into WebCenter Sites. However, we recommend using the WebCenter Sites Admin interface to modify flex definitions.

In certain cases, resources must be manually synchronized using either the Synchronization tool in the Eclipse IDE or (for advanced developers) the command-line tool. Manual synchronization is required when:

- The Eclipse editor is not used to edit resources stored in the Developer Tools workspace. For example, when resources are copied to the Developer Tools workspace from a shared network file system or a version control system.
- WebCenter Sites resources are modified in the WebCenter Sites interfaces.

> **Note:** The Eclipse IDE provides an embedded WebCenter Sites Admin interface. However, Eclipse does not detect the changes that are made using this interface. Therefore, working in the embedded Admin interface is the same as working in a standalone browser running the Admin interface.

- WebCenter Sites is not running while you are creating or editing resources in the Eclipse IDE. Once WebCenter Sites is restarted, you must manually synchronize the resources you created or edited.

Using the command-line tool to synchronize resources is mainly for deployment purposes, such as nightly builds that are deployed to test servers. For example, an advanced developer can embed a synchronization command into a script for an automated deployment procedure. For information about running and using the command-line tool, see Chapter 58, "Developer Tools: Command-Line Tool."

56.3 Dependency Resolution

WebCenter Sites resources often depend on other resources. For example, a flex asset requires an associated flex definition to exist before it can be created. In turn, the flex definition depends on a set of attributes and possibly other resources. Therefore, all flex constructs require that the flex family exist on the system. To import a flex asset into an empty WebCenter Sites system, you must first create a flex family to which the flex asset will be associated. Then, create the following:

1. Create the flex attributes. For example, name, address, age, and so on.
2. Create the desired flex parent definitions.
3. Create flex definitions.

4. Create the desired flex parents.
5. Create flex assets.

When you export a flex asset, the Developer Tools kit performs all dependency resolutions for that asset and automatically exports all of its dependencies. Therefore, you only need to select the desired resource (such as the desired flex asset) and the Developer Tools kit computes all of the asset's dependencies.

> **Note:** The Developer Tools kit does not resolve a resource's dependency on site definitions. This enables you to choose whether you want to export or import an entire site, a subset of sites, or completely ignore site definitions (for example, if you are using the command-line tool to create a reusable module that can be imported into any site). For a detailed example of creating a reusable module, see Chapter 61, "Developer Tools: Using the Command-line Tool to Create Reusable Modules."

56.4 ID Mapping

Each resource created in WebCenter Sites is assigned a unique local identifier. A resource's local identifier is unique only to the WebCenter Sites instance on which it was created. Since multiple WebCenter Sites instances will be used to create resources, it is possible for two different resources, on separate WebCenter Sites instances, to have the same local identifier.

This section contains the following topics:

- Section 56.4.1, "ID Mapping Overview"
- Section 56.4.2, "Overriding a Resource's fw_uid"
- Section 56.4.3, "Using Developer Tools with Pre-Existing Resources"

56.4.1 ID Mapping Overview

To uniquely identify resources, the Developer Tools kit assigns each resource a globally unique identifier (`fw_uid`), which is unique across all WebCenter Sites instances. In addition, when you import a resource into a WebCenter Sites instance, the Developer Tools kit assigns a new local identifier to that resource on that instance. If the resource references other assets (such as associations, asset pointers, and flex definitions), a new local identifier is generated for each of those assets. On subsequent imports to that WebCenter Sites instance, the resources are assigned the same local identifier. The Developer Tools kit maintains the resources' `fw_uid` values across all WebCenter Sites instances. If the resource and its referenced assets are imported back into their original WebCenter Sites instance, the Developer Tools kit re-maps their local identifiers back to their original value.

> **Note:** Certain WebCenter Sites resources, such as Template assets, flex attributes, and tree tabs have unique name constraints. To avoid name conflicts, make sure each resource is uniquely named across all WebCenter Sites instances.

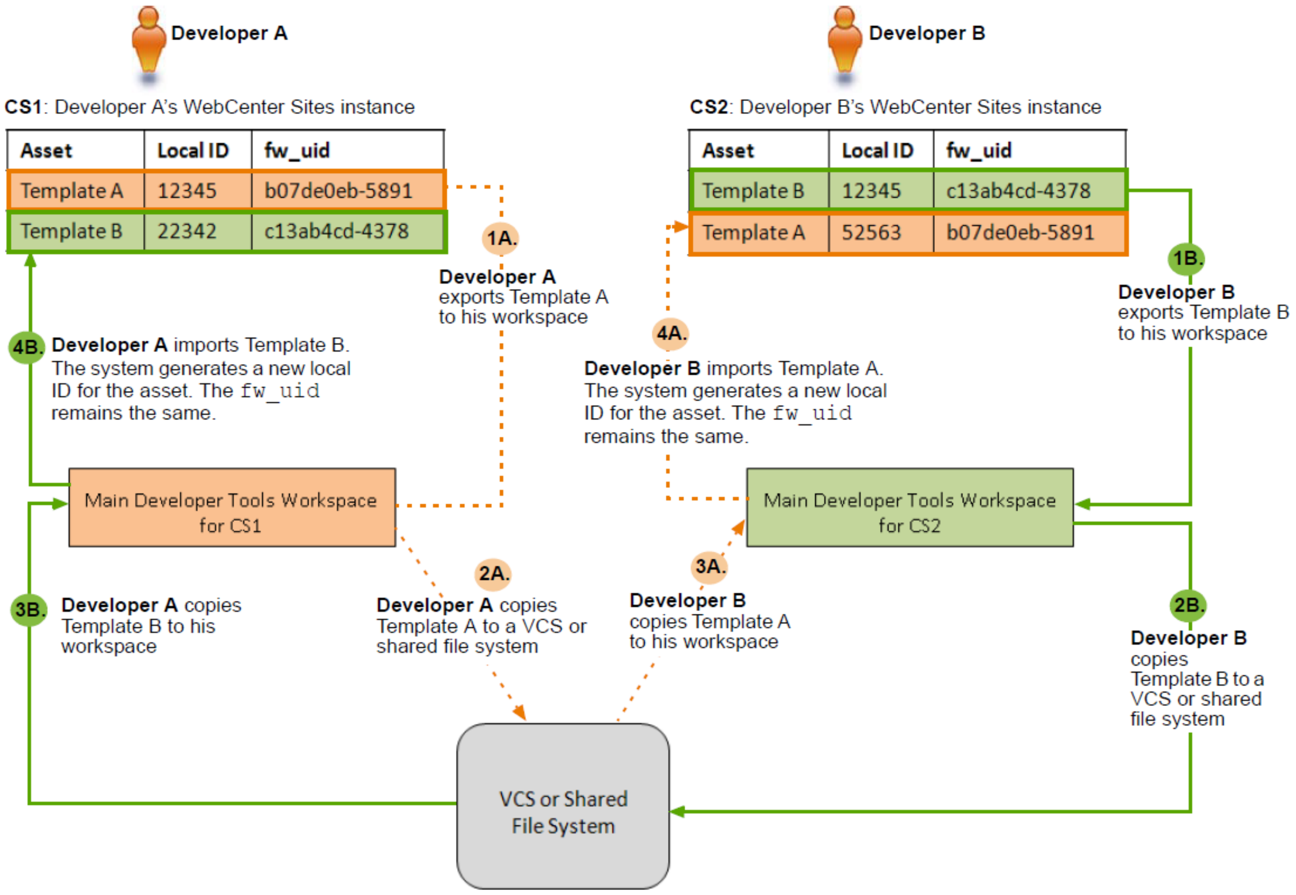
For example, (as shown in Figure 56–1) Developer A is working with a WebCenter Sites instance named CS1 and Developer B is working with a WebCenter Sites instance named CS2. Both developers created a completely different Template asset. Developer

A created Template A and Developer B created Template B. The two Template assets have different `fw_uid` values and different names. However, since local identifiers are randomly assigned, both Template assets, by chance, have been assigned the same local identifier (`12345`). Developers A and B want to exchange Template assets between each other's WebCenter Sites instances. Developer A wants to import Template B into the CS1 instance, and Developer B wants to import Template A into the CS2 instance.

Figure 56–1 illustrates the steps both developers take to exchange Template assets between their WebCenter Sites instances. Both Template assets' local identifiers are re-mapped when imported into the other developer's WebCenter Sites instance. When Template A is imported into the CS2 instance, the system assigns it the local identifier `52563`. When Template B is imported into the CS1 instance, the system assigns it the local identifier `22342`. In each case, the `fw_uid` values for both Template assets remain the same.

Note: To exchange resources between WebCenter Sites instances, the developers in our examples use a VCS or shared file system. For information about using a VCS, see Chapter 59, "Developer Tools: Integrating with Version Control Systems."

Figure 56–1 Exchanging two different assets with the same local identifier between two WebCenter Sites instances



CS1: Developer A's WebCenter Sites instance

Asset	Local ID	fw_uid
Template A	12345	b07de0eb-5891
Template B	22342	c13ab4cd-4378

CS2: Developer B's WebCenter Sites instance

Asset	Local ID	fw_uid
Template B	12345	c13ab4cd-4378
Template A	52563	b07de0eb-5891

1A. **Developer A** exports Template A to his workspace

2A. **Developer A** copies Template A to a VCS or shared file system

3A. **Developer B** copies Template A to his workspace

4A. **Developer B** imports Template A. The system generates a new local ID for the asset. The `fw_uid` remains the same.

1B. **Developer B** exports Template B to his workspace

2B. **Developer B** copies Template B to a VCS or shared file system

3B. **Developer A** copies Template B to his workspace

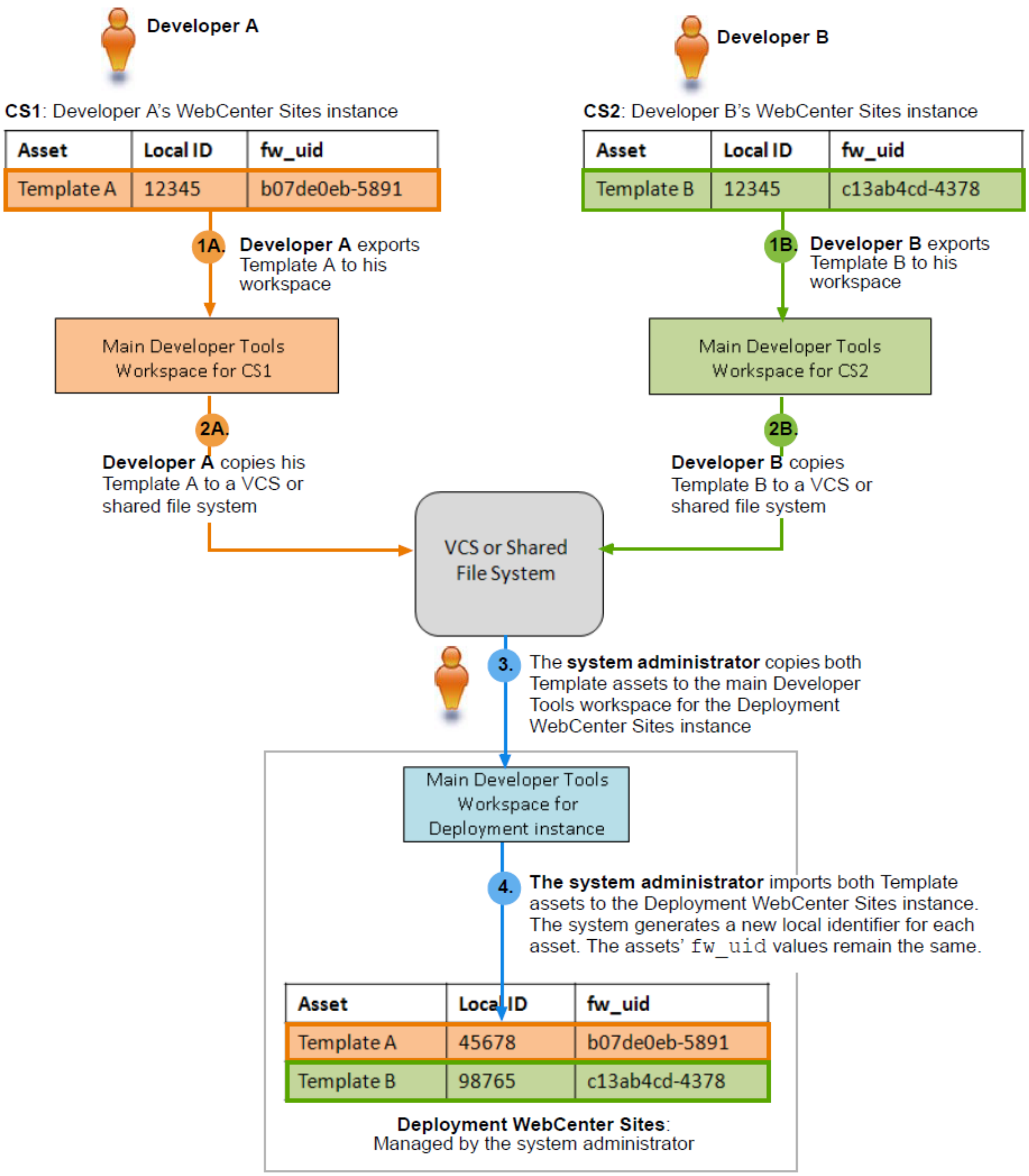
4B. **Developer A** imports Template B. The system generates a new local ID for the asset. The `fw_uid` remains the same.

Main Developer Tools Workspace for CS1
Main Developer Tools Workspace for CS2
VCS or Shared File System

In Figure 56–2, Developer A wants to deploy Template A to the Deployment WebCenter Sites instance (managed by the system administrator) and Developer B wants to deploy Template B to the same instance. Both Template assets have the same local identifier (12345).

Developers A and B each export their Template to the main Developer Tools workspace for their WebCenter Sites instance. They then copy their Templates to a VCS or shared file system. From here, the system administrator copies both Template assets to the Deployment WebCenter Sites' main Developer Tools workspace. The system administrator then imports the two Template assets from the workspace to the Deployment WebCenter Sites. Upon import, the system assigns both Templates a new local identifier. Template A is assigned the local identifier of 45678 and Template B is assigned the local identifier of 98765. The assets' fw_uid values remain the same.

Figure 56–2 Deploying two different assets with the same local identifier to a third WebCenter Sites instance



CS1: Developer A's WebCenter Sites instance

Asset	Local ID	fw_uid
Template A	12345	b07de0eb-5891

CS2: Developer B's WebCenter Sites instance

Asset	Local ID	fw_uid
Template B	12345	c13ab4cd-4378

1A. **Developer A** exports Template A to his workspace

1B. **Developer B** exports Template B to his workspace

2A. **Developer A** copies his Template A to a VCS or shared file system

2B. **Developer B** copies Template B to a VCS or shared file system

3. The **system administrator** copies both Template assets to the main Developer Tools workspace for the Deployment WebCenter Sites instance

4. **The system administrator** imports both Template assets to the Deployment WebCenter Sites instance. The system generates a new local identifier for each asset. The assets' fw_uid values remain the same.

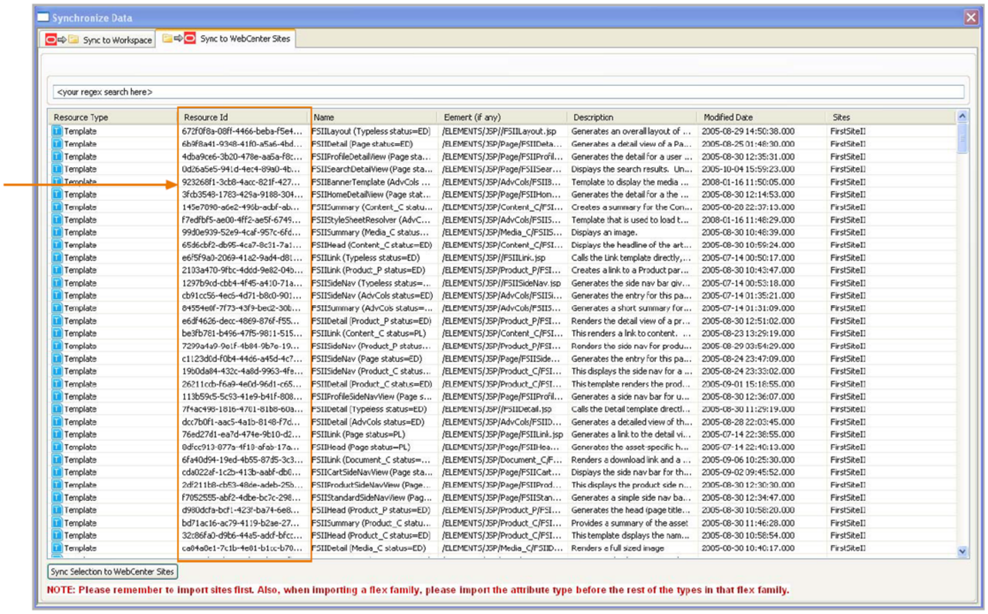
Asset	Local ID	fw_uid
Template A	45678	b07de0eb-5891
Template B	98765	c13ab4cd-4378

Deployment WebCenter Sites: Managed by the system administrator

Developer A
Developer B
Main Developer Tools Workspace for CS1
Main Developer Tools Workspace for CS2
VCS or Shared File System
Main Developer Tools Workspace for Deployment instance

When a resource is exported to a workspace, it is identified only by its `fw_uid`. ElementCatalog and SiteCatalog entries are not assigned an `fw_uid` because these entries are uniquely identified by element name.

The **Resource ID** column lists each WebCenter Sites resource by its `fw_uid` (with the exception of ElementCatalog and SiteCatalog entries).



Resource Type	Resource Id	Name	Element (if any)	Description	Modified Date	Sites
Template	672f0f8a-08ff-4466-beba-f5e4…	FSIILayout (Typeless status=ED)	/ELEMENTS/JSP//FSIILayout.jsp	Generates an overall layout of …	2005-08-29 14:50:38.000	FirstSiteII
Template	6b9f8a41-9348-41f0-a5a6-4bd…	FSIIDetail (Page status=ED)	/ELEMENTS/JSP/Page/FSIIDeta…	Generates a detail view of a Pa…	2005-08-25 01:48:30.000	FirstSiteII
Template	4dba9ce6-3b20-478e-aa5a-f8c…	FSIIProfileDetailView (Page sta…	/ELEMENTS/JSP/Page/FSIIProfil…	Generates the detail for a user …	2005-08-30 12:35:31.000	FirstSiteII
Template	0d26a5e5-941d-4ec4-89a0-4b…	FSIISearchDetailView (Page sta…	/ELEMENTS/JSP/Page/FSIISear…	Displays the search results. Un…	2005-10-04 15:59:23.000	FirstSiteII
Template	923268f1-3cb8-4acc-821f-427…	FSIIBannerTemplate (AdvCols …	/ELEMENTS/JSP/AdvCols/FSIIB…	Template to display the media …	2008-01-16 11:50:05.000	FirstSiteII
Template	3fcb3548-1783-429a-9188-304…	FSIIHomeDetailView (Page stat…	/ELEMENTS/JSP/Page/FSIIHom…	Generates the detail for a the …	2005-08-30 12:14:53.000	FirstSiteII
Template	145e7090-a6e2-496b-adbf-ab…	FSIISummary (Content_C statu…	/ELEMENTS/JSP/Content_C/FSI…	Creates a summary for the Con…	2005-08-28 22:37:13.000	FirstSiteII
Template	f7edfbf5-ae00-4ff2-ae5f-6749…	FSIIStyleSheetResolver (AdvC…	/ELEMENTS/JSP/AdvCols/FSIIS…	Template that is used to load t…	2008-01-16 11:48:29.000	FirstSiteII
Template	99d0e939-52e9-4caf-957c-6fd…	FSIISummary (Media_C status…	/ELEMENTS/JSP/Media_C/FSIIS…	Displays an image.	2005-08-30 10:48:39.000	FirstSiteII
Template	65d6cbf2-db95-4ca7-8c31-7a1…	FSIIHead (Content_C status=ED)	/ELEMENTS/JSP/Content_C/FSI…	Displays the headline of the art…	2005-08-30 10:59:24.000	FirstSiteII
Template	e6f5f9a0-2069-41a2-9ad4-d81…	FSIILink (Typeless status=ED)	/ELEMENTS/JSP//FSIILink.jsp	Calls the Link template directly,…	2005-07-14 00:50:17.000	FirstSiteII
Template	2103a470-9fbc-4ddd-9e82-04b…	FSIILink (Product_P status=ED)	/ELEMENTS/JSP/Product_P/FSI…	Creates a link to a Product par…	2005-08-30 10:43:47.000	FirstSiteII
Template	1297b9cd-cbb4-4f45-a410-71a…	FSIISideNav (Typeless status=…	/ELEMENTS/JSP//FSIISideNav.jsp	Generates the side nav bar giv…	2005-07-14 00:53:18.000	FirstSiteII
Template	cb91cc56-4ec6-4d71-b8c0-901…	FSIISideNav (AdvCols status=ED)	/ELEMENTS/JSP/AdvCols/FSIISi…	Generates the entry for this pa…	2005-07-14 01:35:21.000	FirstSiteII
Template	84554e0f-7f73-43f9-bed2-30b…	FSIISummary (AdvCols status=…	/ELEMENTS/JSP/AdvCols/FSIIS…	Generates a short summary for…	2005-07-14 01:31:09.000	FirstSiteII
Template	e6df4626-decc-4869-876f-f55…	FSIIDetail (Product_P status=ED)	/ELEMENTS/JSP/Product_P/FSI…	Renders the detail view of a pr…	2005-08-30 12:51:02.000	FirstSiteII
Template	be3fb781-b496-47f5-9811-515…	FSIILink (Content_C status=PL)	/ELEMENTS/JSP/Content_C/FSI…	This renders a link to content. …	2005-08-23 13:29:19.000	FirstSiteII
Template	7299a4a9-9e1f-4b84-9b7e-19…	FSIISideNav (Product_P status…	/ELEMENTS/JSP/Product_P/FSI…	Renders the side nav for produ…	2005-08-29 03:54:29.000	FirstSiteII
Template	c1123d0d-f0b4-44d6-a45d-4c7…	FSIISideNav (Page status=ED)	/ELEMENTS/JSP/Page/FSIISide…	Generates the entry for this pa…	2005-08-24 23:47:09.000	FirstSiteII
Template	19b0da84-432c-4a8d-9963-4fe…	FSIISideNav (Product_C status…	/ELEMENTS/JSP/Product_C/FSI…	This displays the side nav for a …	2005-08-24 23:33:02.000	FirstSiteII
Template	26211ccb-f6a9-4e0d-96d1-c65…	FSIIDetail (Product_C status=ED)	/ELEMENTS/JSP/Product_C/FSI…	This template renders the prod…	2005-09-01 15:18:55.000	FirstSiteII
Template	113b59c5-5c93-41e9-b41f-808…	FSIIProfileSideNavView (Page s…	/ELEMENTS/JSP/Page/FSIIProfil…	Generates a side nav bar for u…	2005-08-30 12:36:07.000	FirstSiteII
Template	7f4ac498-1816-4701-81b8-60a…	FSIIDetail (Typeless status=ED)	/ELEMENTS/JSP//FSIIDetail.jsp	Calls the Detail template directl…	2005-08-30 11:29:19.000	FirstSiteII
Template	dcc7b0f1-aac5-4a1b-8148-f7d…	FSIIDetail (AdvCols status=ED)	/ELEMENTS/JSP/AdvCols/FSIID…	Generates a detailed view of th…	2005-08-28 22:03:45.000	FirstSiteII
Template	76ed27d1-ea7d-474e-9b10-d2…	FSIILink (Page status=PL)	/ELEMENTS/JSP/Page/FSIILink.jsp	Generates a link to the detail vi…	2005-07-14 22:38:55.000	FirstSiteII
Template	8dfcc913-877a-4f18-afab-17a…	FSIIHead (Page status=PL)	/ELEMENTS/JSP/Page/FSIIHea…	Generates the asset-specific h…	2005-07-14 22:40:13.000	FirstSiteII
Template	6fa40d94-19ed-4b55-87d5-3c3…	FSIILink (Document_C status=…	/ELEMENTS/JSP/Document_C/F…	Renders a download link and a …	2005-09-06 10:25:30.000	FirstSiteII
Template	cda022af-1c2b-413b-aabf-db0…	FSIICartSideNavView (Page sta…	/ELEMENTS/JSP/Page/FSIICart…	Displays the side nav bar for th…	2005-09-02 09:45:52.000	FirstSiteII
Template	2df211b8-cb53-48de-adeb-25b…	FSIIProductSideNavView (Page…	/ELEMENTS/JSP/Page/FSIIProd…	This displays the product side n…	2005-08-30 12:30:30.000	FirstSiteII
Template	f7052555-abf2-4dbe-bc7c-298…	FSIIStandardSideNavView (Pag…	/ELEMENTS/JSP/Page/FSIIStan…	Generates a simple side nav ba…	2005-08-30 12:34:47.000	FirstSiteII
Template	d980dcfa-bcf1-423f-ba74-6e8…	FSIIHead (Product_P status=ED)	/ELEMENTS/JSP/Product_P/FSI…	Generates the head (page title…	2005-08-30 10:58:20.000	FirstSiteII
Template	bd71ac16-ac79-4119-b2ae-27…	FSIISummary (Product_C statu…	/ELEMENTS/JSP/Product_C/FSI…	Provides a summary of the asset	2005-08-30 11:46:28.000	FirstSiteII
Template	32c86fa0-d9b6-44a5-adcf-bfcc…	FSIIHead (Product_C status=ED)	/ELEMENTS/JSP/Product_C/FSI…	This template displays the nam…	2005-08-30 10:58:54.000	FirstSiteII
Template	ca04a0e1-7c1b-4e01-b1cc-b70…	FSIIDetail (Media_C status=ED)	/ELEMENTS/JSP/Media_C/FSIID…	Renders a full sized image	2005-08-30 10:40:17.000	FirstSiteII

NOTE: Please remember to import sites first. Also, when importing a flex family, please import the attribute type before the rest of the types in that flex family.

56.4.2 Overriding a Resource's fw_uid

When a resource is created, a UUID value is automatically generated as its globally unique identifier and stored in an asset attribute named `fw_uid`. Advanced developers can use the Asset API to override the default `fw_uid` scheme with their own by modifying the `fw_uid` attribute. For information about using the asset API, see the *Oracle Fusion Middleware WebCenter Sites Java API Reference*.

> **Note:** We recommend using the default WebCenter Sites `fw_uid` scheme. If you override a resource's default `fw_uid` value, you must make sure the value is unique across all WebCenter Sites instances. Once you set a resource's `fw_uid` attribute, **do not** change the value.

56.4.3 Using Developer Tools with Pre-Existing Resources

If your Oracle WebCenter Sites system is an upgrade from FatWire Content Server, some of its pre-existing resources may have their `fw_uid` values set to `CSSystem:[type]:id`. However, as of Content Server version 7.6, a resource's `fw_uid` is generated as a UUID value. Developer Tools can map resources with either type of `fw_uid` value, as long as the resource's `fw_uid` value is globally unique. Therefore, you can continue to use a pre-existing resource's current `fw_uid` value (in the format of `CSSystem:[type]:id`).

When using Developer Tools to work with pre-existing resources, do one of the following (or both):

- We recommend continuing to use the pre-existing resource's `fw_uid` value of `CSSystem:[type]:id`. However, you must ensure that no other WebCenter Sites instance has generated the same `fw_uid` value for a different resource. For example, if you have a WebCenter Sites development instance and you published resources to a management instance, the `fw_uid` values of the published resources remain the same on both instances. Therefore, synchronizing resources between these two instances using Developer Tools will not result in ID conflicts.

- If you have pre-existing resources that were created on separate FatWire Content Server instances, but with identical `fw_uid` values, each of those resources must be assigned a new, unique `fw_uid` value. To avoid ID conflicts, you can either remove the current `fw_uid` value and allow Developer Tools to generate a new UUID value when you export the resource from a WebCenter Sites instance, or you can assign your own unique identifier to the resource. For instructions, see Section 56.4.2, "Overriding a Resource's fw_uid."

> **Note:** If you assign a resource a new `fw_uid`, make sure to assign the new `fw_uid` value to every instance of that resource. For example, if you published the resource to another WebCenter Sites instance before modifying its `fw_uid` value, make sure you assign the same `fw_uid` to both copies of that resource.

56.5 Site Mappings

Most WebCenter Sites resources, such as assets, are associated with at least one site. When a resource is exported from a WebCenter Sites instance to a workspace, it stores a complete (canonical) list of sites with which it is associated in its `.main.xml` file. The resource's canonical list remains the same on every WebCenter Sites instance, unless you add a new site affiliation, remove a current one, or (if you are an advanced developer) override the resource's natural site mapping using the command-line tool.

This section contains the following topics:

- Section 56.5.1, "Natural Site Mappings"
- Section 56.5.2, "Overriding Natural Site Mappings With the Command-line Tool"

56.5.1 Natural Site Mappings

By default, Developer Tools maps resources to their associated sites by referencing the canonical list stored in a resource's `.main.xml` file. If any of the sites referenced in this list exist on the WebCenter Sites instance to which the resource is imported, Developer Tools maps the resource to those sites. If none of the sites referenced in the resource's canonical list exist on the WebCenter Sites instance, the import fails.

For example, Developer A installs two sites; News and Sports. On a separate WebCenter Sites instance, Developer B also installs two sites; News and Weather. Both developers import the same Template asset into their WebCenter Sites instances. This Template asset is associated with both the Sports and Weather sites (both sites are referenced in the asset's canonical list). Upon import, Developer Tools references the Template asset's canonical list and then maps the asset to the Sports site on Developer A's environment and the Weather site on Developer B's environment.

When Developers A and B share the changes they made to the Template asset with each other, Developer Tools maps the asset to the appropriate sites on both WebCenter Sites instances. The canonical list enables Developer Tools to recognize the sites with which the Template asset is associated, even when the asset is exported into an instance where some of those sites are not installed.

56.5.2 Overriding Natural Site Mappings With the Command-line Tool

Advanced developers can use the command-line tool to import a resource into sites that are not referenced in its canonical list. The command-line tool enables you to create reusable modules, which are workspaces containing resources that can be imported into any site.

For example, a developer creates a blogging solution within the FirstSiteII sample site. This solution includes resources such as a flex family, assets, and Templates. The developer wants the resources to be imported into various sites, including sites that do not exist yet. Since he is an advanced developer, he uses the command-line tool to export the desired resources to an empty workspace, and then archives the content of this workspace (using a `.zip` or `.tar` format). Using the command-line tool, other developers can then customize the site mappings of the resources contained in this module and manually specify the sites into which the module will be imported.

For more information about using the command-line tool, see Chapter 58, "Developer Tools: Command-Line Tool." For a detailed scenario of creating a reusable module, see Chapter 61, "Developer Tools: Using the Command-line Tool to Create Reusable Modules."

57

Developer Tools: Workspaces

This chapter contains information about how Developer Tools stores resources exported from an integrated WebCenter Sites instance.

This chapter contains the following sections:

57.1 Introduction to Workspaces

A workspace is a disk-based repository of serialized WebCenter Sites data which represent resources from either the workspace's WebCenter Sites instance or another instance's workspace. Workspaces can store any type of WebCenter Sites resource including assets, flex families, sites, and so on. Each workspace is associated with one WebCenter Sites instance.

By default, Eclipse provides each WebCenter Sites instance with a main Developer Tools workspace (located in the Eclipse project folder) which is used for continuous development when working in the Eclipse IDE. Custom workspaces can be created by advanced developers using the Developer Tools command-line tool. Custom workspaces can be used for special projects such as creating modules. For more information about creating custom workspaces, see Chapter 58, "Developer Tools: Command-Line Tool."

With the use of a version control system (such as Subversion) or a shared file system, resources stored on one workspace can be exchanged with other workspaces. Any resource exported from a WebCenter Sites instance into the associated workspace can be copied to another WebCenter Sites instance's workspace. This makes the resource available for import into the second workspace's associated WebCenter Sites instance. For more information about sharing resources between different workspaces, see Chapter 59, "Developer Tools: Integrating with Version Control Systems."

57.2 Workspace Structure

Workspaces are created under the `export/envision` folder inside the WebCenter Sites installation directory. The main Developer Tools workspace is located under the export/envision/cs_workspace folder. The main Developer Tools workspace is the only visible workspace in the Eclipse project folder.

All workspaces have the same structure. Each resource contained in a workspace is stored as a single file or several interrelated files. The main file for each resource ends in .main.xml and contains resource-specific metadata. This main file also contains links to other files associated with the resource (such as an attached document, a JSP file, or a blob). This enables each resource to be fully self-contained, as long as all of a resource's associated files are stored in the workspace. Otherwise, the resource is incomplete.

If a resource has multiple files, those files are listed in the bottom section of the .main.xml file as storable0, storable1, and so on. The associated files of any given resource have similar names. This way, all of a resource's associated files appear together, except ElementCatalog entries which are stored separately to preserve their original root path.

The location of a resource's files in the workspace depends on the type of resource. The workspace is divided into the following sections:

- src/_metadata: The metadata section of a given resource which contains assets, asset types, sites, roles, and so on. In addition, legacy XML code is stored under the ELEMENTS/ subfolder.
- src/jsp/cs_deployed: This section stores a resource's JSP file under its proper path.

Since workspaces have a highly consistent structure, resources from one workspace can be copied to another. As with all file system copy operations, ensure you are not overwriting files that have the same name.

57.3 Asset Storage Structure

Assets are stored under folders named src/_metadata/ASSET/asset type. Under this structure there is a two-level hash-based hierarchy, which contains asset data. The name of the asset file is based on the asset name and its fw_uid value. If the asset includes attached documents or blobs, the file name is based on the asset name, attribute name, fw_uid value, and the name of the document or blob (if any).

For example, a Document_C asset named *FSII IES_Manual.pdf* contains an attached document called *IES_MDPlayer_Manual.pdf*. Therefore, this asset is stored as two separate files:

- The first is the .main.xml file, which contains the asset's metadata and links to the files associated with the asset:

```
.src/_metadata/ASSET/Document_C/8/0/FSII IES_MDPlayer_
    Manual.pdf(aa0b47b5-f558-49d4-a6ac2ee012d1b75).main.xml
```

- The second is the actual document, which is a PDF file in this example:

```
.src/_metadata/ASSET/Document_C/8/0/FSII IES_MDPlayer_
    Manual.pdf.FSIIDocumentFile(aa0b47b5-f558-49d4-8a6a-
    c2ee012d1b75).IES_MDPlayer_Manual.pdf
```

Note: Since all file names of the asset are based on the asset's name, renaming the asset also renames the file. If you are tracking the asset in a VCS, delete the file with the old name.

57.4 Code-Based Resource Storage Structure

Templates, CSElements, and ElementCatalog entries are stored under the storage path required by their code elements. The JSP files associated with code-based resources are stored in the workspace under `src/jsp/cs_deployed` and the XML elements are stored under `src/_metadata/ELEMENTS`. The metadata files of code-based resources are stored under the same name as the resource's JSP with the appended `.main.xml` extension. Therefore, the code-based resource's metadata, JSP, and XML files are grouped together in the workspace.

57.5 Attribute Editor Storage Structure

Attribute editors are tracked as assets, but also have implicit references to a set of ElementCatalog entries. An attribute editor's ElementCatalog entries are tracked independently.

For example, the TextArea editor uses the `OpenMarket/Gator/AttributeTypes/TEXTAREA` ElementCatalog entry, which is registered as a dependency. Developer Tools maintains the following files for the TextArea editor:

- The `.main.xml` file:

  ```
  src/_metadata/ASSET/AttrTypes/9/10/TextArea(e64f983d-9c7c-489baedb-
      476d56f8121e).main.xml
  ```

- The `urlxml` metadata file:

  ```
  src/_metadata/ASSET/AttrTypes/9/10/TextArea.urlxml(e64f983d-9c7c-
      489b-aedb-476d56f8121e).1095346398911.txt
  ```

- The ElementCatalog entry, tracked as an independent resource:
  - The `.main.xml` file of the ElementCatalog entry:

    ```
    src/_metadata/ELEMENTS/OpenMarket/Gator/AttributeTypes/TEXTAREA
        .xml.main.xml
    ```

  - The attribute editor's element code:

    ```
    src/_metadata/ELEMENTS/OpenMarket/Gator/AttributeTypes/
        TEXTAREA.xml
    ```

57.6 Asset Type Storage Structure

Asset types have a main metadata part and a set of elements. For example, the following is the structure of a Page asset type:

- The main metadata of the page is stored in the `.main.xml` file:

  ```
  src/_metadata/Asset_Type/Page(b8d8ae9-14cc-4554-b80e-0c22e39a3ec8)
      .main.xml
  ```

- The associated elements are tracked independently (each element has its own `.main.xml` file):

  ```
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
      SearchForm.xml
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
      CheckDelete.xml
  ```

```
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ContentForm.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ContentDetails.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    LoadSiteTree.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexReplace.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    LoadTree.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexAdd.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    SearchForm.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexReplace.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PreviewPage.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    LoadTree.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PreUpdate.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    Tile.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    SimpleSearch.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    SimpleSearch.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ContentForm.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    AppendSelectDetailsSE.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    LoadSiteTree.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    AppendSelectDetails.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ManageSchVars.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PreviewPage.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    CheckDelete.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ManageSchVars.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/
    Page/PreUpdate.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    AppendSelectDetails.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexCreateVerity.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ContentDetails.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PostUpdate.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexAdd.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexCreateVerity.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    Tile.xml
```

```
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    AppendSelectDetailsSE.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PostUpdate.xml.main.xml
```

58

Developer Tools: Command-Line Tool

This chapter is for advanced developers and provides information about running and using the command-line tool.

The Developer Tools command-line tool can be used for deployment and other resource movement activities. Unlike the Eclipse integration, which enables you to work only with the Developer Tools workspace, the command-line tool enables you to work with any workspace. The command-line tool also provides import and export features which are not available when working in the Eclipse IDE. For example, developers can create reusable modules, which are workspaces containing resources that can be imported into any site.

This chapter contains the following sections:

- Section 58.1, "Running and Using the Command-Line Tool"
- Section 58.2, "Example Commands"
- Section 58.3, "Creating Modules"

58.1 Running and Using the Command-Line Tool

To run the command-line tool:

1. Unzip `csdt.zip`, which is located in the WebCenter Sites distribution package. Open the `csdt-client` folder and place the `csdt-client.jar` file in the classpath. Make sure you have met all the requirements listed in the *Oracle Fusion Middleware WebCenter Sites Certification Matrix* available from the Oracle Technology Network at `http://otn.oracle.com`.

2. Run the command-line tool (`cmd`) and type the following command:

```
java com.fatwire.csdt.client.main.CSDT  [ContentServer url]
    username= username   password= password
    cmd=export|import|listcs|listds    [options]
```

   Replace the placeholder parameters with the information about your development environment and the desired command you wish to execute:

   – `ContentServer url`: The URL of your local WebCenter Sites instance, including the `ContentServer` servlet (for example, `http://localhost:8080/cs/ContentServer`)

   – `username` and `password`: The user name and password of a WebCenter Sites general administrator. This user must be a member of the `RestAdmin` group (for example, `fwadmin/xceladmin`).

   – `cmd`: The command to execute. The following commands are available:

  - `export`: Export data from WebCenter Sites to a workspace
  - `import`: Import data into WebCenter Sites from a workspace
  - `listcs`: List WebCenter Sites content
  - `listds`: List workspace content

- `options`: Specify one of the following to either import or export:
  - `resources`: Specify which resources you wish to import or export in a semicolon-separated list of resource type and resource ID. To specify multiple resources, use a comma-separated list. To specify all resources of a given type, use the `*` symbol. If you are exporting a resource (to a workspace), specify the resource's local ID. For example, use `resources=Content_C:12345;Product_C:*` to export a specific Content_C asset and all Product_C assets.

    If you are importing a resource (to a WebCenter Sites instance), specify the resource's `fw_uid`. To get the resource's `fw_uid`, use the `listds` option.

    The following is a full listing of resource selectors:

    `@SITE`: Specify the desired sites

    `@ROLE`: Specify the desired roles

    `@ASSET_TYPE`: Specify the desired asset types

    `@TREETAB`: Specify the desired tree tabs

    `@STARTMENU`: Specify the desired start menu items

    `@ELEMENTCATALOG`: Specify the desired ElementCatalog entries

    `@SITECATALOG`: Specify the desired site catalog entries

    `@ALL_NONASSETS`: Use this short-hand notation to select all non-asset resources

    `@ALL_ASSETS`: Use this short-hand notation to select all available assets

    `asset type`: Specify assets of a certain type.

Note: To verify that selectors are picking up the correct resources before import or export, use `listcs` for export activities and `listds` for import activities. These commands fine-tune the selectors before execution by providing a list of the resources that will be moved.

If resources have dependencies, they are exported and imported automatically. However, dependencies are not listed using the `listcs` and `listds` commands.

  - `fromSites`: Select resources from specified sites only.
  - `toSites`: (Import only) Override the natural site affiliation during import with a comma-separated list of sites. Specified sites must exist on the target system.
  - `modifiedSince`: (Assets only) Select only resources that have been modified since the specified date. The date format is `yyyy-mm-dd hh:MM:ss`.

  - `datastore`: Specify the workspace you wish to either export WebCenter Sites resources to or import WebCenter Sites resources from. If you do not specify a value for this parameter, the main Developer Tools workspace is specified by default. If you are exporting resources and specify a workspace that does not exist, the command-line tool automatically creates the workspace and exports the desired resources to it.

58.2 Example Commands

The following is a list of example commands that can be executed using the command-line tool:

- This command exports the specified Content_C assets and all Product_C assets that belong to FirstSiteII and were modified since the specified date. Since no workspace is specified, the Developer Tools workspace is used by default:

```
java com.fatwire.csdt.client.main.CSDT  http://localhost:8080/cs/ContentServer
username=bob password=password resources=Content_
C:123432123423,11234234212,111234341234;Product_C:* fromSite=FirstSiteII
modifiedSince=2010-08-08 19:14:00 cmd=export
```

- This command imports the specified Content_C asset and all Product_C assets found in the workspace. Since no workspace is specified, the Developer Tools workspace is used by default:

```
java com.fatwire.csdt.client.main.CSDT  http://localhost:8080/cs/ContentServer
username=bob password=password resources=Content_
C:aad618e9-f04e-4ee4-b902-076224bb6f7b;Product_C:* fromSite=FirstSiteII
cmd=import
```

- This command exports all resources from the site SecondSiteII into a workspace named "TheOutput":

```
java com.fatwire.csdt.client.main.CSDT  http://localhost:8080/cs/ContentServer
username=bob password=password resources=@ALL_ASSETS:*;@ALL_NONASSETS:*
fromSite=SecondSiteII  datastore=TheOutput cmd=export
```

- This command imports all assets and tree tabs from the workspace named "TheInput" into the site MySite:

```
java com.fatwire.csdt.client.main.CSDT  http://localhost:8080/cs/ContentServer
username=bob password=password resources=@ALL_ASSETS:*;@TREETAB:*
toSites=MySite  datastore=TheInput cmd=import
```

58.3 Creating Modules

Modules are sets of related resources exported from your WebCenter Sites instance into a given workspace. The `datastore` parameter enables you to specify the workspace you wish to either export WebCenter Sites resources to or import WebCenter Sites resources from. If you export WebCenter Sites resources to a workspace that does not exist, the command-line tool automatically creates that workspace and exports the desired resources into it.

Modules are reusable, and their content can be imported into any CM sites (even if the site is not listed in the resources' canonical list of sites). To import a module into a CM site, you must execute an import command. In the `datastore` parameter, specify the workspace that contains the desired resources and in the `toSites` parameter, specify the site(s) to which you wish to import those resources. This imports the content of the workspace into the specified CM site(s).

59

Developer Tools: Integrating with Version Control Systems

This chapter provides information about storing the resources, contained in the Developer Tools workspace, in a version control system (VCS). This enables you to share the resources in your Developer Tools workspace with other developers.

This chapter contains the following sections:

- Section 59.1, "About Version Control With Developer Tools"
- Section 59.2, "Integrating Developer Tools With a VCS"
- Section 59.3, "Working With a Developer Tools-Integrated VCS"

59.1 About Version Control With Developer Tools

Version control systems (VCS) provide you with the ability to create source code repositories. A VCS can provide advanced tools for versioning, branching, and managing source files. The file system structure in which the Developer Tools workspace stores WebCenter Sites resources enables those resources to be stored on any VCS and enables complete CM sites to be tracked in a VCS.

59.2 Integrating Developer Tools With a VCS

The Developer Tools workspace is located in the `src` folder of the Eclipse project. This folder can be accessed directly from the WebCenter Sites installation directory (under `export/envision/cs_workspace/src`). To copy the content of your Developer Tools workspace folder to a VCS, you must first determine which VCS you wish to use. Then, check-in the resources stored in the Developer Tools workspace to the VCS. The VCS you choose to use, determines the steps you must take to check resources in from the Eclipse IDE.

In some cases Eclipse supports the VCS you choose to use by providing a plug-in which lets you check resources into the VCS directly from Eclipse. For example, if you use the Subversion repository to store the content of your Developer Tools workspace, the Eclipse IDE supports the Subclipse plug-in. Therefore, you can check resources into the Subversion directory directly from the Eclipse IDE.

The Developer Tools workspace stores all resources as one or more files, depending on the type of resource. If you check a resource into a VCS, you must also check-in all associated files of that resource. For example, an asset that contains attached documents (such as a PDF) is represented by a metadata file (`.main.xml`) and the associated document file(s). All associated files of the asset must be checked in to the VCS. Otherwise, the check-in fails. For a detailed description of the Developer Tools workspace layout and for information about how resources are mapped to workspace files, see Chapter 57, "Developer Tools: Workspaces."

Note: Checking data into a VCS from the Developer Tools workspace does not require an extensive understanding of the Developer Tools workspace file structure. Instead, most VCS clients detect incremental changes to the Developer Tools workspace folder and indicate those changes during a VCS commit operation.

59.3 Working With a Developer Tools-Integrated VCS

When you check WebCenter Sites resources into a VCS from your Developer Tools workspace, you are able to exchange those resources with other developers and track changes to those resources over time. The following is an example of a development team using a VCS to share WebCenter Sites resources:

Developer A creates a resource in WebCenter Sites and exports it to the Developer Tools workspace. Developer A then checks that resource into a VCS. From the VCS, Developer B can then check-out the resource to his own Developer Tools workspace. This developer can now modify the resource and then check the changes back into the VCS. Developer A, as well as the rest of the development team, can now see the changes made to the resource from the VCS. This enables the members of the development team to synchronize their Developer Tools workspaces with the most recent changes made to the resource. Additional developers can join the group by checking-out resources from the VCS into their own, respective Developer Tools workspaces. As the project advances, the cycle of adding and modifying resources continues.

Note: WebCenter Sites provides a revision tracking system for resources that are kept within a given WebCenter Sites instance. The WebCenter Sites revision tracking system cannot be integrated with a VCS.

60

Developer Tools: Development Team Integration Use Case

This chapter contains a development scenario involving a team of developers using Developer Tools to create a CM site and resources. The development team uses the synchronization tool provided by Developer Tools to manage and exchange resources between multiple WebCenter Sites instances. Using the command-line tool, the CM site and its resources will then be deployed as a nightly build.

The sequence of events for the scenario are as follows:

- Section 60.1, "Today: Develop a Site and Associated Resources"
- Section 60.2, "Three Days Later... Deployment"

60.1 Today: Develop a Site and Associated Resources

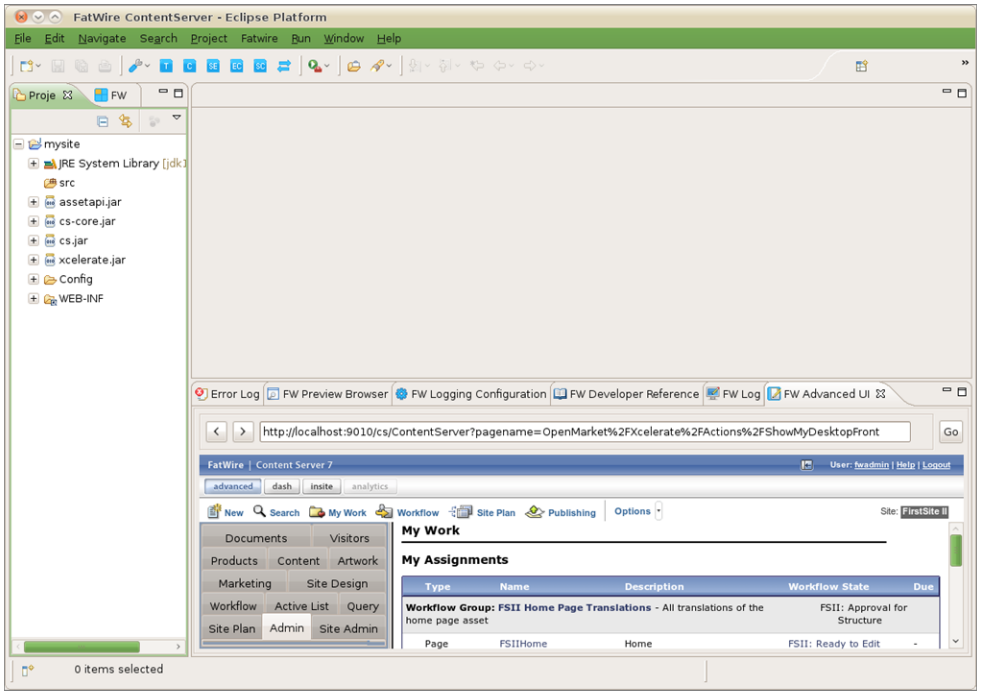
7:14 am: The New Project is Assigned

Artie the architect wakes up and finds himself appointed the leader of a new web-based project.

7:34 am: Setting Up Developer Tools

Artie gets some coffee and installs a WebCenter Sites instance on his laptop. He then starts the Eclipse IDE and configures the Developer Tools kit.

> **Note:** To successfully integrate Eclipse with a WebCenter Sites instance, the configuration screen requires Artie to enter the user name and password of a general administrator. This user must be a member of the `RestAdmin` group.

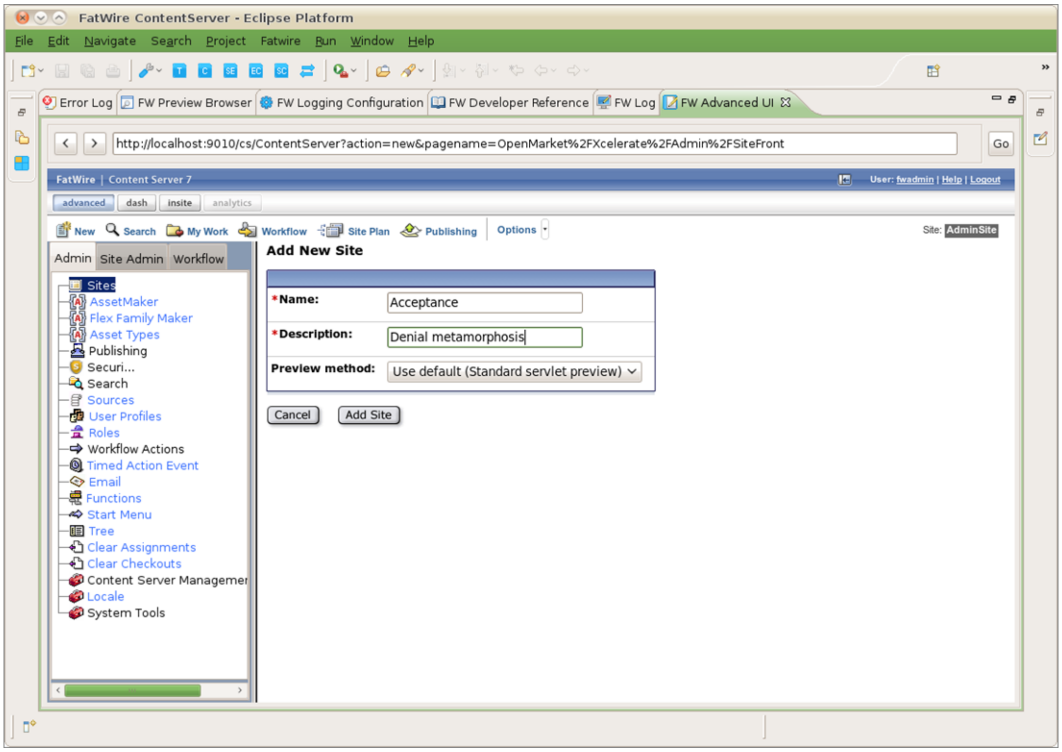

FatWire ContentServer - Eclipse Platform
File Edit Navigate Search Project Fatwire Run Window Help
Proje FW
mysite
JRE System Library [jdk]
src
assetapi.jar
cs-core.jar
cs.jar
xcelerate.jar
Config
WEB-INF
Error Log FW Preview Browser FW Logging Configuration FW Developer Reference FW Log FW Advanced UI
Go
FatWire | Content Server 7
User: fwadmin | Help | Logout
advanced dash insite analytics
New Search My Work Workflow Site Plan Publishing Options
Site: FirstSite II
Documents Visitors
Products Content Artwork
Marketing Site Design
Workflow Active List Query
Site Plan Admin Site Admin
My Work
My Assignments
0 items selected

http://localhost:9010/cs/ContentServer?pagename=OpenMarket%2FXcelerate%2FActions%2FShowMyDesktopFront

Type	Name	Description	Workflow State	Due
Workflow Group: FSII Home Page Translations - All translations of the home page asset			FSII: Approval for Structure	
Page	FSIIHome	Home	FSII: Ready to Edit	-

7:45 am: Create the Site Definition

Artie creates the site definition (naming the site "Acceptance" in this scenario) by using the embedded WebCenter Sites Admin interface view in Eclipse.


FatWire ContentServer - Eclipse Platform
File Edit Navigate Search Project Fatwire Run Window Help
Error Log FW Preview Browser FW Logging Configuration FW Developer Reference FW Log FW Advanced UI
Go
FatWire | Content Server 7
User: fwadmin | Help | Logout
advanced dash insite analytics
New Search My Work Workflow Site Plan Publishing Options
Site: AdminSite
Admin Site Admin Workflow
Sites
AssetMaker
Flex Family Maker
Asset Types
Publishing
Securi...
Search
Sources
User Profiles
Roles
Workflow Actions
Timed Action Event
Email
Functions
Start Menu
Tree
Clear Assignments
Clear Checkouts
Content Server Managemer
Locale
System Tools
Add New Site
Cancel
Add Site

http://localhost:9010/cs/ContentServer?action=new&pagename=OpenMarket%2FXcelerate%2FAdmin%2FSiteFront

*Name: Acceptance

*Description: Denial metamorphosis

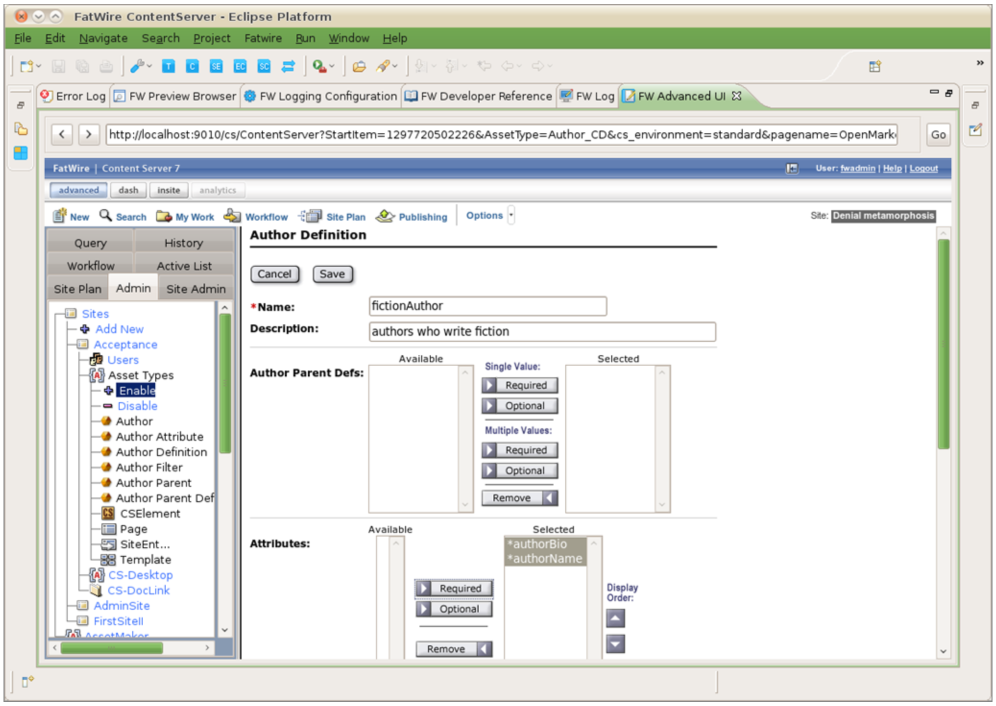
Preview method: Use default (Standard servlet preview)

Artie could have used a separate browser window running the WebCenter Sites Admin interface to create the site definition. However, being a huge Eclipse fan, he indulges in the fact that he can usually write complete WebCenter Sites CM sites without leaving Eclipse.

7:46 am: Create Resources for the Site

Artie primes the site:

- Enables asset types.
- Assigns permissions.
- Creates and enables a flex family to store information assets (author information assets in this scenario) for the site:
  - Flex Attribute: `Author_A`
  - Flex Parent Definition: `Author_PD`
  - Flex Definition: `Author_CD`
  - Flex Parent: `Author_P`
  - Flex Asset: `Author_C`
  - Flex Filter: `Author_F`
- Creates flex attributes (authorName and authorBio) and a flex definition (fictionAuthor). He then adds the attributes to the flex definition.



FatWire ContentServer - Eclipse Platform

http://localhost:9010/cs/ContentServer?StartItem=1297720502226&AssetType=Author_CD&cs_environment=standard&pagename=OpenMark

FatWire | Content Server 7

User: fwadmin | Help | Logout

Site: Denial metamorphosis

Author Definition

*Name: fictionAuthor

Description: authors who write fiction

Author Parent Defs:

Attributes: Selected: *authorBio, *authorName

8:12 am: The VCS Discussion

Artie arrives at the office and meets with the rest of the development team; Sonoko (coder), Matthäus (coder), and Yogesh (system engineer). The discussion is about whether to use a version control system for the project:

Yogesh: I can set up a version control system in-house, but I would like to avoid doing extra work. Do you guys really want one?

Artie: Well, we expect this project to last several months. We could just create a shared folder on the network and synchronize all our work to it. However, we have to be careful not to overwrite each other's work. For example, if two people are working on the same Template asset, they will have to wait for each other.

Sonoko: Artie, do you remember how the last project turned out to be very intense toward the end? Waiting for other people to finish their work is so unnerving when you have all this pressure from the management. I would much rather use a version control system. Also, can we keep the repository on the web this time so I can work from Stellarbucks when I'm bored?

Matthäus: I have to agree with Sonoko. We can get SVN hosting for next to nothing. We can even get an SVN with SSL for peace of mind.

Yogesh: If I don't have time to set up an in-house SVN, I could at least get you an SVN hosting subscription.

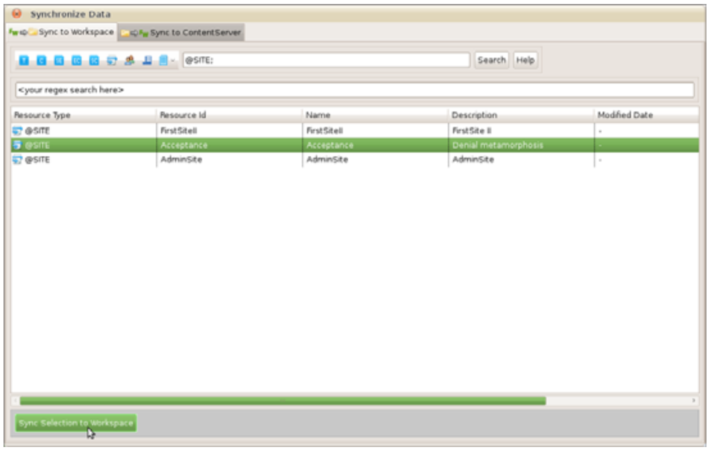
Artie: OK then, I guess we'll go with SVN. Anything else?

Artie and the rest of the development team decide to use SVN to track the resources of their site.

9:42 am: Synchronizing Workspaces With a VCS

Artie and his team install the Subclipse plug-in from `http://subclipse.tigris.org/`. Now, Artie needs to check in the site and resources he created earlier:

1. Using the Developer Tools Synchronization screen in Eclipse, Artie accesses the "Sync to Workspace" tab and enters the `@Site` selector in the search field to retrieve a listing of all the sites on his WebCenter Sites instance.



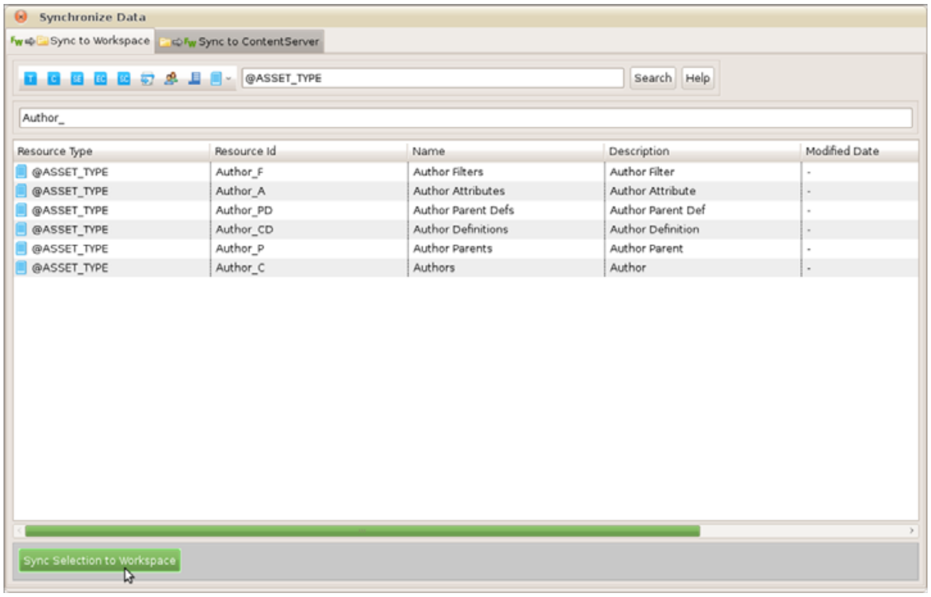
@SITE:

<your regex search here>

Resource Type	Resource Id	Name	Description	Modified Date
@SITE	FirstSiteII	FirstSiteII	FirstSite II	-
@SITE	Acceptance	Acceptance	Denial metamorphosis	-
@SITE	AdminSite	AdminSite	AdminSite	-

   Artie selects the site he created earlier ("Acceptance" site) and clicks the **Sync Selection to Workspace** option to export the site definition from his WebCenter Sites instance to his workspace.

2. Next, Artie exports the site's associated flex family to the workspace. He uses the `@ASSET_TYPE` selector to list all the assets on his WebCenter Sites instance. To narrow down the results, he uses the `Author_` search string. Artie then selects all listed items and clicks **Sync Selection to Workspace**.



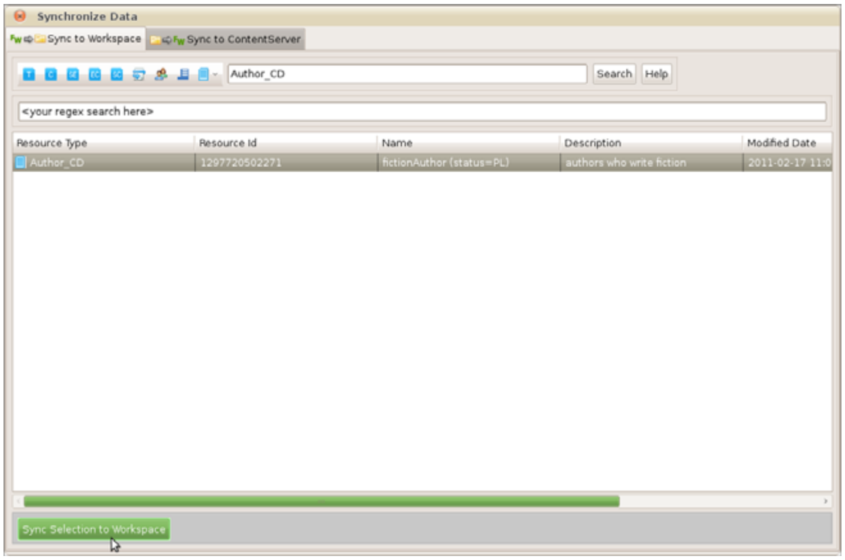
@ASSET_TYPE

Author_

Resource Type	Resource Id	Name	Description	Modified Date
@ASSET_TYPE	Author_F	Author Filters	Author Filter	-
@ASSET_TYPE	Author_A	Author Attributes	Author Attribute	-
@ASSET_TYPE	Author_PD	Author Parent Defs	Author Parent Def	-
@ASSET_TYPE	Author_CD	Author Definitions	Author Definition	-
@ASSET_TYPE	Author_P	Author Parents	Author Parent	-
@ASSET_TYPE	Author_C	Authors	Author	-

The flex family types are serialized to the workspace, including their type-specific ElementCatalog entries.

3. Now, Artie exports the flex definition to his workspace. He uses the `Author_CD` selector, which lists all available definitions of that type. In this case, there is only one definition (fictionAuthor).



Author_CD

<your regex search here>

Resource Type	Resource Id	Name	Description	Modified Date
Author_CD	1297720502271	fictionAuthor (status=PL)	authors who write fiction	2011-02-17 11:0

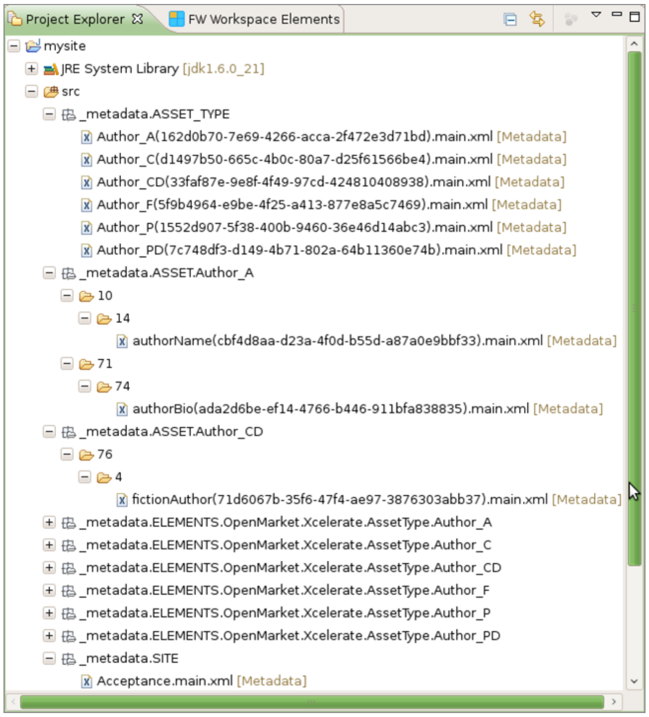
Note: Artie did not select the flex attributes (Author_A instances) on which the site definition depends because he knows the Developer Tools kit synchronizes them automatically with the definition.

---

4. Artie takes a quick look at his workspace in the Eclipse "Project Explorer" view to verify that all his work is there. From top to bottom, he sees the following under the project's `src` folder:

   – `_metadata.ASSET_TYPE` entries for each asset type he synchronized

- `_metadata.ASSET.Author_A` files for both of the Author_A attributes
- `_metadata.ASSET.Author_CD` file for the serialized definition
- `_metadata.ELEMENTS` entries for ElementCatalog entries related to each of the serialized asset types
- `_metadata.SITE` entry for the site definition

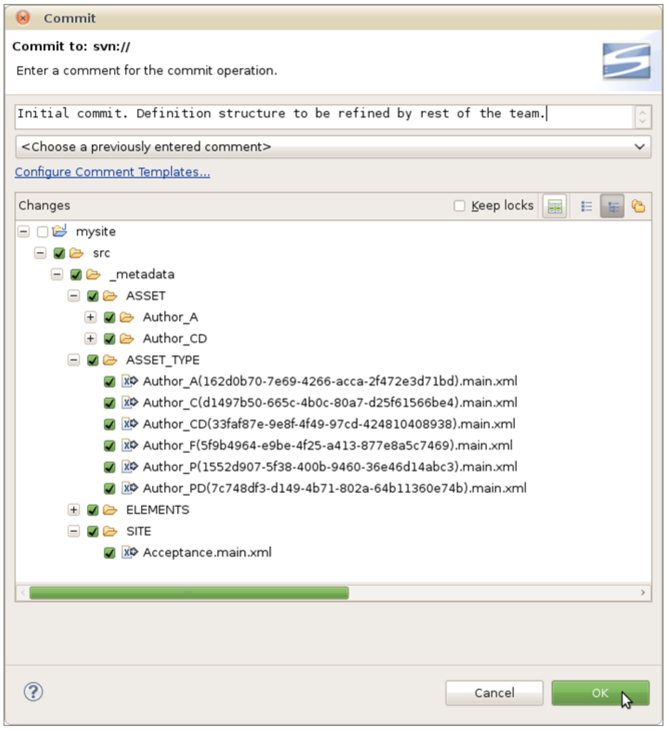


```
mysite
  JRE System Library [jdk1.6.0_21]
  src
    _metadata.ASSET_TYPE
      Author_A(162d0b70-7e69-4266-acca-2f472e3d71bd).main.xml [Metadata]
      Author_C(d1497b50-665c-4b0c-80a7-d25f61566be4).main.xml [Metadata]
      Author_CD(33faf87e-9e8f-4f49-97cd-424810408938).main.xml [Metadata]
      Author_F(5f9b4964-e9be-4f25-a413-877e8a5c7469).main.xml [Metadata]
      Author_P(1552d907-5f38-400b-9460-36e46d14abc3).main.xml [Metadata]
      Author_PD(7c748df3-d149-4b71-802a-64b11360e74b).main.xml [Metadata]
    _metadata.ASSET.Author_A
      10
        14
          authorName(cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33).main.xml [Metadata]
      71
        74
          authorBio(ada2d6be-ef14-4766-b446-911bfa838835).main.xml [Metadata]
    _metadata.ASSET.Author_CD
      76
        4
          fictionAuthor(71d6067b-35f6-47f4-ae97-3876303abb37).main.xml [Metadata]
    _metadata.ELEMENTS.OpenMarket.Xcelerate.AssetType.Author_A
    _metadata.ELEMENTS.OpenMarket.Xcelerate.AssetType.Author_C
    _metadata.ELEMENTS.OpenMarket.Xcelerate.AssetType.Author_CD
    _metadata.ELEMENTS.OpenMarket.Xcelerate.AssetType.Author_F
    _metadata.ELEMENTS.OpenMarket.Xcelerate.AssetType.Author_P
    _metadata.ELEMENTS.OpenMarket.Xcelerate.AssetType.Author_PD
    _metadata.SITE
      Acceptance.main.xml [Metadata]
```

Note: Artie could have looked in the `export/envision/cs_workspace` folder in his WebCenter Sites installation directory to see the same data.

Looks like all the resources are in Artie's workspace now. However, this is all on Artie's laptop and the team has no access to it. Time to check-in.

5. Using Subclipse, Artie connects to the development team's SVN repository and shares his Developer Tools project by committing his main Developer Tools workspace folder (`src` folder) to the SVN repository.



Commit to: svn://

Enter a comment for the commit operation.

```
Initial commit. Definition structure to be refined by rest of the team.
```

<Choose a previously entered comment>
Configure Comment Templates...
Changes
☐ Keep locks

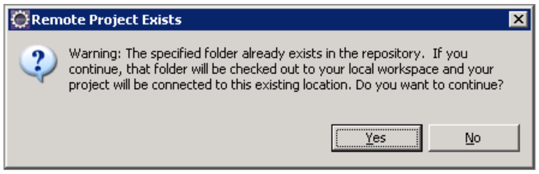
```
⊟ ☐ mysite
  ⊟ ☑ src
    ⊟ ☑ _metadata
      ⊟ ☑ ASSET
        ⊞ ☑ Author_A
        ⊞ ☑ Author_CD
      ⊟ ☑ ASSET_TYPE
          ☑ Author_A(162d0b70-7e69-4266-acca-2f472e3d71bd).main.xml
          ☑ Author_C(d1497b50-665c-4b0c-80a7-d25f61566be4).main.xml
          ☑ Author_CD(33faf87e-9e8f-4f49-97cd-424810408938).main.xml
          ☑ Author_F(5f9b4964-e9be-4f25-a413-877e8a5c7469).main.xml
          ☑ Author_P(1552d907-5f38-400b-9460-36e46d14abc3).main.xml
          ☑ Author_PD(7c748df3-d149-4b71-802a-64b11360e74b).main.xml
      ⊞ ☑ ELEMENTS
      ⊟ ☑ SITE
          ☑ Acceptance.main.xml
```

Note: The main Developer Tools workspace is located under the `src` folder in the Eclipse "Project Explorer" view. Only commit the files that are located inside the `src` folder. All other files are auxiliary local resources and must not be committed.

10:12 am: The Other Team Members Synchronize their Workspaces to the SVN Repository

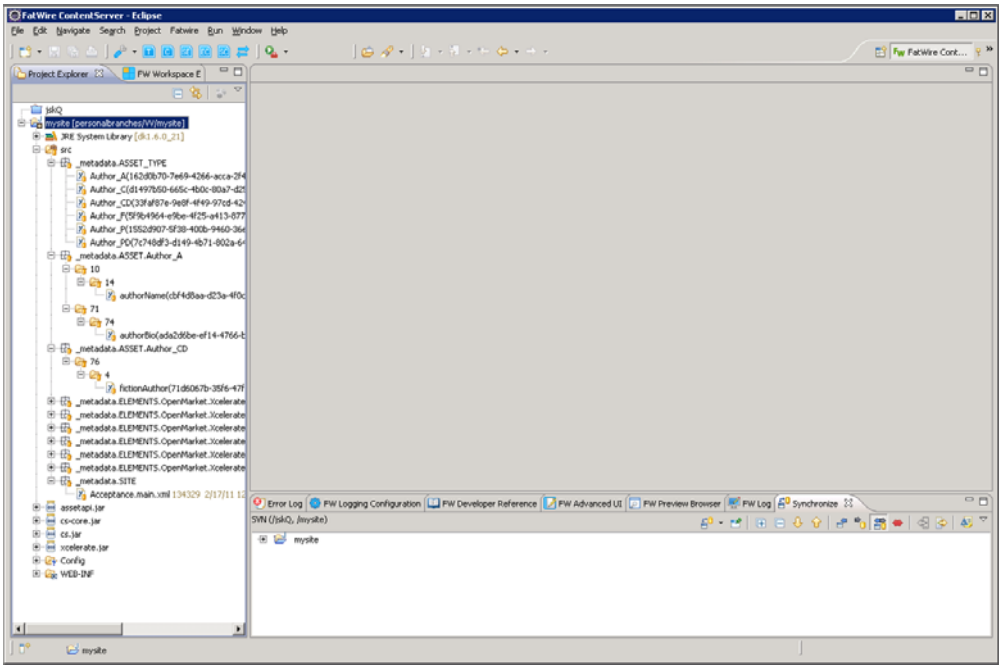
Sonoko and Matthäus just finished setting up their own, individual Eclipse-integrated WebCenter Sites instances. They both connect their Eclipse projects to the SVN repository.

Since Artie checked the site and its resources into the SVN repository earlier, Subclipse detects that the target location already exists:



Warning: The specified folder already exists in the repository. If you continue, that folder will be checked out to your local workspace and your project will be connected to this existing location. Do you want to continue?

Sonoko and Matthäus both synchronize their main Developer Tools workspaces with the resources Artie made available in the SVN repository. Those resources are now accessible on both Sonoko's and Matthäus' main Developer Tools workspaces.



```
jskQ
mysite [personalbranches/VV/mysite]
  JRE System Library [jdk1.6.0_21]
  src
    _metadata.ASSET_TYPE
      Author_A(162d0b70-7e69-4266-acca-2f4
      Author_C(d1497b50-665c-4b0c-80a7-d2
      Author_CD(33faf87e-9e8f-4f49-97cd-42
      Author_F(5f9b4964-e9be-4f25-a413-877
      Author_P(1552d907-5f38-400b-9460-36
      Author_PD(7c748df3-d149-4b71-802a-6
    _metadata.ASSET.Author_A
      10
        14
          authorName(cbf4d8aa-d23a-4f0c
      71
        74
          authorBio(ada2d6be-ef14-4766-b
    _metadata.ASSET.Author_CD
      76
        4
          fictionAuthor(71d6067b-35f6-47f
    _metadata.ELEMENTS.OpenMarket.Xcelerate
    _metadata.ELEMENTS.OpenMarket.Xcelerate
    _metadata.ELEMENTS.OpenMarket.Xcelerate
    _metadata.ELEMENTS.OpenMarket.Xcelerate
    _metadata.ELEMENTS.OpenMarket.Xcelerate
    _metadata.ELEMENTS.OpenMarket.Xcelerate
    _metadata.SITE
      Acceptance.main.xml 134329  2/17/11 12
  assetapi.jar
  cs-core.jar
  cs.jar
  xcelerate.jar
  Config
  WEB-INF
```

SVN (/jskQ, /mysite)

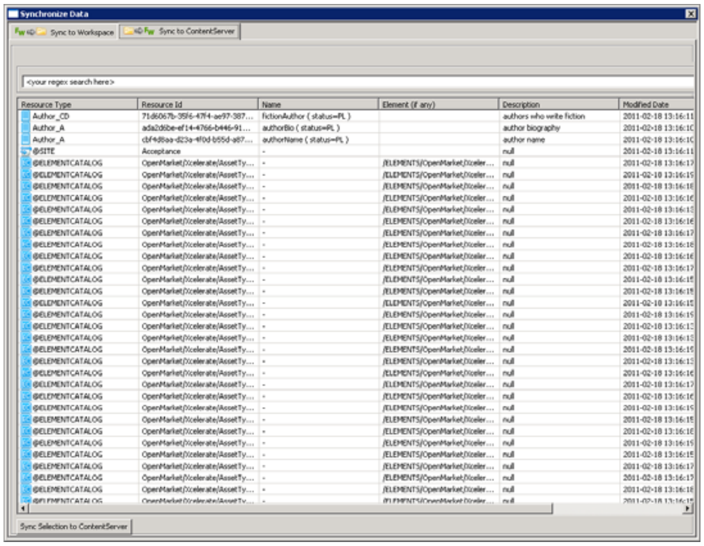
mysite

mysite

However, the resources are not synchronized with Sonoko's or Matthäus' WebCenter Sites instances yet.

10:18 am: Synchronize the Workspace to the WebCenter Sites Instance

Sonoko opens the Developer Tools Synchronization screen and selects the **Sync to WebCenter Sites** tab. All resources contained in Sonoko's main Developer Tools workspace are listed.



<your regex search here>

Resource Type	Resource Id	Name	Element (if any)	Description	Modified Date
Author_CD	71d6067b-35f6-47f4-ae97-387...	fictionAuthor (status=PL)		authors who write fiction	2011-02-18 13:16:11
Author_A	ada2d6be-ef14-4766-b446-91...	authorBio (status=PL)		author biography	2011-02-18 13:16:10
Author_A	cbf4d8aa-d23a-4f0d-b55d-a67...	authorName (status=PL)		author name	2011-02-18 13:16:10
@SITE	Acceptance	-		null	2011-02-18 13:16:11
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:17
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:18
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:16
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:13
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:16
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:17
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:18
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:16
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:17
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:15
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:15
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:15
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:13
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:13
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:13
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:16
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:17
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:16
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:15
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:16
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:15
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:17
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:17
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:18
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:15

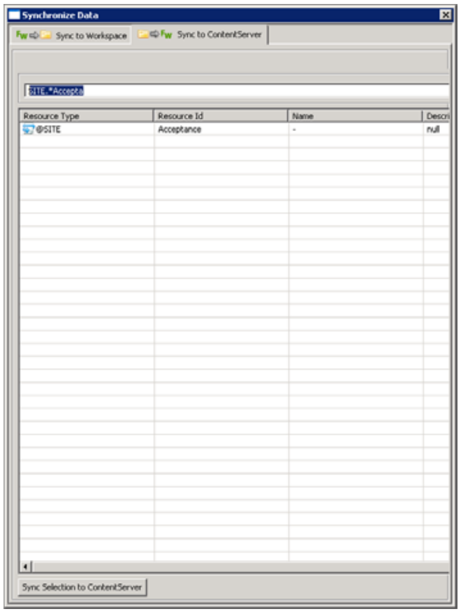
As required, she will first import the site definition, then the flex family, and then the assets, in separate runs as described below:

Note: Matthäus will do the same later, when he finishes his meeting with Marketing.

---

1. Import the site definition ("Acceptance" in this scenario):

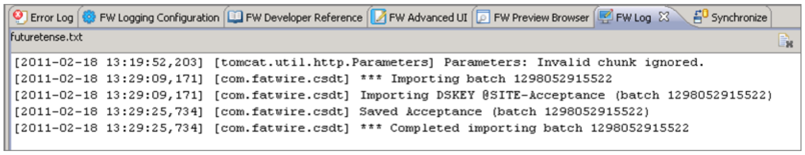
   Sonoko imports the site definition first. She narrows down her search by using the `Site.*Accepta` expression in the search field. She then selects the site ("Acceptance") and synchronizes it to her WebCenter Sites instance by clicking **Sync to WebCenter Sites**.



SITE.*Accepta

Resource Type	Resource Id	Name	Descri
@SITE	Acceptance	-	null

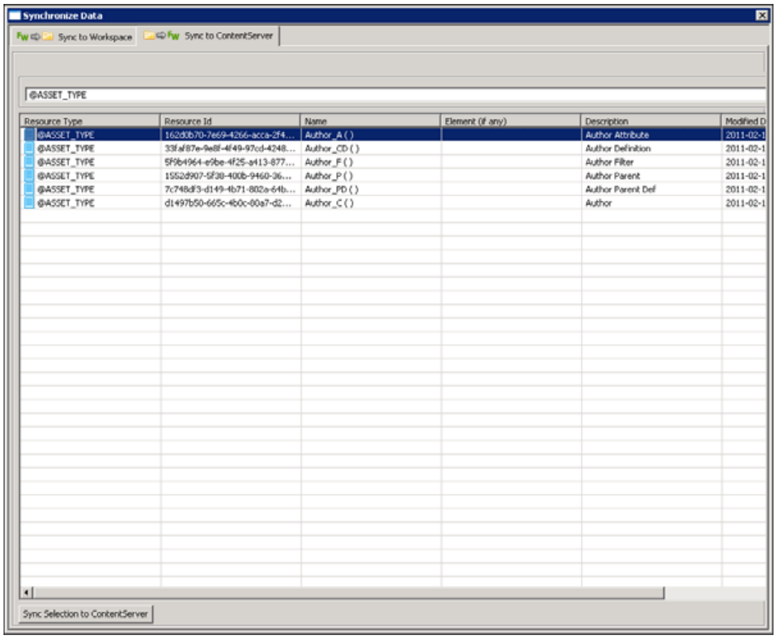
   Using the "FW Log" view, Sonoko verifies that the site is imported successfully:



```
[2011-02-18 13:19:52,203] [tomcat.util.http.Parameters] Parameters: Invalid chunk ignored.
[2011-02-18 13:29:09,171] [com.fatwire.csdt] *** Importing batch 1298052915522
[2011-02-18 13:29:09,171] [com.fatwire.csdt] Importing DSKEY @SITE-Acceptance (batch 1298052915522)
[2011-02-18 13:29:25,734] [com.fatwire.csdt] Saved Acceptance (batch 1298052915522)
[2011-02-18 13:29:25,734] [com.fatwire.csdt] *** Completed importing batch 1298052915522
```

2. Sonoko opens the synchronization screen again, and import the site's flex family, starting with the flex attribute (Author_A in this scenario):

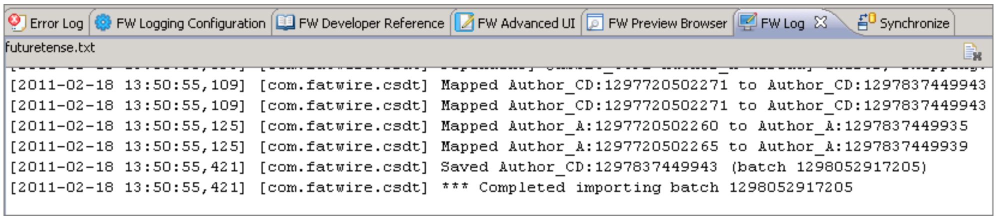
   Since Sonoko did not set up the "Acceptance" site's flex family on her WebCenter Sites instance, she must first import the flex attribute (Author_A) to her WebCenter Sites instance. Once the flex attribute is imported, she can then synchronize the rest of the asset types that comprise the site's flex family to her WebCenter Sites instance.



Resource Type	Resource Id	Name	Element (if any)	Description	Modified D
@ASSET_TYPE	162d0b70-7e69-4266-acca-2f4...	Author_A ( )		Author Attribute	2011-02-1
@ASSET_TYPE	33faf87e-9e8f-4f49-97cd-4248...	Author_CD ( )		Author Definition	2011-02-1
@ASSET_TYPE	5f9b4964-e9be-4f25-a413-877...	Author_F ( )		Author Filter	2011-02-1
@ASSET_TYPE	1552d907-5f38-400b-9460-36...	Author_P ( )		Author Parent	2011-02-1
@ASSET_TYPE	7c748df3-d149-4b71-802a-64b...	Author_PD ( )		Author Parent Def	2011-02-1
@ASSET_TYPE	d1497b50-665c-4b0c-80a7-d2...	Author_C ( )		Author	2011-02-1

3. As a final step, Sonoko synchronizes the flex definition, which automatically imports the required attributes.

   The "FW Log" view shows that the local asset identifiers of all the site's resources are re-mapped when imported into the new WebCenter Sites instance.

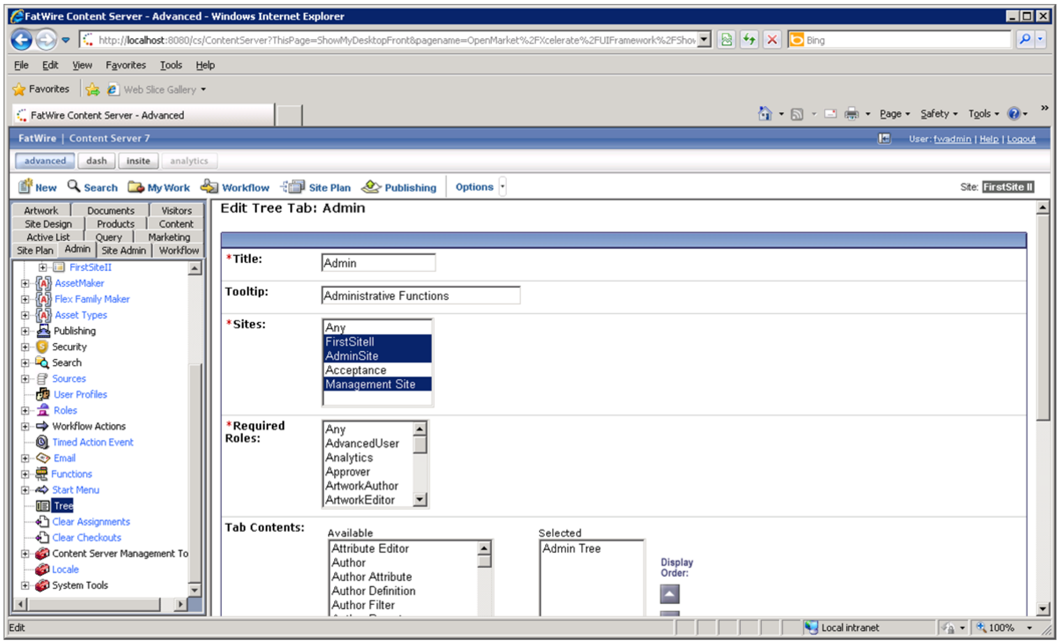


```
[2011-02-18 13:50:55,109] [com.fatwire.csdt] Mapped Author_CD:1297720502271 to Author_CD:1297837449943
[2011-02-18 13:50:55,109] [com.fatwire.csdt] Mapped Author_CD:1297720502271 to Author_CD:1297837449943
[2011-02-18 13:50:55,125] [com.fatwire.csdt] Mapped Author_A:1297720502260 to Author_A:1297837449935
[2011-02-18 13:50:55,125] [com.fatwire.csdt] Mapped Author_A:1297720502265 to Author_A:1297837449939
[2011-02-18 13:50:55,421] [com.fatwire.csdt] Saved Author_CD:1297837449943 (batch 1298052917205)
[2011-02-18 13:50:55,421] [com.fatwire.csdt] *** Completed importing batch 1298052917205
```

10:21 am: Assign Site Permissions

After synchronizing the resources to her WebCenter Sites instance, Sonoko assigns site permissions to herself. These permissions enable her to access the site and its resources from the WebCenter Sites Admin interface.

Note: To access the tree applet in the new site, Sonoko must assign at least one tree tab to the site.


FatWire Content Server - Advanced - Windows Internet Explorer
File Edit View Favorites Tools Help
FatWire | Content Server 7
advanced dash insite analytics
New Search My Work Workflow Site Plan Publishing Options
Site: FirstSite II
Artwork Documents Visitors Site Design Products Content Active List Query Marketing Site Plan Admin Site Admin Workflow
FirstSiteII
AssetMaker
Flex Family Maker
Asset Types
Publishing
Security
Search
Sources
User Profiles
Roles
Workflow Actions
Timed Action Event
Email
Functions
Start Menu
Tree
Clear Assignments
Clear Checkouts
Content Server Management To
Locale
System Tools
Local intranet
100%

Edit Tree Tab: Admin

*Title: Admin

Tooltip: Administrative Functions

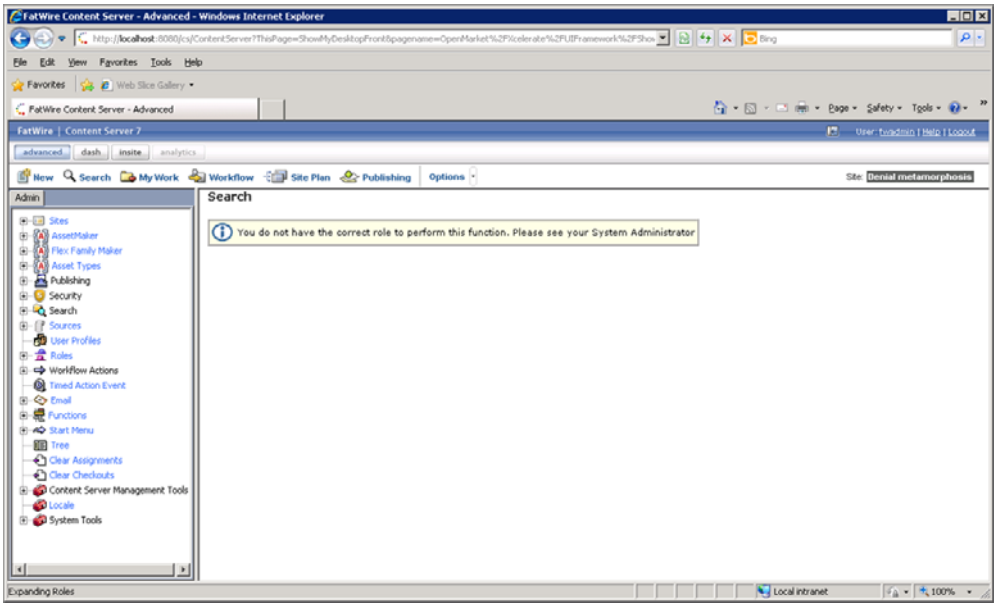
*Sites: Any, FirstSiteII, AdminSite, Acceptance, Management Site

*Required Roles: Any, AdvancedUser, Analytics, Approver, ArtworkAuthor, ArtworkEditor

Tab Contents: Available: Attribute Editor, Author, Author Attribute, Author Definition, Author Filter; Selected: Admin Tree; Display Order:

10:22 am: The Start Menu Issue

Sonoko logs into the site, and clicks the **New** option. However, she finds there are no start menu items available. Of course, Artie did not check the site's start menu items into the SVN repository.


FatWire Content Server - Advanced - Windows Internet Explorer
File Edit View Favorites Tools Help
FatWire | Content Server 7
advanced dash insite analytics
New Search My Work Workflow Site Plan Publishing Options
Site: Denial metamorphosis
Admin
Sites
AssetMaker
Flex Family Maker
Asset Types
Publishing
Security
Search
Sources
User Profiles
Roles
Workflow Actions
Timed Action Event
Email
Functions
Start Menu
Tree
Clear Assignments
Clear Checkouts
Content Server Management Tools
Locale
System Tools
Expanding Roles
Local intranet
100%

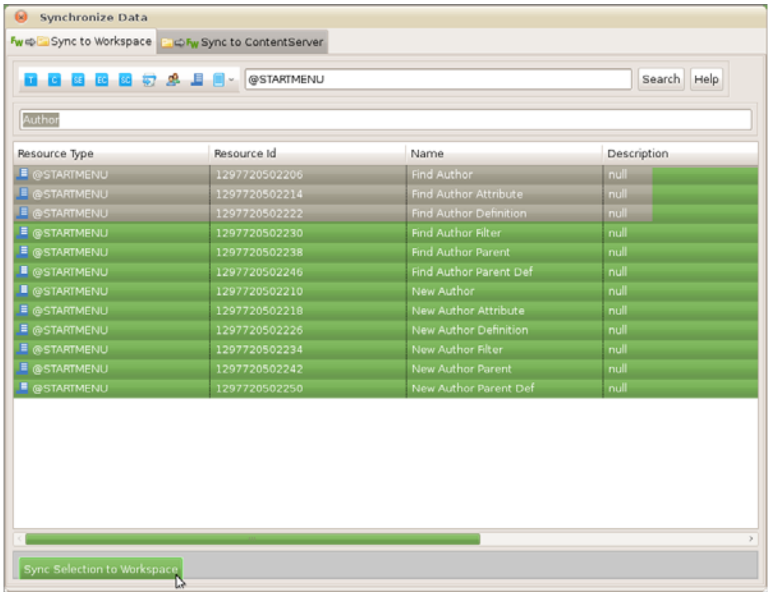
Search

You do not have the correct role to perform this function. Please see your System Administrator

10:24 am: Resolving the Start Menu Issue

Sonoko sends Artie an IM informing him that he forgot to check in the new site's start menu items.

1. Artie synchronizes the site's start menu items to his main Developer Tools workspace.

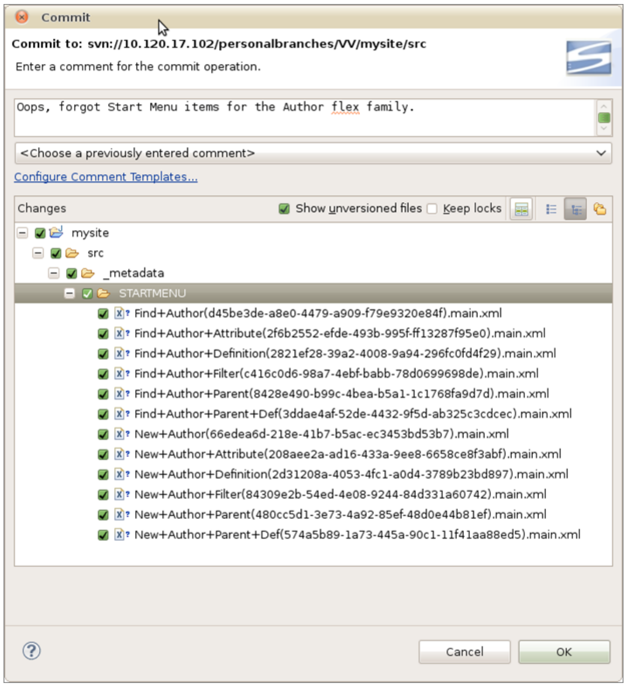


@STARTMENU

Author

Resource Type	Resource Id	Name	Description
@STARTMENU	1297720502206	Find Author	null
@STARTMENU	1297720502214	Find Author Attribute	null
@STARTMENU	1297720502222	Find Author Definition	null
@STARTMENU	1297720502230	Find Author Filter	null
@STARTMENU	1297720502238	Find Author Parent	null
@STARTMENU	1297720502246	Find Author Parent Def	null
@STARTMENU	1297720502210	New Author	null
@STARTMENU	1297720502218	New Author Attribute	null
@STARTMENU	1297720502226	New Author Definition	null
@STARTMENU	1297720502234	New Author Filter	null
@STARTMENU	1297720502242	New Author Parent	null
@STARTMENU	1297720502250	New Author Parent Def	null

2. Artie then checks the site's start menu items into the SVN repository.



Commit to: svn://10.120.17.102/personalbranches/VV/mysite/src

Enter a comment for the commit operation.

Oops, forgot Start Menu items for the Author flex family.

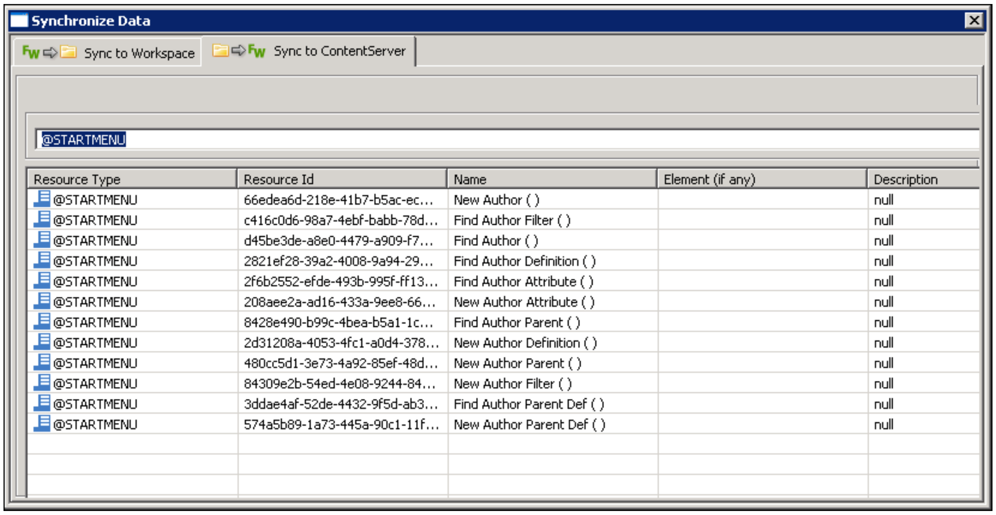
<Choose a previously entered comment>

Configure Comment Templates...

☑ Show unversioned files ☐ Keep locks

- ☑ mysite
  - ☑ src
    - ☑ _metadata
      - ☑ STARTMENU
        - ☑ Find+Author(d45be3de-a8e0-4479-a909-f79e9320e84f).main.xml
        - ☑ Find+Author+Attribute(2f6b2552-efde-493b-995f-ff13287f95e0).main.xml
        - ☑ Find+Author+Definition(2821ef28-39a2-4008-9a94-296fc0fd4f29).main.xml
        - ☑ Find+Author+Filter(c416c0d6-98a7-4ebf-babb-78d0699698de).main.xml
        - ☑ Find+Author+Parent(8428e490-b99c-4bea-b5a1-1c1768fa9d7d).main.xml
        - ☑ Find+Author+Parent+Def(3ddae4af-52de-4432-9f5d-ab325c3cdcec).main.xml
        - ☑ New+Author(66edea6d-218e-41b7-b5ac-ec3453bd53b7).main.xml
        - ☑ New+Author+Attribute(208aee2a-ad16-433a-9ee8-6658ce8f3abf).main.xml
        - ☑ New+Author+Definition(2d31208a-4053-4fc1-a0d4-3789b23bd897).main.xml
        - ☑ New+Author+Filter(84309e2b-54ed-4e08-9244-84d331a60742).main.xml
        - ☑ New+Author+Parent(480cc5d1-3e73-4a92-85ef-48d0e44b81ef).main.xml
        - ☑ New+Author+Parent+Def(574a5b89-1a73-445a-90c1-11f41aa88ed5).main.xml

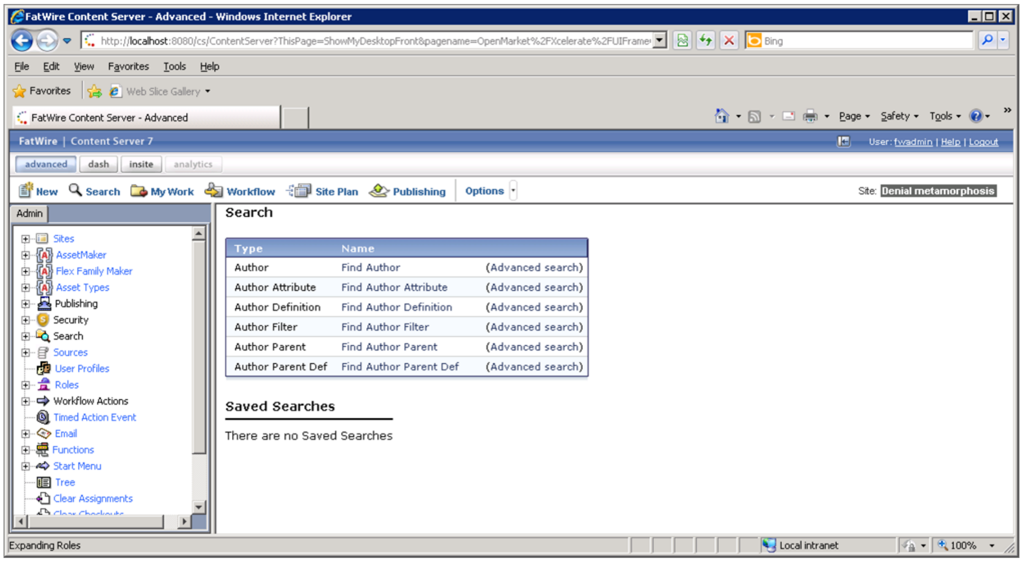
3. Sonoko just got a cup of earl grey tea with two sugars. She comes back to her desk to find that Artie committed the start menu items to the SVN. Sonoko then updates her Eclipse project. She accesses the SVN repository and synchronizes the start menu items to her main Developer Tools workspace. She then imports those start menu items to her WebCenter Sites instance.



Resource Type	Resource Id	Name	Element (if any)	Description
@STARTMENU	66edea6d-218e-41b7-b5ac-ec...	New Author ( )		null
@STARTMENU	c416c0d6-98a7-4ebf-babb-78d...	Find Author Filter ( )		null
@STARTMENU	d45be3de-a8e0-4479-a909-f7...	Find Author ( )		null
@STARTMENU	2821ef28-39a2-4008-9a94-29...	Find Author Definition ( )		null
@STARTMENU	2f6b2552-efde-493b-995f-ff13...	Find Author Attribute ( )		null
@STARTMENU	208aee2a-ad16-433a-9ee8-66...	New Author Attribute ( )		null
@STARTMENU	8428e490-b99c-4bea-b5a1-1c...	Find Author Parent ( )		null
@STARTMENU	2d31208a-4053-4fc1-a0d4-378...	New Author Definition ( )		null
@STARTMENU	480cc5d1-3e73-4a92-85ef-48d...	New Author Parent ( )		null
@STARTMENU	84309e2b-54ed-4e08-9244-84...	New Author Filter ( )		null
@STARTMENU	3ddae4af-52de-4432-9f5d-ab3...	Find Author Parent Def ( )		null
@STARTMENU	574a5b89-1a73-445a-90c1-11f...	New Author Parent Def ( )		null

4. Without restarting her WebCenter Sites instance, Sonoko clicks **Search**.

   The start menu items she imported into her WebCenter Sites instance are listed:



Search

Type	Name	
Author	Find Author	(Advanced search)
Author Attribute	Find Author Attribute	(Advanced search)
Author Definition	Find Author Definition	(Advanced search)
Author Filter	Find Author Filter	(Advanced search)
Author Parent	Find Author Parent	(Advanced search)
Author Parent Def	Find Author Parent Def	(Advanced search)

Saved Searches

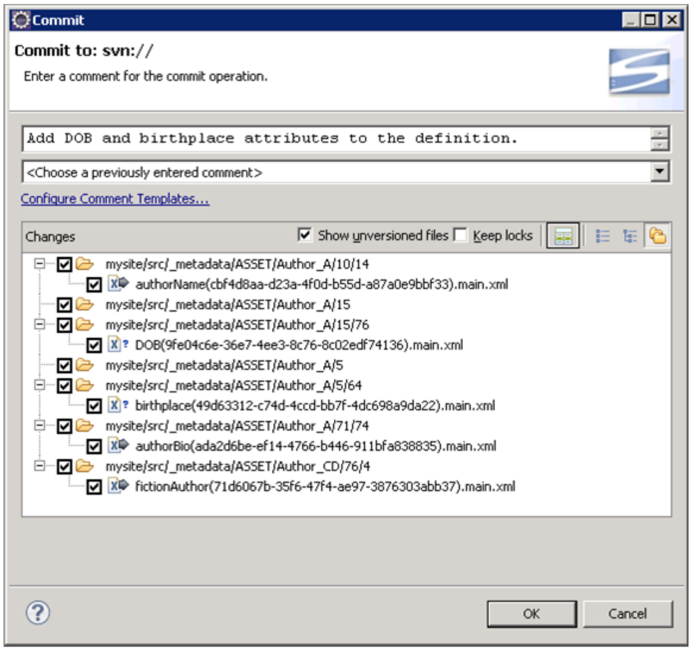
There are no Saved Searches

11:17 am: Marketing Requests Changes

```
Subject: Proposed Author Definition Changes

Date: Wed, 16 Feb 2011 11:17:39

From: matthäus.companynone.com

To: Tech-Development

Team,

I just synchronized your changes into my system. As per my meeting with
Marketing, we must have date of birth and birthplace attributes in the
Author Definition. I noticed these attributes do not exist, so I will add
them. Artie, can you review the changes I make when you have the chance?

Regards,

Matthäus
```

11:22 am: Adding New Attributes to the Author Definition

Matthäus creates the attributes Marketing requested and adds them to the flex definition (Author definition in this scenario). He then exports the new attributes and the flex definition to his main Developer Tools workspace and commits them to the SVN repository.



Commit to: svn://

Enter a comment for the commit operation.

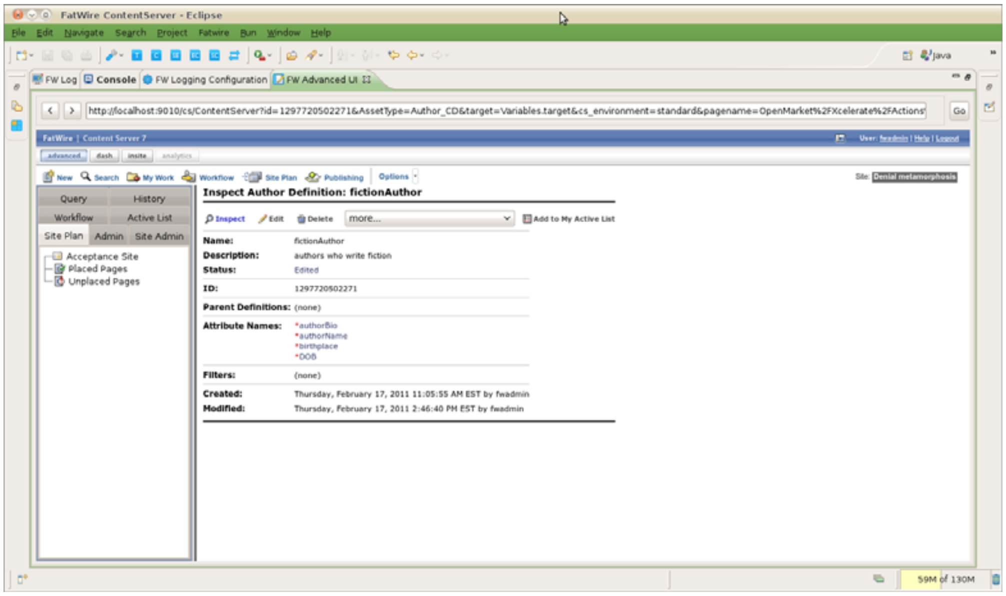
```
Add DOB and birthplace attributes to the definition.
```

<Choose a previously entered comment>
Configure Comment Templates...
Changes
☑ Show unversioned files □ Keep locks

```
☑ mysite/src/_metadata/ASSET/Author_A/10/14
    ☑ authorName(cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33).main.xml
☑ mysite/src/_metadata/ASSET/Author_A/15
☑ mysite/src/_metadata/ASSET/Author_A/15/76
    ☑ DOB(9fe04c6e-36e7-4ee3-8c76-8c02edf74136).main.xml
☑ mysite/src/_metadata/ASSET/Author_A/5
☑ mysite/src/_metadata/ASSET/Author_A/5/64
    ☑ birthplace(49d63312-c74d-4ccd-bb7f-4dc698a9da22).main.xml
☑ mysite/src/_metadata/ASSET/Author_A/71/74
    ☑ authorBio(ada2d6be-ef14-4766-b446-911bfa838835).main.xml
☑ mysite/src/_metadata/ASSET/Author_CD/76/4
    ☑ fictionAuthor(71d6067b-35f6-47f4-ae97-3876303abb37).main.xml
```

11:25 am: Reviewing the Changes to the Site

Artie retrieves the modified Author definition from SVN and imports it into his WebCenter Sites instance.



FatWire ContentServer - Eclipse
File Edit Navigate Search Project Fatwire Run Window Help
Java
FW Log Console FW Logging Configuration FW Advanced UI
Go
FatWire | Content Server 7
advanced dash insite analytics
New Search My Work Workflow Site Plan Publishing Options
Query History Workflow Active List Site Plan Admin Site Admin
Acceptance Site
Placed Pages
Unplaced Pages
Inspect Edit Delete more... Add to My Active List
59M of 130M

http://localhost:9010/cs/ContentServer?id=1297720502271&AssetType=Author_CD&target=Variables.target&cs_environment=standard&pagename=OpenMarket%2FXcelerate%2FActions

Inspect Author Definition: fictionAuthor

Name: fictionAuthor
Description: authors who write fiction
Status: Edited
ID: 1297720502271
Parent Definitions: (none)
Attribute Names: *authorBio
*authorName
*birthplace
*DOB
Filters: (none)
Created: Thursday, February 17, 2011 11:05:55 AM EST by fwadmin
Modified: Thursday, February 17, 2011 2:46:40 PM EST by fwadmin

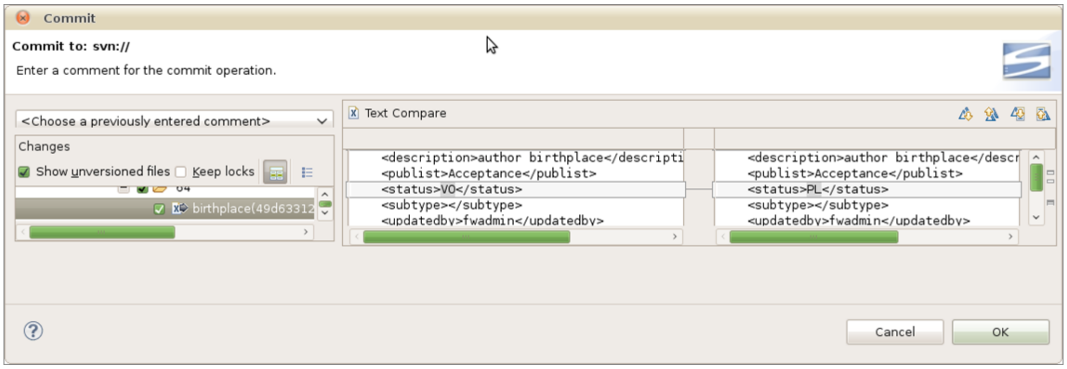
```
Subject: RE: Proposed Author Definition Changes

Date: Wed, 16 Feb 2011 11:37:31

From: artie.companynone.com
```

```
To: matthäus.companynone.com

Matthäus,

Thank you for taking care of this. Corporate standards require us to
capitalize the first letter of each subsequent word. I will delete the
birthplace attribute and add birthPlace instead.

Thank you,

Artie
```

11:44 am: Modifying the Attributes of the Author Definition

1. Artie creates the "birthPlace" attribute and adds it to the flex definition. He then removes the original "birthplace" attribute from the site definition.
2. Artie commits the new attribute and the changes to the Author definition to the SVN repository. He then verifies that the "birthplace" attribute has a status of "VO," indicating the attribute is voided.

   When Sonoko and Matthäus update their WebCenter Sites instances, the "birthplace" attribute will correspondingly be voided on their own workspaces.


Commit

Commit to: svn://

Enter a comment for the commit operation.

<Choose a previously entered comment>
Changes
☑ Show unversioned files □ Keep locks
64
☑ birthplace(49d63312
Text Compare

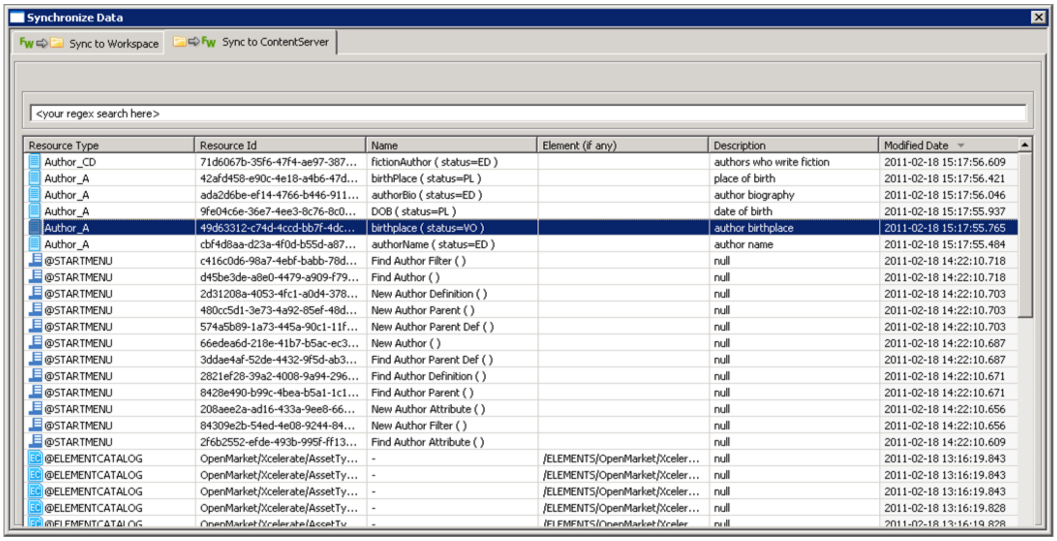
```
<description>author birthplace</descripti
<publist>Acceptance</publist>
<status>VO</status>
<subtype></subtype>
<updatedby>fwadmin</updatedby>
```

```
<description>author birthplace</descr
<publist>Acceptance</publist>
<status>PL</status>
<subtype></subtype>
<updatedby>fwadmin</updatedby>
```

11:53 am: The Team Updates Their Workspaces and WebCenter Sites Instances

1. Sonoko and Matthäus update their main Developer Tools workspaces with the resources Artie checked in to the SVN repository.
2. They then import the resources in their workspaces to their WebCenter Sites instances by opening the "Synchronize to WebCenter Sites" tab. For convenience, they sorted by the "Modified Date" column so the most recent changes are shown on top.

   Any voided attributes (such as the "birthplace" attribute Artie voided) show a status hint (`status=VO`) in the "Name" column.



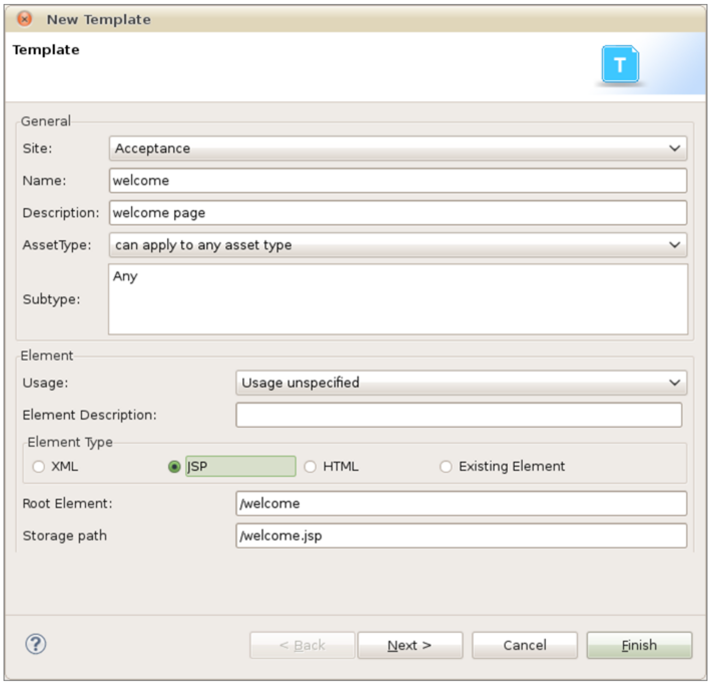
<your regex search here>

Resource Type	Resource Id	Name	Element (if any)	Description	Modified Date
Author_CD	71d6067b-35f6-47f4-ae97-387...	fictionAuthor (status=ED)		authors who write fiction	2011-02-18 15:17:56.609
Author_A	42afd458-e90c-4e18-a4b6-47d...	birthPlace (status=PL)		place of birth	2011-02-18 15:17:56.421
Author_A	ada2d6be-ef14-4766-b446-911...	authorBio (status=ED)		author biography	2011-02-18 15:17:56.046
Author_A	9fe04c6e-36e7-4ee3-8c76-8c0...	DOB (status=PL)		date of birth	2011-02-18 15:17:55.937
Author_A	49d63312-c74d-4ccd-bb7f-4dc...	birthplace (status=VO)		author birthplace	2011-02-18 15:17:55.765
Author_A	cbf4d8aa-d23a-4f0d-b55d-a87...	authorName (status=ED)		author name	2011-02-18 15:17:55.484
@STARTMENU	c416c0d6-98a7-4ebf-babb-78d...	Find Author Filter ( )		null	2011-02-18 14:22:10.718
@STARTMENU	d45be3de-a8e0-4479-a909-f79...	Find Author ( )		null	2011-02-18 14:22:10.718
@STARTMENU	2d31208a-4053-4fc1-a0d4-378...	New Author Definition ( )		null	2011-02-18 14:22:10.703
@STARTMENU	480cc5d1-3e73-4a92-85ef-48d...	New Author Parent ( )		null	2011-02-18 14:22:10.703
@STARTMENU	574a5b89-1a73-445a-90c1-11f...	New Author Parent Def ( )		null	2011-02-18 14:22:10.703
@STARTMENU	66edea6d-218e-41b7-b5ac-ec3...	New Author ( )		null	2011-02-18 14:22:10.687
@STARTMENU	3ddae4af-52de-4432-9f5d-ab3...	Find Author Parent Def ( )		null	2011-02-18 14:22:10.687
@STARTMENU	2821ef28-39a2-4008-9a94-296...	Find Author Definition ( )		null	2011-02-18 14:22:10.671
@STARTMENU	8428e490-b99c-4bea-b5a1-1c1...	Find Author Parent ( )		null	2011-02-18 14:22:10.671
@STARTMENU	208aee2a-ad16-433a-9ee8-66...	New Author Attribute ( )		null	2011-02-18 14:22:10.656
@STARTMENU	84309e2b-54ed-4e08-9244-84...	New Author Filter ( )		null	2011-02-18 14:22:10.656
@STARTMENU	2f6b2552-efde-493b-995f-ff13...	Find Author Attribute ( )		null	2011-02-18 14:22:10.609
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19.843
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19.843
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19.843
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy...	-	/ELEMENTS/OpenMarket/Xceler...	null	2011-02-18 13:16:19.828
@ELEMENTCATALOG	OpenMarket/Xcelerate/AssetTy	-	/ELEMENTS/OpenMarket/Xceler	null	2011-02-18 13:16:19.828

3. Sonoko and Matthäus import these changes from their workspaces to their WebCenter Sites instances. Their workspaces and WebCenter Sites instances are now up to date.

12:27 pm: The Team Creates a Template Asset for the Site

1. (12:27 pm) Matthäus creates a Template asset for the site's "Welcome" page.



General

Site: Acceptance

Name: welcome

Description: welcome page

AssetType: can apply to any asset type

Subtype: Any

Element

Usage: Usage unspecified

Element Description:

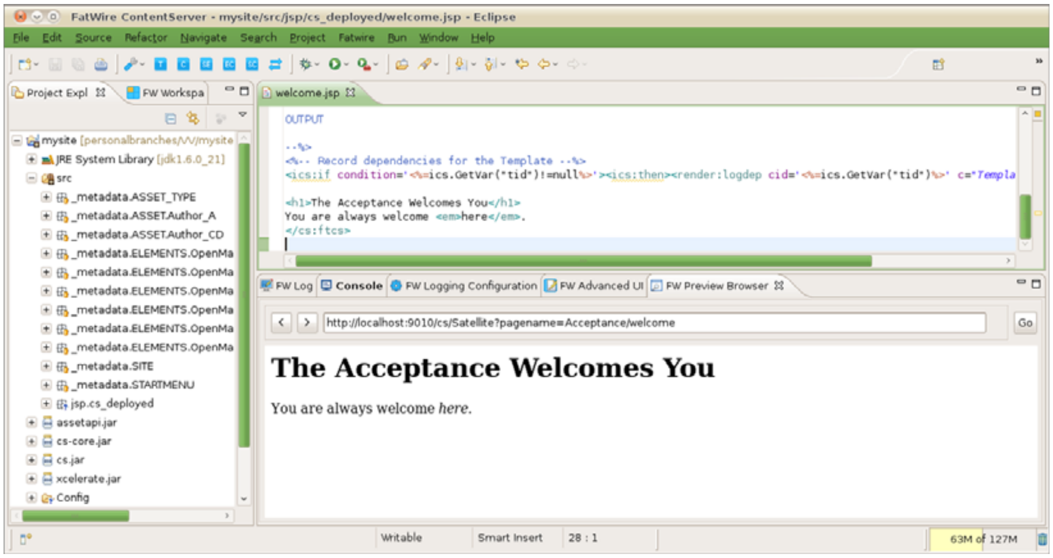
Element Type

○ XML ● JSP ○ HTML ○ Existing Element

Root Element: /welcome

Storage path /welcome.jsp

2. (12:34 pm) Matthäus edits the Template asset and previews the changes in the "FW Preview Browser" view. As soon as he saves the changes made to the Template asset's JSP, he uses the **Ctrl-r** keyboard command to refresh the preview browser.



```
OUTPUT

--%>
<%-- Record dependencies for the Template --%>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep cid='<%=ics.GetVar("tid")%>' c="Templa

<h1>The Acceptance Welcomes You</h1>
You are always welcome <em>here</em>.
</cs:ftcs>
```

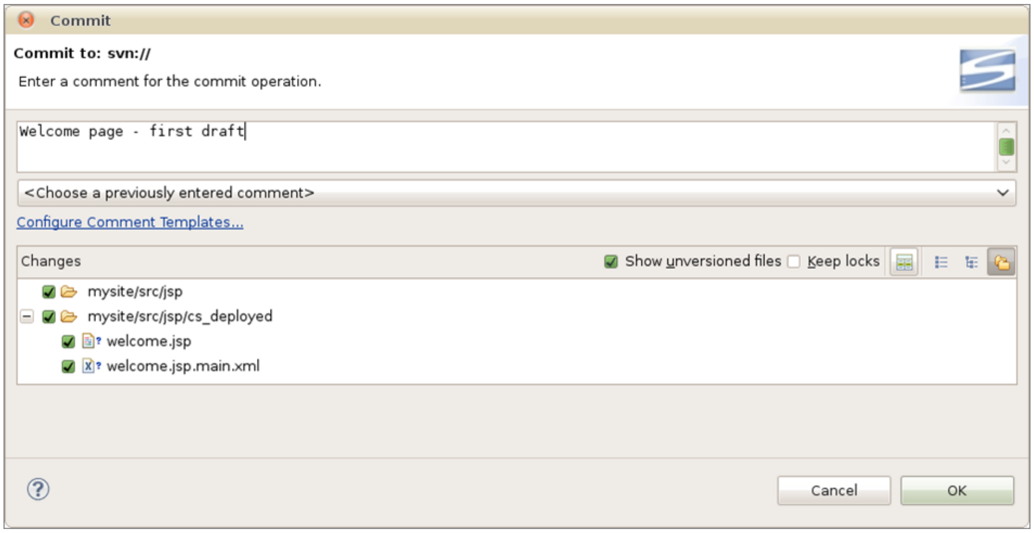
http://localhost:9010/cs/Satellite?pagename=Acceptance/welcome

The Acceptance Welcomes You

You are always welcome *here*.

3. (12:39 pm) Matthäus commits the Template's `.jsp` and `.main.xml` files to the SVN repository.

   Subclipse finds all changes to the project and brings those changes to the attention of the developer. Since the only new asset was the Template asset, Matthäus is able to deduce that the `.main.xml` file is the Template's metadata and the JSP file is the Template's code.



Commit to: svn://

Enter a comment for the commit operation.

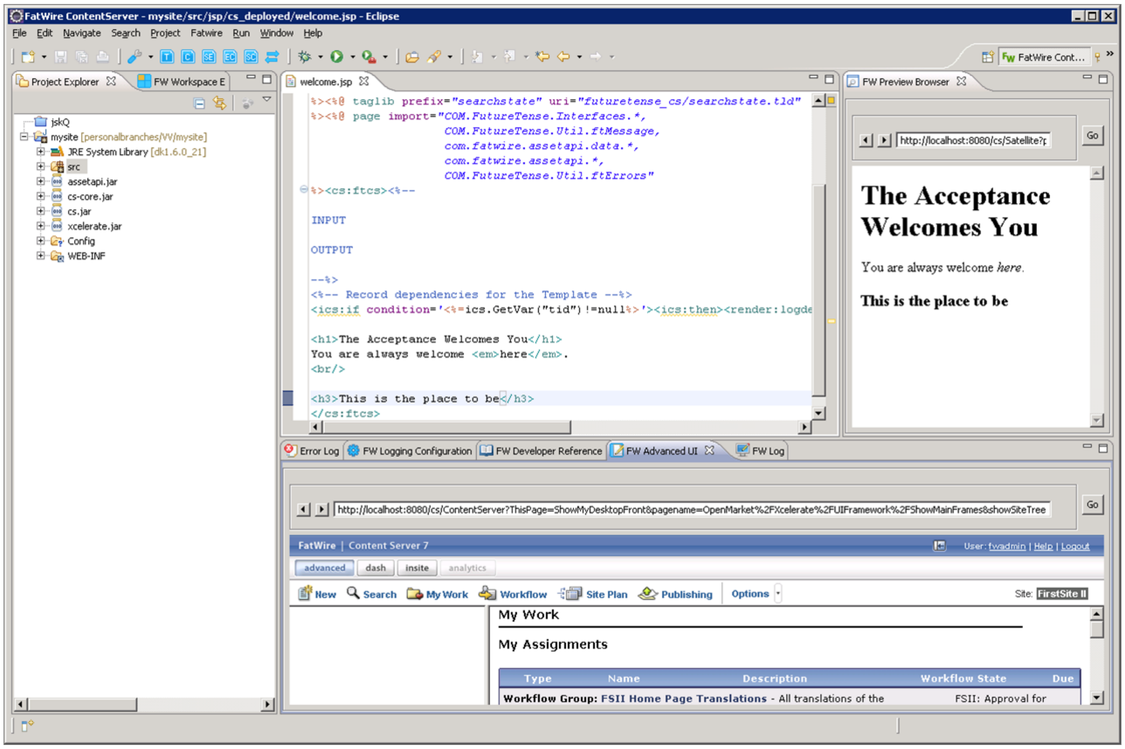
Welcome page - first draft

<Choose a previously entered comment>

☑ Show unversioned files ☐ Keep locks

☑ mysite/src/jsp
☑ mysite/src/jsp/cs_deployed
☑ welcome.jsp
☑ welcome.jsp.main.xml

4. (12:44 pm) Sonoko makes some touch ups to the Template's JSP file in her own workspace.



FatWire ContentServer - mysite/src/jsp/cs_deployed/welcome.jsp - Eclipse

welcome.jsp

```
%><%@ taglib prefix="searchstate" uri="futuretense_cs/searchstate.tld"
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   com.fatwire.assetapi.data.*,
                   com.fatwire.assetapi.*,
                   COM.FutureTense.Util.ftErrors"
%><cs:ftcs><%--

INPUT

OUTPUT

--%>
<%-- Record dependencies for the Template --%>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde

<h1>The Acceptance Welcomes You</h1>
You are always welcome <em>here</em>.
<br/>

<h3>This is the place to be</h3>
</cs:ftcs>
```

FW Preview Browser: http://localhost:8080/cs/Satellite?p

The Acceptance Welcomes You

You are always welcome *here*.

This is the place to be

FW Advanced UI: http://localhost:8080/cs/ContentServer?ThisPage=ShowMyDesktopFront&pagename=OpenMarket%2FXcelerate%2FUIFramework%2FShowMainFrames&showSiteTree

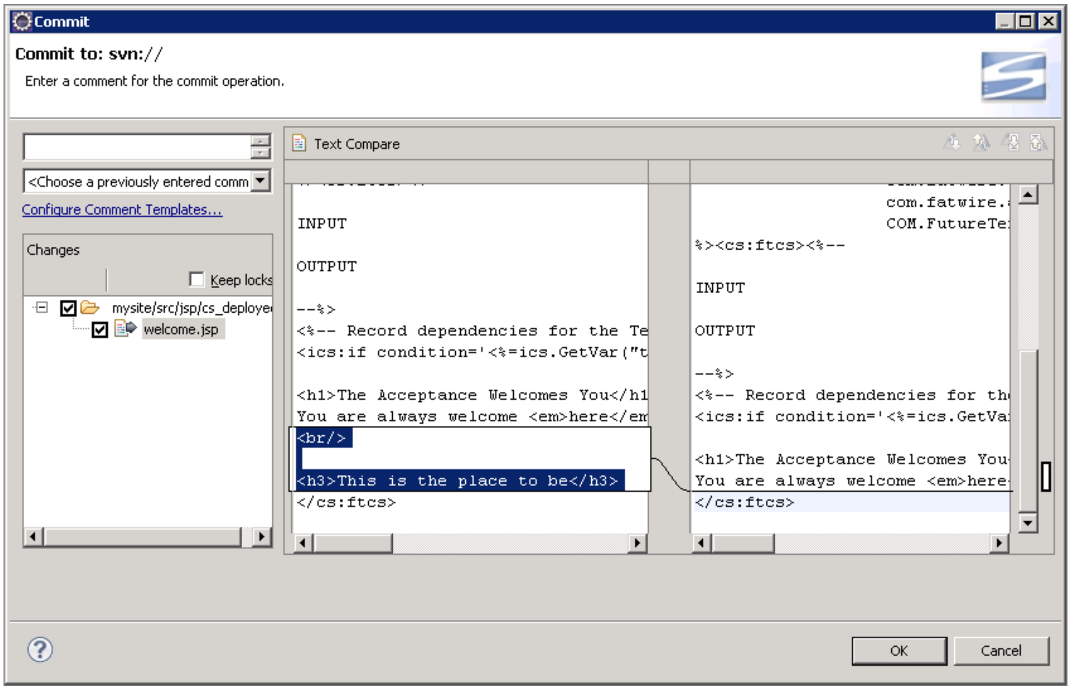
FatWire | Content Server 7 — User: fwadmin | Help | Logout — Site: FirstSite II

My Work

My Assignments

Type	Name	Description	Workflow State	Due
Workflow Group: FSII Home Page Translations - All translations of the			FSII: Approval for	

5. Sonoko reviews the changes to the JSP file and then commits those changes to the SVN.



Commit

Commit to: svn://

Enter a comment for the commit operation.

Text Compare

```
INPUT

OUTPUT

--%>
<%-- Record dependencies for the Te
<ics:if condition='<%=ics.GetVar("t

<h1>The Acceptance Welcomes You</h1
You are always welcome <em>here</em
<br/>

<h3>This is the place to be</h3>
</cs:ftcs>
```

```
                      com.fatwire.
                      COM.FutureTe
%><cs:ftcs><%--

INPUT

OUTPUT

--%>
<%-- Record dependencies for th
<ics:if condition='<%=ics.GetVa

<h1>The Acceptance Welcomes You
You are always welcome <em>here
</cs:ftcs>
```

<Choose a previously entered comm
Configure Comment Templates...
Changes
□ Keep locks
☑ mysite/src/jsp/cs_deploye
☑ welcome.jsp
OK
Cancel

Note: If another team member were to modify and check in this file at the same time as Sonoko, SVN would indicate to Sonoko that another version of the file is already checked in. She would then be able to integrate those changes with her own to avoid inadvertent overwrites.

60.2 Three Days Later... Deployment

Yogesh uses the command-line tool to deploy the site. For information about the commands used in this section, see Chapter 58, "Developer Tools: Command-Line Tool."

9:32 am: Preparing for Deployment

Yogesh finally got around to setting up the test environment and is preparing to deploy the current build using the command-line tool. He installed a WebCenter Sites system on hardware that matches the environment used in production.

To test the Developer Tools import before adding it to a fully-automated nightly script

1. Using the command-line tool, Yogesh checks the "Acceptance" site and its resources out of SVN and into the workspace of the target WebCenter Sites instance.

   Command:

   ```
   ## go to the workspace location under export/envision/
      cs_workspace in the CS install directory
   ## create if not there
   /home$ mkdir /opt/cs/export/envision/cs_workspace
   /home$ cd  /opt/cs/export/envision/cs_workspace

   ## checkout site from svn
   /opt/cs/export/envision/cs_workspace$ svn checkout   svn://
      yoursvnhost/projects/mysite/src
   ```

   Output:

   ```
   A    mysite/src
   A    mysite/src/_metadata
   A    mysite/src/_metadata/ASSET
   A    mysite/src/_metadata/ASSET/Author_A
   A    mysite/src/_metadata/ASSET/Author_A/10
   A    mysite/src/_metadata/ASSET/Author_A/10/14
   A    mysite/src/_metadata/ASSET/Author_
   A/10/14/authorName(cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33).main.xml
   A    mysite/src/_metadata/ASSET/Author_A/11
   A    mysite/src/_metadata/ASSET/Author_A/11/79
   A    mysite/src/_metadata/ASSET/Author_
   A/11/79/birthPlace(42afd458-e90c-4e18-a4b6-47d322b46414).main.xml
   A    mysite/src/_metadata/ASSET/Author_A/5
   A    mysite/src/_metadata/ASSET/Author_A/5/64
   A    mysite/src/_metadata/ASSET/Author_
   A/5/64/birthplace(49d63312-c74d-4ccd-bb7f-4dc698a9da22).main.xml
   A    mysite/src/_metadata/ASSET/Author_A/15
   A    mysite/src/_metadata/ASSET/Author_A/15/76
   A    mysite/src/_metadata/ASSET/Author_
   A/15/76/DOB(9fe04c6e-36e7-4ee3-8c76-8c02edf74136).main.xml
   A    mysite/src/_metadata/ASSET/Author_A/71
   A    mysite/src/_metadata/ASSET/Author_A/71/74
   ```

```
A    mysite/src/_metadata/ASSET/Author_
A/71/74/authorBio(ada2d6be-ef14-4766-b446-911bfa838835).main.xml
A    mysite/src/_metadata/ASSET/Author_CD
A    mysite/src/_metadata/ASSET/Author_CD/76
A    mysite/src/_metadata/ASSET/Author_CD/76/4
A    mysite/src/_metadata/ASSET/Author_
CD/76/4/fictionAuthor(71d6067b-35f6-47f4-ae97-3876303abb37).main.xml
A    mysite/src/_metadata/ASSET_TYPE
A    mysite/src/_metadata/ASSET_TYPE/Author_
F(5f9b4964-e9be-4f25-a413-877e8a5c7469).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_
P(1552d907-5f38-400b-9460-36e46d14abc3).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_
A(162d0b70-7e69-4266-acca-2f472e3d71bd).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_
CD(33faf87e-9e8f-4f49-97cd-424810408938).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_
PD(7c748df3-d149-4b71-802a-64b11360e74b).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_
C(d1497b50-665c-4b0c-80a7-d25f61566be4).main.xml
A    mysite/src/_metadata/STARTMENU
A    mysite/src/_
metadata/STARTMENU/Find+Author+Attribute(2f6b2552-efde-493b-995f-ff13287f95e0).
main.xml
...
```

2. Yogesh runs a workspace listing (`cmd=listds`) to verify that the site and all of its resources will be imported into the WebCenter Sites instance. He uses the `@ALL_ASSETS` and `@ALL_NONASSETS` selectors to generate listings of all asset and non-asset resources in the workspace:

   – **Command** to use the `@ALL_ASSETS` selector:

   ```
       /opt/cs/export/envision/cs_workspace$ export
         CLASSPATH=csdt-client-1.0.2.jar
       /opt/cs/export/envision/cs_workspace$ java
         com.fatwire.csdt.client.main.CSDT
       http://localhost:9010/cs/ContentServer username=fwadmin
     password=xceladmin resources=@ALL_ASSETS cmd=listds
   ```

   Output:

   ```
   Resource Type |||    Resource Id   |||    Name     |||
   Description   ||| Modified On
   -----------------------------------------------------------------
   Author_A ||| cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 ||| authorName
   ( status=ED ) ||| author name ||| 2011-02-17 15:26:34.000
   Author_A ||| 42afd458-e90c-4e18-a4b6-47d322b46414 ||| birthPlace
   ( status=PL ) ||| place of birth ||| 2011-02-17 15:26:34.000
   Author_A ||| 9fe04c6e-36e7-4ee3-8c76-8c02edf74136 ||| DOB
   ( status=PL ) ||| date of birth ||| 2011-02-17 15:26:34.000
   Author_CD ||| 71d6067b-35f6-47f4-ae97-3876303abb37 |||
   fictionAuthor ( status=ED ) ||| authors who write fiction |||
   2011-02-17 15:26:34.000
   Author_A ||| ada2d6be-ef14-4766-b446-911bfa838835 ||| authorBio
   ( status=ED ) ||| author biography ||| 2011-02-17 15:26:34.000
   Author_A ||| 49d63312-c74d-4ccd-bb7f-4dc698a9da22 ||| birthplace
   ( status=VO ) ||| author birthplace ||| 2011-02-17 15:12:43.000
   Template ||| 89b05c0f-227b-4dcb-961e-2ab6e6af2dae ||| welcome
   (Typeless status=PL) ||| welcome page ||| 2011-02-17 23:18:18.000
   ```

– **Command** to use the `@ALL_NONASSETS` selector:

```
/opt/cs/export/envision/cs_workspace$ export
          CLASSPATH=csdt-client-1.0.2.jar
        /opt/cs/export/envision/cs_workspace$ java
          com.fatwire.csdt.client.main.CSDT
        http://localhost:9010/cs/ContentServer username=fwadmin
          password=xceladmin resources=@ALL_NONASSETS cmd=listds
```

Output:

```
Resource Type |||    Resource Id   |||    Name     |||
Description  ||| Modified On
-----------------------------------------------------------------
@STARTMENU ||| 66edea6d-218e-41b7-b5ac-ec3453bd53b7 ||| New
Author ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| c416c0d6-98a7-4ebf-babb-78d0699698de ||| Find
Author Filter ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| 162d0b70-7e69-4266-acca-2f472e3d71bd ||| Author_A
( ) ||| Author Attribute ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 2821ef28-39a2-4008-9a94-296fc0fd4f29 ||| Find
Author Definition ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| d45be3de-a8e0-4479-a909-f79e9320e84f ||| Find
Author ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 2f6b2552-efde-493b-995f-ff13287f95e0 ||| Find
Author Attribute ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| 7c748df3-d149-4b71-802a-64b11360e74b ||| Author_
PD ( ) ||| Author Parent Def ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 208aee2a-ad16-433a-9ee8-6658ce8f3abf ||| New
Author Attribute ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 8428e490-b99c-4bea-b5a1-1c1768fa9d7d ||| Find
Author Parent ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| d1497b50-665c-4b0c-80a7-d25f61566be4 ||| Author_C
( ) ||| Author ||| 2011-02-18 11:02:23.000
…
```

3. Yogesh then makes sure all necessary asset types will be imported by using the `@ASSET_TYPE:*` selector:

   Command:

```
/opt/cs/export/envision/cs_workspace$
   java com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin
   password=xceladmin 'resources=@ASSET_TYPE:*' cmd=listds
```

   Output

```
Resource Type |||    Resource Id   |||    Name     |||   Description   |||
Modified On
-------------------------------------------------------------------------------
---------
Author_A ||| cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 ||| authorName ( status=ED )
||| author name ||| 2011-02-17 15:26:34.000
Author_A ||| 42afd458-e90c-4e18-a4b6-47d322b46414 ||| birthPlace ( status=PL )
||| place of birth ||| 2011-02-17 15:26:34.000
Author_A ||| 9fe04c6e-36e7-4ee3-8c76-8c02edf74136 ||| DOB ( status=PL ) |||
date of birth ||| 2011-02-17 15:26:34.000
Author_CD ||| 71d6067b-35f6-47f4-ae97-3876303abb37 ||| fictionAuthor (
status=ED ) ||| authors who write fiction ||| 2011-02-17 15:26:34.000
Author_A ||| ada2d6be-ef14-4766-b446-911bfa838835 ||| authorBio ( status=ED )
||| author biography ||| 2011-02-17 15:26:34.000
```

```
Author_A ||| 49d63312-c74d-4ccd-bb7f-4dc698a9da22 ||| birthplace ( status=VO )
||| author birthplace ||| 2011-02-17 15:12:43.000
Template ||| 89b05c0f-227b-4dcb-961e-2ab6e6af2dae ||| welcome (Typeless
status=PL) ||| welcome page ||| 2011-02-17 23:18:18.000
```

4. Yogesh notes that all necessary resources for the site will be imported into the build.

10:04 am: Deploying the Site and its Resources

Using the command-line tool, Yogesh runs the import sequence.

1. First, he imports the site:

   Command:

   ```
   /opt/cs/export/envision/cs_workspace$ java
      com.fatwire.csdt.client.main.CSDT
   http://localhost:9999/cs/ContentServer username=fwadmin
      password=xceladmin 'resources=@SITE:Acceptance' cmd=import
   ```

   Output:

   ```
    *** Importing batch 1297868431526
   Importing DSKEY @SITE-Acceptance (batch 1297868431526)
   Saved Acceptance (batch 1297868431526)
   *** Completed importing batch 1297868431526
   ```

2. Then, the flex family:

   Command:

   ```
   /opt/cs/export/envision/cs_workspace$ java
      com.fatwire.csdt.client.main.CSDT
   http://localhost:9999/cs/ContentServer username=fwadmin
      password=xceladmin 'resources=@ASSET_TYPE:*' cmd=import
   ```

   Output:

   ```
   *** Importing batch 1298064678765
   Importing DSKEY @ASSET_TYPE-162d0b70-7e69-4266-acca-2f472e3d71bd (batch
   1298064678765)
   Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_
   A/LoadTree (batch 1298064678765)
   Saved OpenMarket/Xcelerate/AssetType/Author_A/LoadTree (batch 1298064678765)
   …
   ```

3. Next, the assets:

   Command:

   ```
   /opt/cs/export/envision/cs_workspace java
      com.fatwire.csdt.client.main.CSDT
   http://localhost:9999/cs/ContentServer username=fwadmin
      password=xceladmin 'resources=@ALL_ASSETS' cmd=import
   ```

   Output:

   ```
   *** Importing batch 1298064679760
   Importing DSKEY Author_A-cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 (batch
   1298064679760)
   Dependency @ASSET_TYPE-Author_A already exists, skipping.
   Saved Author_A:1295889071437 (batch 1298064679760)
   Importing DSKEY Author_A-42afd458-e90c-4e18-a4b6-47d322b46414 (batch
   ```

```
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071441 (batch 1298064679760)
Importing DSKEY Author_A-9fe04c6e-36e7-4ee3-8c76-8c02edf74136 (batch
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071445 (batch 1298064679760)
Importing DSKEY Author_CD-71d6067b-35f6-47f4-ae97-3876303abb37 (batch
1298064679760)
Importing DSKEY Author_A-ada2d6be-ef14-4766-b446-911bfa838835 (batch
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071449 (batch 1298064679760)
Dependency @ASSET_TYPE-Author_C already exists, skipping.
Dependency @ASSET_TYPE-Author_P already exists, skipping.
Dependency @ASSET_TYPE-Author_CD already exists, skipping.
Dependency @ASSET_TYPE-Author_PD already exists, skipping.
Dependency @ASSET_TYPE-Author_F already exists, skipping.
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_CD:1295889071453 (batch 1298064679760)
Importing DSKEY Author_A-49d63312-c74d-4ccd-bb7f-4dc698a9da22 (batch
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071460 (batch 1298064679760)
Importing DSKEY Template-89b05c0f-227b-4dcb-961e-2ab6e6af2dae (batch
1298064679760)
Saved Template:1295889071461 (batch 1298064679760)
*** Completed importing batch 1298064679760
```

4. Since this is a delivery install, start menu items are optional. However, Yogesh imports the start menu items because he wants to use the WebCenter Sites Admin interface to verify that all of the resources are successfully imported.

   Command:

```
/opt/cs/export/envision/cs_workspace$ java
   com.fatwire.csdt.client.main.CSDT
http://localhost:9999/cs/ContentServer username=fwadmin
password=xceladmin 'resources=@STARTMENU:*' cmd=import
```

   Output:

```
*** Importing batch 1298064681075
Importing DSKEY @STARTMENU-66edea6d-218e-41b7-b5ac-ec3453bd53b7 (batch
1298064681075)
Saved 1297720502210 (batch 1298064681075)
Importing DSKEY @STARTMENU-c416c0d6-98a7-4ebf-babb-78d0699698de (batch
1298064681075)
Saved 1297720502230 (batch 1298064681075)
Importing DSKEY @STARTMENU-2821ef28-39a2-4008-9a94-296fc0fd4f29 (batch
1298064681075)
Saved 1297720502222 (batch 1298064681075)
Importing DSKEY @STARTMENU-d45be3de-a8e0-4479-a909-f79e9320e84f (batch
1298064681075)
Saved 1297720502206 (batch 1298064681075)
Importing DSKEY @STARTMENU-2f6b2552-efde-493b-995f-ff13287f95e0 (batch
1298064681075)
Saved 1297720502214 (batch 1298064681075)
Importing DSKEY @STARTMENU-208aee2a-ad16-433a-9ee8-6658ce8f3abf (batch
1298064681075)
Saved 1297720502218 (batch 1298064681075)
```

```
Importing DSKEY @STARTMENU-8428e490-b99c-4bea-b5a1-1c1768fa9d7d (batch
1298064681075)
Saved 1297720502238 (batch 1298064681075)
Importing DSKEY @STARTMENU-2d31208a-4053-4fc1-a0d4-3789b23bd897 (batch
1298064681075)
Saved 1297720502226 (batch 1298064681075)
Importing DSKEY @STARTMENU-480cc5d1-3e73-4a92-85ef-48d0e44b81ef (batch
1298064681075)
Saved 1297720502242 (batch 1298064681075)
Importing DSKEY @STARTMENU-84309e2b-54ed-4e08-9244-84d331a60742 (batch
1298064681075)
*** Completed importing batch 1298064681075
```

10:55 am: The Deployment is Successful

Yogesh concludes that the import sequence was successful. He plans to automated daily installs on this system by writing the following script:

```
## Reinstall ContentServer to start with a clean slate.
## Optionally skip this and just do an update
Reinstall_CS()

## Bring in the latest source from SVN
SVN_Update()

## Prepare for import: compile any Java code such as url assemblers and flex
filters, etc
## Prepare the database with any custom settings, etc.
preImport()

## Run the CSDT import sequence
CSDT_Import()

## Run the test suite - sanity, performance, acceptance tests
runTestSuite()

## Report results to the team by email so they know about any failures first thing
in the morning
runReports()
```

The script will run as a cron job at five past midnight every night.

61

Developer Tools: Using the Command-line Tool to Create Reusable Modules

The Developer Tools kit provides the ability to reuse and share resources in the form of modules. Modules are workspaces that are not site-specific and contain resources such as Templates, flex families, and ElementCatalog entries. Unlike the standard export/import functionality where assets are added to sites using natural mappings, modules typically utilize site overriding so they can be imported into any site you designate.

This chapter contains the following sections:

- Section 61.1, "Creating a Reusable Model"
- Section 61.2, "List the Resources in the WebCenter Sites Instance"
- Section 61.3, "List Start Menu Items"
- Section 61.4, "Export All Resources to the Desired Workspace"
- Section 61.5, "Inspect the Module's Content"
- Section 61.6, "Archive the Module"
- Section 61.7, "Import the Module to a WebCenter Sites Instance"

61.1 Creating a Reusable Model

Artie has a flex family with a flex definition that he wants to reuse in other sites. He also has a Template asset associated with the flex definition. In the following scenario, Artie will create a module containing these resources. This scenario uses the command-line tool to create a module containing the resources Artie and his team developed in Chapter 60, "Developer Tools: Development Team Integration Use Case."

> **Note:** To use the command-line tool, Artie must specify the user name and password of a general administrator in each command he executes. This user must be a member of the `RestAdmin` group. In this scenario, Artie uses `fwadmin/xceladmin`.

See the following sections:

- Section 61.2, "List the Resources in the WebCenter Sites Instance"
- Section 61.3, "List Start Menu Items"
- Section 61.4, "Export All Resources to the Desired Workspace"

- Section 61.5, "Inspect the Module's Content"
- Section 61.6, "Archive the Module"
- Section 61.7, "Import the Module to a WebCenter Sites Instance"

61.2 List the Resources in the WebCenter Sites Instance

Artie uses the command-line tool to browse his WebCenter Sites instance. He uses the `resources=@ALL_ASSETS` and the `fromSites=Acceptance` selectors to list all the assets of the "Acceptance" site. The command Artie uses is `listcs`, which lists all the resources on his WebCenter Sites instance.

Command:

```
/opt/cs/export/envision/cs_workspace$ export CLASSPATH=csdt-client-1.0.2.jar
/opt/cs/export/envision/cs_workspace$ java com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin password=xceladmin
   resources=@ALL_ASSETS fromSites=Acceptance cmd=listcs
```

Output:

```
Resource Type ||| Resource Id ||| Name ||| Description ||| Modified On
--------------------------------------------------------------------------------
Author_CD ||| 1297720502271 ||| fictionAuthor (status=ED) ||| authors who write
fiction ||| 2011-02-17 15:10:41
Author_A ||| 1297720502260 ||| authorName (status=ED) ||| author name |||
2011-02-17 14:46:40
Author_A ||| 1297720502265 ||| authorBio (status=ED) ||| author biography |||
2011-02-17 14:46:40
Author_A ||| 1297720502289 ||| 1297720502289 (status=VO) ||| author birthplace |||
2011-02-17 15:12:35
Author_A ||| 1297720502293 ||| DOB (status=PL) ||| date of birth ||| 2011-02-17
14:46:40
Author_A ||| 1297720502305 ||| birthPlace (status=PL) ||| place of birth |||
2011-02-17 15:10:22
Template ||| 1297720502331 ||| welcome (Typeless, status=ED) ||| welcome page |||
2011-02-17 23:18:18
```

Artie notes that there are five Author_A flex attribute instances (one of which is voided), one Author_CD flex definition, and a Template asset.

61.3 List Start Menu Items

Artie further uses the command-line tool to browse for any start menu items that are assigned to the "Acceptance" site.

Command:

```
/opt/cs/export/envision/cs_workspace$ java com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin password=xceladmin
   'resources=@STARTMENU:*' fromSites=Acceptance cmd=listcs
```

Output:

```
Resource Type ||| Resource Id ||| Name ||| Description ||| Modified On
--------------------------------------------------------------------------------
@STARTMENU ||| 1297720502206 ||| Find Author ||| null ||| -
@STARTMENU ||| 1297720502214 ||| Find Author Attribute ||| null ||| -
@STARTMENU ||| 1297720502222 ||| Find Author Definition ||| null ||| -
@STARTMENU ||| 1297720502230 ||| Find Author Filter ||| null ||| -
```

```
@STARTMENU ||| 1297720502238 ||| Find Author Parent ||| null ||| -
@STARTMENU ||| 1297720502246 ||| Find Author Parent Def ||| null ||| -
@STARTMENU ||| 1297720494070 ||| Find CSElement, FirstSiteII ||| Find CSElement
||| -
@STARTMENU ||| 1297720494086 ||| Find Page, FirstSiteII ||| Find Page ||| -
@STARTMENU ||| 1297720494078 ||| Find SiteEntry, FirstSiteII ||| Find SiteEntry
||| -
@STARTMENU ||| 1297720494066 ||| Find Template, FirstSiteII ||| Find Template |||
-
@STARTMENU ||| 1297720502210 ||| New Author ||| null ||| -
@STARTMENU ||| 1297720502218 ||| New Author Attribute ||| null ||| -
@STARTMENU ||| 1297720502226 ||| New Author Definition ||| null ||| -
@STARTMENU ||| 1297720502234 ||| New Author Filter ||| null ||| -
@STARTMENU ||| 1297720502242 ||| New Author Parent ||| null ||| -
@STARTMENU ||| 1297720502250 ||| New Author Parent Def ||| null ||| -
@STARTMENU ||| 1297720501427 ||| New CSElement ||| null ||| -
@STARTMENU ||| 1297720494052 ||| New Page, FirstSiteII ||| New Page ||| -
@STARTMENU ||| 1297720501431 ||| New SiteEntry ||| null ||| -
@STARTMENU ||| 1297720501435 ||| New Template ||| null ||| -
```

61.4 Export All Resources to the Desired Workspace

Artie wants to create a module using all of the resources listed in Section 61.2, "List the Resources in the WebCenter Sites Instance" and Section 61.3, "List Start Menu Items." He runs the following command to export all of the resources, at one time, into the specified workspace:

Command:

```
/opt/cs/export/envision/cs_workspace$ java com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin password=xceladmin
   'resources=@STARTMENU:*;@ALL_ASSETS' fromSites=Acceptance cmd=export
datastore=authorModule
```

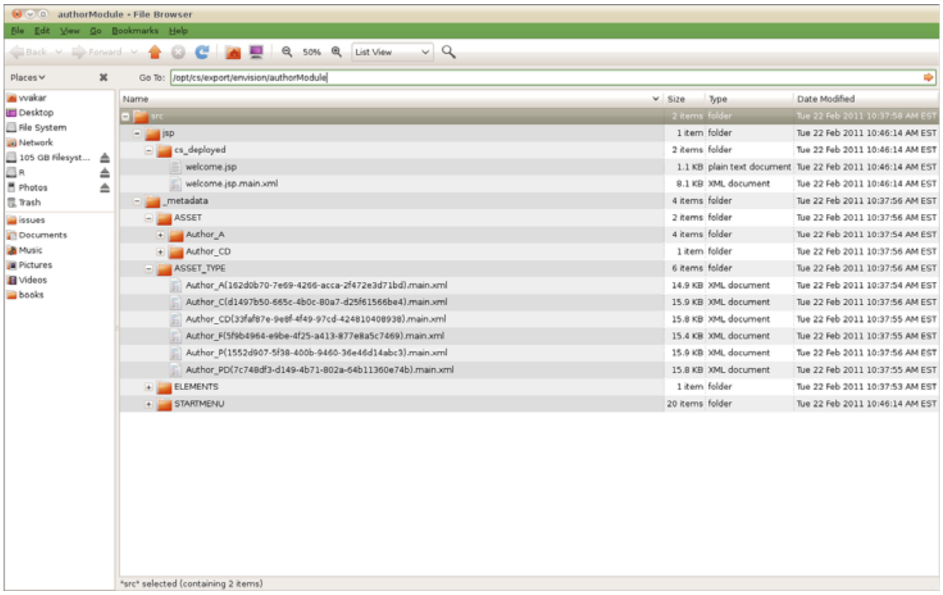
Output:

```
*** Exporting batch 1298385511005
Exporting ASSETDATA Author_CD:1297720502271 (batch 1298385511005)
Exporting ASSETDATA Author_A:1297720502260 (batch 1298385511005)
Exporting ASSET_TYPE Author_A (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/LoadSiteTree
(batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_
A/AppendSelectDetails (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_
A/AppendSelectDetailsSE (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/IndexAdd (batch
1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/IndexReplace
(batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/IndexCreateVerity
(batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/ContentDetails
(batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/ContentForm
(batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/PostUpdate (batch
1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/PreUpdate (batch
1298385511005)
...
```

All asset types for the flex family are included in the export. In addition, all elements belonging to those types are included as well. This information, although not usually modified, is necessary in order to make the module Artie is creating reusable on other WebCenter Sites instances.

61.5 Inspect the Module's Content

Artie inspects the authorModule workspace on his file system.



Go To: /opt/cs/export/envision/authorModule

Name	Size	Type	Date Modified
src	2 items	folder	Tue 22 Feb 2011 10:37:58 AM EST
jsp	1 item	folder	Tue 22 Feb 2011 10:46:14 AM EST
cs_deployed	2 items	folder	Tue 22 Feb 2011 10:46:14 AM EST
welcome.jsp	1.1 KB	plain text document	Tue 22 Feb 2011 10:46:14 AM EST
welcome.jsp.main.xml	8.1 KB	XML document	Tue 22 Feb 2011 10:46:14 AM EST
_metadata	4 items	folder	Tue 22 Feb 2011 10:37:56 AM EST
ASSET	2 items	folder	Tue 22 Feb 2011 10:37:56 AM EST
Author_A	4 items	folder	Tue 22 Feb 2011 10:37:54 AM EST
Author_CD	1 item	folder	Tue 22 Feb 2011 10:37:56 AM EST
ASSET_TYPE	6 items	folder	Tue 22 Feb 2011 10:37:56 AM EST
Author_A(162d0b70-7e69-4266-acca-2f472e3d71bd).main.xml	14.9 KB	XML document	Tue 22 Feb 2011 10:37:54 AM EST
Author_C(d1497b50-665c-4b0c-80a7-d25f61566be4).main.xml	15.9 KB	XML document	Tue 22 Feb 2011 10:37:56 AM EST
Author_CD(33faf87e-9e8f-4f49-97cd-424810408938).main.xml	15.8 KB	XML document	Tue 22 Feb 2011 10:37:55 AM EST
Author_F(5f9b4964-e9be-4f25-a413-877e8a5c7469).main.xml	15.4 KB	XML document	Tue 22 Feb 2011 10:37:55 AM EST
Author_P(1552d907-5f38-400b-9460-36e46d14abc3).main.xml	15.9 KB	XML document	Tue 22 Feb 2011 10:37:56 AM EST
Author_PD(7c748df3-d149-4b71-802a-64b11360e74b).main.xml	15.8 KB	XML document	Tue 22 Feb 2011 10:37:55 AM EST
ELEMENTS	1 item	folder	Tue 22 Feb 2011 10:37:53 AM EST
STARTMENU	20 items	folder	Tue 22 Feb 2011 10:46:14 AM EST

"src" selected (containing 2 items)

Artie notes that the Template asset, flex family members, asset types, and start menu items were all exported to the workspace on his file system.

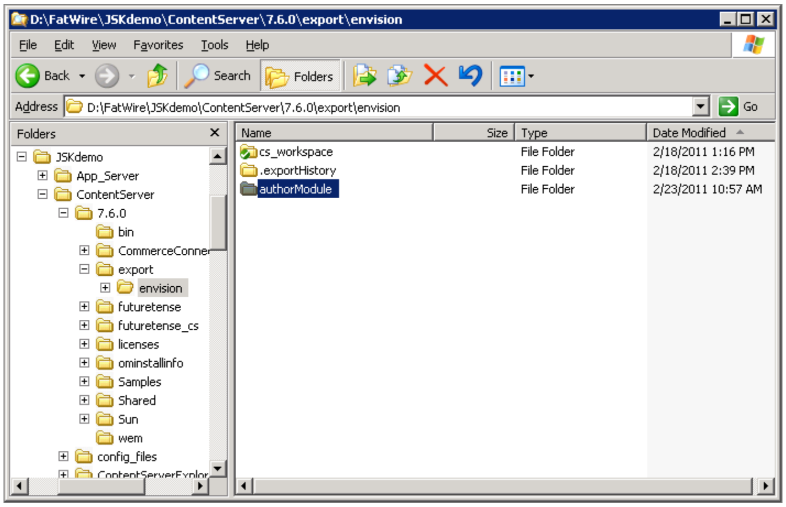
61.6 Archive the Module

Artie creates a `.zip` file archive of the authorModule workspace and saves it.

61.7 Import the Module to a WebCenter Sites Instance

Artie decides to import the module into the FirstSiteII sample site.

1. Artie unzips the module into the workspace location of the target WebCenter Sites instance.



Name	Size	Type	Date Modified
cs_workspace		File Folder	2/18/2011 1:16 PM
.exportHistory		File Folder	2/18/2011 2:39 PM
authorModule		File Folder	2/23/2011 10:57 AM

2. Using the command-line tool, Artie imports the asset types and start menu items into the target WebCenter Sites instance.

Command:

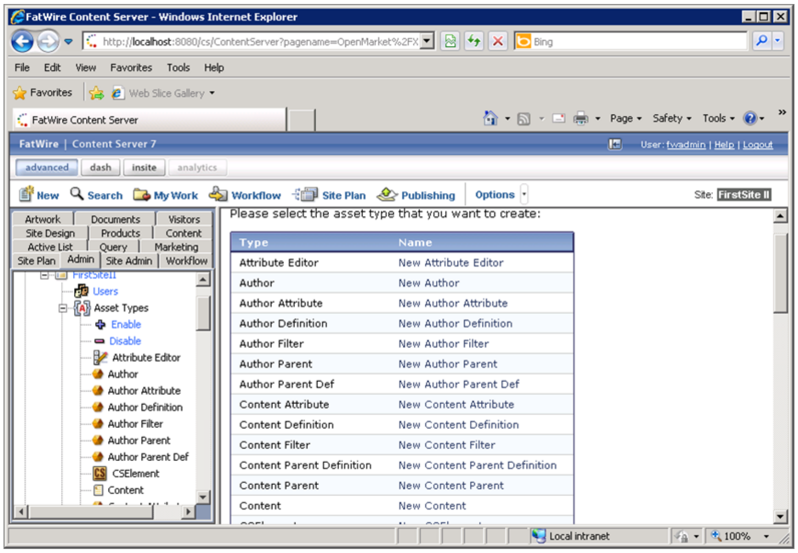
```
D:\FatWire\JSKdemo\ContentServer>java com.fatwire.csdt
   .client.main.CSDT
http://localhost:8080/cs/ContentServer username=fwadmin password=xceladmin
   resources=@ALL_NONASSETS cmd=import datastore=authorModule
   toSites=FirstSiteII
```

Output:

```
*** Importing batch 1298052933085
Importing DSKEY @STARTMENU-4340b65d-a9e4-4131-ac7f-51185a79b18d (batch
1298052933085)
Saved 1297720494070 (batch 1298052933085)
Importing DSKEY @STARTMENU-0a2decd4-b6be-418c-9992-a4332480bb20 (batch
1298052933085)
Saved 1297720501435 (batch 1298052933085)
Importing DSKEY @STARTMENU-66edea6d-218e-41b7-b5ac-ec3453bd53b7 (batch
1298052933085)
Saved 1297720502210 (batch 1298052933085)
Importing DSKEY @STARTMENU-c416c0d6-98a7-4ebf-babb-78d0699698de (batch
1298052933085)
Saved 1297720502230 (batch 1298052933085)
Importing DSKEY @ASSET_TYPE-162d0b70-7e69-4266-acca-2f472e3d71bd (batch
1298052933085)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_
A/LoadSiteTree (batch 1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/LoadSiteTree (batch
1298052933085)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_
A/AppendSelectDetails (batch 1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetails (batch
1298052933085)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_
A/AppendSelectDetailsSE (batch 1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetailsSE (batch
1298052933085)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_
A/IndexAdd (batch 1298052933085)
```

```
Saved OpenMarket/Xcelerate/AssetType/Author_A/IndexAdd (batch 1298052933085)
...
```

3. Artie access the WebCenter Sites Admin interface for the FirstSiteII sample site and confirms that the asset types and start menu items were imported successfully.



Please select the asset type that you want to create:

Type	Name
Attribute Editor	New Attribute Editor
Author	New Author
Author Attribute	New Author Attribute
Author Definition	New Author Definition
Author Filter	New Author Filter
Author Parent	New Author Parent
Author Parent Def	New Author Parent Def
Content Attribute	New Content Attribute
Content Definition	New Content Definition
Content Filter	New Content Filter
Content Parent Definition	New Content Parent Definition
Content Parent	New Content Parent
Content	New Content

4. Now, Artie imports the assets.

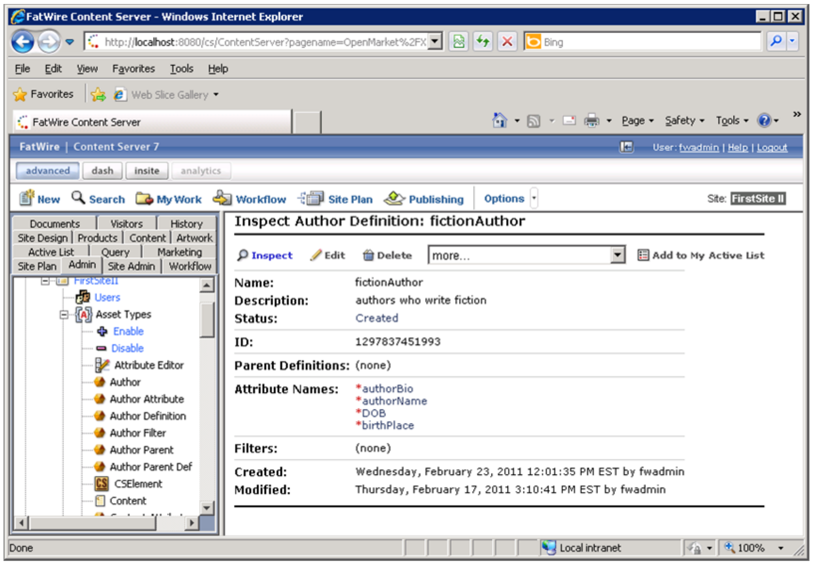
Command:

```
D:\FatWire\JSKdemo\ContentServer>java com.fatwire.csdt
   .client.main.CSDT http://localhost:8080/cs/ContentServer username=fwadmin
   password=xceladmin resources=@ALL_ASSETS cmd=import datastore=authorModule
   toSites=FirstSiteII
```

Output:

```
*** Importing batch 1298480206533
Importing DSKEY Author_A-cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 (batch
1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451977 (batch 1298480206533)
Importing DSKEY Author_A-42afd458-e90c-4e18-a4b6-47d322b46414 (batch
1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451981 (batch 1298480206533)
Importing DSKEY Author_A-9fe04c6e-36e7-4ee3-8c76-8c02edf74136 (batch
1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451985 (batch 1298480206533)
Importing DSKEY Author_CD-71d6067b-35f6-47f4-ae97-3876303abb37 (batch
1298480206533)
Importing DSKEY Author_A-ada2d6be-ef14-4766-b446-911bfa838835 (batch
1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451989 (batch 1298480206533)
Dependency @ASSET_TYPE-Author_C already exists, skipping.
Dependency @ASSET_TYPE-Author_P already exists, skipping.
Dependency @ASSET_TYPE-Author_CD already exists, skipping.
Dependency @ASSET_TYPE-Author_PD already exists, skipping.
Dependency @ASSET_TYPE-Author_F already exists, skipping.
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_CD:1297837451993 (batch 1298480206533)
Importing DSKEY Template-89b05c0f-227b-4dcb-961e-2ab6e6af2dae (batch
1298480206533)
Saved Template:1297837452000 (batch 1298480206533)
*** Completed importing batch 1298480206533
```

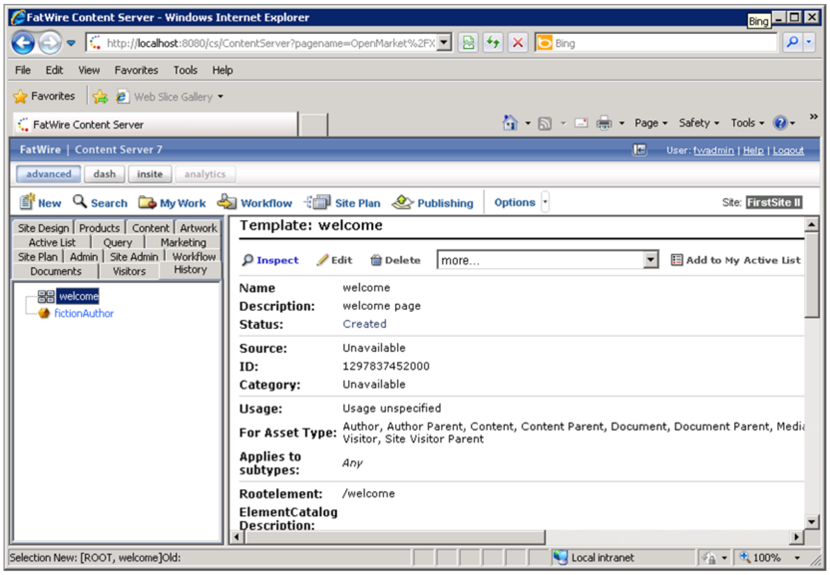
5. Artie verifies that the flex definition is imported into the FirstSiteII sample site successfully:


FatWire Content Server - Windows Internet Explorer
http://localhost:8080/cs/ContentServer?pagename=OpenMarket%2FX
Bing
File Edit View Favorites Tools Help
Favorites
Web Slice Gallery
FatWire Content Server
Page Safety Tools
FatWire | Content Server 7
User: fwadmin | Help | Logout
advanced dash insite analytics
New Search My Work Workflow Site Plan Publishing Options
Site: FirstSite II
Documents Visitors History
Site Design Products Content Artwork
Active List Query Marketing
Site Plan Admin Site Admin Workflow
FirstSiteII
Users
Asset Types
Enable
Disable
Attribute Editor
Author
Author Attribute
Author Definition
Author Filter
Author Parent
Author Parent Def
CSElement
Content
Inspect Edit Delete more... Add to My Active List
Done
Local intranet
100%

Inspect Author Definition: fictionAuthor

Name: fictionAuthor
Description: authors who write fiction
Status: Created
ID: 1297837451993
Parent Definitions: (none)
Attribute Names: *authorBio
*authorName
*DOB
*birthPlace
Filters: (none)
Created: Wednesday, February 23, 2011 12:01:35 PM EST by fwadmin
Modified: Thursday, February 17, 2011 3:10:41 PM EST by fwadmin

6. Using the command-line tool, he also imports the Template asset. He then accesses the WebCenter Sites Admin interface again to verify the Template asset is imported correctly.


FatWire Content Server - Windows Internet Explorer
Bing
http://localhost:8080/cs/ContentServer?pagename=OpenMarket%2FX
File Edit View Favorites Tools Help
Favorites
Web Slice Gallery
FatWire Content Server
Page Safety Tools
FatWire | Content Server 7
User: fwadmin | Help | Logout
advanced dash insite analytics
New Search My Work Workflow Site Plan Publishing Options
Site: FirstSite II
Site Design Products Content Artwork
Active List Query Marketing
Site Plan Admin Site Admin Workflow
Documents Visitors History
welcome
fictionAuthor
Inspect Edit Delete more... Add to My Active List
Local intranet
100%

Template: welcome

Name welcome
Description: welcome page
Status: Created
Source: Unavailable
ID: 1297837452000
Category: Unavailable
Usage: Usage unspecified
For Asset Type: Author, Author Parent, Content, Content Parent, Document, Document Parent, Medi
Visitor, Site Visitor Parent
Applies to subtypes: Any
Rootelement: /welcome
ElementCatalog Description:

Selection New: [ROOT, welcome]Old:

The entire module is imported successfully into the FirstSiteII sample site. This module can be reused and imported into any desired WebCenter Sites instance.

Part III

Customizing the Contributor Interface

This part begins with an overview of the development environment for customizing the WebCenter Sites Contributor interface. Later chapters provide information about customizable interface components, customization methods, and supporting code.

Note: Sample code for this part is located in the `misc\Samples\UICustomization\` directory of the unbundled WebCenter Sites distribution file.

- Code for customizing search views and the dashboard is provided in the `CustomAttrEditor.zip` file.
- Code for customizing asset forms is provided in the `sample_elements.zip` file.

This part contains the following chapters:

62

About Customizing the Oracle WebCenter Sites Contributor Interface

This guide describes procedures for customizing components of the WebCenter Sites Contributor interface. This chapter summarizes the customizable components and guides you to the location of sample code provided with WebCenter Sites.

This chapter contains the following topics:

- Section 62.1, "Before You Begin"
- Section 62.2, "What Can You Customize in the Contributor Interface?"
- Section 62.3, "Where to Find Sample Code?"
- Section 62.4, "Where to Begin?"

> **Note:** The WebCenter Sites Contributor interface is designed to be used by content providers, rather than developers. Therefore, the following system-defined asset types can be displayed (inspected) in the Contributor interface, but they cannot be created (or edited) in the Contributor interface: Template, CSElement, SiteEntry, DimensionSet, Dimension, Attribute Editor, Parent Definitions, Attributes, and Flex Definitions. Similarly, the following system-defined asset types are not accessible from the Contributor interface: FW_Application, and FW_View. Assets of these types can be created (edited) and displayed only in the WebCenter Sites Admin interface.

62.1 Before You Begin

Developers using this guide are required to have a working knowledge of the Contributor interface; experience with Java, JavaScript, and HTML; and solid familiarity with WebCenter Sites development tools.

Information about the Contributor interface is available in the *Oracle Fusion Middleware WebCenter Sites User's Guide* and *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

62.2 What Can You Customize in the Contributor Interface?

The following list summarizes the components you can customize in the Contributor interface:

- Dashboard. See Chapter 64, "Contributor Interface: Customizing the Dashboard."

- Search views. See Chapter 65, "Contributor Interface: Customizing Search Views."
- Global and site-specific configuration properties, toolbar, and menu bar. See Chapter 66, "Contributor Interface: Customizing Global Properties, Toolbar, and Menu Bar."
- Asset forms. See Chapter 67, "Contributor Interface: Customizing Asset Forms."

62.3 Where to Find Sample Code?

Some of the sample code for illustrating interface customization is provided in this guide. Other code is either packaged in WebCenter Sites, or available independently. Paths to such code are listed in the individual chapters of this guide.

62.4 Where to Begin?

The Contributor interface framework contains a component called the *UI Controller*, which handles most of the interface-related requests, except for those pertaining to asset forms. The UI Controller is described in Chapter 63, "Contributor Interface: Understanding the Framework and UI Controller."

- If you are customizing any of the following components: dashboard, search views, configuration properties, toolbars, or menu bars, we recommend starting with Chapter 63 to obtain basic information about the concepts and code you will be using in most of this guide and your customization process.
- If you are customizing asset forms, you can skip to Chapter 67, "Contributor Interface: Customizing Asset Forms" for information about modifying asset form headers and building an attribute editor.

63

Contributor Interface: Understanding the Framework and UI Controller

This chapter describes the framework of the WebCenter Sites Contributor interface, particularly the UI Controller, which handles all requests pertaining to the interface. This chapter also discusses elements that are executed by the UI Controller. The concepts introduced here are basic to customizing the Contributor interface. They are used throughout this guide.

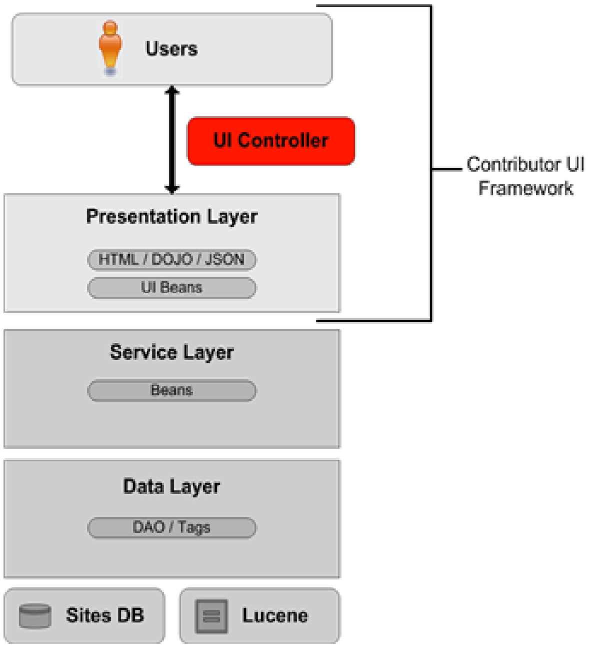
This chapter contains the following topics:

- Section 63.1, "Overview of the Contributor Framework"
- Section 63.2, "UI Controller"
- Section 63.3, "Custom Elements"

63.1 Overview of the Contributor Framework

The framework of the WebCenter Sites Contributor interface sits on top of the Services Layer and handles client requests. As shown in Figure 63–1, the framework consists of the Presentation Layer and UI Controller.

Figure 63–1 Contributor Interface Framework



The Presentation Layer consists of elements that render views and elements that generate a response. The UI Controller is used to process the requests it receives from the Contributor interface, as explained in Section 63.2, "UI Controller."

> **Note:** The UI Controller is not used to process requests pertaining to asset forms, given that asset forms exist outside the Contributor framework. Asset forms are discussed in Chapter 67, "Contributor Interface: Customizing Asset Forms."

63.2 UI Controller

The UI Controller, shown in Figure 63–1, is the entity that processes the requests it receives from the Contributor interface. This section describes the UI Controller's request processing phases, conventions for naming the elements in each phase, and the process by which the UI Controller checks for custom elements.

This section contains the following topics:

- Section 63.2.1, "How the UI Controller Processes Requests"
- Section 63.2.2, "Example: UI Controller Processing an Element Request"

63.2.1 How the UI Controller Processes Requests

The UI Controller can be reached by invoking the `fatwire/ui/controller` SiteCatalog entry. The UI Controller requires the incoming request to provide at least one parameter, `elementName`, which determines the controller element to be executed. For example, the following URL invokes the controller element `Foo/Bar`:

```
http://localhost:7001/sites/ContentServer?pagename=fatwire/ui/controller/c
ontroller&elementName=Foo/Bar
```

A controller element is processed in the following three phases:

1. Configuration phase
2. Action phase
3. Presentation phase

where each phase consists of running a distinct element. For each phase, the corresponding element name is determined by a naming convention, described below.

> **Note:** A controller element is any element that can be invoked through the UI Controller.
>
> In each of the phases described above, the UI Controller first tests for the custom element specific to that phase. The process flow is illustrated in the steps of Section 63.2.2, "Example: UI Controller Processing an Element Request."

1. Configuration Phase

This phase consists of evaluating the configuration element. The configuration element is meant to contain configuration settings used by the controller element being invoked. The expected element name is `<controllerElementName>Config`, where `<controllerElementName>` is the value of the `elementName` parameter. For instance, in our example, where the controller element name is assumed to be `Foo/Bar`, the expected name of the configuration element is `Foo/BarConfig`.

The Configuration phase is based on Apache Commons Configuration and requires configuration data to be formatted as a valid XML document. For example:

```
<myconfig>
   <foo>123</foo>
   <bar>foobar</bar>
</myconfig>
```

The XML configuration data is evaluated into a configuration object, that is, an instance of `org.apache.commons.configuration.beanutils.ConfigurationDynaBean`, which is kept in the request scope, where it is identified by the name of the XML root element. In our example, the configuration object can be accessed in the Action phase or Presentation phase as follows:

```
ConfigurationDynaBean configBean =
(ConfigurationDynaBean)request.getAttribute("myconfig");
```

where `myconfig` matches the name of the top-level XML element in the configuration element. More information about Apache commons configuration can be obtained at the following URL: `http://commons.apache.org/configuration`.

Note: About Configuration Elements in the Configuration Phase:

1. The Configuration phase is conditional. If the element `<controllerElementName>Config` does not exist, the UI Controller skips this phase and moves on to the next phase without creating a configuration object.
2. Unlike in the other two phases, the configuration element is not evaluated directly (using, for example, `ics.callElement`). Instead, it is invoked through the `fatwire/ui/controller/readConfiguration` SiteCatalog entry, using `ics.ReadPage()`, allowing to capture its output.
3. While creating the configuration object, the controller checks if there is a custom configuration available for the requested element under CustomElements. If there is one, the controller, by default, merges the custom configuration with the corresponding system configuration. This is the default behavior. If you do not want to merge the custom configuration with the system configuration, specify `merge=false` for the field **resdetails1** or **resdetails2** in the ElementCatalog entry for the custom configuration element.

   As an example - if the system configuration is:

   ```
   <myconfig>
      <foo>123</foo>
      <bar>foobar</bar>
   </myconfig>
   ```

   And if the custom configuration is:

   ```
   <myconfig>
      <foo1>456</foo1>
      <bar1>foo1bar1<bar1>
   </myconfig>
   ```

   Then the controller merges the system configuration and custom configuration to have the following:

   ```
   <myconfig>
      <foo>123</foo>
      <bar>foobar</bar>
      <foo1>456</foo1>
      <bar1>foo1bar1</bar1>
   </myconfig>
   ```

2. Action Phase

In this phase, the UI Controller evaluates the action element. The expected name of the action element is `<controllerElementName>Action`. In our example, the action element name is `Foo/BarAction`.

The action element is meant to contain arbitrary business logic. It typically builds java objects in the request scope, in order to be consumed by the next phase.

Note: The Action phase is conditional. If the element `<controllerElementName>Action` does not exist, the UI Controller skips this phase and moves on to the Presentation phase.

3. Presentation Phase

In this last phase, the UI Controller evaluates the presentation element, whose name depends on the content type of the generated output. The UI Controller can serve either HTML (the default behavior) or JSON. The element name would then be `<controllerElementName>Html` or `<controllerElementName>Json`.

In our example, the UI Controller would attempt to evaluate `Foo/BarHtml`, because HTML is the default content type. If you wish to generate JSON data instead, you must explicitly specify a response type as follows:

```
http://localhost:7001/sites/ContentServer?pagename=fatwire/ui/controller/c
ontroller&elementName=Foo/Bar&responseType=json
```

In this case, the UI Controller will attempt to evaluate the presentation element called `Foo/BarJson`.

63.2.2 Example: UI Controller Processing an Element Request

When the UI Controller processes any element request, it tests for the custom element as follows: In each phase (Configuration, Action, and Presentation), the UI Controller first looks for the custom element specific to that phase (for more information, see Section 63.3.2, "How the UI Controller Locates Elements"). If the custom element is not found, the UI Controller looks for the default element. If the default element is not found, the UI Controller skips the phase and moves on to the next phase.

The steps below explain, by example, how the UI Controller processes an element request. In this example, the request is for an existing element named `UI/Layout/LeftNavigation`, and the response type is `Html`:

1. **Configuration Phase.** The UI Controller looks for the `LeftNavigation` element's configuration. That is, the UI Controller looks for the element named `LeftNavigationConfig.jsp` under `CustomElements` (in the `ElementCatalog`). If the element exists, the UI Controller reads this element. Otherwise, it reads the default element `LeftNavigationConfig.jsp` (in `UI/Layout/`). The UI Controller then generates the configuration object and keeps this object in the request scope.

   An alternative is to pass the configuration file name as an argument to the UI Controller call. The passed parameter is named `configName`. If `configName` is passed, the UI Controller looks for the element specified in that parameter.

2. **Action Phase.** The UI Controller now looks for the element `LeftNavigationAction.jsp`. If it finds the element under `CustomElements`, the UI Controller executes this element. Otherwise, it executes the default `LeftNavigationAction.jsp` element (in `UI/Layout/`).

3. **Presentation Phase.** In the current example, the response type is Html. Therefore, the UI Controller looks for the element `LeftNavigationHtml.jsp`. If it finds the element under `CustomElements`, the UI Controller executes this element to generate an `Html` response. Otherwise, it executes the default `LeftNavigationHtml.jsp` element (in `UI/Layout/`).

63.3 Custom Elements

When customizing the Contributor interface, store your custom elements in the recommended location as discussed in this section, and ensure you have a clear understanding of how they are located by the UI Controller.

This section contains the following topics:

- Section 63.3.1, "Element Storage"

- Section 63.3.2, "How the UI Controller Locates Elements"
- Section 63.3.3, "Element Naming Conventions in This Guide"

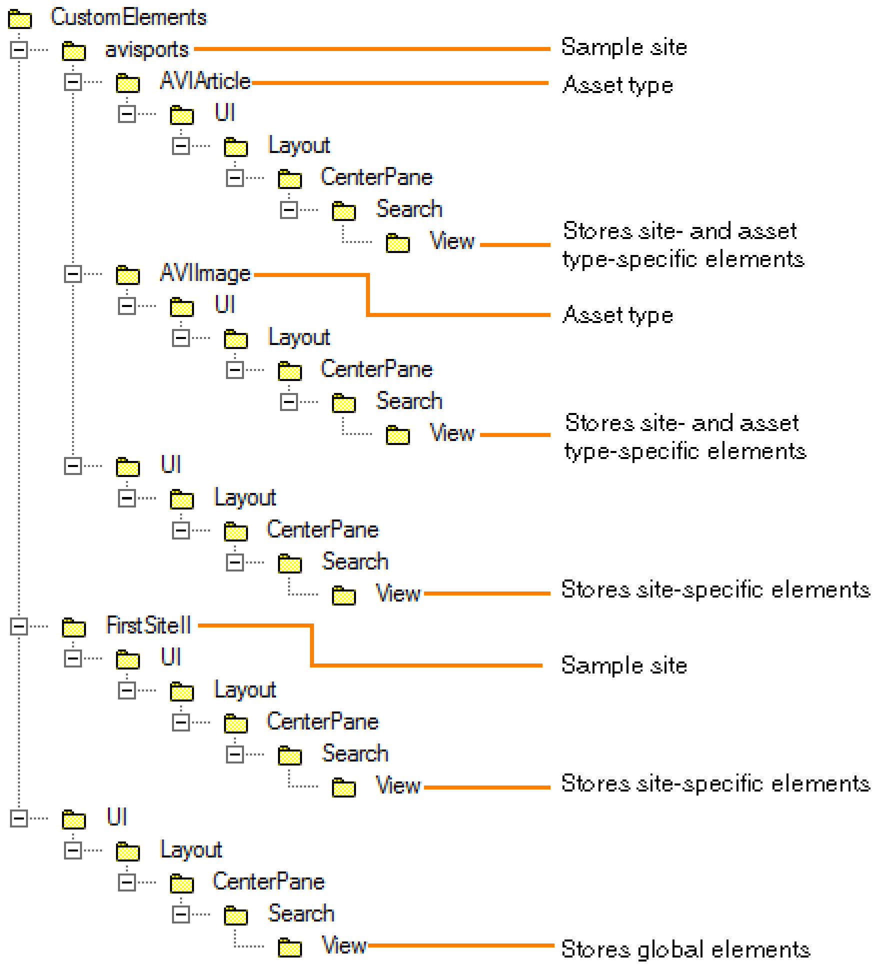
63.3.1 Element Storage

The framework of the Contributor interface allows developers to keep their custom elements separate from the system default elements, in accordance with best practices.

> **Note:** We recommend not modifying the system's default configurations. Instead, create your own custom elements and store them under `CustomElements` of the `ElementCatalog` to ensure their preservation during upgrades.

The path to a custom element depends on whether the element is global, site-specific, site- and asset type- specific, or just asset type-specific. For an example, see Figure 63–2.

Figure 63–2 Paths to Custom Elements



63.3.2 How the UI Controller Locates Elements

When the UI Controller looks for an element:

1. The UI Controller first looks for the customized version of the element by traversing all paths under `CustomElements` in the following order:

   a. Site-specific and asset type-specific paths

   b. Asset type-specific paths

   c. Site-specific paths

   d. Global paths

   (For an example of paths, see Figure 63–2.)

2. If the custom element is not found, the UI Controller uses the system-defined element.

> **Note:** For the UI Controller to use the asset type-specific element, the `assetTypeParam` parameter must be passed with a valid asset type as its value.

63.3.3 Element Naming Conventions in This Guide

When referring to a system-defined element or sample element packaged with WebCenter Sites, this guide provides the full path to the element. The full path always begins with UI/Layout/. For example, the system-defined element `DashBoardContentsConfig.jsp` is presented as follows in this guide:

```
UI/Layout/CenterPane/DashBoardContentsConfig
```

When referring to a custom-defined element that you create, this guide provides only the name of the element (JSP), given that its path is unknown. For example:

```
DashBoardContentsConfig.jsp
```

It is assumed that the custom element is stored under `CustomElements`.

64

Contributor Interface: Customizing the Dashboard

This chapter describes how to customize the dashboard of the WebCenter Sites Contributor interface. It familiarizes you with the dashboard configuration and provides sample code, which you can use while you perform the procedures described in this chapter.

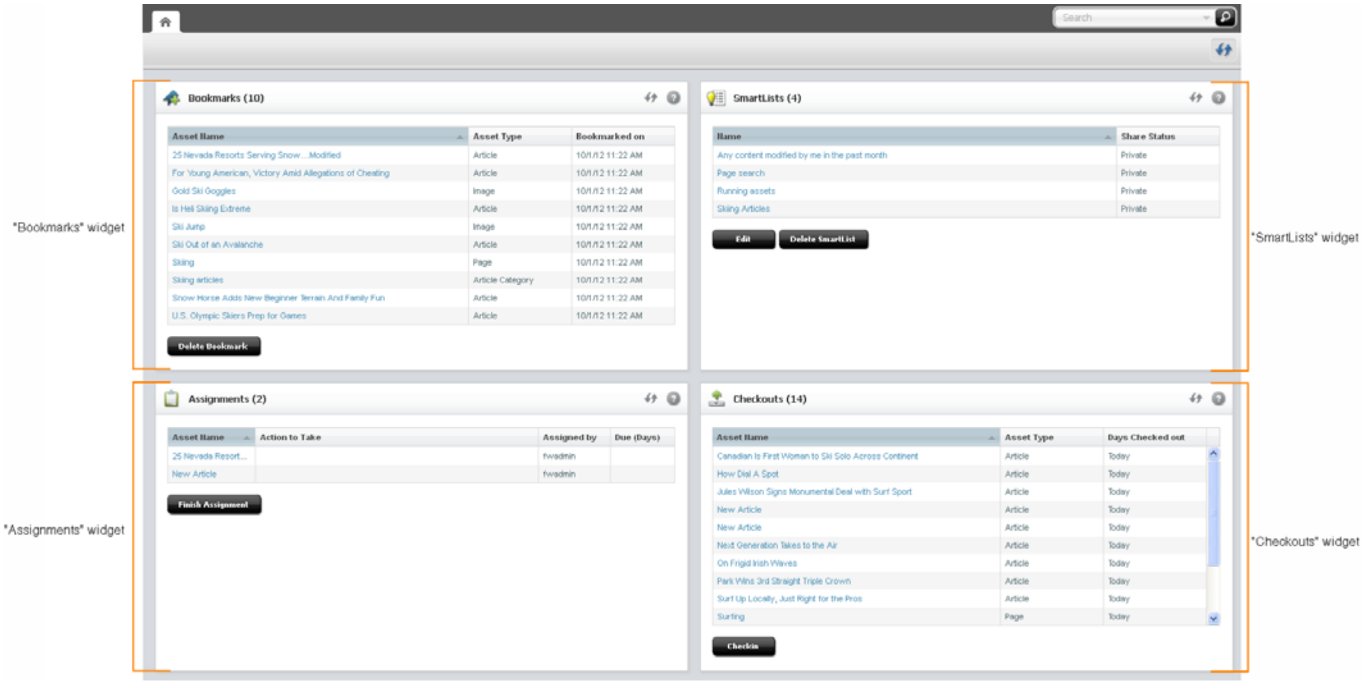
This chapter contains the following topics:

- Section 64.1, "Overview of Dashboard Customization"
- Section 64.2, "Customizing the Dashboard"
- Section 64.3, "Examples of Customizing the Dashboard"

64.1 Overview of Dashboard Customization

When you log in to the Contributor interface, the dashboard is displayed. By default, the dashboard displays the following ready-to-use widgets: Bookmarks, SmartLists, Checkouts, and Assignments, as shown in Figure 64–1.

Figure 64–1   Dashboard with Default Widgets



Asset Name	Asset Type	Bookmarked on
25 Nevada Resorts Serving Snow....Modified	Article	10/1/12 11:22 AM
For Young American, Victory Amid Allegations of Cheating	Article	10/1/12 11:22 AM
Gold Ski Goggles	Image	10/1/12 11:22 AM
Is Heli Skiing Extreme	Article	10/1/12 11:22 AM
Ski Jump	Image	10/1/12 11:22 AM
Ski Out of an Avalanche	Article	10/1/12 11:22 AM
Skiing	Page	10/1/12 11:22 AM
Skiing articles	Article Category	10/1/12 11:22 AM
Snow Horse Adds New Beginner Terrain And Family Fun	Article	10/1/12 11:22 AM
U.S. Olympic Skiers Prep for Games	Article	10/1/12 11:22 AM

Name	Share Status
Any content modified by me in the past month	Private
Page search	Private
Running assets	Private
Skiing Articles	Private

Asset Name	Action to Take	Assigned by	Due (Days)
25 Nevada Resort...		fwadmin	
New Article		fwadmin	

Asset Name	Asset Type	Days Checked out
Canadian Is First Woman to Ski Solo Across Continent	Article	Today
How Dial A Spot	Article	Today
Jules Wilson Signs Monumental Deal with Surf Sport	Article	Today
New Article	Article	Today
New Article	Article	Today
Next Generation Takes to the Air	Article	Today
On Frigid Irish Waves	Article	Today
Park Wins 3rd Straight Triple Crown	Article	Today
Surf Up Locally, Just Right for the Pros	Article	Today
Surfing	Page	Today

You can customize the following portions of the dashboard and its widgets:

- Number of columns
- Column width
- Widget's display name, height, and position on the dashboard
- You can also add new widgets.

64.2 Customizing the Dashboard

The system-defined dashboard is generated by the controller element `UI/Layout/CenterPane/DashboardContentsConfig`. You can override this element.

Your dashboard configuration can be global or site-specific. You can customize the default widgets, add new widgets, and delete the ones not required.

To customize the dashboard

Override the element `UI/Layout/CenterPane/DashBoardContentsConfig` by creating your own `DashBoardContentsConfig.jsp` under `CustomElements` and customizing its properties.

> **Note:** When a new widget is added or an existing widget is updated, you must clear user preferences in the WEM UI for the changes to take place.

The `UI/Layout/CenterPane/DashBoardContentsConfig` element is shown next, followed by property descriptions in Table 64–1.

Element `UI/Layout/CenterPane/DashboardContentsConfig`

```
<dashboardconfig>
```

```
   <dashboardlayout>
      <numberofcolumns></numberofcolumns>
      <columnwidths></columnwidths>
   </dashboardlayout>
   <components>
      <component id="widgetId">
         <name>widgetName</name>
         <url>widgetURL</url>
         <height>height_in_px</height>
         <dragRestriction>true | false </dragRestriction>
         <column>number_of_column_in_which_to_display_widget</column>
     </component>
         …
         …
         …
   </components>
</dashboardconfig>
```

Table 64–1 Properties in `UI/Layout/CenterPane/DashBoardContentsConfig.jsp`

Property	Description	Value
`<numberofcolumns>`	Number of columns in the dashboard display.	Integer greater than 0. The system default is `2`.
`<columnwidths>`	Comma-separated widths of columns.	For example, if there are `3` columns in `<numberofcolumns>` then the `<columnwidths>` can be `30,30,40`.
`<components>`	This section is used to define dashboard widgets.	N/A
`<component>`	Used to define a single widget.	N/A
`<id>`	ID of the widget.	Alpha-numeric value unique across widgets. Special characters are not allowed.
`<name>`	Displayed name of the widget.	Arbitrary string.
`<url>`	Controller URL.	The file location of the widget in the `UI/Layout/CenterPane/DashBoard/<Your_Element>/` directory.
`<height>`	Height of the widget.	Height in pixels. For example, `300px`.
`<dragRestriction>`	Restricts dragging of the widget.	`true | false`
`<column>`	The column in which the widget is displayed.	`1` to *`n`*, where n is the value specified in `<numberofcolumns>`.

64.3 Examples of Customizing the Dashboard

You can add new widgets to the WebCenter Sites Contributor dashboard. Adding a new widget involves two basic steps:

1. Creating the widget element.
2. Registering the new widget in your custom `DashBoardContentsConfig.jsp` element.

This section illustrates the process of adding a widget to the dashboard. This section contains the following examples:

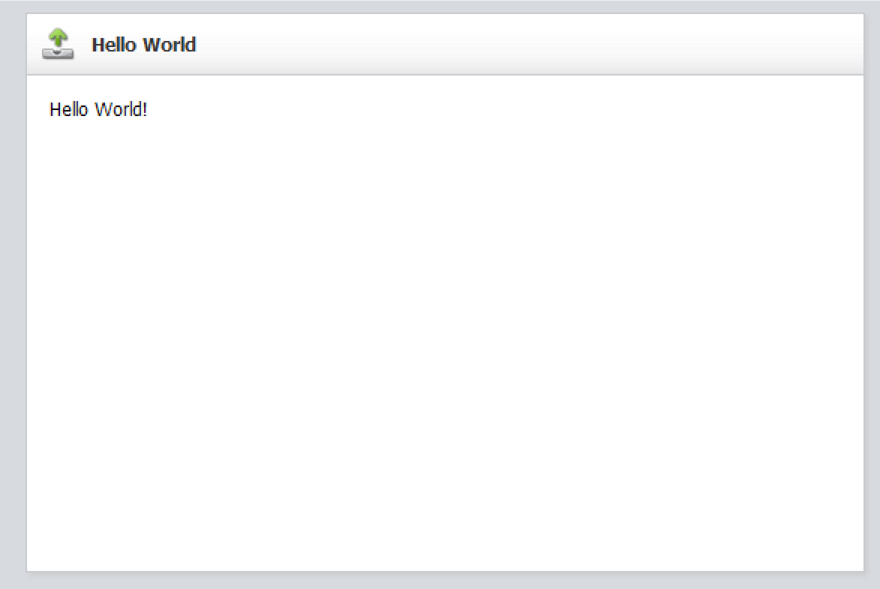
- Section 64.3.1, "Adding a 'Hello World' Widget"

- Section 64.3.2, "Adding a Widget that Shows Recently Modified Assets"

64.3.1 Adding a 'Hello World' Widget

In this section, you will create and register a simple widget, shown in Figure 64–2.

Figure 64–2 'Hello World' Widget



Hello World!

To add your widget to the dashboard

1. Create your widget:

   a. Create a JSP element under CustomElements. In this example, we name the element `HelloWorldHtml`.

   b. For widget code, you can navigate to the sample file provided with this guide and copy its content.

2. Register your widget (add it to the dashboard):

   a. Open your custom `DashBoardContentsConfig.jsp`, locate the `<components>` section, and add the newly created widget's specifications. For example:

```
<component id="helloworld">
  <name>Hello World</name>
  <url>Path_to_your_widget_under_CustomElements</url>
  <height>300px</height>
  <closable>false</closable>
  <open>true</open>
  <dragRestriction>true</dragRestriction>
  <style>checkoutPortlet</style>
  <column>2</column>
</component>
```

   b. Go to the `<applicationServer_install_directory>/webapps/<cs_context>/WEB-INF/classes/ReqAuthConfig.xml` file and add the path to the sample element, under the `excludedControllerElements` list. In our example, the path is:

```
<property name="excludedControllerElements">
```

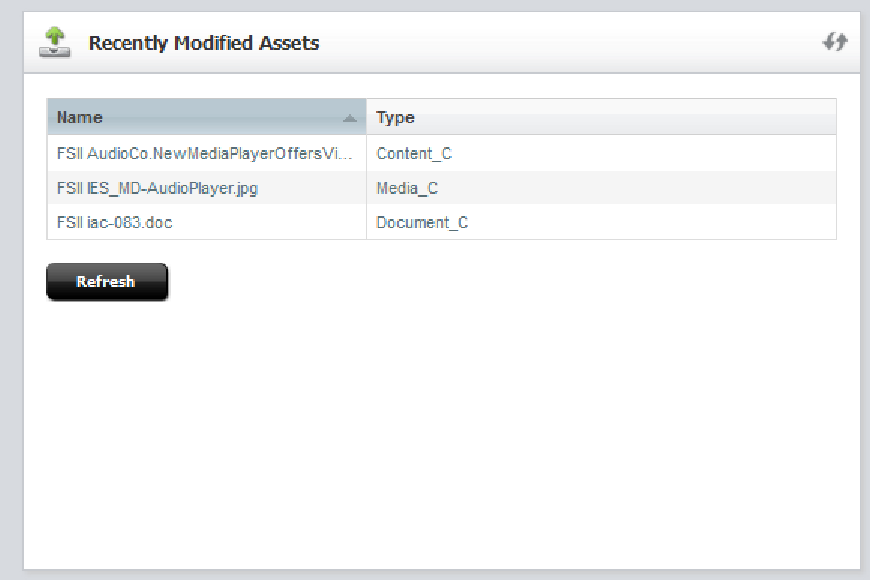
```
        <list>
            <value>/UI/Layout/CenterPane/DashBoard/HelloWorld</value>
        </list>
    </property>
```

c. Refresh the home page of your Contributor interface. The new widget is displayed on your dashboard (Figure 64–2).

64.3.2 Adding a Widget that Shows Recently Modified Assets

In this section, you will create a widget that shows which assets were modified in the past week. After completing the steps in this section, your dashboard will display a widget similar to the one in Figure 64–3.

Figure 64–3 'Recently Modified Assets' Widget



Name	Type
FSII AudioCo.NewMediaPlayerOffersVi...	Content_C
FSII IES_MD-AudioPlayer.jpg	Media_C
FSII iac-083.doc	Document_C

To add your widget to the dashboard

1. Create your widget:

   a. Create an `Action` JSP element under `CustomElements`. In this example, we name the element `RecentlyModifiedAssetsAction.jsp`. For the widget code, you can navigate to the sample file provided with this guide and copy its content.

   b. Create a `Json` JSP element for the Action element created in the previous step. In this example, we name the element `RecentlyModifiedAssetsJson.jsp`. For the code, you can navigate to the sample file provided with this guide and copy its content. Place the element in the same location as the `RecentlyModifiedAssetsAction.jsp` element.

   c. Create a presentation element under CustomElements for your widget. Name the element after the widget element. In this example, we name the display element `RecentlyModifiedAssetsHtml.jsp`. For the code, you can navigate to the sample file provided with this guide and copy its content.

Note: The presentation element will call the `RecentlyModifiedAssetsAction.jsp` element. Enter the path to that element.

2. Register your widget (add it to the dashboard):

   a. Open your custom `DashBoardContentsConfig.jsp`, locate the `<components>` section, and add the newly created widget's specifications. For example:

```
<component id="myrecent">
<!-- a unique identifier for the component. This must be unique among all
the components. It can be alpha numeric but no special characters allowed
-->
      <name>Recently Modified Assets</name>
      <url>Path_to_your_custom_widget's presentation_element</url>
      <height>300px</height>
      <closable>false</closable>
      <open>true</open>
      <dragRestriction>false</dragRestriction>
      <style>checkoutPortlet</style>
      <column>2</column>
</component>
```

   b. Go to the `<applicationServer_install_directory>/webapps/<cs_context>/WEB-INF/classes/ReqAuthConfig.xml` file and add the path to the sample element, under the `excludedControllerElements` list. In our example, the path is:

```
<property name="excludedControllerElements">
   <list>
      <value>/UI/Layout/CenterPane/DashBoard/RecentlyModifiedAssets</value>
   </list>
</property>
```

   c. Refresh the dashboard to see the newly configured widget. For example, see Figure 64–3.

65 Contributor Interface: Customizing Search Views

This chapter describes how to customize the List and Thumbnail search views of the WebCenter Sites Contributor interface.

This chapter contains the following topics:

- Section 65.1, "Overview of Search View Customization"
- Section 65.2, "Customization Processes"
- Section 65.3, "Customizing Undocked Views"
- Section 65.4, "Customizing Docked Views"
- Section 65.5, "Customizing Sort Menus and Tooltips"

65.1 Overview of Search View Customization

When users log in to the WebCenter Sites Contributor interface and access their sites, they can perform a simple or advanced search to locate the required assets. Search results are then presented in either List view or Thumbnail view. This section describes the different search views, which of their features can be customized, and which elements control the configuration of search views. If you need information about search functionality and views, see the *Oracle WebCenter Sites User's Guide*.

This section contains the following topics:

- Section 65.1.1, "Types of Search Views"
- Section 65.1.2, "What You Can Customize in Search Views"
- Section 65.1.3, "View-Rendering Process"
- Section 65.1.4, "Configuration Elements for Search Views"

65.1.1 Types of Search Views

The search results panel can be either undocked or docked and displayed as a List view or Thumbnail view. Thus, the Contributor interface displays the following views:

- List Undocked
- List Docked
- Thumbnail Undocked
- Thumbnail Docked

An undocked view opens only when no assets are open for editing. A docked view is attached to assets in edit mode and therefore opens only when an asset is open in edit mode.

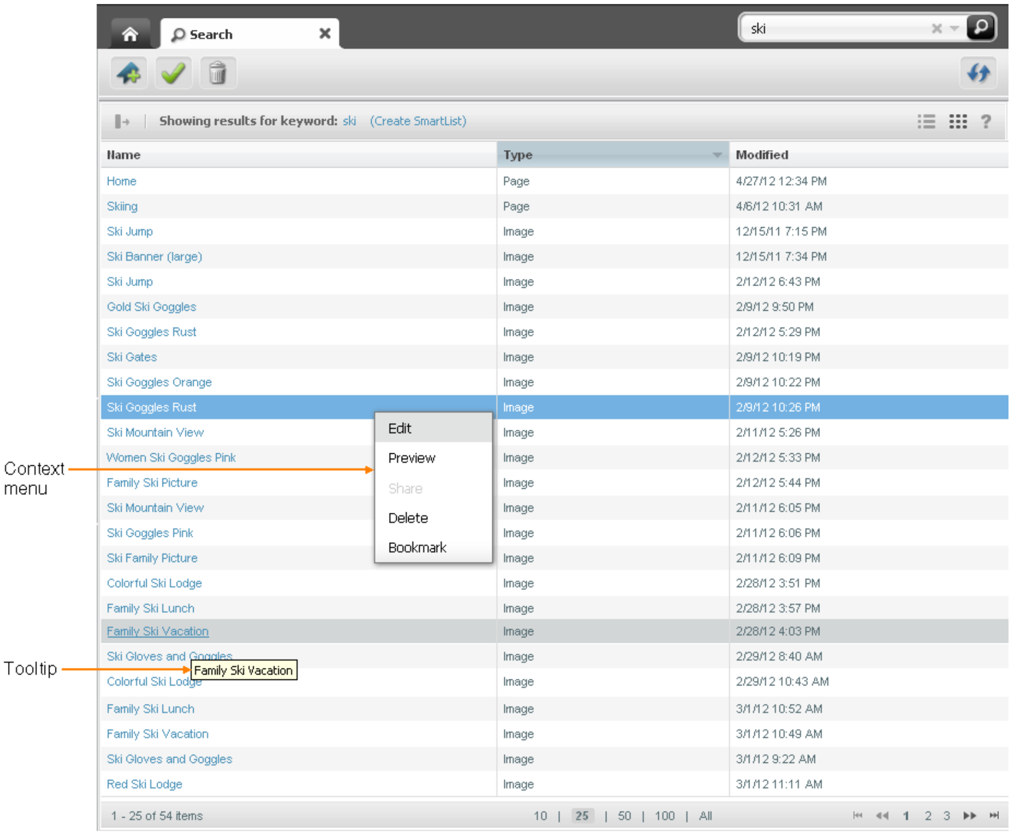
65.1.2 What You Can Customize in Search Views

Figure 65–1 summarizes the features you can customize in List view. Figure 65–2 summarizes the features you can customize in Thumbnail view. For more information, see Section 65.3, "Customizing Undocked Views," which also applies to docked views.

Sort menus and tooltips are customized separately. For more information, see Section 65.5, "Customizing Sort Menus and Tooltips."

Which view opens by default for a given mode depends on your configuration settings and the user's search habits. For example, if you set Thumbnail view as the default view for undocked mode, Thumbnail view will open when the user first runs search in undocked mode and will continue to open until the user switches to List view (search remembers the user's choice until browser cookies are cleared).

Figure 65–1 Customizable Features in List View



Showing results for keyword: ski (Create SmartList)

Name	Type	Modified
Home	Page	4/27/12 12:34 PM
Skiing	Page	4/6/12 10:31 AM
Ski Jump	Image	12/15/11 7:15 PM
Ski Banner (large)	Image	12/15/11 7:34 PM
Ski Jump	Image	2/12/12 6:43 PM
Gold Ski Goggles	Image	2/9/12 9:50 PM
Ski Goggles Rust	Image	2/12/12 5:29 PM
Ski Gates	Image	2/9/12 10:19 PM
Ski Goggles Orange	Image	2/9/12 10:22 PM
Ski Goggles Rust	Image	2/9/12 10:26 PM
Ski Mountain View	Image	2/11/12 5:26 PM
Women Ski Goggles Pink	Image	2/12/12 5:33 PM
Family Ski Picture	Image	2/12/12 5:44 PM
Ski Mountain View	Image	2/11/12 6:05 PM
Ski Goggles Pink	Image	2/11/12 6:06 PM
Ski Family Picture	Image	2/11/12 6:09 PM
Colorful Ski Lodge	Image	2/28/12 3:51 PM
Family Ski Lunch	Image	2/28/12 3:57 PM
Family Ski Vacation	Image	2/28/12 4:03 PM
Ski Gloves and Goggles	Image	2/29/12 8:40 AM
Colorful Ski Lodge	Image	2/29/12 10:43 AM
Family Ski Lunch	Image	3/1/12 10:52 AM
Family Ski Vacation	Image	3/1/12 10:49 AM
Ski Gloves and Goggles	Image	3/1/12 9:22 AM
Red Ski Lodge	Image	3/1/12 11:11 AM

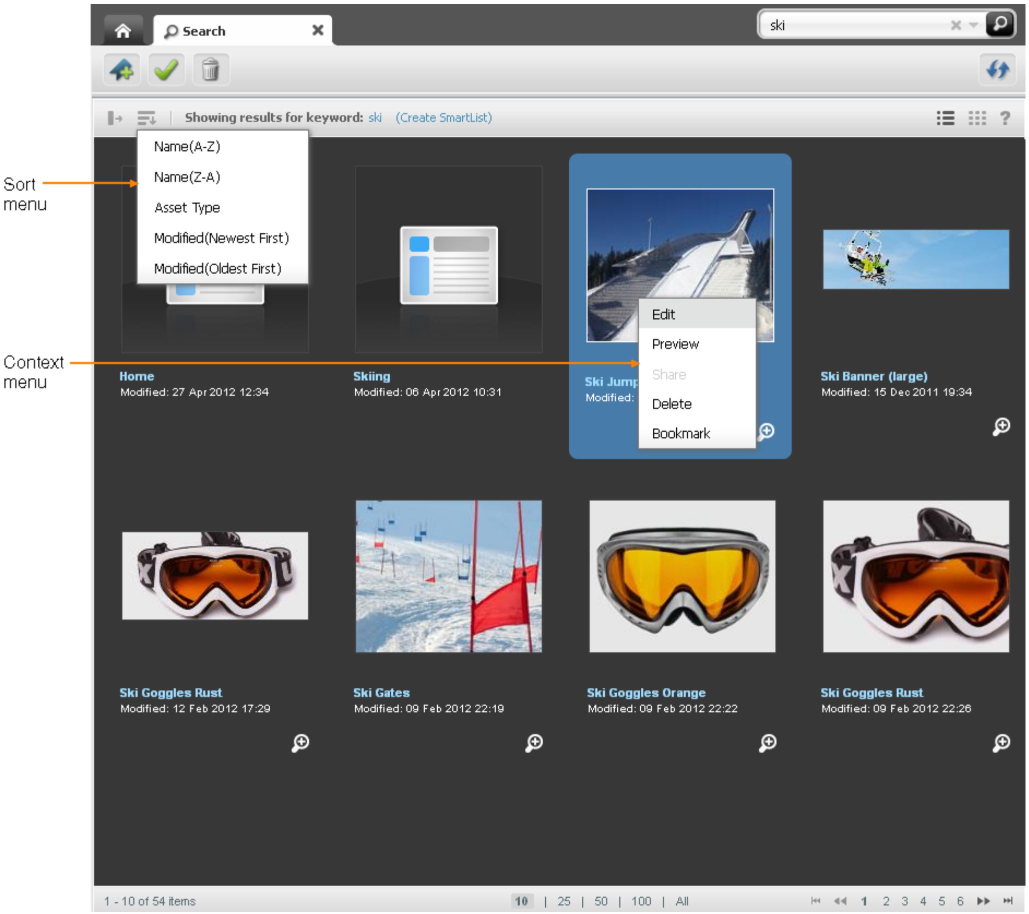
1 - 25 of 54 items 10 | 25 | 50 | 100 | All 1 2 3

Customizable features for List view include:

- Maximum number of items to return
- Number of rows per page
- Fields (columns) to display
- Column display name
- Column width

- Format of date and other fields
- Default sort field and sort order
- Sort menu (docked mode)
- Context (right-click) menu
- Tooltip (docked mode)

Figure 65–2 Customizable Features in Thumbnail View



Sort menu

Context menu

Customizable features for Thumbnail view include:

- Maximum number of items to return
- Number of rows per page
- Asset types for which special thumbnails will be shown
- Fields to display
- Format of date and other fields
- Default sort field and sort order
- Sort menu
- Context (right-click) menu
- Tooltip (docked mode)

65.1.3 View-Rendering Process

System-defined and custom-defined views are rendered by similar processes. To illustrate, we begin with system-defined views.

System-defined views are rendered by the following elements (JSPs), whose names for undocked and docked views differ only by the `Docked` prefix.

When undocked:

- List view is rendered by the element:
  `UI/Layout/CenterPane/Search/View/ListViewHtml`
- Thumbnail view is rendered by the element:
  `UI/Layout/CenterPane/Search/View/ThumbnailViewHtml`

When docked:

- List view is rendered by the element:
  `UI/Layout/CenterPane/Search/View/DockedListViewHtml`
- Thumbnail view is rendered by the element:
  `UI/Layout/CenterPane/Search/View/DockedThumbnailViewHtml`

Rendering of undocked and docked views is similar (except that the names of elements for docked views start with `Docked`). The steps below illustrate the rendering of undocked views.

1. When a user runs a search routine, the search functionality determines the user's current view, which is either the default view or a subsequently chosen view.

   > **Note:** "Default view" is the view that the system renders the first time search is run. (List view is the system-defined default view for both undocked and docked modes.) If the user switches to a different view, search remembers and continues to display the user's choice until browser cookies are cleared.

2. Search functionality reads `UI/Layout/CenterPane/Search/SearchResultsConfig` to obtain the path to the element that will initiate the rendering of the view:
   - If the user is running search for the first time, or continues using the default view, search reads the value of the `<defaultview>` property.
   - If the user's view is other than the default view, search reads the value of either the `<listview>` or `<thumbnailview>` property (depending on which view was determined in step 1).
3. If search determines that List view must be rendered, it reads the element `UI/Layout/CenterPane/Search/View/ListViewConfig` and invokes `UI/Layout/CenterPane/Search/View/ListViewHtml`, which then renders the list view. If search determines that the Thumbnail view must be rendered, it reads the element `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig` and invokes `UI/Layout/CenterPane/Search/View/ThumbnailViewHtml`, which then renders the Thumbnail view.

You can override all of the above system-defined elements by customizing your own identically named elements and placing them under `CustomElements` to actualize the changes shown in Figure 65–1 and Figure 65–2. You can also customize individual features, such as sort menus and tooltips, by using the elements `UI/Layout/CenterPane/Search/View/SearchTopBarConfig` and `UI/Layout/CenterPane/Search/View/SearchToolTipHtml` respectively.

For a comprehensive list of elements, see Section 65.1.4, "Configuration Elements for Search Views."

65.1.4 Configuration Elements for Search Views

This section summarizes the JSP elements you will use to customize search views.

- **System-defined configuration elements**: You will be configuring identically named elements to customize search views and searches that are global or specific to a site, asset type(s), or site and asset type(s). All of your customized elements should be stored under `CustomElements` (for an example, see Figure 65–3). For a summary of the elements, see the following tables:
  - Table 65–1 lists system-defined configuration elements that define ready-to-use undocked views (all of the element names end with `Config`).
  - Table 65–2 lists system-defined configuration elements that define ready-to-use docked views (all of the element names end with `Config`).
  - Table 65–3 lists system-defined elements for customizing a search view's individual features, such as sort menus and tooltips (element names end with either `Config` or `Html`).
- **Custom elements**: Table 65–4 lists sample custom elements that are packaged with WebCenter Sites to help illustrate customization code.

Table 65–1 Configuration Elements for Undocked Search Views

Path to Configuration Element (JSP)	Description	See ...
`UI/Layout/CenterPane/Search/SearchResultsConfig`	Element for setting the default search view (List view or Thumbnail view) in undocked mode.	Section 65.3.2, "Setting the Default Undocked View to List or Thumbnail"
`UI/Layout/CenterPane/Search/View/ListViewConfig`	Element for configuring the undocked List view.	Section 65.3.3, "Customizing the Undocked List View"
`UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`	Element for configuring the undocked Thumbnail view.	Section 65.3.4, "Customizing the Undocked Thumbnail View"

Table 65–2 Configuration Elements for Docked Search Views

Path to Configuration Element (JSP)	Description	See ...
`UI/Layout/CenterPane/Search/DockedSearchResultsConfig`	Element for setting the default search view (List view or Thumbnail view) in docked mode.	Section 65.2, "Customization Processes"
`UI/Layout/CenterPane/Search/View/DockedListViewConfig`	Element for configuring the docked List view.	Section 65.2, "Customization Processes"
`UI/Layout/CenterPane/Search/View/DockedThumbnailViewConfig`	Element for configuring the docked Thumbnail view.	Section 65.2, "Customization Processes"

Table 65–3 Configuration and Presentation Elements for Other Features in Search Views

Path to Configuration Element (JSP)	Description	See ...
`UI/Layout/CenterPane/Search/View/SearchTopBarConfig`	Element for configuring fields as sort options in the sort drop-down menus for docked List, undocked Thumbnail, and docked Thumbnail views.	Section 65.5.1, "Customizing Sort Menus"
`UI/Layout/CenterPane/Search/View/SearchToolTipHtml`	Element for configuring tooltips for docked views (List and Thumbnail). This element enables you to configure tooltip appearance and custom messages.	Section 65.5.2, "Customizing Tooltips for Search Results"
`UI/Config/GlobalHtml`	Element for configuring context (right-click) menus. This element is valid for all search views.	Section 65.5.3, "Customizing Context Menus"

Table 65–4 Custom Sample Elements for Search Views

Path to Sample Element	Description
`CustomElements/avisports/AVIArticle/UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`	Configuration element for undocked Thumbnail view for the AVIArticle asset type in the avisports sample site.
`CustomElements/avisports/AVIArticle/UI/Layout/CenterPane/Search/View/DockedThumbnailViewConfig`	Configuration element for docked Thumbnail view for AVIArticle asset type in avisports site.
`CustomElements/avisports/AVIImage/UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`	Configuration element for undocked Thumbnail view for the AVIImage asset type in the avisports sample site.
`CustomElements/avisports/AVIImage/UI/Layout/CenterPane/Search/View/DockedThumbnailViewConfig`	Configuration element for docked Thumbnail view for the AVIImage asset type in the avisports sample site.
`CustomElements/avisports/UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`	Configuration element for undocked Thumbnail view for the avisports sample site.
`CustomElements/avisports/UI/Layout/CenterPane/Search/View/DockedThumbnailViewConfig`	Configuration element for docked Thumbnail view for the avisports sample site.

65.2 Customization Processes

When customizing views in undocked and docked mode, you will follow similar procedures. The main differences are the following:

- **Customizing undocked and docked views:**

  When customizing undocked views, you will follow instructions in Section 65.3, "Customizing Undocked Views" and name your configuration elements (JSPs) as shown in that section (also in Table 65–1). When customizing docked views, you will also follow instructions in Section 65.3, "Customizing Undocked Views," but name your configuration elements as shown in Table 65–2 (that is, include the `Docked` prefix).

- **Customizing sort menus and tooltips for search views:**

  Elements for creating sort menus and tooltips apply to both undocked and docked mode. You will name the elements exactly as shown in Table 65–3 (and Section 65.5, "Customizing Sort Menus and Tooltips," regardless of mode.

- **If you wish to display a field in docked List view or docked Thumbnail view:**

  By default, the UI/Layout/CenterPane/Search/View/DockedListViewConfig element points to the UI/Layout/CenterPane/Search/View/ListViewConfig element to get only the first listed field and display its name in docked List view. The field is defined in the first <field> property, as follows:

```
<field>
    <fieldname>fieldname</fieldname>
    <displayname>DisplayName</displayname>
```

  If you want to display any other field name in the docked List view, you will have to specify that name in your custom DockedListViewConfig.jsp element. The same logic applies to displaying a field name in docked Thumbnail view (except that your configuration elements are named ThumbnailViewConfig and DockedThumbnailViewConfig).

65.3 Customizing Undocked Views

Customizing the undocked List and Thumbnail views involves overriding the system-defined elements shown in Table 65–1 by configuring your own identically named elements and placing them under CustomElements.

This section contains the following topics:

- Section 65.3.1, "Basic Steps for Customizing Undocked Views"
- Section 65.3.2, "Setting the Default Undocked View to List or Thumbnail"
- Section 65.3.3, "Customizing the Undocked List View"
- Section 65.3.4, "Customizing the Undocked Thumbnail View"

65.3.1 Basic Steps for Customizing Undocked Views

To customize an undocked view, you can take any combination of the following steps:

- Set the default undocked view to be List or Thumbnail for all asset types or your choice of asset types. To set the view(s), you will override the element UI/Layout/CenterPane/Search/SearchResultsConfig, as shown in Section 65.3.2, "Setting the Default Undocked View to List or Thumbnail."
- Configure the undocked List and/or Thumbnails views. You can specify the number of columns to be displayed in the view(s), configure column names and column widths, specify the sort order of returned items, and more (see Figure 65–1 and Figure 65–2).
  - To configure the List view, you will override the element UI/Layout/CenterPane/Search/View/ListViewConfig, described in Section 65.3.3, "Customizing the Undocked List View."
  - To configure the Thumbnail view, you will override the element UI/Layout/CenterPane/Search/View/ThumbnailViewConfig, described in Section 65.3.4, "Customizing the Undocked Thumbnail View."
- Configure additional features, such as sort menus for the views. In this step, you will be configuring JSP elements that are specific to the features of the view (such as a sort menu), rather than the view itself. For more information, see Section 65.5, "Customizing Sort Menus and Tooltips."

65.3.2 Setting the Default Undocked View to List or Thumbnail

When setting the default search view (List or Thumbnail), you can set it globally for all asset types. You can also specify a default search view for selected asset types of your choice.

To set the default search view(s)

Override the element `UI/Layout/CenterPane/Search/SearchResultsConfig` by creating your own `SearchResultsConfig.jsp` under `CustomElements` and customizing its properties.

The `UI/Layout/CenterPane/Search/SearchResultsConfig` element is shown next, followed by property descriptions in Table 65–5.

Element **`UI/Layout/CenterPane/Search/SearchResultsConfig`**:

```
<searchconfig>
<listview>UI/Layout/CenterPane/Search/View/ListView</listview>
   <thumbnailview>UI/Layout/CenterPane/Search/View/ThumbnailView</thumbnailview>
   <defaultview>listview</defaultview>
   <assettypeviews>
       <assettype id="Page" name="Page">listview</assettype>
         …
         …
         …
   </assettypeviews>
</searchconfig>
```

Table 65–5 Properties in `UI/Layout/CenterPane/Search/SearchResultsConfig`

Property	Description	Value
`<listview>`	Path to the ListView controller element.	`UI/Layout/CenterPane/Search/View/ListView` **Note:** Do not change the value of this property.
`<thumbnailview>`	Path to the ThumbnailView controller element.	`UI/Layout/CenterPane/Search/View/ThumbnailView` **Note:** Do not change the value of this property.
`<defaultview>`	Specifies whether List or Thumbnail is the default view. **Note:** The default view is the view that opens the first time search is run. If the user switches the view, search remembers the user's choice until browser cookies are cleared.	`listview` | `thumbnailview` **Note:** The value of this property is case-sensitive.
`<assettypeviews>`	Used to selectively configure a default view for one or more asset types.	N/A
`<assettype id= name= >`	Used to specify the asset type and its default view (which remains until the user either switches to a different view or clears browser cookies). You can specify as many asset types as necessary (one per `<assettype>`).	`<assettype id="`*`unique_identifier`*`" name="`*`AssetTypeName`*`"> listview | thumbnailview </assettype>`

65.3.3 Customizing the Undocked List View

When customizing the List view, you can set the type of content to be returned and its presentation.

To customize the undocked List view

Override the `UI/Layout/CenterPane/Search/View/ListViewConfig` element by creating your own `ListViewConfig.jsp` under `CustomElements` and customizing its properties.

The `UI/Layout/CenterPane/Search/View/ListViewConfig` element is shown next, followed by property descriptions in Table 65–6.

Element **`UI/Layout/CenterPane/Search/View/ListViewConfig`**:

```
<listviewconfig>
    <numberofitems>1000</numberofitems>
    <numberofitemsperpage>100</numberofitemsperpage>
    <defaultsortfield>  </defaultsortfield>
    <defaultsortorder>  </defaultsortorder>
    <fields>
      <field id="name">
        <fieldname>name</fieldname>
        <displayname>Name</displayname>
        <width>350px</width>
        <formatter>fw.ui.GridFormatter.nameFormatter</formatter>
        <displayintooltip>true</displayintooltip>
    </field>
    <field id="updateDate">
        <fieldname>updateddate</fieldname>
        <displayname>Modified</displayname>
        <!-- <dateformat>MM/dd/yyyy hh:mm a z </dateformat> -->
        <javadateformat>SHORT</javadateformat>
        <width>auto</width>
        <formatter></formatter>
        <displayintooltip>true</displayintooltip>
      </field>
        …
        …
        …
      </fields>
</listviewconfig>
```

Table 65–6 Properties in `UI/Layout/CenterPane/Search/View/ListViewConfig`

Property	Description	Value
`<numberofitems>`	Maximum number of items returned by search.	Integer greater than `0`. **Note:** If `-1` is entered for instance, then all results matching the search criteria are returned.
`<numberofitemsperpage>`	Number of rows per page needed in the search results.	`100` is the default.
`<defaultsortfield>`	Default field that search should sort when fetching search results.	The default is empty. Therefore, search results are displayed by relevance. Configure this element if any other field should be set as the default for sorting.

Table 65–6 (Cont.) Properties in `UI/Layout/CenterPane/Search/View/ListViewConfig`

Property	Description	Value
`<defaultsortorder>`	Sort order used by search.	`ascending | descending` Required only when `<defaultsortfield>` is specified.
`<fields>`	Columns that will be shown in List view. These columns will be shown in the same order as listed under `<fields>`. **Note:** If you are creating an asset type-specific configuration and you wish to display asset type-specific attributes in the search results, you will have to enable the asset type index and attribute search. For more information, see the following sections in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*: ■ Adding Asset Types to the Search Index ■ Configuring Attributes for Asset Type Index If you skip this procedure, search will use the global index.	N/A
`<field id= >`	Defines a column to be shown in List view.	`<field id="`*`unique_ identifier`*`">`
`<fieldname>`	Asset's field name to render in the column.	This name must match the column name in the Lucene index. **Note:** If `locale` is added as the field name, it will be displayed only if the site dimension is enabled.
`<displayname>`	Display name shown in the column header.	Alphanumeric string
`<width>`	Width of the column in pixels.	Width in units of `px` (e.g., `350px`). **Note:** We recommend setting the width to `auto` for the last field.
`<formatter>`	Dojo formatter function to display column values in your preferred format.	The formatter must be made available in a dojo module. See the `modules` property in `UI/Config/GlobalHtml`.

Table 65–6 (Cont.) Properties in UI/Layout/CenterPane/Search/View/ListViewConfig

Property	Description	Value
<displayintooltip>	Indicates whether the associated field must be listed in the tooltip for docked List view. **Note:** The element UI/Layout/CenterPane/Search/View/SearchToolTipHtml renders tooltips and uses the value of this property to determine whether to list the associated field name in the tooltip (the field value will also be listed). Tooltips can be customized only for docked views. For instructions, see Section 65.5.2, "Customizing Tooltips for Search Results."	true | false
<dateformat>	Applies to date fields only. This is an option to specify a custom date format if the date needs to be displayed in a format other than javadateformat.	A valid date format string. **Note:** If <dateformat> is used, it takes precedence over <javadateformat>.
<javadateformat>	Applies to date fields only.	Valid values are SHORT, MEDIUM, LONG, and FULL. **Note:** If <javadateformat> is omitted or left blank, the system uses SHORT by default. If <dateformat> is used, it takes precedence over <javadateformat>.

65.3.4 Customizing the Undocked Thumbnail View

When customizing the Thumbnail view, you can set the type of content to be returned and its presentation.

To customize the undocked Thumbnail view

Override the element UI/Layout/CenterPane/Search/View/ThumbnailViewConfig by creating your own ThumbnailViewConfig.jsp under CustomElements and customizing its properties.

The UI/Layout/CenterPane/Search/View/ThumbnailViewConfig element is shown below, followed by property descriptions in Table 65–7.

Note: Pay particular attention to the following properties: <formatter> and <assettypes>. While the element UI/Layout/CenterPane/Search/View/ThumbnailViewConfig is mostly the same as UI/Layout/CenterPane/Search/View/ListViewConfig, the <formatter> property is defined differently. Also, the <assettypes> property is exclusive to ThumbnailViewConfig, where it is used to render thumbnails.

The <assettypes> property is described in detail in Section 65.3.4.1, "More About the <assettypes> Section in the ThumbnailViewConfig Element," where its usage is illustrated with examples. One of the examples shows you how to supplement video assets with a custom element that displays a video player.

Element `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`:

```
<thumbnailviewconfig>
    <numberofitems>1000</numberofitems>
    <defaultsortfield></defaultsortfield>
    <defaultsortorder></defaultsortorder>
    <numberofitemsperpage>12</numberofitemsperpage>
    <formatter>fw.ui.GridFormatter.thumbnailFormatter</formatter>
    <fields>
      <field id="name">
        <fieldname>name</fieldname>
        <displayname>Name</displayname>
        <displayintooltip>true</displayintooltip>
      </field>
      <field id="updateDate">
        <fieldname>updateddate</fieldname>
        <displayname>Modified</displayname>
        <!-- <dateformat>MM/dd/yyyy hh:mm a z </dateformat> -->
        <javadateformat>SHORT</javadateformat>
        <displayintooltip>true</displayintooltip>
      </field>
       …
       …
       …
    </fields>
  <assettypes>
     <assettype id="unique_identifier">
        <type>AVIImage</type>
        <subtype>Image</subtype>
        <element>UI/Layout/CenterPane/Search/View/ImageThumbnail</element>
        <attribute>imageFile</attribute>
      </assettype>
         …
         …
         …
    </assettypes>
</thumbnailviewconfig>
```

Table 65–7 Properties in `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`

Property	Description	Value
`<numberofitems>`	Maximum number of items to be returned by search.	Integer greater than `0`. **Note:** If `-1` is entered for instance, then all results matching the search criteria are returned.
`<numberofitemsperpage>`	Number of rows per page needed in the search results.	`100` is the default value.
`<formatter>`	Dojo formatter function to display values in your preferred format.	The formatter must be made available in a dojo module. See the `modules` property in `UI/Config/GlobalHtml`.
`<defaultsortfield>`	Default sort field that search should sort when fetching search results.	The default is empty. Therefore, search results are displayed by relevance. Configure this element if any other field should be set as a default for sorting.
`<defaultsortorder>`	Sort order used by search.	`ascending | descending` This is required only when `<defaultsortfield>` is specified.

Table 65–7 (Cont.) Properties in `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`

Property	Description	Value
`<fields>`	Fields that will be shown below the thumbnails in Thumbnail view. These fields will be shown in the same order as listed under `<fields>`. **Note:** If you are creating an asset type-specific configuration and you wish to display asset type-specific attributes in the search results, you will have to enable the asset type index and attribute search. For more information, see the following sections in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*: ▪ Adding Asset Types to the Search Index ▪ Configuring Attributes for Asset Type Index If you skip this procedure, search will use the global index.	N/A
`<field id=>`	Describes a field under the thumbnail.	`<field id="`*`unique_identifier`*`">`
`<fieldname>`	Asset's field name to render below the thumbnail.	This name must match the column name in the Lucene index. **Note:** If `locale` is added as the field name, it will be displayed only if the site dimension is enabled.
`<displayname>`	Display name to render below the thumbnail.	Alphanumeric string
`<dateformat>`	Applies to date fields only. This is an option to specify a custom date format if the date needs to be displayed in a format other than `javadateformat`.	A valid date format string. **Note:** If `<dateformat>` is used, it takes precedence over `<javadateformat>`.
`<javadateformat>`	Applies to date fields only.	Valid values are `SHORT`, `MEDIUM`, `LONG`, and `FULL`. **Note:** If `<javadateformat>` is omitted or left blank, the system uses `SHORT` by default. If `<dateformat>` is used, it takes precedence over `<javadateformat>`.
`<displayintooltip>`	Indicates whether the associated field must be listed in the tooltip for docked Thumbnail view. **Note:** The element `UI/Layout/CenterPane/Search/View/SearchToolTipHtml` renders tooltips. It uses the value of the `<displayintooltip>` property to determine whether to list the associated field in the tooltip (the field value will also be listed). Tooltips can be customized only for docked views. For instructions, see Section 65.5.2, "Customizing Tooltips for Search Results."	`true | false`

Table 65–7 (Cont.) Properties in `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`

Property	Description	Value
`<assettypes>`	This section specifies the asset types for which special thumbnails will be shown. Each asset type must have an attribute whose content will be rendered as a thumbnail. For more information as to when this section must be customized, see Section 65.3.4.1, "More About the `<assettypes>` Section in the ThumbnailViewConfig Element."	N/A
`<assettype id= >`	Describes the asset type for which a special thumbnail will be shown.	`<assettype id="`*`unique_identifier`*`">`
`<type>`	Name of the asset type for which a thumbnail will be rendered.	For more information, see Section 65.3.4.1.2 and Section 65.3.4.1.3.
`<subtype>`	Subtype of the asset type.	For more information, see Section 65.3.4.1.2 and Section 65.3.4.1.3.
`<element>`	Path to the controller element that renders the content specified in `<attribute>` as a thumbnail.	For more information, see Section 65.3.4.1.2 and Section 65.3.4.1.3. **Note:** If you do not specify an element, the system-defined element `UI/Layout/CenterPane/Search/View/GlobalThumbnail` will be used to render static icons, stored in the `images/search` directory. For more information, see Section 65.3.4.1.1.
`<attribute>`	Attribute whose content will be shown as a thumbnail.	For more information, see Section 65.3.4.1.2 and Section 65.3.4.1.3.

65.3.4.1 More About the `<assettypes>` Section in the ThumbnailViewConfig Element

Table 65–7 contains the `<assettypes>` section, which may need to be configured, depending on which features you choose to customize. Various `<assettypes>` configuration scenarios are discussed below in the context of the most commonly performed customizations.

This section contains the following topics:

- Section 65.3.4.1.1, "If You Wish to Use Static Icons"
- Section 65.3.4.1.2, "If You Wish to Re-use the System-Defined Image Thumbnail Element"
- Section 65.3.4.1.3, "If You Wish to Use a Custom Thumbnail-Rendering Element"

65.3.4.1.1 If You Wish to Use Static Icons

If you plan to use your own static thumbnails (stored in the file system), there is no need to customize the `<assettypes>` section of the `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig` element, as long as you observe the following conventions:

- The name of the thumbnail icon should not contain spaces (they will be replaced with underscores). The name must be in one of the following formats, depending on the size of the thumbnail:

  - `<assettypename>.png` or `<assettypename>-<subtype>.png` (small thumbnail, 96x96, docked view)
  - `<assettypename>_large.png` or `<assettypename>-<subtype>_large.png` (large thumbnail, 170x170, undocked view)
- The storage location of the icon is the `/images/search` directory of the file system.

If the above conventions are followed, the icon will be automatically rendered as a thumbnail by the `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig` element, which is coded to look for icons in the `/images/search` directory. Naming the icon after the asset type and subtype automatically associates the icon with assets of that type and subtype.

65.3.4.1.2 If You Wish to Re-use the System-Defined Image Thumbnail Element

Customizing the `<assettypes>` section of the `ThumbnailViewConfig.jsp` element is a requirement if you wish to dynamically render *custom images* as thumbnails by re-using the system-defined element `ImageThumbnailHtml.jsp`. This element processes images that are associated with image attributes belonging to specific asset types and/or subtypes.

To re-use the System-Defined ImageThumbnailHtml.jsp

In your custom `ThumbnailViewConfig.jsp`, do the following:

1. Specify the asset types that require a custom image thumbnail. **Each asset type must have an image attribute.**

```
<assettypes>
    <assettype>
        <type>Name_of_AssetType_containing_the_image_attribute</type>
        <subtype>Name_of_subtype_containing_the_image_attribute</subtype>
        <element>UI/Layout/CenterPane/Search/View/ImageThumbnail</element>
        <attribute>Name_of_imageAttribute_containing_the_image</attribute>
    </assettype>
     …
     …
    </assettypes>
```

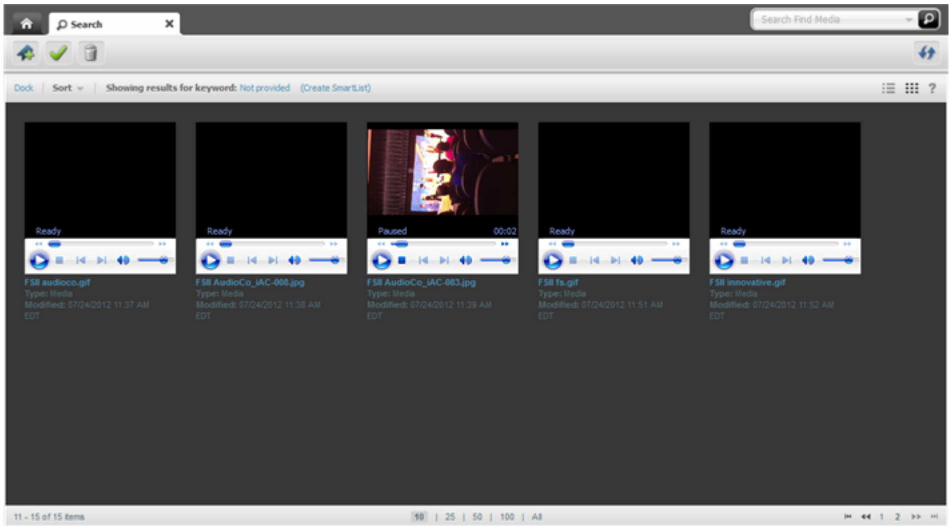
2. For `<element>`, specify the path to the system-defined element `ImageThumbnailHtml.jsp`, exactly as shown in the sample code above.

65.3.4.1.3 If You Wish to Use a Custom Thumbnail-Rendering Element

Customizing the `<assettypes>` section of the `ThumbnailViewConfig.jsp` is a requirement if you plan to use a custom element that dynamically renders the content of an asset type's (or subtype's) `blob` attribute as a thumbnail.

In the example below, you will create elements that work together to render video thumbnails. Figure 65–3 displays a sample video thumbnail view, which you can reproduce by following the steps in this section.

Figure 65–3 Sample Video Thumbnail View



The steps below provide guidelines for (1) creating elements that work together to dynamically render video thumbnails, and (2) customizing the `<assettypes>` section of the `ThumbnailViewConfig` element.

Note: To make this sample work, ensure that you have assets with a `blob` attribute and video files for that `blob` attribute are uploaded to your site's directory.

To play the video file the browser plugin should be available.

To create elements that render video thumbnails

1. Write a video thumbnail `Action` element that uses the AssetAPI and gets the URL of the `blob` using `BlobUtil` for the video attribute specified in the element. (The element can be named as you wish, but it must end in `Action`. The element should be stored in a directory under `CustomElements`.)

   A sample element named `VideoThumbnailAction.jsp` is available in the WebCenter Sites distribution in the `misc\Samples\UICustomization\sample_elements.zip` file, under the `search_view_elements` folder.

2. Write a video thumbnail `Html` element, which takes the URL built in the previous step and renders the video and other asset details below the thumbnail. (The element can be named as you wish, but it must end in `Html`. The element should be stored in a directory under `CustomElements`.) This `Html` element calls the `Action` element in step 1.

   A sample element named `VideoThumbnailHtml.jsp` is available in the WebCenter Sites distribution in the `misc\Samples\UICustomization\sample_elements.zip` file, under the `search_view_elements` folder.

3. To use the video thumbnail `Html` element, configure the `<assettype>` property in your custom `ThumbnailViewConfig.jsp` element as shown below:

```
<assettype>
    <type>Name_of_AssetType_containing_blob_attribute</type>
    <subtype>Name_of_asset_subtype</subtype>
    <element>CustomElements/path_to_your_element/Element</element>
    <attribute>Name_of_attribute_containing_video</attribute>
```

```
</assettype>
```

A sample element named `ThumbnailViewConfig.jsp` is available in the WebCenter Sites distribution in the `misc\Samples\UICustomization\sample_elements.zip` file, under the `search_view_elements` folder.

65.4 Customizing Docked Views

Methods for customizing docked views are similar to those for undocked views. The main differences are outlined in Section 65.2, "Customization Processes."

65.5 Customizing Sort Menus and Tooltips

Features discussed in this section can be customized for undocked views, docked views, or both, as shown in Table 65–8.

This section contains the following topics:

- Section 65.5.1, "Customizing Sort Menus"
- Section 65.5.2, "Customizing Tooltips for Search Results"
- Section 65.5.3, "Customizing Context Menus"

Table 65–8 Customizing Other Features for Search Views

Customization Option	Undocked List	Undocked Thumbnail	Docked List View	Docked Thumbnail	See ...
Sort Menus	No	Yes	Yes	Yes	Section 65.5.1
Tooltips for Search Results	No	No	Yes	Yes	Section 65.5.2
Context Menus	Yes	Yes	Yes	Yes	Section 65.5.3

65.5.1 Customizing Sort Menus

Sort menus can be customized only for the views listed in Table 65–8. You can specify which sort fields to display in a sort menu. You can also specify sort order for each field.

To customize a sort menu

Override the element `UI/Layout/CenterPane/Search/View/SearchTopBarConfig` by creating your own `SearchTopBarConfig.jsp` under `CustomElements` and customizing its properties.

The `UI/Layout/CenterPane/Search/View/SearchTopBarConfig` element is shown next, followed by property descriptions in Table 65–9.

Element `UI/Layout/CenterPane/Search/View/SearchTopBarConfig`:

```
<sortconfig>>
     <sortfields>
        <sortfield id="unique_identifier">
           <fieldname>name</fieldname>
           <displayname>Name(A-Z)</displayname>
           <sortorder>ascending</sortorder>
        </sortfield>
        <sortfield id="unique_identifier">
           <fieldname>name</fieldname>
           <displayname>Name(Z-A)</displayname>
```

```
            <sortorder>descending</sortorder>
         </sortfield>
         <sortfield id="unique_identifier">
            <fieldname>AssetType_Description</fieldname>
            <displayname>Asset Type</displayname>
            <sortorder>ascending</sortorder>
         </sortfield>
               …
               …
      </sortfields>
</sortconfig>
```

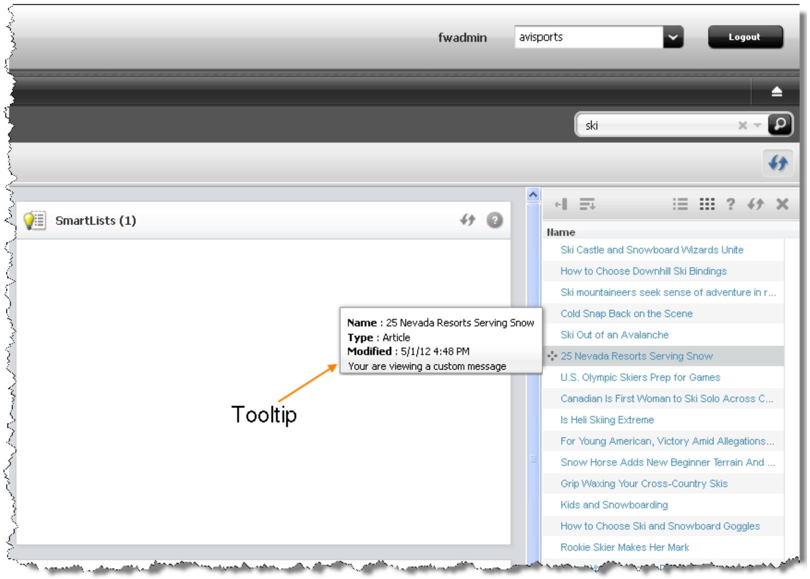
Table 65–9 Properties in `UI/Layout/CenterPane/Search/View/SearchTopBarConfig`

Property	Description	Value
`<sortfield id= >`	Describes the search index field by which to sort search results.	`id=`*`unique_identifier`*
`<fieldname>`	Name of the search index field. **Note:** The same field can be repeated multiple times to provide multiple sort orders.	For example, `name` in the code above.
`<displayname>`	Display name of the user-readable field.	For example, `Name` in the code above.
`<sortorder>`	Sort order.	`ascending` | `descending`

65.5.2 Customizing Tooltips for Search Results

Tooltips can be customized only for docked views. Docked views are displayed in a limited space and therefore provide a limited amount of information about the assets that are returned as search results. Tooltips are a way of displaying more information about the returned assets. For example, you can customize tooltips to display field names and values in addition to those already displayed in docked mode, as shown in Figure 65–4. You can also customize tooltips to display custom messages, and you can modify the appearance of tooltips.

Figure 65–4 Tooltip in Undocked List View



Name : 25 Nevada Resorts Serving Snow
Type : Article
Modified : 5/1/12 4:48 PM
Your are viewing a custom message

Name

- Ski Castle and Snowboard Wizards Unite
- How to Choose Downhill Ski Bindings
- Ski mountaineers seek sense of adventure in r...
- Cold Snap Back on the Scene
- Ski Out of an Avalanche
- 25 Nevada Resorts Serving Snow
- U.S. Olympic Skiers Prep for Games
- Canadian Is First Woman to Ski Solo Across C...
- Is Heli Skiing Extreme
- For Young American, Victory Amid Allegations...
- Snow Horse Adds New Beginner Terrain And ...
- Grip Waxing Your Cross-Country Skis
- Kids and Snowboarding
- How to Choose Ski and Snowboard Goggles
- Rookie Skier Makes Her Mark

The default tooltip for docked search results is rendered by the element `UI/Layout/CenterPane/Search/View/SearchToolTipHtml`. This element renders the tooltip as a box (as shown in Figure 65–4). Within the box, it renders the name of each field in the `<fields>` section of `UI/Layout/CenterPane/Search/View/ListViewConfig` (or `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`), **but only if the field's `<displayintooltip> property is set to true`**. For example, the `Name`, `Type`, and `Modified` fields in the `ListViewConfig.jsp` below are displayed as part of the tooltip in Figure 65–4, given that `<displayintooltip>` is set to `true`:

```
<fields>
        <field>
            <fieldname>name</fieldname>
            <displayname>Name</displayname>
            <width>350px</width>
            <formatter>fw.ui.GridFormatter.nameFormatter</formatter>
            <displayintooltip>true</displayintooltip>
        </field>
        <field>
            <fieldname>type</fieldname>
                  <displayname>Type</displayname>
                  <width>auto</width>
                  <formatter></formatter>
                  <displayintooltip>true</displayintooltip>
        </field>
        <field>
                  <fieldname>updateddate</fieldname>
                  <displayname>Modified</displayname>
                  <javadateformat>SHORT</javadateformat>
                  <width>auto</width>
                  <formatter></formatter>
                  <displayintooltip>true</displayintooltip>
        </field>
```

Note: The `UI/Layout/CenterPane/Search/View/SearchToolTipHtml` element also renders field values. However, customized messages and changes to tooltip appearance must be coded in the custom `SearchToolTipHtml.jsp` element.

To create a tooltip or add fields to the tooltip

1. To create a tooltip, override the element `UI/Layout/CenterPane/Search/View/SearchToolTipHtml` by creating your own `SearchToolTipHtml.jsp` under `CustomElements`.
2. To add fields to the tooltip, add the fields to your custom `ListViewConfig.jsp` or `ThumbnailViewConfig.jsp` and set each field's `<displayintooltip>` property to `true`.
3. To display a custom message in the tooltip (custom or system-defined) or to change the appearance of the tooltip, code a custom `SearchToolTipHtml.jsp` element. For example:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><cs:ftcs>
<style>
.customSearchTooltip {
     font-weight: bold;
     color: #333;
     font-style: italic;
}
</style>

<div class='customSearchTooltip'>
     You are Viewing a Custom Tooltip
</div>
</cs:ftcs>
```

65.5.3 Customizing Context Menus

Context menus can be customized for all search views.

To customize a context menu

Override the element `UI/Config/GlobalHtml` by creating your own `MyConfig.jsp` with the following code:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<cs:ftcs>
   webcenter.sites['${param.namespace}'] = function (config) {
     config.contextMenus = {
       "default":["bookmark"],
       "asset":["edit","preview", "share", "bookmark", "tagasset"],
       "asset/Page":["edit", "preview", "delete", "bookmark"],
       "proxy":["preview", "bookmark", "tagasset"],
       ;
   }
</cs:ftcs>
```

66

Contributor Interface: Customizing Global Properties, Toolbar, and Menu Bar

This chapter describes UI/Config/GlobalHtml, the global configuration element. This chapter also shows you how to customize the features the global element defines for the WebCenter Sites Contributor interface.

This chapter contains the following topics:

- Section 66.1, "Customizing Global Configuration Properties"
- Section 66.2, "Customizing the Toolbar"
- Section 66.3, "Customizing the Menu Bar"
- Section 66.4, "Customizing Context Menus"

66.1 Customizing Global Configuration Properties

Global configuration properties are used to set display conditions for the Contributor interface across all content management sites.

This section contains the following topics:

- Section 66.1.1, "Overview of the Configuration Properties"
- Section 66.1.2, "Modifying Default Configuration Properties"
- Section 66.1.3, "Adding Custom Configuration Properties"

66.1.1 Overview of the Configuration Properties

The client-side framework retrieves its main configuration settings from the server-side controller element `UI/Config/GlobalHtml`. This presentation element serves JavaScript code, which is executed by the client-side application at startup. The JavaScript code defines a JavaScript function, whose name is given as a request parameter by the client-side application:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<cs:ftcs>
webcenter.sites['${param.namespace}'] = function (config) {
    config.maxTabCount = 50;
    config.defaultView =  …;
     … merge
}
</cs:ftcs>
```

The `config` object is then manipulated as needed in the function body, by setting the properties expected by the client-side application.

In addition, as explained below, the client-side application is capable of retrieving additional configuration properties from the server-side, which allows to merge settings from multiple sources, without having to duplicate the global properties in multiple locations.

66.1.2 Modifying Default Configuration Properties

Table 66–1 describes the system-defined configuration properties and indicates which properties can be modified.

Table 66–1 Configuration Properties in `UI/Config/GlobalHtml`

Property Name	Description	Values and Examples
`maxTabCount`	Maximum number of tabs that can remain open simultaneously. A tab is the tab of an open asset.	Any integer greater than `0`. Ex: `config.maxTabCount = 30;`
`enableContextMenu`	Indicates whether the default browser context (right-click) menu should be enabled when users work in web mode.	`true | false` Ex: `config.enableContextMenu = true;`
`enableWebMode`	Indicates whether web mode should be enabled. When this property is set to `false`, users are able to work only with assets in form mode and use the preview functionality. By default, this property takes the value of the `xcelerate.enableinsite` property, found in `futuretense_xcel.ini`.	`true | false` **Ex:** `config.enableWebMode = true;`
`enableDatePreview`	Indicates whether date-based preview should be enabled. By default, this property takes the value of the `cs.sitepreview` property, found in `futuretense_xcel.ini`.	`true | false` **Ex:** `config.enableDatePreview = false;`
`enablePreview`	Indicates whether preview is allowed. By default, this property takes the value of the "Preview method" attribute in the Edit Site form (accessible from the Administrator interface: Select **Admin** tab, expand **Sites**, double-click ***SampleSite***, and select **Edit**).	`true | false` **Ex:** `config.enablePreview = true;`

Table 66–1 (Cont.) Configuration Properties in `UI/Config/GlobalHtml`

Property Name	Description	Values and Examples

`defaultView`	Defines the preferred view for working with assets (i.e., whether assets will be viewed, by default, in form mode or web mode).
	Note: An asset will be opened in web mode only if the asset is associated with a default template.
		The expected value is one of the following:
		- `"default": "form" | "web"`
		- `"`*`assetType`*`" : "form" | "web"`
		- `"`*`assetType/subtype`*`": "form" | "web"`
		where *`assetType`* is a valid asset type name, and *`subtype`* is a valid subtype or definition name.
		Ex:
		`config.defaultView = {`
		` "default": "form",`
		` "AVIArticle": "web",`
		` "Page/AVISection": "web"`
		`}`

`toolbars`	For each type of view, this property defines the list of available toolbar actions.	See Section 66.2, "Customizing the Toolbar."
`toolbarButtons`	Used to define the behavior of specific toolbar buttons.	See Section 66.2.2.3, "Customizing the Toolbar with Custom Actions."
`menubar`	Defines the list of available actions in the menu bar.	See Section 66.3, "Customizing the Menu Bar."
`documents`	Registers available implementations of documents.	Do not modify the value of this property. The only supported value is `asset`.
`views`	Registers view implementations.	Do not modify the value of this property.
`controllers`	Registers controller implementations and the set of actions supported by each controller.	Do not modify the value of this property.
`roles`	Contains the list of roles for the currently logged in user.	Do not modify the value of this property.
`supportedTypes`	Contains the list of asset types that can be edited from the Contributor interface.	Do not modify the value of this property.
`searchableTypes`	Contains the list of asset types that can be searched from the Contributor interface.	Do not modify the value of this property.
`token`	Used for security when uploading binary file.	Do not modify the value of this property.
`sessionid`	Used for security when uploading binary file.	Do not modify the value of this property.
`contextMenus`	Defines the list of available actions in the context menu.	See Section 66.4, "Customizing Context Menus."

66.1.3 Adding Custom Configuration Properties

In addition to retrieving the global properties, stored in `UI/Config/GlobalHtml`, the Contributor application will attempt to retrieve additional settings in `UI/Config/SiteConfig` and any element present in `UI/Config`. Depending on the requirement, this lets you set global properties, or site-specific properties, without having to replicate all the properties defined in UI/Config/GlobalHtml, but only the properties that actually change.

This section contains the following topics:

- Section 66.1.3.1, "Adding Custom Global Properties"
- Section 66.1.3.2, "Adding Site-Specific Properties"

66.1.3.1 Adding Custom Global Properties

Custom global properties are meant to be shared across all sites on a given content management system. The recommended approach consists of creating a custom configuration element defined as follows:

- The presentation element name must be UI/Config/SiteConfigHtml.
- The element code must follow the pattern shown in Section 66.1.1, "Overview of the Configuration Properties."

For example, you may want to:

- Override the value of maxTabCount for all sites.
- Override the default view for Page assets.
- And, define an additional custom property called foo.

To do this, you could create an element called CustomElements/UI/Config/SiteConfigHtml, containing the following code:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<cs:ftcs>
webcenter.sites['${param.namespace}'] = function (config) {
    // override existing properties
    config.maxTabCount = 60;
    config.defaultView.Page = "form";

    // add custom properties
    config.foo = "bar";

}
</cs:ftcs>
```

66.1.3.2 Adding Site-Specific Properties

In some cases, the Contributor interface must be configured differently for each content management site. The recommended approach consists of overriding the core controller element called UI/Config/SiteConfig by creating an element as follows:

CustomElements/*siteName*/UI/Config/SiteConfigHtml

where *siteName* is the name of the content management site (for instance, avisports).

For example, the avisports demo site enforces web mode as the default mode for assets of type Page and AVIArticle. This is done by defining the JSP element CustomElements/avisports/UI/Config/SiteConfigHtml, and providing the following settings:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<cs:ftcs>
webcenter.sites['${param.namespace}'] = function (config) {
    // default view modes for avisports
    config.defaultView.Page = "web";
    config.defaultView.AVIArticle = "web";
}
</cs:ftcs>
```

Loading of Configuration Elements

The global configuration element is always loaded first. Additional configuration elements are loaded in alphabetic order. For instance, using the examples above, configuration properties would be loaded in the following order:

1. `UI/Config/GlobalHtml`
2. `UI/Config/SiteConfigHtml`

Property Values

The value of some properties is, in some cases, an object. That is:

```
config.someProperty = {
  foo: "bar",
  x: 123
};
```

When partially overriding this property, it is important to distinguish between the following types of code:

```
config.someProperty = {
  x: 3456
};
```

vs.

```
config.someProperty.x = 3456;
```

In the first case, the property `foo` is overridden as "undefined", whereas in the second case, the original value of `foo` is preserved.

66.2 Customizing the Toolbar

The toolbar can be customized to list actions for operating on assets in web mode or form mode. The toolbar can be further customized per asset type and subtype.

This section contains the following topics:

- Section 66.2.1, "Overview of Toolbar Customization"
- Section 66.2.2, "Examples of Toolbar Customization"

66.2.1 Overview of Toolbar Customization

The global configuration element (`UI/Config/GlobalHtml`) describes for each type of view (such as web mode inspect, web mode edit, form mode edit, and form mode inspect), the list of actions to display in the toolbar to the user. This is done through the `toolbars` property. Its value is an object with the following syntax:

```
config.toolbars = {
    "viewAlias": [action_1, action_2,  …],
    or:
    "viewAlias": {
        "view_mode_1": [action_1, action_2,  …],
        "view_mode_2": [action_1, action_2,  …]
    }
  …
}
```

where:

viewAlias indicates for which type of view this toolbar must be used. The alias must match one of the view aliases defined in the `config.views` section.

action_i is an action name. For standard actions, such as `save` and `approve`, the action name is automatically mapped to a given icon, title, alternate text, and so on. For more information about standard actions, custom actions, or customizing the appearance of a custom button, see Section 66.2.2, "Examples of Toolbar Customization."

view_mode_i is one of the modes supported by the view (typically, `edit` or `view`).

66.2.2 Examples of Toolbar Customization

This section contains the following topics:

- Section 66.2.2.1, "Customizing the Toolbar with Standard Actions for Web Mode"
- Section 66.2.2.2, "Customizing the Toolbar with Standard Actions for Asset Type and Subtype"
- Section 66.2.2.3, "Customizing the Toolbar with Custom Actions"

66.2.2.1 Customizing the Toolbar with Standard Actions for Web Mode

The following configuration determines which toolbar actions are available in web mode for all asset types:

```
config.toolbars = {
    ( …)

    "web": {
      "edit": ["form-mode", "inspect", "separator", "save", "preview",
               "approve", "delete", "separator", "changelayout",
               "separator", "checkincheckout", "refresh"],
      "view": ["form-mode", "edit", "separator", "preview", "approve",
               "delete", "separator", "checkincheckout", "refresh"]

    ( …)

}
```

The above configuration defines two lists of actions (`edit` and `view`), corresponding to the asset's views: Edit and Inspect.

> **Note:** To find the set of standard actions, refer to the list of actions specified in the following properties under the `controllers` property:
>
> - `fw.ui.document.AssetDocument` (all actions supported by assets)
> - `fw.ui.controller.InsiteController` (all actions supported by the view controller).

66.2.2.2 Customizing the Toolbar with Standard Actions for Asset Type and Subtype

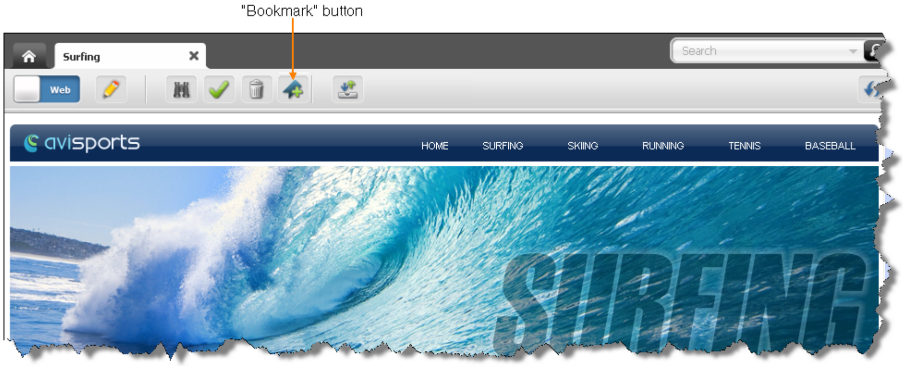
Each toolbar configuration can be customized by asset type and subtype by adding a property named:

- *viewAlias/assetType*
- *viewAlias/assetType/assetSubtype*

For example, we can add the bookmark/unbookmark buttons for Page assets in web mode. In a custom configuration element (such as `CustomElements/avisports/UI/Config/SiteConfigHtml`), we can add the following property:

```
config.toolbars["web/Page/AVISection"] = {
    "edit": config.toolbars.web.edit, // reuse default for edit mode
    "view": ["form-mode", "edit", "separator", "preview", "approve", "delete",
                    "bookmark", "unbookmark", "separator",
                    "checkincheckout",  "refresh"]
}
```

Inspecting the "Surfing" Page asset now shows the following toolbar:



Note: Keep in mind the following:

- We are customizing only the `view` mode. When a Page/AVISection asset is being edited in web mode, the standard toolbar will be shown.
- The "Bookmark" and "Unbookmark" buttons are not shown simultaneously, since they both depend on the asset's current state (whether it is already bookmarked or not).

66.2.2.3 Customizing the Toolbar with Custom Actions

Custom actions can be defined by adding new entries to the `config.toolbarButtons` property, as follows:

```
config.toolbarButtons.<customActionName> = {
  src: <path_to_icon>,
  onClick: <click_handler>
}
```

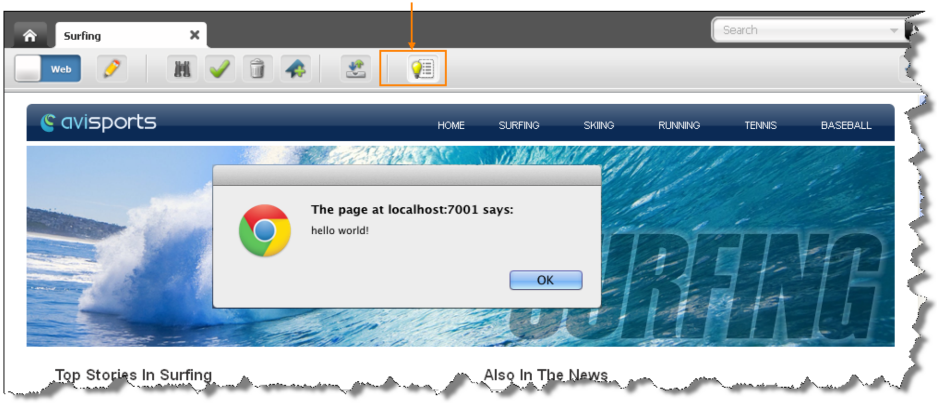
For example, let's define the following `helloWorld` custom action:

```
config.toolbarButtons.helloWorld = {
      src: 'js/fw/images/ui/ui/toolbarButton/smartlist.png',
      onClick: function () {
          alert('Hello World!!');
      }
}
```

The `helloWorld` action can now be referenced from a toolbar configuration as follows (we will reuse our example from the previous section):

```
config.toolbars["web/Page/AVISection"] = {
    "view":
          ["form-mode", "edit", "separator", "preview", "approve",
           "bookmark", "unbookmark", "separator",
           "checkincheckout", "separator", "helloWorld", "refresh"],

    "edit": config.toolbars.web.edit // reuse default web mode toolbar
}
```

Note: In this example, we have added a separator (a vertical line) and the custom button at the end of the toolbar:



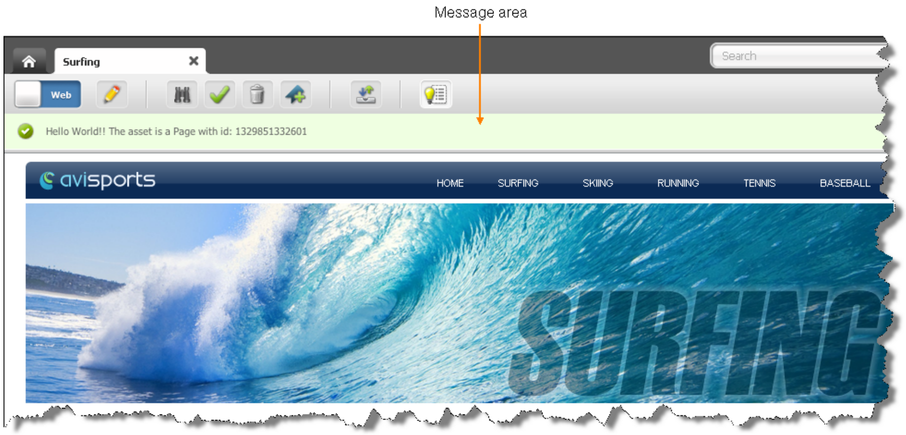
Separator and custom button

The page at localhost:7001 says:

hello world!

A more elaborate example would involve, for instance, retrieving the `id` and type of the current asset, and giving some feedback in the message area:

```
config.toolbarButtons.helloWorld = {
  src: 'js/fw/images/ui/ui/toolbarButton/smartlist.png',
  onClick: function () {
var doc = SitesApp.getActiveDocument(), // the document in the active tab
    asset = doc.get('asset'),      // the asset object
    view = SitesApp.getActiveView();     // the active view

view.info('Hello World!! The asset is a ' + asset.type + ' with id: '
+ asset.id);

  }
}
```



Hello World!! The asset is a Page with id: 1329851332601

66.3 Customizing the Menu Bar

The menu bar can be customized to support menus for operating on assets of a certain type. or type/subtype. Submenus can be actionable items, additional menus, or menu separators.

This section contains the following topics:

- Section 66.3.1, "Overview of Menu Bar Customization"
- Section 66.3.2, "Adding a Custom Action to the Menu Bar"

66.3.1 Overview of Menu Bar Customization

The menu bar configuration is defined by the `config.menubar` property:

```
config.menubar = {
    "key_i": [
          //menu_i
          {
              "id": "menu_id",
              "label": "menu_label",
              "children": [
                  //submenus
                  //- actionable menu item
                  {
                     label: 'menu_item_label',
                     action: 'action_name' | click_handler
                  },
                  //- deferred pop-up menu
                  {
                     label: 'menu_item_label',
                     deferred: 'controller_element',
                     cache: true|false
                  },
                  //- pop-up menu
                  {
                     label: 'menu_item_label',
                     children: [
                         // submenu_1
                         {
                            label: 'menu_item_label',
                            action: 'action_name' | click_handler
                         }'
```

```
                            // submenu_2
                            {
                               label: 'menu_item_label',
                               action: 'action_name' | click_handler
                            },
                             …
                             …
                             …
                        ]
                    },
                    //- menu item separator
                    {separator: true}

                //additional menu_i
                 …
                 …
                 …
                ]
            }
```

where:

key_i is one of the following:

- *`default`*: Defines the default menu bar.
- *`assetType`*: Defines the customized menu bar for all assets of type *`assetType`*.
- *`assetType/subtype`*: Defines the customized menu bar for all assets of type *`assetType`* and subtype *`subtype`*.

//menu_i starts a section that describes each top menu, where:

- *`menu_id`* is the identifier of the menu.
- *`menu_label`* is the display name of the menu.
- `submenus` can be any of the following:
  - An **actionable menu item** (clicking the menu item produces an action), where:
    - `label` specifies the display name of the menu item.
    - `action` can be any action supported by a controller, such as `edit` and `inspect`, or a custom click handler (see the customization example in Section 66.3.2, "Adding a Custom Action to the Menu Bar").

    > **Note:** When a given action is not supported by the current document/view, it appears disabled (greyed out) in the menu.

    For instance, a menu item triggering a `save` action is defined as follows:

    ```
    {
        label: "Save",
        action: "save"
    }
    ```

  - A **deferred pop-up menu** (the pop-up menu is determined dynamically by running a controller element on the server-side), where:
    - `label` specifies the display name of the menu item.

- `deferred` specifies a controller element name, such as `UI/Data/StartMenu/New`.
- `cache` is a boolean indicating whether the output of the controller element should be cached or not.

For instance, the New pop-up menu, which reads all available start menu items for the current site/user is defined as follows:

```
{
    label: New,
    deferred: "UI/Data/StartMenu/New",
    cache: true
}
```

- A **pop-up menu** (the child menu items are hard wired in the configuration itself).
- A **menu item separator** (a horizontal line), which is used to group menu entries together.

66.3.2 Adding a Custom Action to the Menu Bar

In this example, we want to add the `helloWorld` custom action defined in Section 66.2.2.3, "Customizing the Toolbar with Custom Actions" to the menu bar (to run the custom onClick handler). We can add this action by adding a new entry to the menu bar called "Custom Menu", with a single menu item called "Hello World", which will trigger the custom action. Our steps are the following:

1. First, we reuse the default menu bar, and add to it. The simplest way to do this is to make a copy of the original array:

```
config.menubar["Page/AVISection"] = config.menubar["default"].slice(0);
```

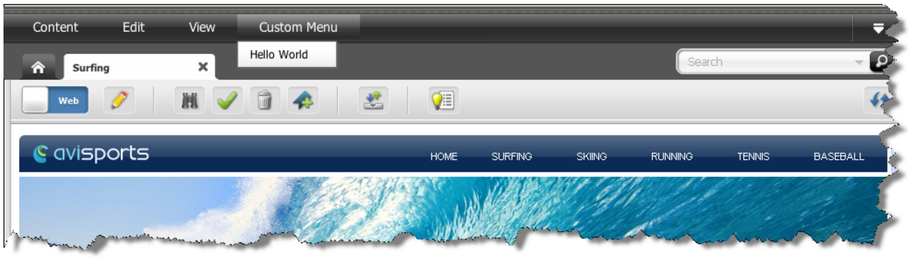
2. We can then add our menu as follows:

```
config.menubar["Page/AVISection"].push(
    "id": "myCustomMenu",
    "label": "Custom Menu",
    "children": [
        // Children go here
    ]
);
```

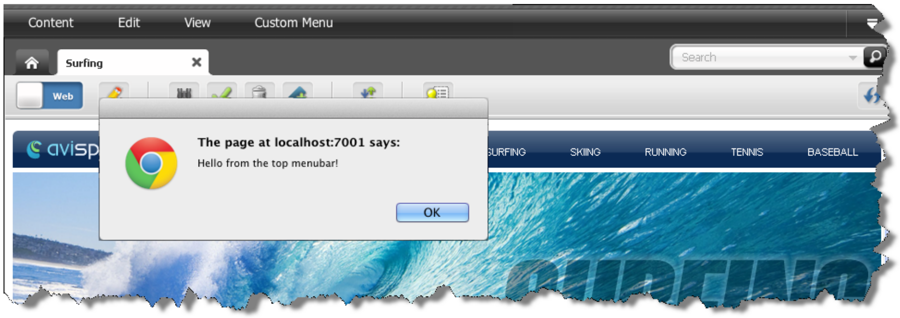
3. Finally, we define the child menu items:

```
config.menubar["Page/AVISection"].push({
    "id": "myCustomMenu",
    "label": "Custom Menu",
    "children": [{
      "label": "Hello World",
      "action": function () {
         alert("Hello from the top menubar!");
      }
    }]
});
```

The **Custom Menu** can now be seen whenever an "AVISection" Page section is viewed:



Selecting the **Hello World** menu item should run the custom onClick handler:



The page at localhost:7001 says:

Hello from the top menubar!

4. To run the exact same code, whether clicked from the menu bar or toolbar, we could write the following:

```
// define the helloWorld code once
config.myActions = {
    hello: function (args) {
      var doc = SitesApp.getActiveDocument(),
        asset = doc.get('asset'),
        view = SitesApp.getActiveView();

      view.info('Hello World!! The asset is a ' + asset.type + ' with id:
            + asset.id);
    }
};

// attach it to the helloWorld button
config.toolbarButtons['helloworld'] = {
    src: 'js/fw/images/ui/ui/toolbarButton/smartlist.png',
    onClick: config.myActions.hello
};

config.toolbars["web/Page/AVISection"] = {
    "edit": config.toolbars.web.edit, // reuse default for edit mode
    "view": [  "form-mode", "edit", "separator", "preview", "approve",
    "delete", "bookmark", "unbookmark", "separator",
    "checkincheckout","separator","helloworld", "refresh"]
}

// attach it to the menubar, under "Custom Menu">"Hello World"
config.menubar['Page/AVISection'] = config.menubar['default'].slice(0);

config.menubar["Page/AVISection"].push({
    "id": "myCustomMenu",
    "label": "Custom Menu",
    "children": [{
        "label": "Hello World",
        "action": config.myActions.hello
        }]
    });
```

66.4 Customizing Context Menus

The context menus are opened with a right-click and allow for a menu that allows work with the immediate object that was right-clicked. The specific items available in a context menu, as well as the order the objects are displayed, can be customized.

The context menus are defined by the `config.contextMenus` property:

```
config.contextMenus = {
   "default": ["action_1", "action_2", … "action_n"]
   "asset": ["action_1", "action_2", … "action_n"],
   "asset/assetType": ["action_1", "action_2", … "action_n"],
}
```

Where the values are defined as:

- `default`: Defines the default list of context menus. Each menu item displayed in the list is an `action`.
- `asset`: Defines the customized list of context menus for all assets. Each menu item displayed in the list is an `action`.
- `asset/assetType`: Defines the customized list of context menu items for the assets of type `assetType`. Each menu item displayed in the list is an `action`.

An example context menu configuration for the asset type Page would be:

```
"asset/Page":["edit", "copy", "preview", "delete", "bookmark", "tagasset"]
```

67 Contributor Interface: Customizing Asset Forms

This chapter discusses customizations that you can apply to asset forms in the WebCenter Sites Contributor interface. It also provides a step-by-step example for building a single-valued and multi-valued attribute editor for some of the supported data types.

This chapter contains the following topics:

67.1 Overview of Asset Forms Customization

You can perform two types of customizations on asset forms in the Contributor interface. One, you can modify the header of an asset form. The other is, you can either customize an existing attribute editor, as explained in Section 18.3, "Customizing Attribute Editors," or you can build a custom attribute editor for the data types supported by WebCenter Sites, as described in Section 67.3, "Building an Attribute Editor."

> **Note:** Unlike other components of the Contributor interface, asset forms are not in the Contributor framework. Therefore, requests for asset forms are not processed by the UI Controller.

67.2 Modifying the Header of Asset Forms

You can modify the header of any asset form by creating a custom assettype-specific element in the `OpenMarket/Xcelerate/AssetType/<AssetTypeName>/` directory. You can include additional stylesheets or JavaScript code instead of modifying the body of the HTML pages. The name of the element must be `Header`.

67.3 Building an Attribute Editor

You can customize the look and feel of some of the existing ready-to-use attribute editors, as described in Section 18.3, "Customizing Attribute Editors." You can also create a custom attribute editor for the data types supported in WebCenter Sites.

This section describes how to build a custom attribute editor that supports a single value of data type `text`, `string`, `integer`, or `money`. This section also provides pointers and sample code for implementing a multi-valued attribute editor for the same data types.

> **Note:** If you want to create a custom attribute editor for the `blob` or `asset` data type, you can base your implementation on the `UPLOADER` attribute editor for the `blob` type and the `PICKASSET` attribute editor for the `asset` data type.

Steps for building an attribute editor are the following:

- Section 67.3.1, "Creating a Dojo Widget and its Template"
- Section 67.3.2, "Defining the Attribute Editor as a Presentation Object"
- Section 67.3.3, "Creating the Attribute Editor Element"
- Section 67.3.4, "Creating the Attribute Editor"
- Section 67.3.5, "Implementing a Multi-Valued Attribute Editor"

67.3.1 Creating a Dojo Widget and its Template

This section describes how to create a Dojo widget to handle a single value of data type `text`, `string`, `integer`, or money.

This section contains the following topics:

- Section 67.3.1.1, "Create a Template for the Dojo Widget"
- Section 67.3.1.2, "Creating a Dojo Widget"

67.3.1.1 Create a Template for the Dojo Widget

To create an HTML template for the Dojo widget:

1. In your WebCenter Sites installation directory, navigate to the `<context_root>/js/` directory.
2. Create a new directory structure under the `js` directory as follows: `extensions/dijit/templates`.
3. In the `<context_root>/js/extensions/dijit/templates` directory, create an HTML template file and give it a meaningful name. For example: `MyWidget.html`
4. In the HTML template file, define the look and feel of the new Dojo widget. The content of this HTML template would look similar to this:

```
<div>
   <div>
       <input type="text" dojoAttachPoint='inputNode'
        name='${name}'size='60' class='valueInputNode'>
       </input>
   </div>
</div>
```

   If you use the above code for the template, then the input node will take the input from the end user and the value of the input node will be maintained in the Dojo widget which you will create in Section 67.3.1.2, "Creating a Dojo Widget."

5. Save your template file.
6. Continue to Section 67.3.1.2, "Creating a Dojo Widget."

67.3.1.2 Creating a Dojo Widget

To create a Dojo widget:

1. Navigate to the `js/extensions/dijit` directory of the WebCenter Sites installation.
2. Create a `dojo` widget, for example, `MyWidget.js` (see Example 67–1) by implementing the following mandatory functions:
   - `_setValueAttr`: This `setter` method sets the value of the attribute.
   - `_getValueAttr`: This `getter` method gets the attribute value.
   - `isValid`: This method runs validations to see if the given value is valid or not.
   - `focus`: This sets the focus on the attribute editor.
   - `onChange`: This method is called whenever the user updates the value of the attribute.
   - `onBlur`: This method updates the widget when the attribute value is entered by the user. An update will be triggered when the user selects another field.

Example 67–1 Sample Code for a Dojo Widget

```
dojo.provide('extensions.dijit.MyWidget');
dojo.require('dijit._Widget');
dojo.require('dijit._Templated');
dojo.declare('extensions.dijit.MyWidget', [dijit._Widget, dijit._Templated], {
  //string.
  //The value of the attribute.
  value: '',
  //int
  //The Attribute editor's MAXALLOWEDCHARS should be assigned to this variable.
  //maxAllowedLength: 15,
  //string
  //   This variable is required only for single valued instance.
  //   The server should recieve information from input element with this name.
  name: '',
  //HTMLElement
  //   This stores the cached template of the widget's representation.
  templateString: dojo.cache('extensions.dijit', 'templates/MyWidget.html'),
  //string
  //   This class will be applied to the top div of widget.
  //   It will help in managing css well.
  baseClass: 'MyWidget',
  postCreate: function() {
      var self = this;
      // Do not allow typing characters more than allowed length.
         dojo.connect(this.inputNode, 'onkeypress', function(e) {
         if (this.value.length >= self.maxAllowedLength && e.keyCode !=
             dojo.keys.BACKSPACE)
         e.preventDefault();
      });
},
// Start -  Mandatory functions
_setValueAttr: function(value) {
    //  summary:
    //     Set the value to 'value' attribute and input node
    if (value === undefined || !this._isValid(value)) return;
    this.value = value;
    this._setInputNode(value);
},
_getValueAttr: function() {
   // summary:
   //Get the latest value and return it.
   return this.value;
},
_isValid: function(newVal) {
   //summary:
   //Verify if the given value is as per the expectation or not.
   if (newVal.length > this.maxAllowedLength) {
   return false;
}
   return true;
},
   focus: function() {
   //summary:
   //Set the focus to the representation node i.e. input node here.
   if (typeof this.inputNode.focus === 'function')
   this.inputNode.focus();
},
onBlur: function() {
   //summary:
   //Custom selected browser event when the value should be updated
   //Any activity which leads to value change should update
   //the widget value as well.
   this.updateValue();
},
_onChange: function(newValue) {
   //summary:
   //Internal onChange method
   this.onChange(newValue);
},
onChange: function(newValue) {
   //summary:
   //A public hook for onChange.
},
   // End -  Mandatory functions
   // Extra functions used in Mandatory functions
_setInputNode: function(value) {
  //summary:
  //Sets the value to input node.
  this.inputNode.value = value;
},
updateValue: function() {
  //summary:
  //Validate the newly entered value and if it is successful
  //then update widget's value.
  var newVal = this.inputNode.value;
  if (!this._isValid(newVal)) return;
  if (this.value != newVal)
  this._onChange(newVal);
  this.set('value', newVal);
}
});
```

Note: For information about creating Dojo widgets, see the Dojo documentation at `http://dojotoolkit.org/`.

3. In the `js/extensions/themes/`directory, create a CSS (for example, `MyWidget.css`) for this widget. You can use the following code in the CSS file, or write your own code:

   ```
   .fw .MyWidget .valueInputNode {
   color: blue;
   }
   ```

4. In the `js/extensions/themes/`directory, update the `UI.css` with an import statement for the Dojo widget's CSS. For example, `@import url("MyWidget.css");`
5. Save your work.
6. Continue to Section 67.3.2, "Defining the Attribute Editor as a Presentation Object."

67.3.2 Defining the Attribute Editor as a Presentation Object

This section describes how to define input tags (presentation objects) for flex attributes. It also describes how to assign arguments that the input tags can pass from the attribute editor to the display elements.

To define the attribute editor:

1. In your WebCenter Sites installation, navigate to the `Sites\11gR1\Sites\11.1.1.6.1\presentationobject.dtd` file.
2. In the `presentationobject.dtd` file, do the following;
   a. Add a new tag (presentation object) to the list in the `<!ELEMENT PRESENTATIONOBJECT …>` statement. In this example, the new tag is named `MYATTREDITOR`.

      In the following line, `MYATTREDITOR` is your custom attribute editor whose name matches the name of the element you will create in Section 67.3.3, "Creating the Attribute Editor Element." All other tags are ready-to-use attribute editors.

      ```
      <!ELEMENT PRESENTATIONOBJECT (TEXTFIELD | TEXTAREA | PULLDOWN |
      RADIOBUTTONS | CHECKBOXES | PICKFROMTREE | EWEBEDITPRO | REMEMBER |
      PICKASSET | FIELDCOPIER | DATEPICKER | IMAGEPICKER | REALOBJECT |
      CKEDITOR | DATEPICKER | IMAGEPICKER | REALOBJECT | CKEDITOR | FCKEDITOR |
      UPLOAD | MAGEEDITOR | RENDERFLASH | PICKORDERASSET | TYPEAHEAD | UPLOADER |
      MYATTREDITOR)>
      ```

   b. Add an `<!ELEMENT … >` section that defines the new tag (presentation object) and the arguments it takes. This new tag includes elements that supply the logic behind the format and behavior of the attribute when it is displayed on a form. Ensure that `MAXALLOWEDCHARS` is marked as a required attribute.

      ```
      <!ELEMENT MYATTREDITOR ANY>
      <!ATTLIST MYATTREDITOR MAXALLOWEDCHARS CDATA #REQUIRED>
      <!ATTLIST MYATTREDITOR MAXVALUES CDATA #IMPLIED>
      ```

   c. Save and close the `presentationobject.dtd` file.
3. Continue to Section 67.3.3, "Creating the Attribute Editor Element."

67.3.3 Creating the Attribute Editor Element

This section describes how to create an element that displays an "edit" view of an attribute (single-valued) when it appears in a "New" or "Edit" form. This element must be located in the OpenMarket/Gator/AttributeTypes directory in the ElementCatalog table. The element name must exactly match the name of the tag you defined in Section 67.3.2, "Defining the Attribute Editor as a Presentation Object," so that it can be invoked by the tag (in this example, MYATTREDITOR).

1. Navigate to the OpenMarket/Gator/AttributeTypes directory in your ElementCatalog.
2. Create an attribute element for your new editor (in this example, MYATTREDITOR.jsp. See Example 67–2.) Ensure that the name of this element matches the tag name you defined in the presentationobject.dtd file (Section 67.3.2, "Defining the Attribute Editor as a Presentation Object").
3. To prevent the default rendering of the attribute editor, set the doDefaultDisplay variable to no.
4. To display the attribute name, call the element OpenMarket/Gator/FlexibleAssets/Common/DisplayAttributeName. The code is:

   ```
   <ics:callelement
     element="OpenMarket/Gator/FlexibleAssets/Common/DisplayAttributeName"/>
   ```

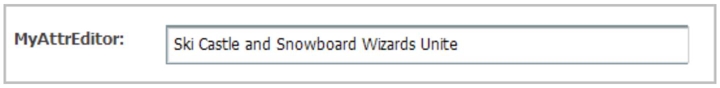
5. To render the widget, call the element OpenMarket/Gator/AttributeTypes/CustomTextAttributeEditor by using the following parameters:

   - editorName: Name of the widget created in Section 67.3.1.2, "Creating a Dojo Widget." In this example it is extensions.dijit.MyWidget.
   - editorParams: This argument passes the JSON string of parameters to the widget. In this example, it passes the maxAllowedLength value. For example, the value can look like this: { maxAllowedLength: "10" }
   - maximumValues: Required only for a multi-valued widget. This is the maximum number of values allowed to be rendered in a multi-valued widget.

     For a single-valued widget, the complete code with the initialization parameters and formatting styles should look like the code in Chapter 67–2, "Sample Single-Valued Attribute Editor (MYATTREDITOR)." To implement a multi-valued widget, see Section 67.3.5, "Implementing a Multi-Valued Attribute Editor."

   If you use the code given in Example 67–2, then the single-valued attribute editor would look like the editor in Figure 67–1.

Figure 67–1 Single-Valued Attribute Editor



MyAttrEditor: Ski Castle and Snowboard Wizards Unite

Note: In Example 67–2, the following core logic is implemented to render the single-valued attribute using the new attribute editor:

```
<ics:if condition='<%= "no".equals(ics.GetVar("MultiValueEntry"))
%>'>
<ics:then>
     <div dojoType='<%= ics.GetVar("editorName") %>'
     name='<%= ics.GetVar("cs_SingleInputName") %>'
     value='<%= attributeValue %>'
     >
     </div>
</ics:then>
```

The name that is coded in the element must be `ics.GetVar("cs_SingleInputName")` to ensure that the input node in the Dojo template will have the same name. The input node value will be sent to the server for saving the attribute.

Example 67–2 Sample Single-Valued Attribute Editor (MYATTREDITOR)

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
<%//
// OpenMarket/Gator/AttributeTypes/MYATTREDITOR
//
// INPUT
//
// OUTPUT
//%>
<%@ page import="COM.FutureTense.Interfaces.FTValList" %>
<%@ page import="COM.FutureTense.Interfaces.ICS" %>
<%@ page import="COM.FutureTense.Interfaces.IList" %>
<%@ page import="COM.FutureTense.Interfaces.Utilities" %>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<cs:ftcs>
<ics:setvar name="doDefaultDisplay" value="no" />
<script>
     dojo.require('extensions.dijit.MyWidget');
</script>
<link href="<%=ics.GetVar("cs_imagedir")%>/../js/extensions/themes/MyWidget.css"
     rel="stylesheet" type="text/css"/>
<%
FTValList args = new FTValList();
args.setValString("NAME", ics.GetVar("PresInst"));
args.setValString("ATTRIBUTE", "MAXALLOWEDCHARS");
args.setValString("VARNAME", "MAXALLOWEDCHARS");
ics.runTag("presentation.getprimaryattributevalue", args);
args.setValString("NAME", ics.GetVar("PresInst"));
args.setValString("ATTRIBUTE", "MAXVALUES");
args.setValString("VARNAME", "MAXVALUES");
ics.runTag("presentation.getprimaryattributevalue", args);
String maximumValues = ics.GetVar("MAXVALUES");
maximumValues = null == maximumValues ? "-1" : maximumValues;
String editorParams = "{ maxAllowedLength: "
     + ics.GetVar("MAXALLOWEDCHARS") + " }";
%>
```

```
<tr>
<ics:callelement
  element="OpenMarket/Gator/FlexibleAssets/Common/DisplayAttributeName"/>
    <td></td>
    <td>
    <ics:callelement
      element="OpenMarket/Gator/AttributeTypes/CustomTextAttributeEditor">
         <ics:argument name="editorName" value="extensions.dijit.MyWidget" />
         <ics:argument name="editorParams" value='<%= editorParams %>' />
         <ics:argument name="maximumValues" value="<%= maximumValues %>" />
</ics:callelement>
    </td>
 </tr>
</cs:ftcs>
```

6. Continue to Section 67.3.4, "Creating the Attribute Editor."

67.3.4 Creating the Attribute Editor

This section describes how to create an attribute editor asset to make it available to content contributors on their content management sites. This asset will support the input types you defined in Section 67.3.3, "Creating the Attribute Editor Element," for example, check boxes, radio options, and drop-down lists. The developer selects this editor when creating or modifying the attribute.

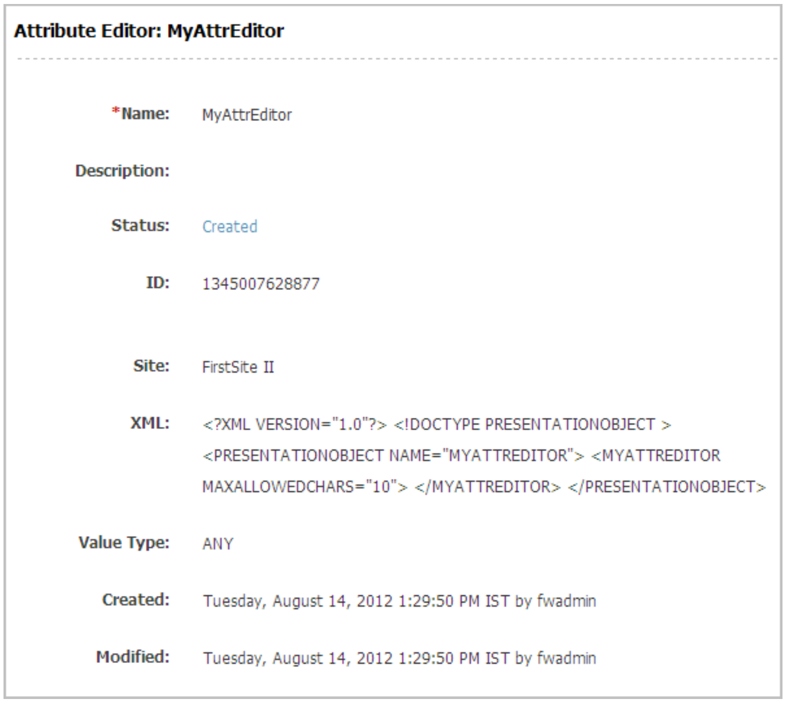
1. Log in to the Admin interface of your site.
2. On the **New** page, under the **Name** column, click **New Attribute Editor**.
3. In the **Name** field, enter a meaningful name for your editor. For example, `MyAttrEditor`.
4. In the **XML** box, enter the XML code for your attribute editor. Ensure that the name of the attribute editor in this code is exactly the same as the element name. For example:

   ```
   <?XML VERSION="1.0"?>
   <!DOCTYPE PRESENTATIONOBJECT >
   <PRESENTATIONOBJECT NAME="MYATTREDITOR">
   <MYATTREDITOR MAXALLOWEDCHARS="10"> </MYATTREDITOR> </PRESENTATIONOBJECT>
   ```

5. In the **Attribute Type** box, accept the appropriate value(s).
6. Click the **Save** icon.

   The attribute editor similar to the editor in Figure 67–2 is created for your site.

Figure 67–2 Sample Attribute Editor for a Site



Attribute Editor: MyAttrEditor

***Name:**	MyAttrEditor
Description:	
Status:	Created
ID:	1345007628877
Site:	FirstSite II
XML:	<?XML VERSION="1.0"?> <!DOCTYPE PRESENTATIONOBJECT > <PRESENTATIONOBJECT NAME="MYATTREDITOR"> <MYATTREDITOR MAXALLOWEDCHARS="10"> </MYATTREDITOR> </PRESENTATIONOBJECT>
Value Type:	ANY
Created:	Tuesday, August 14, 2012 1:29:50 PM IST by fwadmin
Modified:	Tuesday, August 14, 2012 1:29:50 PM IST by fwadmin

7. Continue to Section 67.3.5, "Implementing a Multi-Valued Attribute Editor."

67.3.5 Implementing a Multi-Valued Attribute Editor

In Section 67.3.3, "Creating the Attribute Editor Element," Example 67–2 shows the implementation for a single-valued attribute editor. To implement a multi-valued attribute editor for `text`, `integer`, `string`, or `money` data types, you can write code similar to the code in Example 67–3.

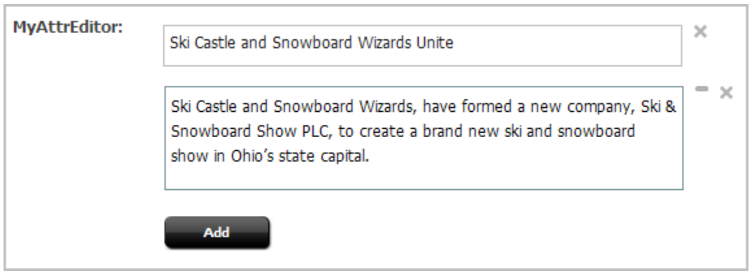
Example 67–3 Sample Multi-Valued Attribute Editor (CustomTextAttributeEditor)

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
<%//
// OpenMarket/Gator/AttributeTypes/CustomTextAttributeEditor
//
// INPUT
//
// OUTPUT
//%>
<%@ page import="COM.FutureTense.Interfaces.FTValList" %>
<%@ page import="COM.FutureTense.Interfaces.ICS" %>
<%@ page import="COM.FutureTense.Interfaces.IList" %>
<%@ page import="COM.FutureTense.Interfaces.Utilities" %>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<cs:ftcs>
<%
IList attributeValueList = ics.GetList("AttrValueList", false);
boolean hasValues = null != attributeValueList && attributeValueList.hasData();
String attributeValue = hasValues ? attributeValueList.getValue("value") : "";
%>
```

```
<ics:if condition='<%= "no".equals(ics.GetVar("MultiValueEntry")) %>'>
<ics:then>
       <div dojoType='<%= ics.GetVar("editorName") %>'
            name='<%= ics.GetVar("cs_SingleInputName") %>'
            value='<%= attributeValue %>'
       >
       </div>
</ics:then>
<ics:else>
<ics:callelement
  element="OpenMarket/Gator/AttributeTypes/RenderMultiValuedTextEditor">
    <ics:argument name="editorName"
      value='<%= ics.GetVar("editorName") %>' />
    <ics:argument name="editorParams"
      value='<%= ics.GetVar("editorParams") %>' />
    <ics:argument name="multiple"
      value="true" />
    <ics:argument name="maximumValues"
      value='<%= ics.GetVar("maximumValues") %>' />
</ics:callelement>
</ics:else>
</ics:if>
</cs:ftcs>
```

If the code in Example 67–3 is used, then the multi-valued attribute editor will look similar to the editor in Figure 67–3.

Figure 67–3 Multi-Valued Attribute Editor



MyAttrEditor:

Ski Castle and Snowboard Wizards Unite

Ski Castle and Snowboard Wizards, have formed a new company, Ski & Snowboard Show PLC, to create a brand new ski and snowboard show in Ohio's state capital.

Note the following points about Example 67–3:

- You can instantiate a multi-valued widget, which uses a single-valued widget to render multi-valued representations.
- The `MultiValueEntry` variable with the `no` value indicates that the attribute editor renders a single value. Changing the variable value to `yes` will enable the attribute editor to render multiple values.
- You can implement a multi-valued widget that accepts values in the JSON object or in any other format.
- For a multi-valued attribute editor, the `RenderMultiValuedTextEditor` element creates hidden input nodes required for `Save` logic. The value of each node is sent to the server.
- The multi-valued widget is rendered by calling the `OpenMarket/Gator/AttributeTypes/RenderMultiValuedTextEditor` element using the following code in Example 67–3:

  ```
  <ics:else>
    <ics:callelement
  ```

```
    element="OpenMarket/Gator/AttributeTypes/RenderMultiValuedTextEditor">
          <ics:argument name="editorName"
           value='<%= ics.GetVar("editorName") %>' />
          <ics:argument name="editorParams"
           value='<%= ics.GetVar("editorParams") %>' />
          <ics:argument name="multiple"
           value="true" />
          <ics:argument name="maximumValues"
            value='<%= ics.GetVar("maximumValues") %>' />
     </ics:callelement>
</ics:else>
</ics:if>
```

If you want to create a custom attribute editor for the `blob` or `asset` data type, you can base your implementation on the `UPLOADER` attribute editor for the `blob` type and the `PICKASSET` attribute editor for the `asset` data type.

Part IV

Configuring Oracle WebCenter Sites: Mobility

This part provides information on configuring Oracle WebCenter Sites: Mobility

This part contains the following chapter:

68

Configuring Oracle WebCenter Sites: Mobility to Support Mobile Websites

Oracle WebCenter Sites: Mobility, also called Mobility, enables its users to create, preview, and deliver websites to a variety of mobile devices such as phones and tablets.

Implementing Mobility to support mobile-optimized websites requires configuring its framework. This chapter first presents the key concepts and outlines major features of the framework. Configuration procedures follow.

This chapter includes the following sections:

68.1 Prerequisites for Developers

Configuring the Mobility framework requires you to have experience with core WebCenter Sites features, including site plans and templates. Also required is an understanding of mobile devices and their user agents.

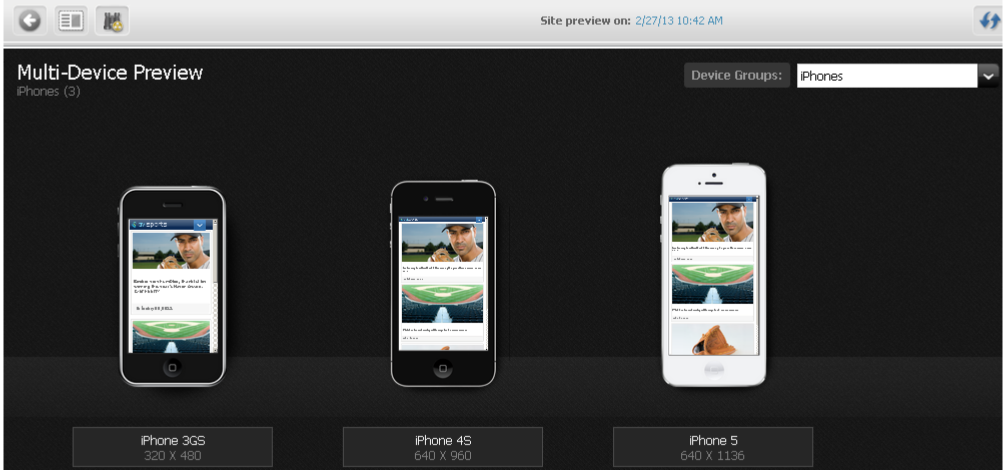
68.2 Key Mobility Concepts

WebCenter Sites uses a built-in device detection mechanism to identify the device that requests website content. Once the device is identified, WebCenter Sites finds the appropriate site plan (site navigation) and invokes the correct template to render the website.

The mechanics of this device detection process involve new features, such as device repository, device groups and suffixes, device assets, template variants, and site plans (which have been extended to support mobile websites). This section describes the concepts behind the new features. Later sections provide procedures for configuring the features to support mobile websites.

Once the features are configured, it is possible in the Contributor interface to create, preview, and deliver mobile sites, such as the site shown in Figure 68–1. In this figure, the Home page of the avisports sample site is displayed in the context of three devices (multi-device preview).

Figure 68–1 Preview of Avisports Site Home Page in the Context of Multiple Devices in the Contributor Interface



This section contains the following topics:

- Section 68.2.1, "Device Repository"
- Section 68.2.2, "Device Groups and Suffixes"
- Section 68.2.3, "Device Assets"
- Section 68.2.4, "Site Plans"
- Section 68.2.5, "Mobile Templates"

68.2.1 Device Repository

The device repository is a file that WebCenter Sites uses to detect mobile devices. The device repository contains the properties of devices and uniquely identifies each device based on its user agent. Device detection is accomplished by WebCenter Sites matching the user agent of the device to the user agent that is specified in the device repository.

Note: A user agent is a software agent that acts on behalf of users. The format of a user-agent string is a list of product tokens (keywords) with optional comments. Most browsers specify the following format:

```
Mozilla[version] (system and browser information]) [ platform]
([platform details]) [extensions]
```

For example:

```
Mozilla/so(iPad;u;CPUOS 3.2.1 like Mac
OSx;en-us)ppleWebkit/531.21.10(KHTML,like Gecko) Mobile/7B405
```

The two types of device repositories are:

- `devices.xml`: This is the default repository included with WebCenter Sites. This repository is updated at the time of a product release and includes only popular devices. You can add more data to this repository as and when required.

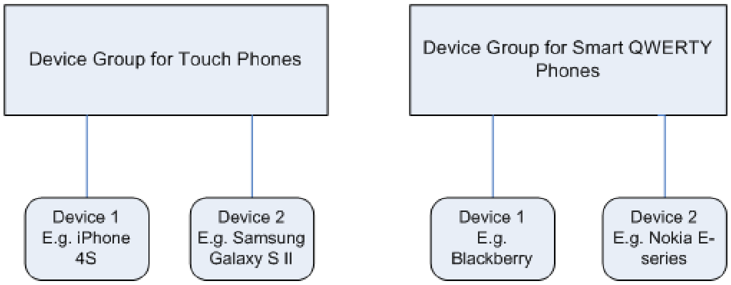
- WURFL: This is a third-party device information database from ScientiaMobile. WURFL is much more comprehensive than `devices.xml`, and it is updated regularly by ScientiaMobile. We recommend using WURFL to ensure you have the latest devices. A licensed copy can be obtained from ScientiaMobile. It is not included with WebCenter Sites.

For procedures on using the desired device repository, see Section 68.4.2, "Step 2: Setting Up the Device Repository."

68.2.2 Device Groups and Suffixes

Device Groups enable features-based grouping of devices. A device group is an asset that defines a collection of devices with common characteristics, for which a common website can be delivered. For instance, all Touch phones could belong in one group, whereas NonTouch (QWERTY) phones could be in a separate group, as illustrated in Figure 68–2.

Figure 68–2 Device Type-Based Grouping


Device Group for Touch Phones
Device Group for Smart QWERTY Phones
Device 1 E.g. iPhone 4S
Device 2 E.g. Samsung Galaxy S II
Device 1 E.g. Blackberry
Device 2 E.g. Nokia E-series

One website can be delivered to all the Touch devices. Another website can be delivered to all the NonTouch (QWERTY) devices. Therefore, two sets of templates must be coded: one set for the group of Touch devices, and another set for the group of NonTouch devices.

This is where we introduce the concept of suffixes. Suffixes are used to associate templates to the correct device groups. The association is made when the template and device group have the same suffix, such as `_Touch` (or `_NonTouch`). For example, if the base template `HomeLayout` is used to render the desktop website, you would create template variants, that is, templates named `HomeLayout_`*`suffix`*. In this example, you would create the `HomeLayout_Touch` template to render content on devices in the Touch device group; you would also create the `HomeLayout_NonTouch` template to render content on devices in the NonTouch device group.

At runtime, WebCenter Sites will match the suffixes of the device groups to the names of the template variants so that if `HomeLayout` is requested from a Touch device, WebCenter Sites will use the `HomeLayout_Touch` template to render the website. This is part of device detection.

> **Note:** A template without a suffix is called the *base template*. Templates with suffixes are called *variants of the base template*. The base template must exist before its variants can be created.

Suffixes are also used to associate device groups to site plans, which enables you to implement different website navigation for devices in different device groups. Templates are coded with the `device:siteplan` tag to specify website navigation on the delivery system.

To create a device group, you will create an asset of type **DeviceGroup**. In the process, you will specify:

- A suffix.
- Either the registered device name(s), or criteria that WebCenter Sites will use to associate devices to the group you are creating.

> **Note:** If you create multiple device groups, you will have to prioritize them for proper matching of devices to device groups during device detection.

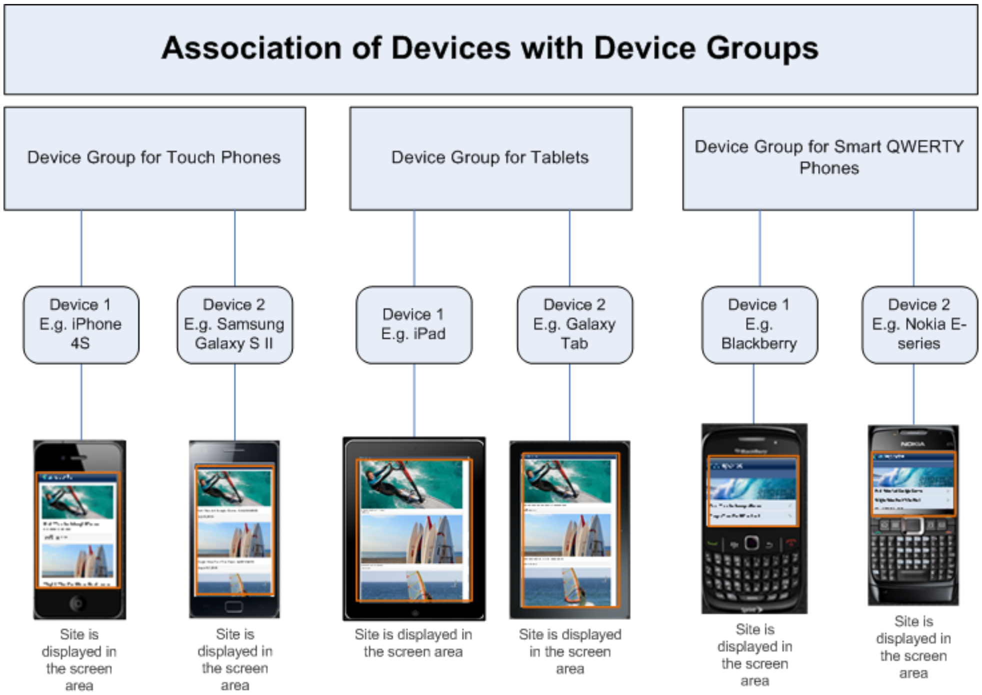
For procedures on configuring device groups, see Section 68.4.3, "Step 3: Configuring Device Groups." For procedures on prioritizing device groups, see Section 68.4.4, "Step 4: Prioritizing Device Groups."

68.2.3 Device Assets

Any mobile device can be represented by a device asset, which enables previewing of mobile website content in the context of the mobile device, in the Contributor interface. Device assets are used strictly to support preview. Even when devices are similar enough to be gathered in the same device group, they display variations (for example, screen size). Previewing in the context of a device asset enables content contributors to ensure that content will be rendered properly on the corresponding real device.

A device asset consists of an image and a user agent. The user agent is used to associate the device asset to (1) a real device in the device repository and (2) to a device group with matching criteria. For an example of associations illustrating the relationship between device groups and device assets, see Figure 68–3.

Figure 68–3 Devices Associated with Device Groups


Association of Devices with Device Groups
Device Group for Touch Phones
Device Group for Tablets
Device Group for Smart QWERTY Phones
Device 1 E.g. iPhone 4S
Device 2 E.g. Samsung Galaxy S II
Device 1 E.g. iPad
Device 2 E.g. Galaxy Tab
Device 1 E.g. Blackberry
Device 2 E.g. Nokia E-series
Site is displayed in the screen area
Site is displayed in the screen area
Site is displayed in the screen area
Site is displayed in the screen area
Site is displayed in the screen area
Site is displayed in the screen area

Device assets are associated to device groups by device detection. If a device asset matches the criteria specified in two or more device groups, the device asset is automatically associated to your preferred (highest-priority) device group. If a device fails to match criteria in all device groups, it is automatically assigned to the Default device group (used for serving websites to desktop browsers). Once a device is associated to a device group, templates associated with that device group can render website content in preview mode. The content is superimposed on the device image.

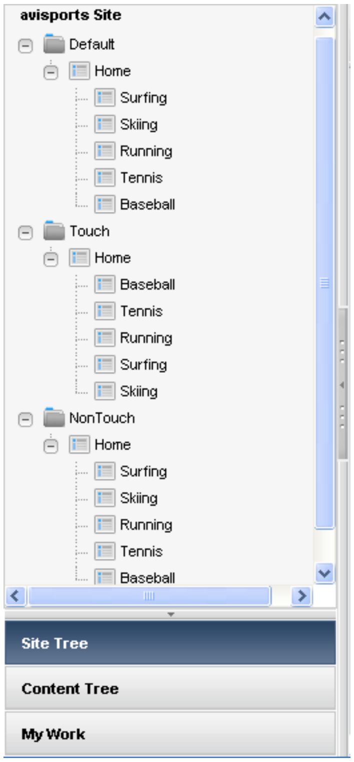
> **Note:** Ensure that content contributors are aware of the following preview behavior:
>
> - While preview of web pages in mobile devices works in all browsers, some features will not be displayed correctly if there is a mismatch between the browser and the user agent. For example, if the Contributor interface is opened in Internet Explorer, whereas the user agent is for FireFox on Android, the browser will use the Internet Explorer engine to render HTML. Therefore, previews in the Contributor interface are likely to differ from the look of the real pages on the real mobile device.
> - Preview renders pages as they would be displayed in full-screen mode on real devices.

For procedures about creating devices, see Section 68.4.5, "Step 5: Creating Device Assets."

68.2.4 Site Plans

A site plan defines the navigational hierarchy of a website. For example, Figure 68–4 shows the following site plans for the avisports sample site in the Contributor interface: **Default**, **Touch**, and **NonTouch**.

Figure 68–4 Site Plans in the Contributor Interface



avisports Site

- Default
  - Home
    - Surfing
    - Skiing
    - Running
    - Tennis
    - Baseball
- Touch
  - Home
    - Baseball
    - Tennis
    - Running
    - Surfing
    - Skiing
- NonTouch
  - Home
    - Surfing
    - Skiing
    - Running
    - Tennis
    - Baseball

When creating a site plan, you will associate device groups to that site plan by selecting a shared suffix. Site navigation defined by that site plan can then be served to devices in those device groups. (Once selected, the suffix is no longer available for other site plans.)

You have two basic approaches to creating site plans:

- Create a single site plan for all devices across all device groups.
- Create multiple site plans. Different site plans for different device groups allows for different navigation. The content can be re-used across site plans, or it can be different across site plans.

For procedure on configuring site plans, see Section 68.4.6, "Step 6: Creating Site Plans."

68.2.5 Mobile Templates

You have two basic approaches for creating templates that render mobile websites:

- Create a single set of templates that adapt to all mobile devices (and therefore all device groups). Adaptive templates adapt to the screen resolution of a device and render content suitably. In this scenario, there is no need for creating templates with suffixes.
- Create different templates for different device groups. (An example was described in Section 68.2.2, "Device Groups and Suffixes.")

  This scenario requires you to create two sets of templates: One set contains the base templates (without any suffixes). The other set contains the template variants (templates with suffixes). To create a template variant, you will append `_suffix` to the name of the base template.

For example, if you create a device group called iPhones with suffix `Touch`, then your template name will be *`BaseTemplateName`*`_Touch`. This naming convention and the matching of suffixes for device groups and templates ensures that WebCenter Sites routes requests from mobile devices to the correct template variants. If a mobile template variant does not exist for a specific device group, the base template is used instead.

> **Note:** WebCenter Sites does not recognize templates with two suffixes (such as `_Touch_NonTouch`), and therefore, such templates are not available through the **Choose Page Layout** option on the asset toolbar in the Contributor interface. If you do create a template with two suffixes, this template will not be considered a variant of an existing template, and so, it will not be used for rendering assets. The base template will be used instead.

You can create template variants at any time after you have created the device groups and therefore, specified the suffixes. For more information about creating templates, see Section 68.4.8, "Step 8: Creating Templates."

68.3 Prerequisites for Configuring Mobility Features

Before you start configuring Mobility features, ensure you have the following credentials and information:

- You have either the `SiteAdmin` or `GeneralAdmin` role. In this chapter, we assume you have the `GeneralAdmin` role.
- An understanding of which device repository you will be using. For more information about your options and procedures, see Section 68.4.2, "Step 2: Setting Up the Device Repository."
- The names of device groups in which to organize different devices.
- Which capabilities (such as user agent, touch, tablet, and screen resolution) you will use to identify and group devices. For more information as to which data you will provide, see Section 68.4.3, "Step 3: Configuring Device Groups."
- Which suffix you will use for each device group. A suffix is required for each device group. (You will append the same suffix to the names of the template variants for these device groups. You will also select the same suffix for the site plan.)
- The list of devices your contributors will be using to preview the mobile website. WebCenter Sites comes with a number of device assets that represent many commonly used devices.

  If you need to create your own device assets, first create your own images of the real devices, and look up the user agents of the real devices. You will use these images and user-agent information to create device assets. Also create their thumbnail images for use in the device selector panel in the Contributor interface.
- Effective site plans for the mobile versions of your website.

68.4 Configuring Mobility Features

This section contains the following topics:

- Section 68.4.1, "Step 1: Enabling the Mobility Tab"

- Section 68.4.2, "Step 2: Setting Up the Device Repository"
- Section 68.4.3, "Step 3: Configuring Device Groups"
- Section 68.4.4, "Step 4: Prioritizing Device Groups"
- Section 68.4.5, "Step 5: Creating Device Assets"
- Section 68.4.6, "Step 6: Creating Site Plans"
- Section 68.4.7, "Step 7: Organizing Site Plans"
- Section 68.4.8, "Step 8: Creating Templates"

68.4.1 Step 1: Enabling the Mobility Tab

The only user who is automatically granted access to the **Mobility** tab in the **Admin** interface is the user whose credentials were used during the WebCenter Sites installation process. This user was automatically assigned the `MobileSitesDeveloper` role. Other users must be explicitly assigned the `MobileSitesDeveloper` role so they can use this tab.

68.4.2 Step 2: Setting Up the Device Repository

In this step, you have the option to use one of the following device repositories:

- `devices.xml`

  If you want to use this file, ensure that it is active. For instructions, see Section 68.4.2.1, "Determining Which Device Repository is Active."

  If you need to make changes to this file, you will have to modify the file and re-upload it to WebCenter Sites. For instructions on setting up `devices.xml`, see Section 68.4.2.2, "Modifying the Default Device Repository."

- WURFL

  You will need to obtain a licensed version of this repository from ScientiaMobile. It is not included with WebCenter Sites.

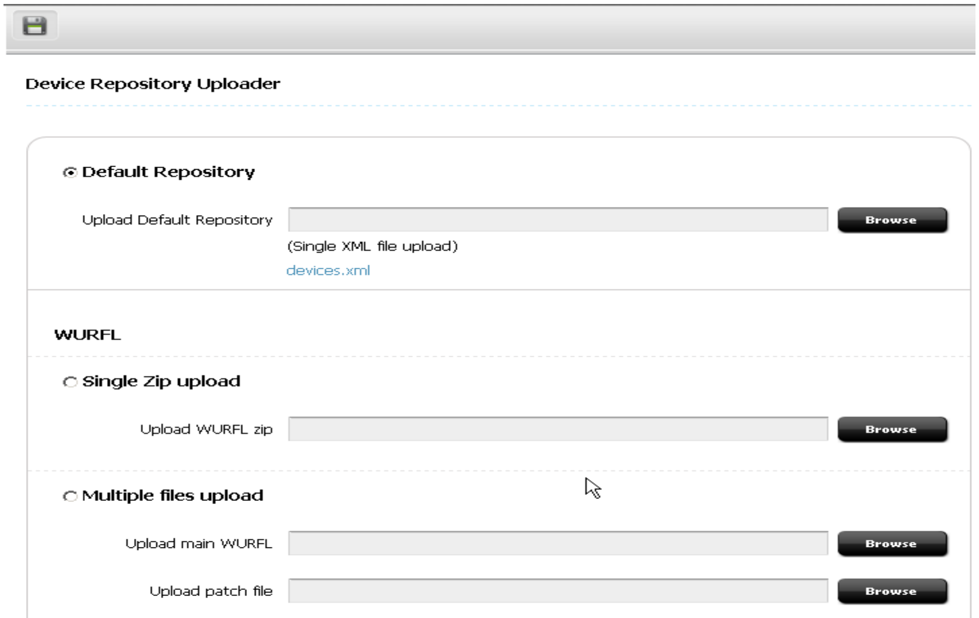
  The WURFL repository can be uploaded in one of two formats:

  - `WURFl.zip`
  - `WURFL.xml` with a WURFL patch file. The patch file is used to override the content of `WURFL.xml`. For more information about patch files, see `http://wurfl.sourceforge.net/`. For instructions on setting up WURFL, see Section 68.4.2.3, "Uploading the Third-Party Device Repository."

68.4.2.1 Determining Which Device Repository is Active

1. On the **Mobility** tab, double-click the **Device Repository** node.
2. In the Device Repository Uploader form, note which repository is selected: **Default Repository** (which means `devices.xml` is active) or one of the WURFL files.

Figure 68–5 Device Repository Uploader



Device Repository Uploader

● Default Repository

Upload Default Repository

(Single XML file upload)

devices.xml

WURFL

○ Single Zip upload

Upload WURFL zip

○ Multiple files upload

Upload main WURFL

Upload patch file

3. Continue with one of the following steps:
   - If you need to change the device repository, complete one of the following sections:
     – Section 68.4.2.2, "Modifying the Default Device Repository"
     – Section 68.4.2.3, "Uploading the Third-Party Device Repository"
   - If you are satisfied with the current active repository, continue to Section 68.4.3, "Step 3: Configuring Device Groups."

68.4.2.2 Modifying the Default Device Repository

In this step, you will ensure the following:

- The `devices.xml` file contains the device names and user agents you require.
- The `devices.xml` file is active.

To set up the `devices.xml` file:

1. Locate `devices.xml` in the following directory: `<WebCenterSites>\11gR1\Sites\11.1.1.8.0\Shared\Storage\DeviceRepository\902\235\devices.xml`, make the necessary changes, and continue with this procedure.
2. Upload `devices.xml` to WebCenter Sites:
   a. In the Admin interface, navigate to the **Mobility** tab.
   b. Double-click **Device Repository**.
   c. In the Device Repository Uploader form, select the **Default Repository** option, if not selected by default.
   d. In the **Upload Default Repository** section, click **Browse** and select the path to `devices.xml`.

> **Note:** If you changed device names in `devices.xml`, you are reminded to update the corresponding device names in all the device groups you might have already been created.

e. Click the **Save** icon.

3. Continue to Section 68.4.3, "Step 3: Configuring Device Groups."

68.4.2.3 Uploading the Third-Party Device Repository

If you need to support more capabilities and devices than already registered in the `devices.xml` file, you can use WURFL as the device repository. WURFL can be uploaded as `WURFL.zip` or `wurfl.xml` and its patch file.

> **Note:** WURFL is a third-party device repository. You must purchase a license in order to use this repository.
>
> If you plan to use the `wurfl.xml` file and patch, ensure the patch file is configured before you start this procedure.

To upload WURFL:

1. In the Admin interface, navigate to the **Mobility** tab.
2. Double-click **Device Repository**.
3. In the Device Repository Uploader form, go to the **WURFL** section, and upload either the complete `WURFL` zip file, or the main `wurfl.xml` file and a patch file.

> **Note:** After the `WURFL.patch` file is uploaded, it is used with the `WURFL.xml` file, regardless of the fact that the record for `WURFL.patch` in the device repository table is still set to `Active=F` (false).

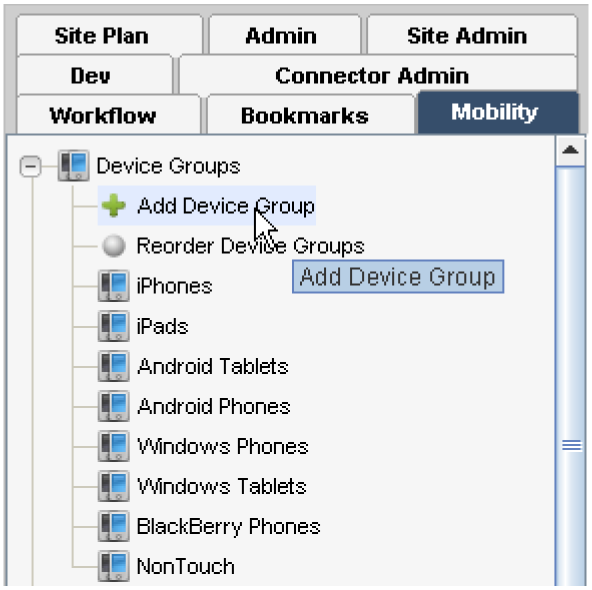
4. Continue to Section 68.4.3, "Step 3: Configuring Device Groups."

68.4.3 Step 3: Configuring Device Groups

To configure a device group:

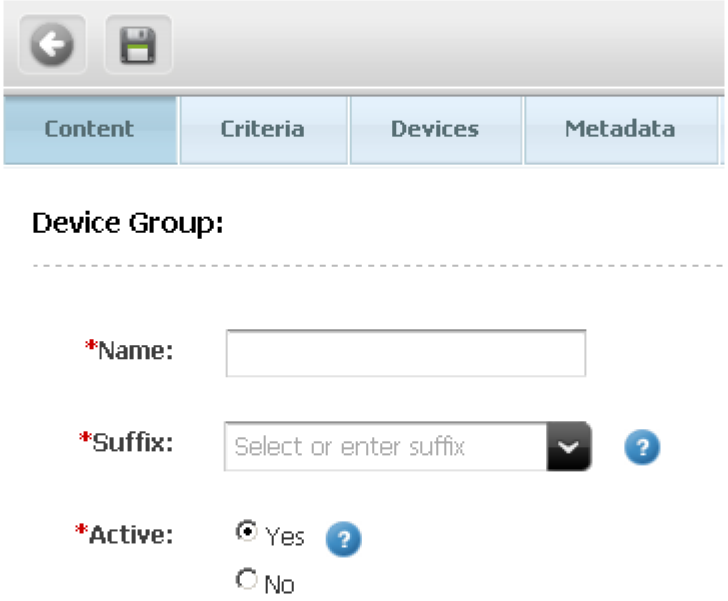
1. On the **Mobility** tab, expand the **Device Groups** node.
2. Click **Add Device Group**, as shown in Figure 68–6.

Figure 68–6 Add Device Group



The Device Group form is displayed, as shown in Figure 68–7.

Figure 68–7 Device Group Form



3. In the Device Group form, select the **Content** tab and do the following:

   a. In the **Name** field, enter a meaningful name for the device group you are creating.

   b. In the **Suffix** section, enter the suffix you plan to use for the templates you will create for this device group (Figure 68–7). You can choose an existing suffix if there is any.

> **Note:** This suffix is not editable. If you want to change it later, you will have to recreate this device group and change this suffix.

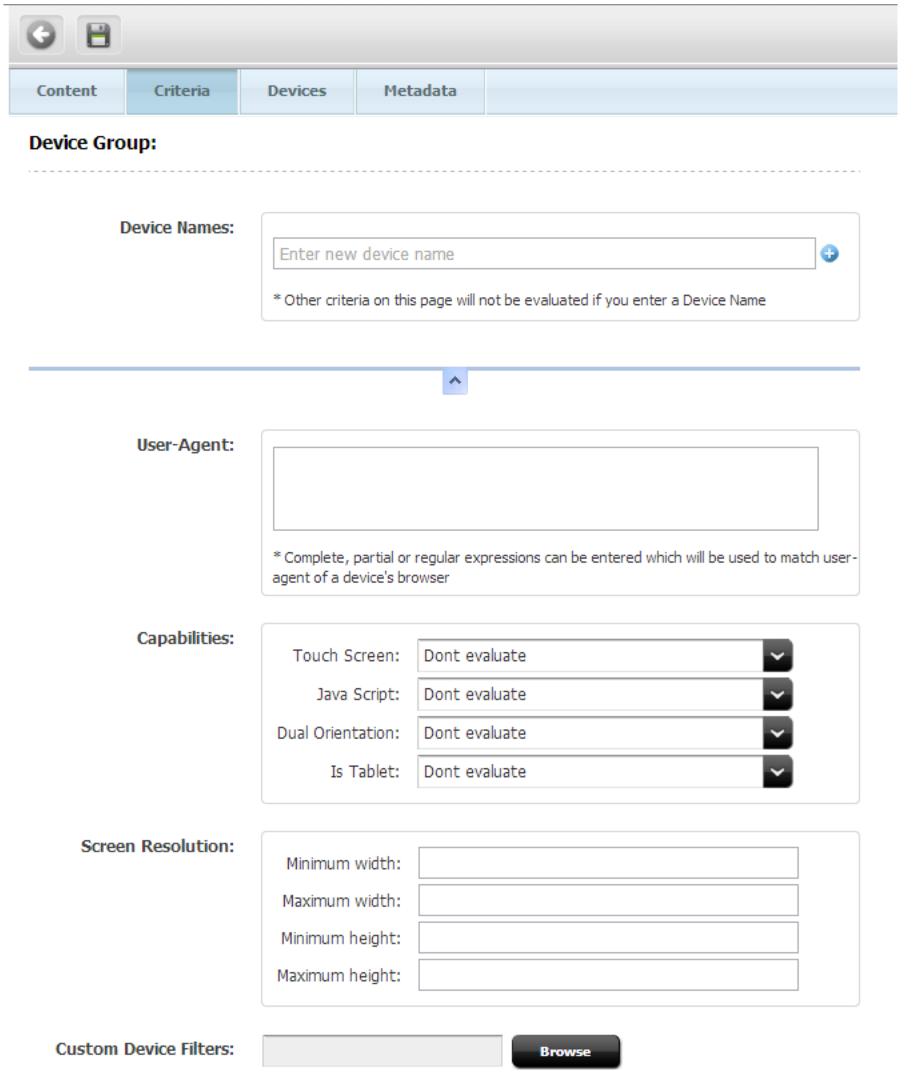
c. In the **Active** field, enable this device group for device detection by selecting **Yes**.

> **Note:** Device groups are global. Once enabled, they become available for all sites. Similarly once disabled, they are disabled for all sites and will no longer be used in device detection.

4. Select the **Criteria** tab (Figure 68–8) to create a set of rules for matching real devices so they can be associated with their correct template variants.

   You can either provide one or more device names (the rest of the form will be disabled (see Figure 68–8), or you can omit the device name(s) and fill in the rest of the form.

Figure 68–8 Add Device Names



Device Group:

Device Names: Enter new device name

* Other criteria on this page will not be evaluated if you enter a Device Name

User-Agent:

* Complete, partial or regular expressions can be entered which will be used to match user-agent of a device's browser

Capabilities:

Touch Screen: Dont evaluate

Java Script: Dont evaluate

Dual Orientation: Dont evaluate

Is Tablet: Dont evaluate

Screen Resolution:

Minimum width:

Maximum width:

Minimum height:

Maximum height:

Custom Device Filters:

a. If you choose to enter device name(s), enter the same name(s) that are registered in the device repository you chose to use in Section 68.4.2, "Step 2: Setting Up the Device Repository." If you uploaded `devices.xml`, enter the device entry name. If you uploaded WURFL, enter the *`model_name`* of the device.

> **Note:** A device can belong to only one device group at a given time.

Continue to Section 68.4.4, "Step 4: Prioritizing Device Groups."

b. If you choose to omit device names, fill in the rest of the form, as follows:

a. **User-Agent Section**. (This section is disabled if you specify the device names.)

In this field, you can enter just a list of device names, or a combination of user agent, screen dimensions, capabilities, and custom filters. A combination of user-agent `regex`, capabilities, screen dimensions, and custom filter requires a device to meet all the rules to match with the device group.

Enter the exact user agent of the browser from which the incoming request will be sent. Or, you can enter a regular expression or a substring to match the set of the user agents. For example, to match all iPhone user agents, specify the user-agent `reg exp` as `(m|M)ozilla/5.0(| )\(i(p|P)(hone|od|rod).*` This expression will match any user-agent string which contains the `iPhone` Java `regex`. Or, you can use the substring `iPhone;` which matches all iPhones.

b. **Capabilities Section**. (This section is disabled if you specify the device names.)

   The capabilities are Touch Screen, JavaScript, Dual Orientation, and Is Tablet. Each capability gives you the following options: **Yes**, **No**, **Don't evaluate**.

   For example, if you want this device group to match only Tablet devices that do not support JavaScript, then for the IsTablet capability, select **Yes**; for the JavaScript capability, select **No**; and for the rest of the capabilities, select **Don't evaluate**.

c. **Screen Resolution Section**. (This section is disabled if you specify the device names.)

   Enter the minimum and maximum width and height for the display area (units are in pixels).

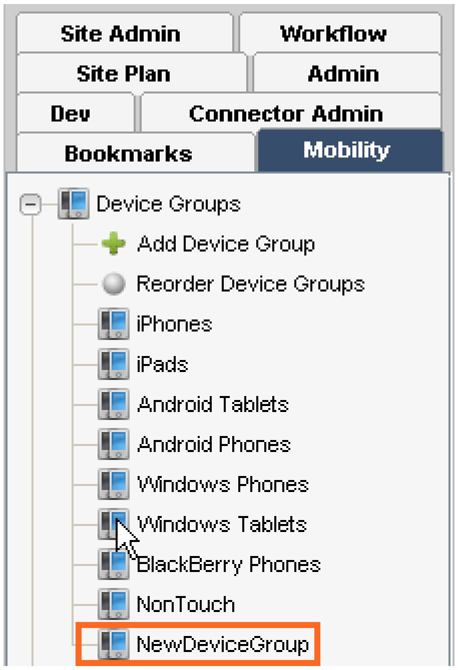
   For example, if you specify a maximum width of 640, then all devices whose screen resolution width is 640 or less will match this device group.

d. In the **Custom Device Filters** section, click **Browse** to choose a custom filter that you might have created to apply conditions on capabilities that are not listed on the Device Group Criteria tab. For information about creating custom filters to set device criteria, see Section 68.4.3.1, "Creating Custom Filters for Device Group Criteria."

5. Click the **Save** icon to save your device group.

   Your new device group is listed on the Mobility tab, at the bottom of the Device Groups node (Figure 68–9).

Figure 68–9 New Device Group



Note: Once you prioritize your device groups and create the device assets, you can verify that they are correctly associated by device detection. To do so, open the device group and select the Devices tab to view the list of device assets associated with the device group.

For procedures on prioritizing device groups, see Section 68.4.4, "Step 4: Prioritizing Device Groups."

For procedures on creating device assets see Section 68.4.5, "Step 5: Creating Device Assets."

6. Continue to Section 68.4.4, "Step 4: Prioritizing Device Groups."

68.4.3.1 Creating Custom Filters for Device Group Criteria

WebCenter provides a default custom filter implementation (`DefaultCustomFilter.java`), which takes an XML file as input. To create a custom filter of your own, you can write an implementation of the `CustomDeviceFilter.java` interface. The XML file for a custom filter is uploaded from the Device Group configuration screen, as described on page 68-14 in Section 68.4.3, "Step 3: Configuring Device Groups."

This section includes the following:

- Using the Default DefaultCustomFIlter.java Custom Filter Provided with WebCenter Sites
- Creating Your Own DeviceGroupFilter Implementation

Using the Default DefaultCustomFIlter.java Custom Filter Provided with WebCenter Sites

The default implementation class of a custom filter is called `COM.FutureTense.Mobility.Filter.DefaultCustomFilter`. It takes input in XML format. In the default custom filter implementation provided with WebCenter Sites, a filter is passed only if *all* its arguments are passed. Similarly, an entire custom filter is passed when *all* its filters are passed. So, in any scenario, *all* arguments must be passed for the filter criteria to work.

Example 68–1 shows a sample filter XML in which device property names are taken from the WURFL repository.

Example 68–1 Sample Filter Based on DefaultCustomFIlter.java and WURFL Repository

```
<?xml version="1.0" encoding="UTF-8"?>
<devicefilters>
       <filter name="tabletFilter"
classname="COM.FutureTense.Mobility.Filter.DefaultCustomFilter"> //Filter 1
             <argument name="is_tablet" value="true" datatype="boolean"/>
//argument 1 of filter 1
             <argument name="pointing_method" value="touchscreen"
datatype="string" operator="equals"/> //argument 2 of filter 1
       </filter>
             <filter name="flashFilter"
classname="COM.FutureTense.Mobility.Filter.DefaultCustomFilter"> //Filter 2
             <argument name=" full_flash_support" value="false"
datatype="boolean"/> //argument 1 of filter 1
       </filter>
</devicefilters>
```

In the above sample, `tabletFilter` has two arguments. `argument 1` requires that the value of the `is_tablet` property should be `true`. `argument 2` requires that the value of the `pointing_method` property should be `touchscreen`. `flashFilter` has only one argument which is, the value of the `full_flash_support` property should be `false`. Each argument is a rule and a complete filter. Only when *every* argument is met, a device can match to the device group containing the above custom filter.

This sample XML uses device property names from the WURFL repository. If the default device repository (`devices.xml`) is used, then this XML looks something like Example 68–2 (notice that the property names have changed). In this filter XML, there are two filters: `tabletFilter` and `flashFilter`.

Example 68–2 Sample Filter Based on DefaultCustomFIlter.java and Devices.xml Repository

```
<?xml version="1.0" encoding="UTF-8"?>
<devicefilters>
       <filter name="tabletFilter"
classname="COM.FutureTense.Mobility.Filter.DefaultCustomFilter"> //Filter 1
           <argument name="tablet" value="true" datatype="boolean"/> //argument 1
of filter 1
           <argument name="touch" value="true" datatype="string"
operator="equals"/> //argument 2 of filter 1
       </filter>
       <filter name="flashFilter"
classname="COM.FutureTense.Mobility.Filter.DefaultCustomFilter"> //Filter 2
           <argument name="flash" value="false" datatype="boolean" /> //argument 1
of filter 1
       </filter>
</devicefilters>
```

While creating a custom filter based on `DefaultCustomFIlter.java`, consider the following:

- In a custom filter, possible values of the `datatype` argument attribute are `string`, `number`, `boolean`. Default value is `string`.
  - When `datatype` = `number`, the possible values of the `operator` attribute are: `<>`, `notequals`, `<`, `lt`, `>`, `gt`, `=`, `equals`. Default value is `equals`.
  - When `datatype` = `boolean`, the possible values of the `operator` attribute are: `=`, `equals`. Any other value is treated as the reverse of `equals`. Default value is `equals`.
  - When `datatype` = `string`, the possible values of the `operator` attribute are `=`, `equals`, `%`, `like`, `!=`, `notequals`, `!%`, `notlike`. Default value is `equals`. The following snippet shows the use of the `like` or `%` value for the `operator` attribute:

    ```
    <argument name="pointing_method" value="touch" datatype="string"
    operator="like"/>
    ```

    Here, the value of the `pointing_method` property must contain the word `touch` either as a substring or an entire word.
- The value of the property attributes must be as per the property names in the current device repository (either `devices.xml` or `WURFL.xml`).

Creating Your Own DeviceGroupFilter Implementation

You can create a Java class that implements the `DeviceGroupFilter` interface (Example 68–3) and uses the `matches` method. Your custom filter can consist of one or more filters. Each filter can contain zero or more arguments. The custom filter implementations can use the `OR` rule, or any custom logic.

Example 68–3 A Java Class That Implements the DeviceGroupFilter Interface and the matches Method

```
public class UserDefinedCustomFilter implements DeviceGroupFilter
{
public boolean matches(DeviceContext context)
    {
       // Logic that returns true/false depending on whether criteria matched or
not.
    }
}
```

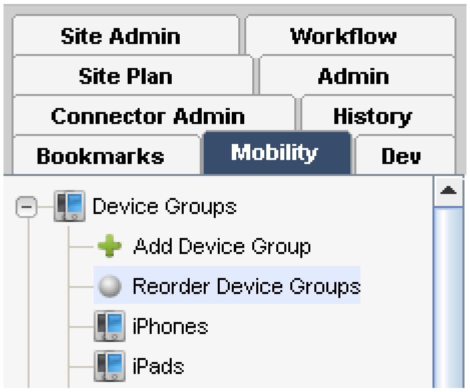
68.4.4 Step 4: Prioritizing Device Groups

Multiple device groups must be prioritized to enable device detection to automatically associate a real device to the highest-priority device group at runtime.

To prioritize device groups:

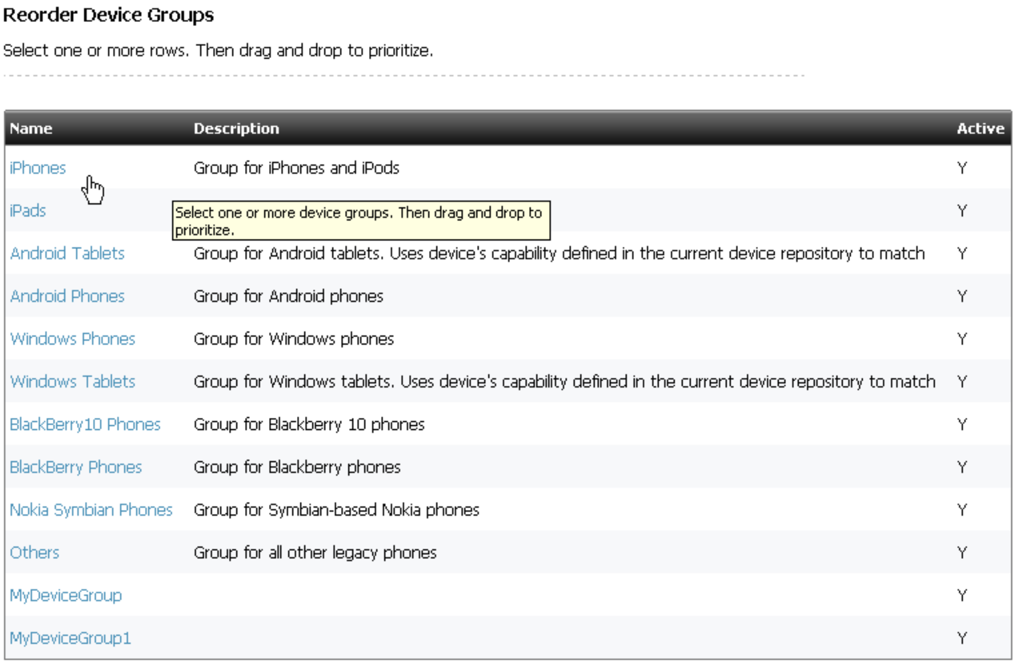
1. On the **Mobility** tab, expand the **Device Groups** node.
2. Double-click **Reorder Device Groups** (Figure 68–10).

Figure 68–10 Reorder Device Groups



3. In the Reorder Device Groups screen, drag and drop the device groups in the order of your preferred priority (Figure 68–11).

Figure 68–11 Drag and Drop Device Groups



Reorder Device Groups

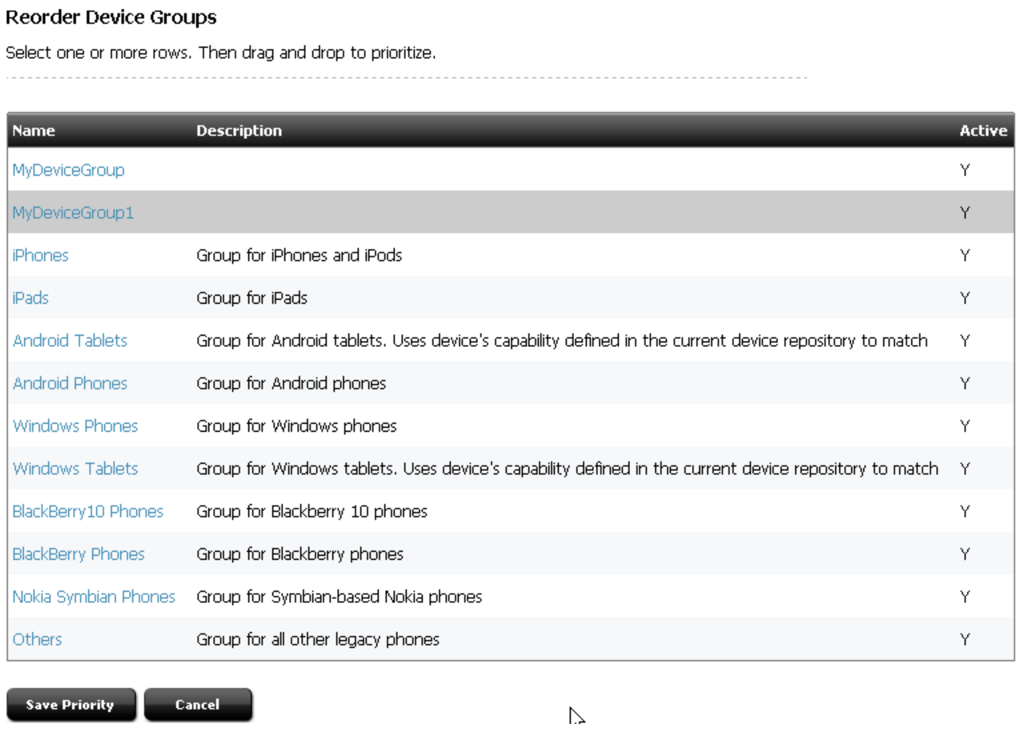
Select one or more rows. Then drag and drop to prioritize.

Select one or more device groups. Then drag and drop to prioritize.

Name	Description	Active
iPhones	Group for iPhones and iPods	Y
iPads		Y
Android Tablets	Group for Android tablets. Uses device's capability defined in the current device repository to match	Y
Android Phones	Group for Android phones	Y
Windows Phones	Group for Windows phones	Y
Windows Tablets	Group for Windows tablets. Uses device's capability defined in the current device repository to match	Y
BlackBerry10 Phones	Group for Blackberry 10 phones	Y
BlackBerry Phones	Group for Blackberry phones	Y
Nokia Symbian Phones	Group for Symbian-based Nokia phones	Y
Others	Group for all other legacy phones	Y
MyDeviceGroup		Y
MyDeviceGroup1		Y

When you have reordered the device groups, the Reorder Device Groups screen displays them in the new sequence (such as the one in Figure 68–12).

Figure 68–12 Device Groups Reordered



Reorder Device Groups

Select one or more rows. Then drag and drop to prioritize.

Name	Description	Active
MyDeviceGroup		Y
MyDeviceGroup1		Y
iPhones	Group for iPhones and iPods	Y
iPads	Group for iPads	Y
Android Tablets	Group for Android tablets. Uses device's capability defined in the current device repository to match	Y
Android Phones	Group for Android phones	Y
Windows Phones	Group for Windows phones	Y
Windows Tablets	Group for Windows tablets. Uses device's capability defined in the current device repository to match	Y
BlackBerry10 Phones	Group for Blackberry 10 phones	Y
BlackBerry Phones	Group for Blackberry phones	Y
Nokia Symbian Phones	Group for Symbian-based Nokia phones	Y
Others	Group for all other legacy phones	Y

4. Click **Save Priority**.

   The **Device Groups have been re-prioritized successfully** message is displayed.

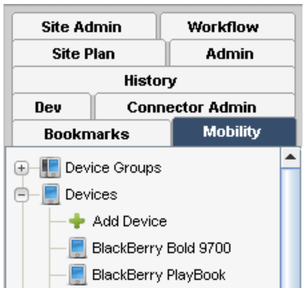
5. Continue to Section 68.4.5, "Step 5: Creating Device Assets."

68.4.5 Step 5: Creating Device Assets

To create a device asset:

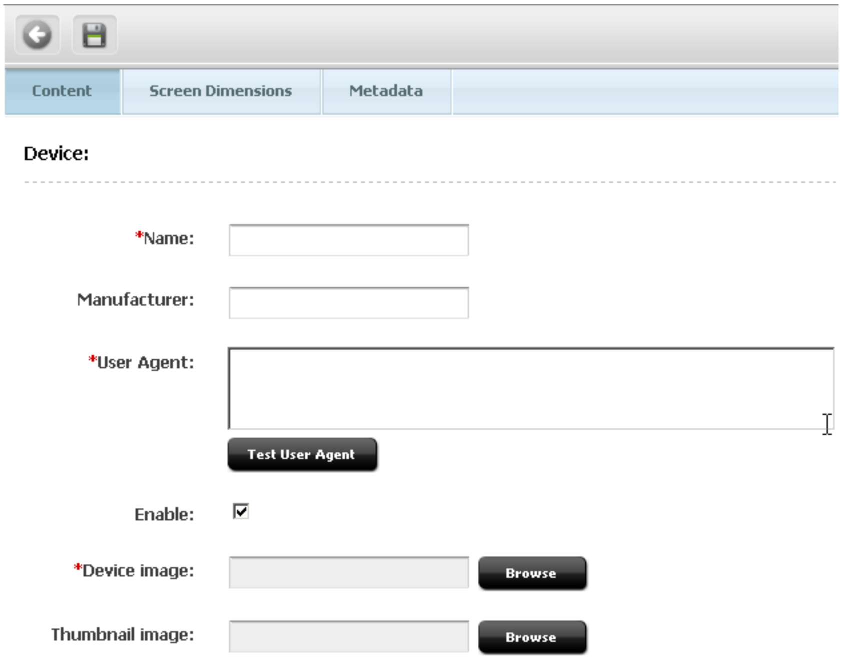
1. On the **Mobility** tab, expand the **Devices** node.
2. Click **Add Device** (Figure 68–13).

Figure 68–13 Add Device



The Device form is displayed, as shown in Figure 68–14.

Figure 68–14 Device Form - Content Tab



Content | Screen Dimensions | Metadata

Device:

*Name:

Manufacturer:

*User Agent:

Test User Agent

Enable: ☑

*Device image: Browse

Thumbnail image: Browse

3. On the **Content** tab (Figure 68–14):

   a. In the **Name** field, enter a name for this device asset.

      The name that you enter will be displayed in the Contributor interface. It does not have to match the name in the device repository.

   b. (Optional). In the **Manufacturer** field, enter the name of the device maker company.

   c. In the **User Agent** field, specify the registered user agent for the device whose image you are adding. You can copy the user agent from the device repository.

      Note: The user-agent field identifies which real device this device asset represents. This field is used in device detection logic to associate this device with the matching device group of highest priority.

   d. Click **Test User Agent** to run device detection and confirm that this device matches a particular device group.

   e. Select the **Enable** checkbox to make this device asset available for association to device groups by device detection.

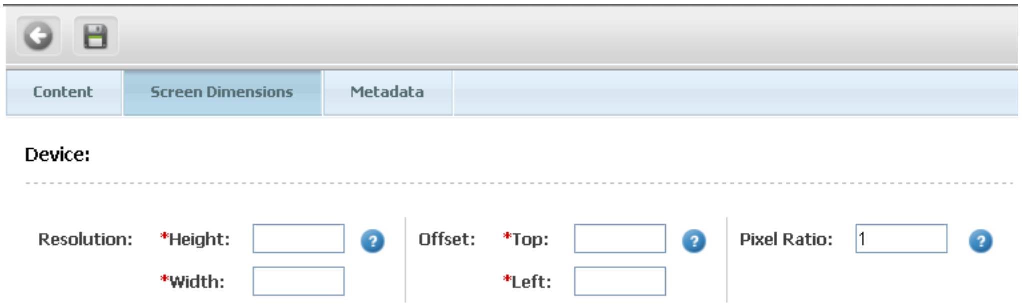
   f. Next to the **Device image** field, click **Browse** to select the device image from the directory in which the image is stored.

   g. (Optional). Next to the **Thumbnail image** field, upload a thumbnail image. This thumbnail image will be displayed in the device selector panel that is associated with the preview feature in the Contributor interface. Selecting the thumbnail displays a page preview in the device image.

4. On the **Screen Dimensions** tab, enter the required pixels in the **Height**, **Width**, **Top**, and **Left** fields and pixel ratio in the **Pixel Ratio** field to determine an appropriate dimension of the screen area.

As you enter the pixels in each field, the screen area of the device image begins to reset accordingly. This feature lets you determine the exact dimension of the device display area on the spot.

Figure 68–15 Screen Dimensions Tab



Pixel Ratio: 1

5. Click the **Save** icon on the form to create the new device.

   The success message is displayed.

6. Continue to Section 68.4.6, "Step 6: Creating Site Plans."

68.4.6 Step 6: Creating Site Plans

Whether you need multiple site plans or a single site plan depends on your design approach.

When creating a site plan, you will associate device groups to that site plan by selecting a common suffix. Once selected, that suffix is no longer available for other site plans.

To create a site plan:

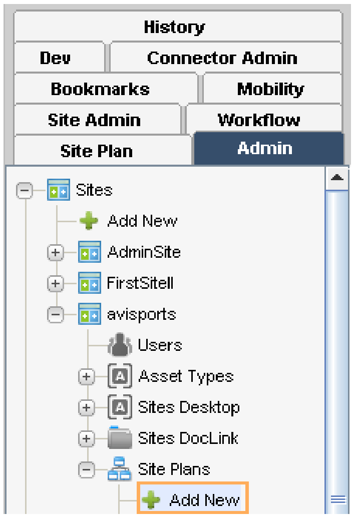
1. In the Admin interface, open the **Admin** tab, and expand the **Sites** node and the site for which you are creating the site plan.

   > **Note:** (If you are assigned the `SiteAdmin` role, use the **Site Admin** tab instead.

2. Expand the **Site Plans** node.

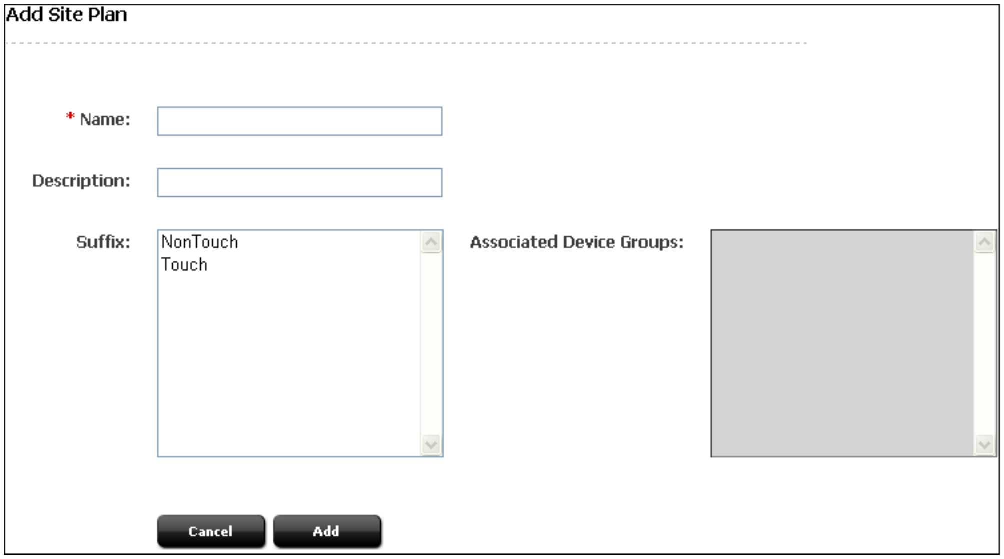
3. Double-click **Add New** (Figure 68–16).

Figure 68–16 Site Plans Node in the Admin Tab



The Add Site Plan page is displayed (Figure 68–17).

Figure 68–17 Add Site Plan

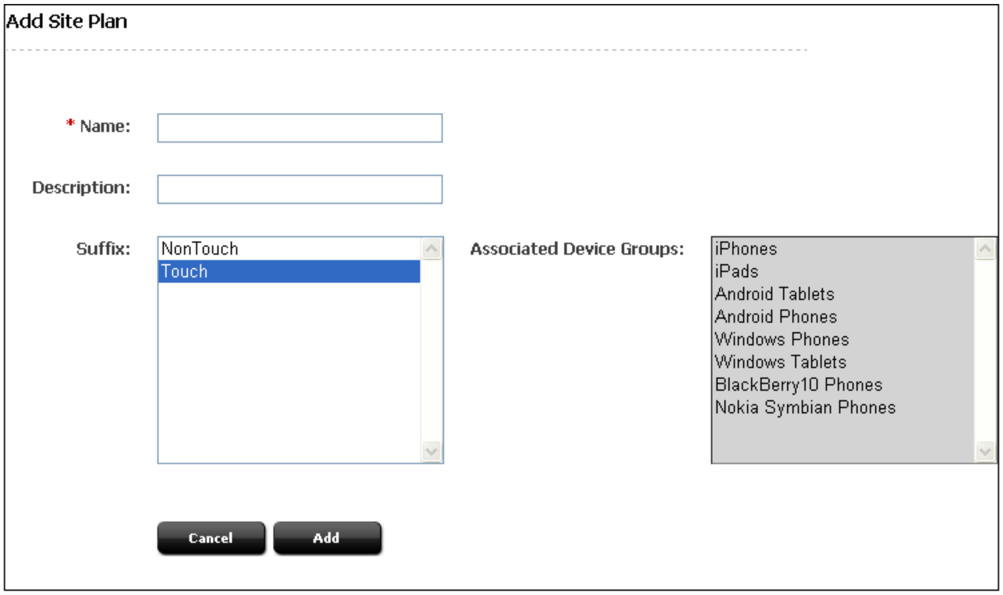


4. In the **Name** field, enter a meaningful name for your site plan.
5. (Optional) In the **Description** field, enter a description for your site plan.
6. To associate device groups with this site plan, go to the **Suffix** section, and select a suffix used by those device groups.

> **Note:** The **Suffix** section lists only suffixes that are not assigned to any site plans.

The Associated Device Groups panel (Figure 68–18) lists the device groups that are associated with your selected suffix.

Figure 68–18 Associated Device Group Panel



Add Site Plan

* Name:

Description:

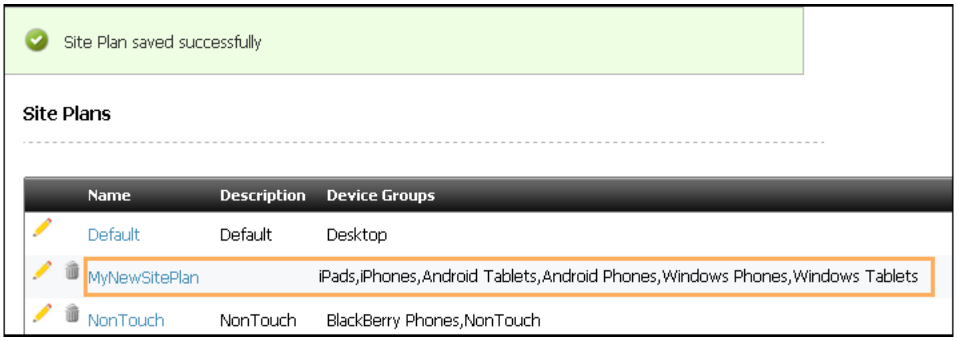
Suffix: NonTouch, Touch

Associated Device Groups: iPhones, iPads, Android Tablets, Android Phones, Windows Phones, Windows Tablets, BlackBerry10 Phones, Nokia Symbian Phones

7. Click **Add** to complete the site plan.

   A screen similar to Figure 68–19 is displayed.

Figure 68–19 New Site Plan Created



Site Plan saved successfully

Site Plans

Name	Description	Device Groups
Default	Default	Desktop
MyNewSitePlan		iPads,iPhones,Android Tablets,Android Phones,Windows Phones,Windows Tablets
NonTouch	NonTouch	BlackBerry Phones,NonTouch

8. Click the **Site Plan** tab to see the newly created site plan.

   To modify the site plan, double-click it under your site's node on the **Admin** tab. On the **Modify Site Plan** page, make the desired changes and then click **Modify**. Continue to Section 68.4.7, "Step 7: Organizing Site Plans."

68.4.7 Step 7: Organizing Site Plans

You can reorder site plans if they should be displayed in a specific order in the Admin and Contributor interfaces.

To organize site plans:

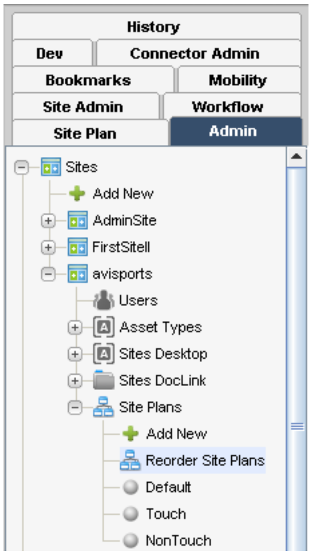
1. In the Admin interface, open the **Admin** tab, and expand the **Sites** node and the site for which you are prioritizing the site plans.

Note: (If you are assigned the `SiteAdmin` role, use the **Site Admin** tab instead.

---

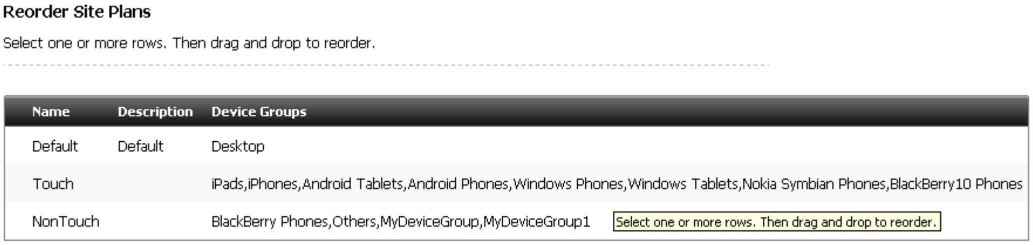
2. Double-click **Reorder Site Plans**.

Figure 68–20 Reorder Site Plans



3. In the Reorder Site Plans screen (Figure 68–21), drag and drop site plans to order them in the preferred sequence.

Figure 68–21 Drag and Drag Site Plans



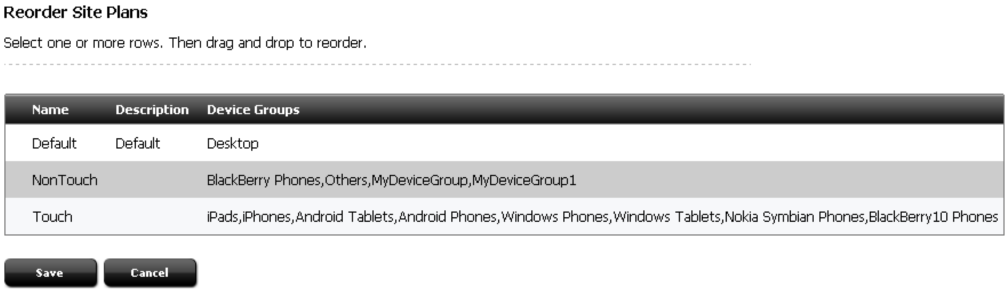
Reorder Site Plans

Select one or more rows. Then drag and drop to reorder.

Name	Description	Device Groups
Default	Default	Desktop
Touch		iPads,iPhones,Android Tablets,Android Phones,Windows Phones,Windows Tablets,Nokia Symbian Phones,BlackBerry10 Phones
NonTouch		BlackBerry Phones,Others,MyDeviceGroup,MyDeviceGroup1

When you have reordered the site plans, the Reorder Site Plans screen looks something like Figure 68–22.

Figure 68–22 Site Plan Reordered



Reorder Site Plans

Select one or more rows. Then drag and drop to reorder.

Name	Description	Device Groups
Default	Default	Desktop
NonTouch		BlackBerry Phones,Others,MyDeviceGroup,MyDeviceGroup1
Touch		iPads,iPhones,Android Tablets,Android Phones,Windows Phones,Windows Tablets,Nokia Symbian Phones,BlackBerry10 Phones

4. Click **Save**.

   The **Modification was successful** message is displayed.

5. Continue to Section 68.4.8, "Step 8: Creating Templates."

68.4.8 Step 8: Creating Templates

You have two basic approaches for creating templates that render mobile websites:

- Create a single set of templates that adapt to all mobile devices (and therefore all device groups).
- Create different templates for different device groups. This scenario requires the use of suffixes.

This section discusses the second approach. This section contains the following topics:

- Section 68.4.8.1, "Basic Guidelines for Creating Template Variants"
- Section 68.4.8.2, "Mobility Tags"
- Section 68.4.8.3, "Tags Modified to Support Device Detection and Page Rendering"

68.4.8.1 Basic Guidelines for Creating Template Variants

When creating different templates for different device groups, note the following requirements:

1. Create the base template (the template that renders the desktop website).
2. Create the template variant by using the name of the base template and appending *_suffix*.
3. If you intend to cache pages based on your templates, you must use the suffix (the d parameter) as one of the cache criteria.

68.4.8.2 Mobility Tags

Table 68–1 describes the Mobility tags you will use when creating templates. For detailed information about the tags, see the *Oracle Fusion Middleware WebCenter Sites Tag Reference*.

Table 68–1 Tags for Mobility

Tag	Description
`<device:load name="<Name of Current Device>" />`	Detects the current device and loads the device information. Similar to `asset:load`.
`<device:get name="<Name of Current Device>" property="useragent|devicegroup|suffix" [output="propName"] />`	For a loaded device, this tag returns the value of one of the following properties: `useragent`, `devicegroup`, or `suffix`. If the optional attribute `output` is provided, this tag sets the value of the property in the `output` variable. If the `output` attribute is absent, this tag sets the value of the property to a variable with the name of that property.
`<device:if name="<Name of Current Device>" property="touch" value="true" [datatype="boolean"] [operator="equals"] > ... conditional code for touch devices only ... </device:if>`	For the currently loaded device, this tag allows you to specify a condition and code that will be executed when the condition is met. The default value for the optional attribute `datatype` is `String`, and the default value for the optional attribute `operator` is `"="`.
`<device:hascapability name="<Name of the Device Loaded Earlier>" capability="<WURFL_CAPABILITY_NAME>" > conditional code </device:hascapabiilty>`	For the currently loaded device, this tag checks if this device supports the capability specified in this tag. If the device supports this capability, the code in the tag is executed.

Table 68–1 (Cont.) Tags for Mobility

Tag	Description
`<device:capability name="<Name of the Device Loaded Earlier>" capability="<WURFL_CAPABILITY_NAME>" output="capabilityValue" />`	For the currently loaded device, this tag sets the value of the specified capability in the `ics` scope.
`<device:siteplan output=<NAME_OF_VARIABLE_HOLDING_RESULTING_SITEPLAN_ID> [pubid = <%=ics.GetVar("pubid")%>] [site=<%=ics.GetVar("site")%>] [d= <%=ics.GetVar("d")%>] />`	You can use the `device:siteplan` tag to look up the site plan for any device. The tag takes the `d` parameter, which is a device group suffix. At runtime, WebCenter Sites computes the value of the `d` parameter by using device detection. The tag then uses the value of the `d` parameter (that is, the suffix) to find the site plan with the same suffix and to return the ID of that site plan. The site plan ID is used by other tags to construct website navigation.

68.4.8.3 Tags Modified to Support Device Detection and Page Rendering

The following tags include Mobility-specific attributes:

- `render:getpageurl`
- `render:calltemplate`
- `render:gettemplateurl`
- `render:gettemplateurlparameters`
- `insite:calltemplate`
- `satellite:page`
- `satellite:link`

The attributes are:

- `d`, the suffix of a device group.
- `resolvetemplatefordevice`, which appends the device group suffix to the template name. The default value is `true`. If the value is set to `false`, the suffix is not appended to the template name and the base template is loaded.

The `d` parameter is automatically passed down the chain in each of the tags above. The tags call the correct template variant by basing their call on the passed `d` attribute.

In the following example for the avisports sample site, `c=Page` and `TestSite` is the current site.

```
<render:calltemplate tname='Detail' args="c,cid,p,d,locale,form-to-render" />
(i) for d=Desktop (default) or blank, the following template is called:
         TestSite/Page/Detail
(ii) for d=Touch, the following template is called:
         TestSite/Page/Detail_Touch
```

For information about how the `d` attribute is used, see Section 68.5, "Device Detection."

68.5 Device Detection

WebCenter Sites uses a built-in device detection mechanism to identify the device that requests website content. Once the device is identified, WebCenter Sites looks for the matching device group and reads its suffix. Using the suffix, WebCenter Sites finds the site plan and invokes the template variant to render the content.

The detailed steps are as follows:

1. A page request from a real device is received by Remote Satellite Server. The header for this request includes the user agent of the device.
2. Remote Satellite Server looks for the page in its own cache. If it fails to find the page, Remote Satellite Server sends the page request to WebCenter Sites.
3. WebCenter Sites responds as follows:
   a. Identifies the device by its user agent in the request header.
   b. Looks for the user agent in the device repository.
   c. If it finds a matching device in the device repository, WebCenter Sites also finds the capabilities of that device.
   d. WebCenter Sites then uses the user agent and device capabilities to find device groups with matching criteria.
   e. Associates the device to the highest-priority device group.
   f. Reads the suffix for this device group.
   g. Assigns the suffix to the d parameter in the `ics` scope.
   h. Appends `_`*`suffix`* to the requested template name in the URL.
   i. Appends `d=`*`suffix`* to the URL of the requested page.

      For example:

      If the suffix is `Touch`, WebCenter Sites converts the original URL `pagename=avisports/HomeLayout1&c=Page&cid=1482760932`

      to the following URL:

      ```
      pagename=avisports/HomeLayout1_Touch&c=Page&cid=1482760932&d=Touch
      ```

   j. If the template variant exists, WebCenter Sites executes the new URL, caches the page, and sends it to Remote Satellite Server.

      If the template variant does not exist, WebCenter Sites executes the original URL, caches that page, and sends it to Remote Satellite Server.
4. Remote Satellite Server caches the page and sends the response back to the device.
5. Remote Satellite Server also caches device detection information and uses it to process subsequent requests from the same device. This prevents WebCenter Sites from re-running device detection.

Part V

Developing Applications with the Web Experience Management (WEM) Framework

This part provides information about the Oracle WebCenter Sites: Web Experience Management (WEM) Framework as it relates to application development. This part of the guide contains an overview of the WEM Framework, and moves on to describe the process of developing applications and custom Representational State Transfer (REST) resources. This part of the guide also provides information about implementing and customizing Single Sign-On (SSO).

This part contains the following chapters:

69

About the Web Experience Management (WEM) Framework

The Oracle WebCenter Sites: Web Experience Management (WEM) Framework provides the technology for developing applications and integrating them with Oracle WebCenter Sites. A single administrative interface, WEM Admin, supports centralized application management and user authorization. Single sign-on enables users to log in once and gain access to all applications allowed to them during the session.

This chapter contains the following sections:

- Section 69.1, "Overview of the WEM Framework"
- Section 69.2, "Prerequisites for Application Development"
- Section 69.3, "Getting Started"

69.1 Overview of the WEM Framework

The WEM Framework requires a content management platform. In this release, the WEM Framework runs on Oracle WebCenter Sites and ships with the WebCenter Sites Representational State Transfer (REST) API. Objects in the WebCenter Sites database, such as sites, users, and data model map to REST resources in the WEM Framework.

When implemented on the WEM Framework, applications communicate with the WebCenter Sites database through REST services. The applications appear in WEM Admin as list items on the **Apps** page (Figure 69–1). Administrators authorize users, which involves configuring access to the applications and their resources. To this end, the WEM Admin interface exposes authorization items (along with applications) through links on the **menu bar**.

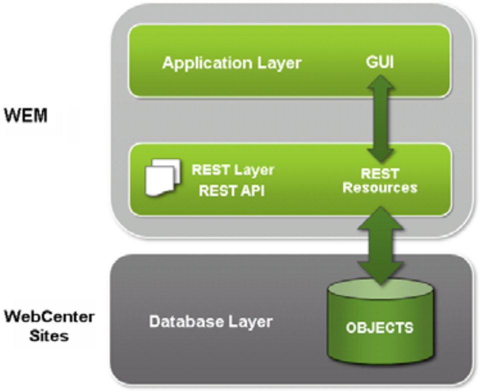
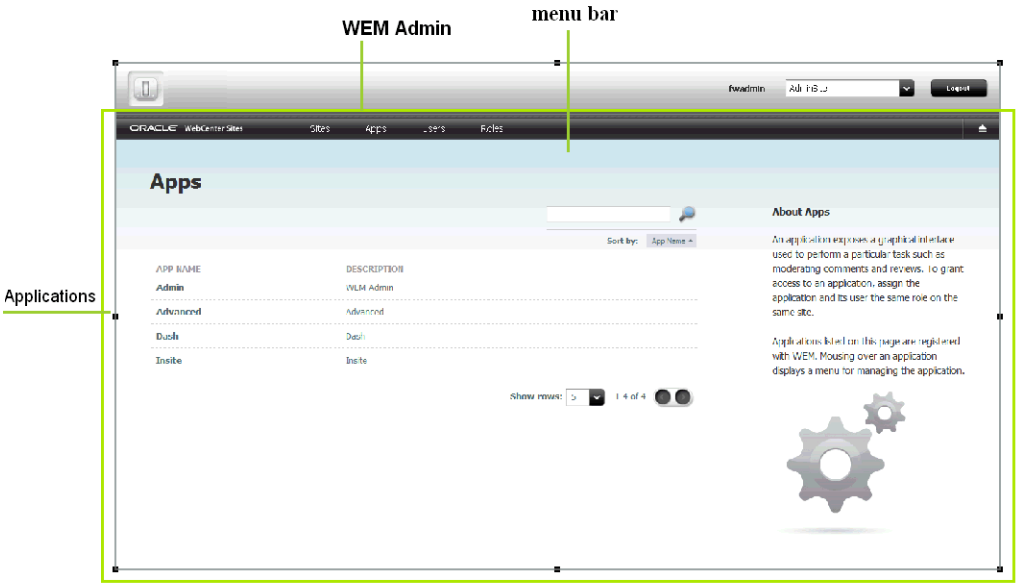


Figure 69–1 Apps Page, WEM Admin



Apps

Sort by: App Name

APP NAME	DESCRIPTION
Admin	WLM Admin
Advanced	Advanced
Dash	Dash
Insite	Insite

Show rows: 5 1-4 of 4

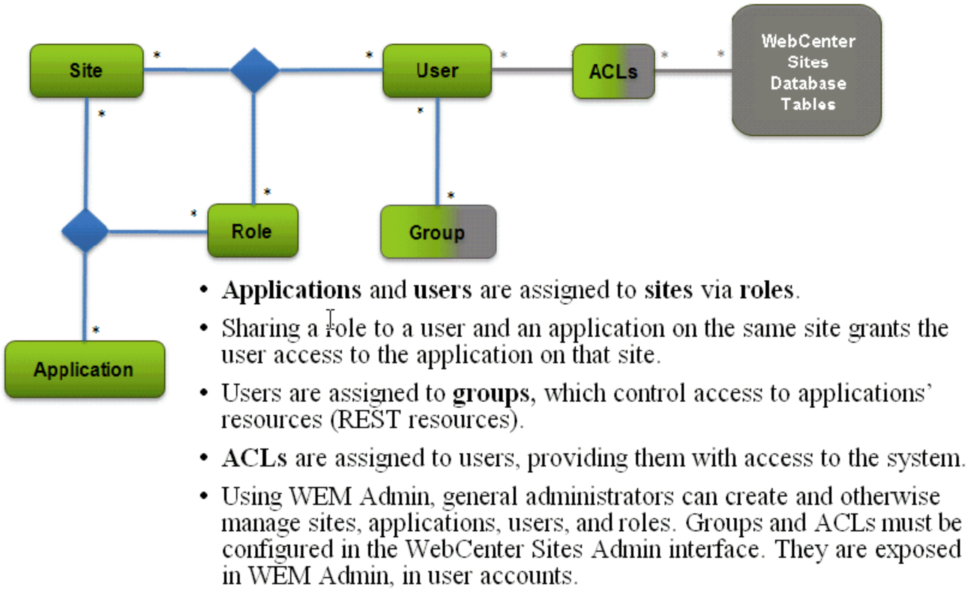
About Apps

An application exposes a graphical interface used to perform a particular task such as moderating comments and reviews. To grant access to an application, assign the application and its user the same role on the same site.

Applications listed on this page are registered with WEM. Mousing over an application displays a menu for managing the application.

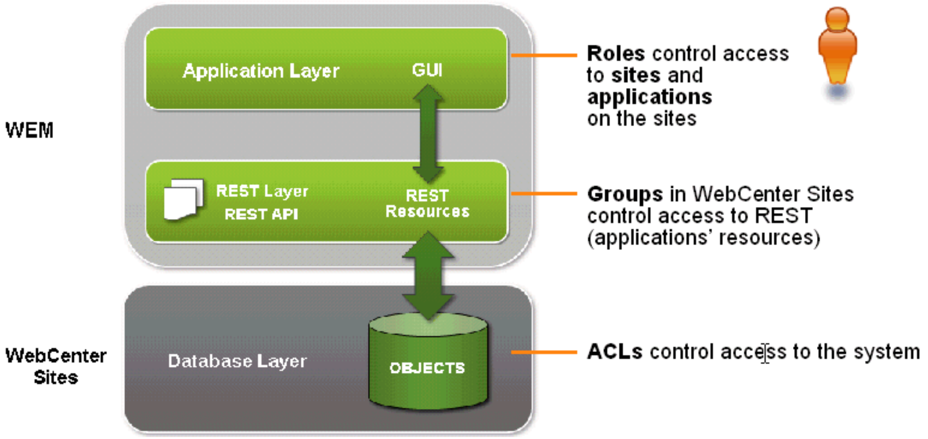
Coupling the items as shown in Figure 69–2 enables applications for users.

Figure 69–2 Authorization Mode



- **Applications** and **users** are assigned to **sites** via **roles**.
- Sharing a role to a user and an application on the same site grants the user access to the application on that site.
- Users are assigned to **groups**, which control access to applications' resources (REST resources).
- **ACLs** are assigned to users, providing them with access to the system.
- Using WEM Admin, general administrators can create and otherwise manage sites, applications, users, and roles. Groups and ACLs must be configured in the WebCenter Sites Admin interface. They are exposed in WEM Admin, in user accounts.

Once the coupling is complete, users are authorized at the database, REST, and application levels.


WEM
Application Layer
GUI
REST Layer
REST API
REST
Resources
WebCenter
Sites
Database Layer
OBJECTS
Roles control access to **sites** and **applications** on the sites
Groups in WebCenter Sites control access to REST (applications' resources)
ACLs control access to the system

Experienced WebCenter Sites developers will recognize that the WEM Framework extends the use of sites and roles to control access to applications. However, unlike WebCenter Sites, the WEM Admin interface does not expose the data model. The REST API does. In this respect, WEM Admin can be thought of as strictly an authorization interface, supported by the WebCenter Sites Admin interface (for configuring ACLs and groups).

Although WEM Admin is seldom used by developers, the concepts behind user authorization can come into play in application development. The next chapters describe the WEM Framework as it relates to application development and provides examples of application code.

69.2 Prerequisites for Application Development

Developing an application involves coding the application's logic, deploying the application, and registering the application to expose it in WEM Admin for administrators to manage and make available to other users. This information is not intended to be a tutorial on application development, but a reference to orient experienced application developers to the WEM Framework. Users of these chapters must be expert WebCenter Sites developers with a working knowledge of the technologies listed in this section. Required resources are also listed below.

This section contains the following topics:

- Section 69.2.1, "Technologies"
- Section 69.2.2, "WebCenter Sites Interfaces, Objects, and APIs"
- Section 69.2.3, "Documentation"
- Section 69.2.4, "Sample Applications and Files"
- Section 69.2.5, "Application Access"

69.2.1 Technologies

- Representational State Transfer (REST), used to communicate with the WebCenter Sites platform
- Central Authentication Service (CAS), which is deployed during WebCenter Sites installation to support single sign-on for WEM

- Java Server Pages Standard Tag Library (JSTL), Java, JavaScript, Jersey, and the Spring MVC framework, in order to follow the code of the "Articles" sample application provided with WEM

69.2.2 WebCenter Sites Interfaces, Objects, and APIs

Developers must have a working knowledge of:

- WebCenter Sites Admin (the WebCenter Sites administrative interface)
- WebCenter Sites basic and flex asset models
- Asset API
- ACLs, which protect database tables and define the types of operations that can be performed on the tables
- Concept of sites and roles

69.2.3 Documentation

To follow this section of the guide you will need this documentation:

- *Oracle Fusion Middleware WebCenter Sites REST API Resource Reference*
- *Oracle Fusion Middleware WebCenter Sites REST API Bean Reference*

Information about ACLs, sites, and roles, and their usage in WebCenter Sites is available in the *Oracle Fusion Middleware WebCenter Sites: Administrator's Guide*.

69.2.4 Sample Applications and Files

- The following sample applications are used in this guide:
  - *Articles*, a lightweight content management application
  - SSO sample application, a small authentication application for production sites. The application is packaged as `wem-sso-api-cas-sample.war`.
  - *Recommendations*, which demonstrates the process of creating REST resources
- The Customizable Single Sign-On facility is used in this section of the guide to illustrate customization of login behavior.
- WEM Framework ships with sample files to illustrate cross-domain implementations and management of assets over REST using our API.

All sample applications and files are located in the `Misc/Samples/WEM Samples` folder in your WebCenter Sites installation directory.

69.2.5 Application Access

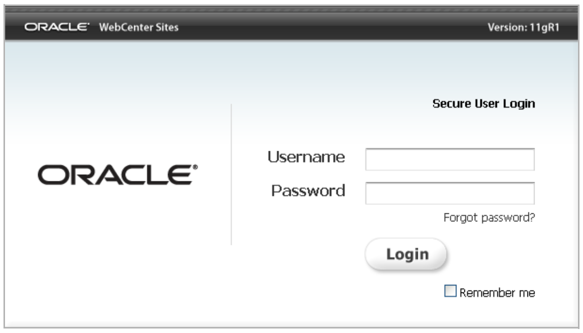
When using this guide, or developing and testing, you will access some or all of the applications listed below:

- **CAS web application**. You will specify its URL in the "Articles" sample application to enable single sign-on:

  ```
  http://<server>:<port>/<cas_application_context>/login
  ```

  where `<server>` is the host name or IP address of the computer running CAS and `<cas_application_context>` is the context path of the CAS web application.
- **WebCenter Sites Admin** interface, if you decide to register applications manually:

```
http://<server>:<port>/<cs_application_
context>/Xcelerate/LoginPage.html
```



Log in with the credentials of the general administrator that was used during the WebCenter Sites installation process (or an equivalent general admin). The default login credentials are `fwadmin/xceladmin` (**same for logging in to WEM Admin**).

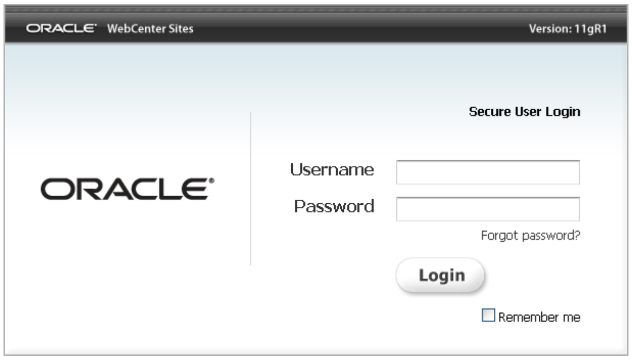
> **Note:** General administrators on WebCenter Sites systems are specially configured for the WEM Framework. During the installation process, `fwadmin` was automatically added to the `RestAdmin` group for unrestricted access to REST services, and enabled on AdminSite where the WEM Admin interface runs. More information about WEM-related changes to WebCenter Sites is available in the *Oracle Fusion Middleware WebCenter Sites Installation Guide*.

- **WEM Admin**, to test the results of your application registration process:

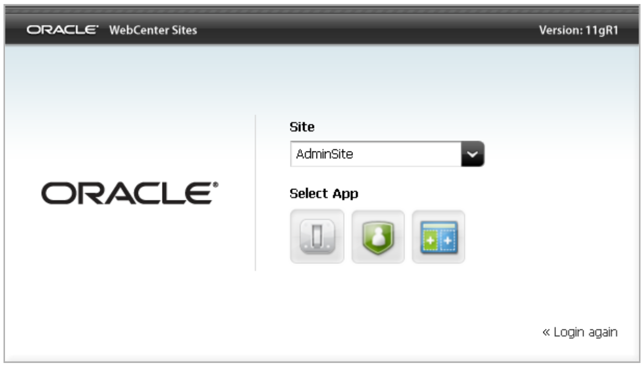
  ```
  http://<server>:<port>/<cs_application_context>/login
  ```

  Log in as `fwadmin` (or an equivalent user). The sequence of screens is the following:

  a. Login Screen:



  b. Transition screen (if you are logging in for the first time or in to a site that you have never accessed before). Select **AdminSite** and the first icon, **Admin**:

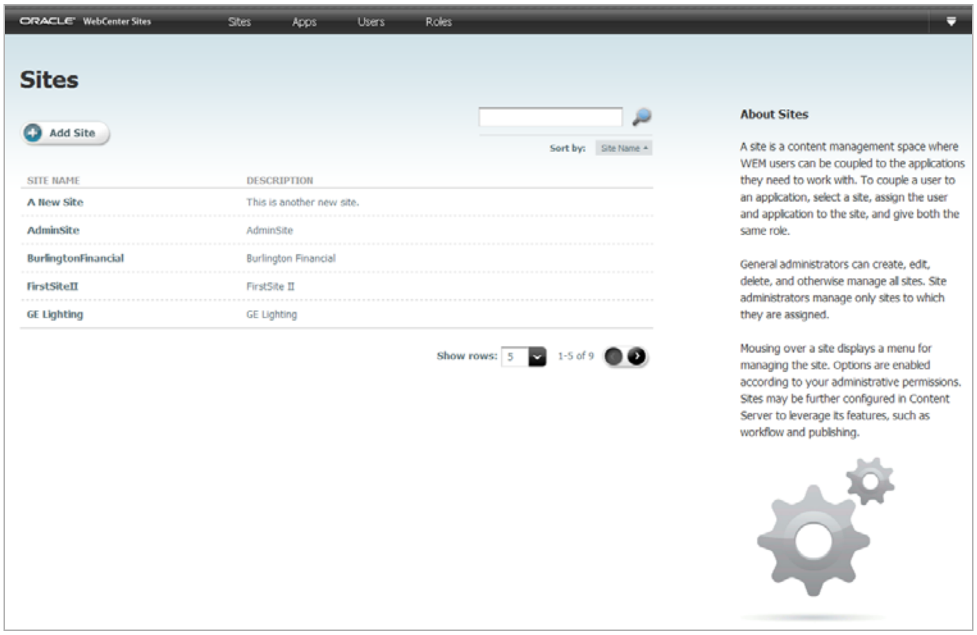


Site

AdminSite

Select App

c. WEM Admin **Sites** page. Registered applications are listed on the **Apps** page.



Sites

SITE NAME	DESCRIPTION
A New Site	This is another new site.
AdminSite	AdminSite
BurlingtonFinancial	Burlington Financial
FirstSiteII	FirstSite II
GE Lighting	GE Lighting

Show rows: 5 1-5 of 9

About Sites

A site is a content management space where WEM users can be coupled to the applications they need to work with. To couple a user to an application, select a site, assign the user and application to the site, and give both the same role.

General administrators can create, edit, delete, and otherwise manage all sites. Site administrators manage only sites to which they are assigned.

Mousing over a site displays a menu for managing the site. Options are enabled according to your administrative permissions. Sites may be further configured in Content Server to leverage its features, such as workflow and publishing.

69.3 Getting Started

The chapters of this section of the guide can be read in any order:

- For information about the WEM Framework, see Chapter 70, "WEM Framework: Understanding the Framework and Services."
- For a demonstration of the "Articles" application, see Chapter 71, "WEM Framework: The Articles Sample Application."
- For information about the "Articles" application code, programmatic application registration, and cross-domain implementations, see Chapter 72, "WEM Framework: Developing Applications." An example of manual application registration is available in Chapter 78, "WEM Framework: Registering Applications Manually."
- For information about creating REST resources, see Chapter 73, "WEM Framework: Developing Custom REST Resources."
- For a demonstration of the SSO sample application, see Chapter 74, "WEM Framework: Single Sign-On for Production Sites."

- For information about system security, see Chapter 75, "WEM Framework: Using REST Resources."
- For information about customizing the login behavior for the WEM Framework, see Chapter 76, "WEM Framework: Customizable Single Sign-On Facility."
- For information about buffering, see Chapter 77, "WEM Framework: Buffering."

70

WEM Framework: Understanding the Framework and Services

The application developer's environment consists of the WEM Framework running on WebCenter Sites via REST services. Applications can be written in any language to make REST calls to WebCenter Sites. Custom-built applications can be deployed to an application server other than the platform's, and therefore written independently of the platform's deployment infrastructure.

This chapter contains the following sections:

70.1 Support for Application Development

Support for application development is in the following components (which are also described in their own sections in this chapter):

- **REST services,** a set of programmatic interfaces that provide access to the WebCenter Sites objects.
- **UI container,** which exposes registered applications. Registration enables rendering of the applications' interfaces. The UI container also supports the WEM Context object, used by applications to get details from the WEM Framework about the logged-in user and current site.
- **Single Sign-On (SSO),** which enables authenticated users to log in only once to access all applications allowed to them during the session. (The WebCenter Sites installation process installs the Central Authentication Service web application to support single sign-on in WEM.)
- **REST authorization model,** which provides fine-grained access control over REST resources, based on group membership. Application development does not directly involve authorization (which is configured graphically in WEM Admin and the WebCenter Sites Admin interface), except when a predefined user is specified in the code.

WEM Admin is also part of the WEM Framework, but seldom used in application development, mainly to test the results of the application registration process, or to obtain administrative information about sites, users, groups, and roles. Information about WEM Admin and the Web Experience Management Framework, is available in the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

70.2 REST Services

The REST API exposes the WebCenter Sites data model:

- Basic asset types and basic assets (read-write)
- Flex asset types and definitions (read only)
- Flex children and parents (read-write)
- Indexing to support asset searches

The following objects are also exposed by the REST API. They are used mainly by administrators in the authorization process (the objects are displayed in the WEM Admin interface):

- Sites (read-write)
- Users (read-write)
- Roles (read-write)
- ACLs (read only)
- Groups (read only), introduced in this release to control access to the REST layer.
- Auxiliary services: user locale and server time zone

(Sites, roles, and users can be configured in WEM Admin. ACLs and groups are exposed in WEM Admin (under Users) as read-only items; they must be configured in the WebCenter Sites Admin interface.)

Objects in WebCenter Sites map to REST resources in WEM. All other features, such as publishing, workflow, database management tools, and page caching must be accessed from the WebCenter Sites Admin interface or via JSP and XML tags.

Among the authorization objects that general administrators manage, sites and roles are the most likely candidates for application development, depending on your requirements. You can also specify "predefined" users to simplify administrators' authorization tasks.

- **Sites:** Using sites in application code is a requirement when the application's asset types and assets must be programmatically installed. The code must specify at least one site on which to enable the asset types (site-specific access to assets requires their asset types to be enabled on at least one site). Otherwise, you can install just the asset types (without naming any sites). Administrators will follow up by using the WebCenter Sites Admin interface to enable the asset types and assets on sites of their own choice.
- **Roles:** in WEM are used to manage access to applications. Sharing a role to a user and an application on the same site grants the user access to the application on that site. Roles can be used in application code to protect interface functions, such as `Edit`. The WebCenter Sites Admin interface exemplifies an application with role-protected interface functions.
- **Users:** The only user you are likely to specify in your application code is the "predefined" user, to simplify administrators' authorization processes. Specifying

the user involves coding a user name and password. Instead of authorizing all application users individually at the REST level, an administrator will authorize your predefined user. Permissions granted to the predefined user will be passed to the logged-in users when they access the application. More information about predefined users and the authorization model can be found in Section 70.5, "Authorization Model."

Keeping track of how sites and roles are used across the system is an administrators task that requires support from application developers. Tracking becomes especially important when the WebCenter Sites platform also functions as a staging system, only because the WEM Framework uses the WebCenter Sites database. For example, sites created in WEM Admin are stored in the database. They might not be used in WebCenter Sites for staging, but they are exposed in the WebCenter Sites Admin interface, along with its dedicated CM sites. Conversely, sites that are created in the WebCenter Sites Admin interface for CM purposes are exposed in WEM Admin, where other applications can be assigned to those sites. For users to be properly authorized, developers must communicate to administrators the nature of the custom-built applications: the resources they use, role-protected interface functions, and predefined users, if any.

70.3 UI Container

The UI container exposes registered applications and supports the Context object, used by applications to get information from the WEM Framework.

This section contains the following topics:

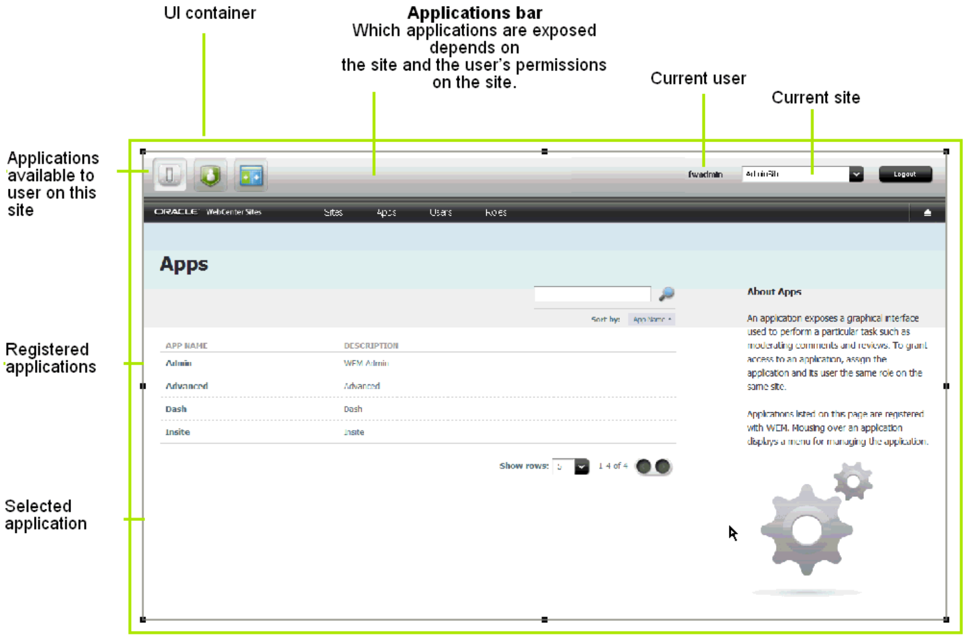
- Section 70.3.1, "Registration"
- Section 70.3.2, "WEM Context Object"

70.3.1 Registration

The purpose of registering an application is to expose the application in WEM Admin for administrators to manage and make available to other users. Registration allows the system to recognize the application as an asset, which in turn allows the system to

- list the application on the Apps page in WEM Admin (Figure 70–1),
- locate the icon you have chosen to represent the application,
- display the application's icon on the WebCenter Sites login page, and in the applications bar on each site to which the application is assigned (Figure 70–1), and
- render the application's interface when the application's icon is selected.

Figure 70–1 Registered Applications in UI Container



APP NAME	DESCRIPTION
Admin	WEM Admin
Advanced	Advanced
Dash	Dash
Insite	Insite

About Apps

An application exposes a graphical interface used to perform a particular task such as moderating comments and reviews. To grant access to an application, assign the application and its user the same role on the same site.

Applications listed on this page are registered with WEM. Mousing over an application displays a menu for managing the application.

Registering an application includes registering its views. While multiple and shared views are supported, applications with a single, unshared view are typical (and used in this guide). Views can be of type iframe, HTML, and JavaScript.

To support registration, the WEM Framework ships with the basic asset types `FW_Application` and `FW_View`. Both are created when the WEM option is selected during the WebCenter Sites installation process. They are enabled by default on AdminSite (also created during the WebCenter Sites installation process).

Registering an application (once it is deployed) requires creating an instance of `FW_Application`, creating an instance of `FW_View` for each view, and associating the `FW_View` instances with the `FW_Application` instance. Applications must be registered on AdminSite, even if they will be used on other sites. Registration allows applications to be assigned to other sites.

Applications can be registered either programmatically via the REST API's `applications` service, or manually from the WebCenter Sites Admin interface. Programmatic registration is preferred. For an example, see Section 71.2, "Launching the Articles Sample Application." For general instructions, see Section 72.6, "Registration Code." An example of manual registration is available in Chapter 78, "WEM Framework: Registering Applications Manually."

70.3.2 WEM Context Object

The UI container provides a JavaScript Context object (`WemContext`) to all applications inside the container. The Context object is used by the applications to get details from the WEM Framework about the logged-in user and site (for example, the current site's name from the UI container). The Context object also provides various utility methods that applications will use to share data. The Context Object can be used by applications running in the same domain as WebCenter Sites or in different domains. For more information, see Section 72.5, "Context Object: Accessing Parameters from the WEM Framework."

70.4 Single Sign-On

Single sign-on is implemented using Central Authentication Service (`http://www.jasig.org/cas`). As shown in the sample "Articles" example, the servlet filter that ships with the WEM Framework are ready-to-use for any application that is deployed as a Java web application. If your application is developed using a different technology, refer to CAS clients specific to your choice of technology, at the following URL:

`http://www.ja-sig.org/wiki/display/CASC/Official+Clients`

When a user tries to access an application protected by CAS, the authentication system responds with the steps below.

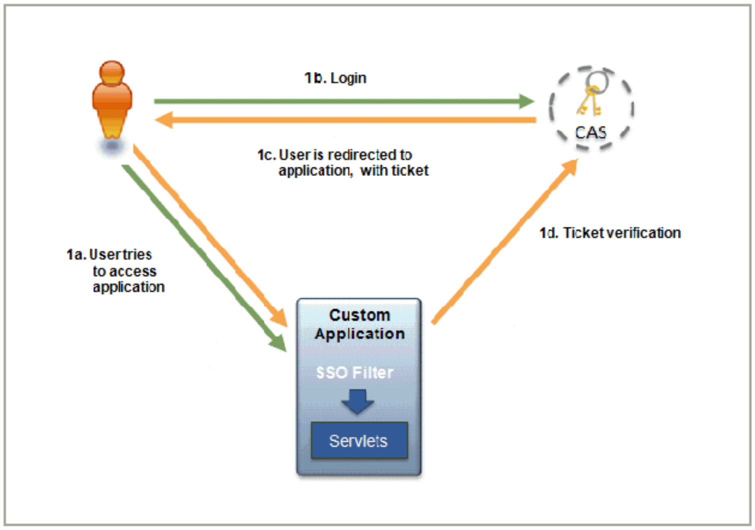
1. **Initial Access**

   a. When the user first attempts to access an application protected by CAS,

   b. the user is redirected to the CAS login page. Upon successful login,

   c. the user is redirected back to the application with a ticket. The cookie for the CAS login page is saved.

   d. The application verifies the user's identity by verifying the ticket against CAS. (On content management systems, CAS authenticates by default against the WebCenter Sites database.)


1b. Login
CAS
1c. User is redirected to application, with ticket
1d. Ticket verification
1a. User tries to access application
Custom Application
SSO Filter
Servlets

2. **Subsequent Access**

   a. When the user attempts to access another application protected by CAS, the user is redirected to the CAS login page.

   b. The cookie is retrieved from the request, implicit login is performed, and the login page is bypassed.

   c. The user is redirected back to the application with a ticket.

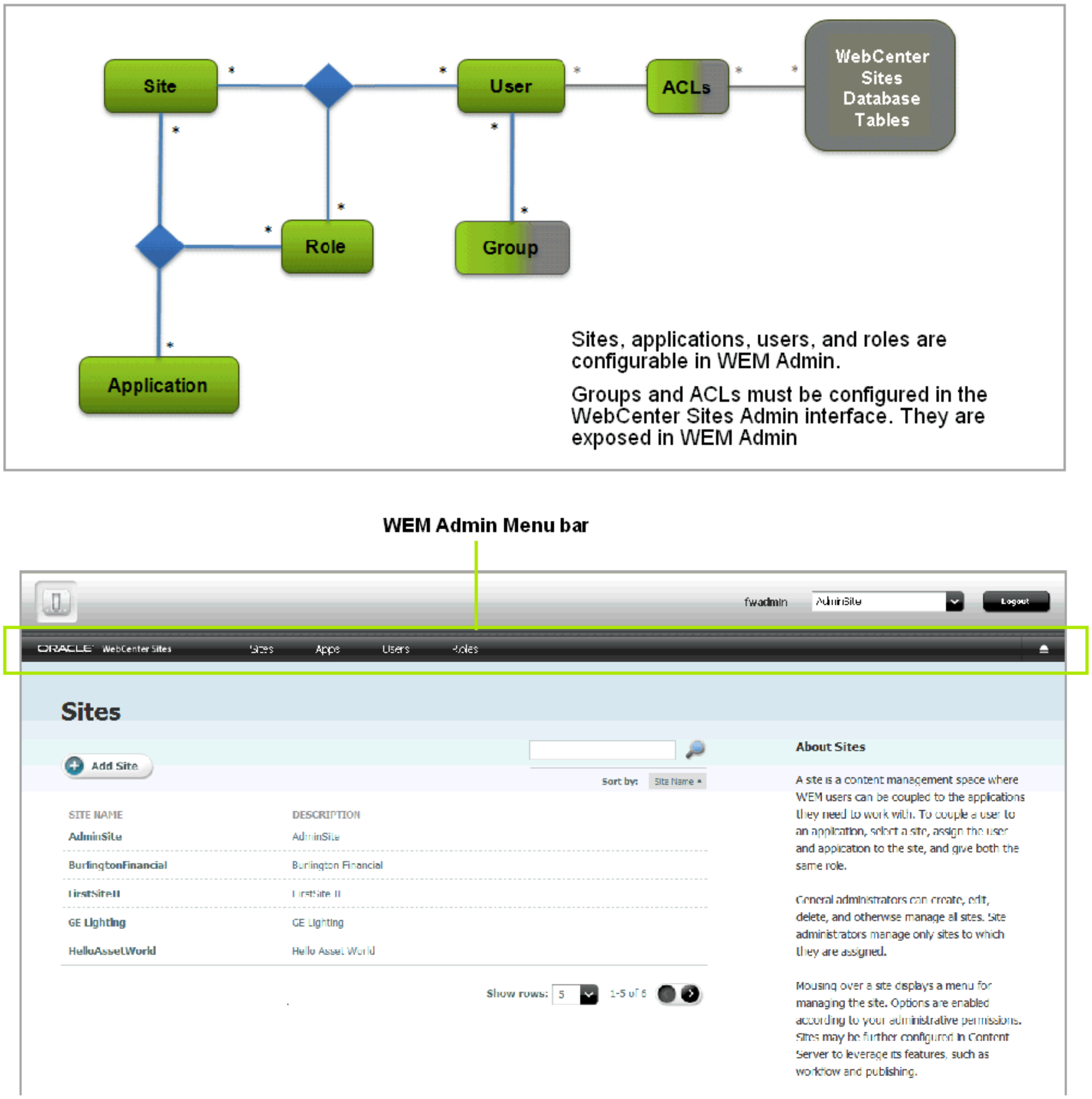
   d. The application verifies the user's identity by verifying the ticket against CAS.

70.5 Authorization Model

Authorization is the process of granting users access to applications. General administrators are responsible for authorization by using WEM Admin to couple objects as shown in Figure 70–2. Developers can simplify the administrator's task by coding a predefined user in their applications. How the user fits into the authorization model is explained below.

In Figure 70–2, Site, Application, User, and Role each have a counterpart menu option in WEM Admin. ACLs and groups are exposed on each user's page.

Figure 70–2 Authorization Model



Sites, applications, users, and roles are configurable in WEM Admin.

Groups and ACLs must be configured in the WebCenter Sites Admin interface. They are exposed in WEM Admin

Sites

SITE NAME	DESCRIPTION
AdminSite	AdminSite
BurlingtonFinancial	Burlington Financial
FirstSiteII	FirstSite II
GE Lighting	GE Lighting
HelloAssetWorld	Hello Asset World

About Sites

A site is a content management space where WEM users can be coupled to the applications they need to work with. To couple a user to an application, select a site, assign the user and application to the site, and give both the same role.

General administrators can create, edit, delete, and otherwise manage all sites. Site administrators manage only sites to which they are assigned.

Mousing over a site displays a menu for managing the site. Options are enabled according to your administrative permissions. Sites may be further configured in Content Server to leverage its features, such as workflow and publishing.

Authorization is managed at three levels: application, REST, and database.

- Application-level authorization requires sharing a role to a user and an application on the same site, which grants the user access to the application on that site. If interface functions are role-protected, their roles as well must be shared to the application users.
- REST-level authorization regulates the user's permission to operate on the application's resources, *assuming ACLs are correctly assigned*. REST-level authorization requires configuring groups with privileges to operate on objects that map to REST resources. Users who are assigned to a group gain the group's privileges.

Developers can define a user in their applications (by user name and password) to act as a proxy for logged-in users, which eliminates the need for administrators to configure REST security for each logged-in user. Once an application is deployed and registered, a general administrator authorizes its predefined user by: 1) configuring the predefined user in WEM Admin for application access, 2) configuring a group (in the WebCenter Sites Admin interface) with privileges to operate on the applications' resources, and 3) assigning the predefined user to the group (by using either the WEM Admin or the WebCenter Sites Admin interface). The group's privileges are passed to the predefined user and then to logged-in users when they access the application. Supported security configurations are described and listed in Section 75.3, "REST Authorization." The "Articles" sample application provided with the WEM Framework specifies a predefined user.

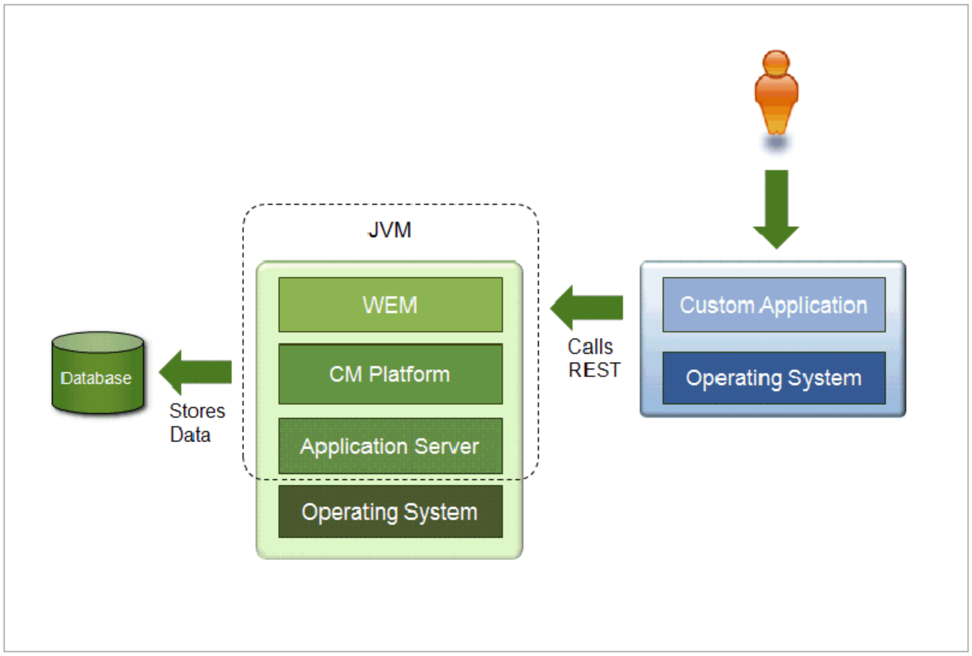
- At the database level, ACLs determine the individual user's access to the system, i.e., permission to log in and operate on the database, *regardless of the user's membership in any groups.* If a user lacks the appropriate ACLs and therefore permissions to the database tables, then membership in a group does not grant those permissions.

  Default ACLs give users almost unrestricted permissions, but not the means, to operate on objects in many of the database tables. Those permissions are modulated at the REST level: Either directly by the user's membership in groups (in the absence of a predefined user), or indirectly by the application's predefined user and his membership in groups. Modifying a group's privileges to operate on objects modifies the group member's privileges to operate on resources. The same user on the WebCenter Sites side remains unaffected by group memberships. Permissions to content are still regulated by ACLs and actuated by sites and roles.

70.6 Custom Applications

Custom applications developed in WEM are often implemented in a loosely coupled manner to the content management platform. Because custom applications utilize the REST API Web services and SSO mechanism enabled by the WEM Framework, they are often deployed to an application server other than the platform's application server. Developers can therefore write custom applications completely independently of the platform's deployment infrastructure. Most custom applications are deployed remotely (Figure 70–3).

Figure 70–3 Remote Application Deployment



Custom applications can be implemented as content management or delivery applications. We recommend getting started with the content management side, as it typically does not require much performance tuning effort.

The WEM Framework ships with several lightweight sample applications, which you can launch and analyze as models for developing your own applications. "Articles" illustrates a content management application. Chapter 71, "WEM Framework: The Articles Sample Application" contains instructions for launching "Articles." Specifications can be found in Chapter 72, "WEM Framework: Developing Applications," source code is provided in the WebCenter Sites `Misc/Samples` folder, and other supporting information is provided in the REST API resource and Bean references. The SSO sample application is for authentication on live sites and the "Recommendations" application illustrates the creation of REST resources.

70.7 Requirements for REST Resources

To authenticate all REST `POST/PUT/DELETE` requests as valid, each request requires a header with the `X-CSRF-Token` as the key and a value of either a CAS ticket (multi or single) or a `sessionid`.

71

WEM Framework: The Articles Sample Application

Articles is a simple content management application with richly documented source code and a self-installation process to help you quickly master information that is most important to developing applications. As the name implies, Articles enables the management of article assets.

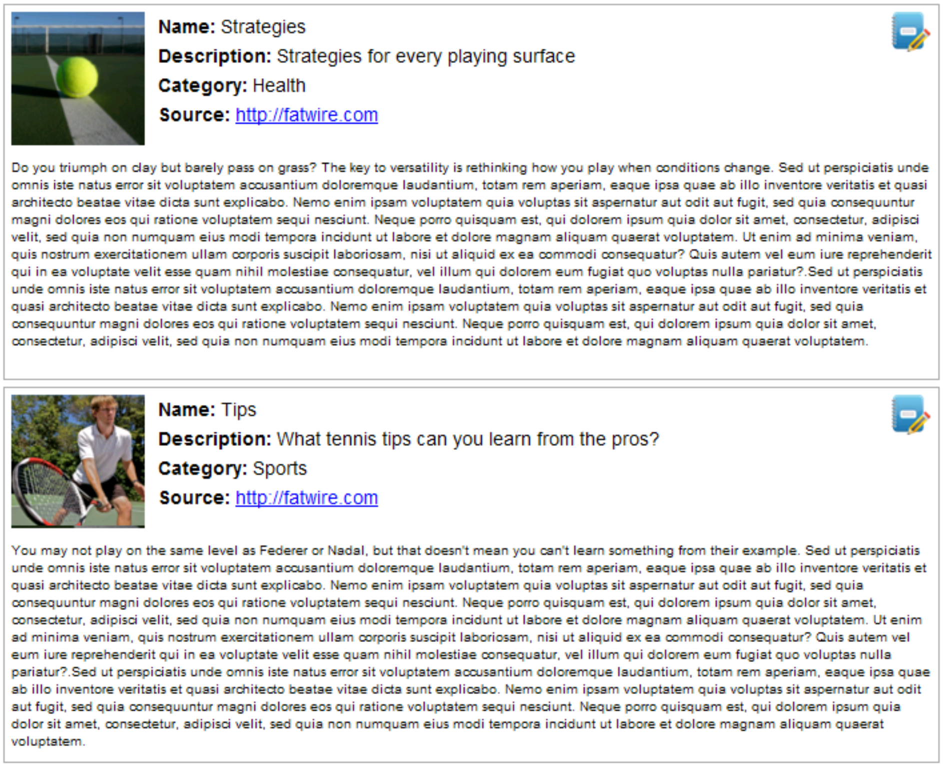
This chapter contains the following sections:

- Section 71.1, "Overview of the Sample Application"
- Section 71.2, "Launching the Articles Sample Application"
- Section 71.3, "Testing the Articles Application"

71.1 Overview of the Sample Application

The Articles home page displays two articles that can be edited directly in WEM, from the custom interface that you see in the figure above. The application demonstrates usage of the WebCenter Sites REST API to perform a search query from Java code and an asset modification query from JavaScript code. The Articles application and REST services can be run on different application servers. Cross-domain restrictions in JavaScript prevent AJAX calls directly from the Articles application to the REST services. This is why a simple ProxyController is introduced. It redirects calls from JavaScript to WEM REST Web Services. Custom implementations may reuse this controller implementation.

The application's home page looks like this:



Name: Strategies

Description: Strategies for every playing surface

Category: Health

Source: http://fatwire.com

Do you triumph on clay but barely pass on grass? The key to versatility is rethinking how you play when conditions change. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?.Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

Name: Tips

Description: What tennis tips can you learn from the pros?

Category: Sports

Source: http://fatwire.com

You may not play on the same level as Federer or Nadal, but that doesn't mean you can't learn something from their example. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?.Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

The Articles application is based on the Spring MVC framework. Articles includes a predefined administrative user named `fwadmin` with password `xceladmin`, who is assigned to the REST group named `RestAdmin`. The application's self-installer contains specifications for registering the Articles application and installing its asset model and sample articles. The application does not have internally configured sites or role-protected functions. It has a single, iframe view. Additional specifications are available in Chapter 71, "WEM Framework: The Articles Sample Application."

71.2 Launching the Articles Sample Application

In this section, you will first build and deploy the Articles application, then run the installer.

This section contains the following topics:

- Section 71.2.1, "Building and Deploying the 'Articles' Application"
- Section 71.2.2, "Registering the Articles Sample Application"

71.2.1 Building and Deploying the 'Articles' Application

1. Determine or create the site to which you will assign the sample articles application. The default site is FirstSiteII (a sample WebCenter Sites CM site). It is possible that FirstSiteII is not installed on your system.

   To select or create a site, log in to WEM Admin at the URL:

   `http://<server>:<port>/<cs_application_context>/login`

   using the credentials of a general administrator (`fwadmin` / `xceladmin` are the default values).

> **Note:** In step 5, you will specify the site you have chosen here, which will allow the installer to enable the application's asset model and assets on that site.

2. Download and install SUN JDK (1.5 or later) from the following URL:

   `http://java.sun.com/`

3. Download the latest Apache Ant from `http://ant.apache.org/` and place the Ant `bin` directory into the system `PATH`.
4. Copy `servlet-api.jar` to the Articles application `lib` folder. The `jar` file can be taken from your application server's home directory (for example, Tomcat's `servlet-api.jar` is located in the `home lib` directory).
5. Set the following parameters in the `applicationContext.xml` file (in `src\articles\src\main\webapp\WEB-INF\`):
   - `casUrl`: Specify the URL of the CAS application: `http://<server>:<port>/<context_path>`
   - `csSiteName`: Specify the name of the site that you selected in step 1.
   - `csUrl`: Specify the URL where the WebCenter Sites platform is running: `http://<server>:<port>/<context>`
   - `csUserName`: The default value is `fwadmin`. This is the application's predefined user, a general administrator with membership in the `RestAdmin` group which has unrestricted permissions to REST services. If you specify a different user, you must name a user equivalent to `fwadmin`. For instructions on creating a general administrator, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.
   - `csPassword`: Specify the predefined user's password.
   - `articlesUrl`: Point to the URL where the sample application will be accessed.
6. Run the Ant build with the default target (enter **ant** on the command line).
7. Deploy the resulting `target/articles-1.0.war` to an application server.

   On deployment, the following content is copied from source to target: The contents of the `lib` folder are copied to `/WEB-INF/lib`. The contents of the `resources` folder are copied to `/WEB-INF/classes/`. For information about the structure of the source application, see Chapter 72, "WEM Framework: Developing Applications."

71.2.2 Registering the Articles Sample Application

The Articles application has a self-installer, which starts running when you log in to the `install.app` page. The installer registers the sample application (including the view) and creates its data model and assets in the WebCenter Sites database.

> **Note:** Specifications for the registration asset types `FW_View` and `FW_Application` can be found in the *Oracle Fusion Middleware WebCenter Sites REST API Bean Reference* and in Chapter 78, "WEM Framework: Registering Applications Manually".

To run the Articles installer

1. Navigate to the `install.app` page:

   ```
   http://<hostname>:<portnumber>/<context_path>/install.app
   ```

   For example:

   ```
   http://localhost:9080/articles-1.0/install.app
   ```

2. Use any credentials to log in (the application's predefined user, specified by `csUserName` and `csPassword` (see Section 71.2.1, "Building and Deploying the 'Articles' Application"), provides you with permissions to the application. The sample application does not perform authorization checks as it does not use roles.)

3. The self-installation process invokes `InstallController.java`, which first registers the application (including the view, in an application Bean), then writes the sample asset type and assets to the database.

   a. `InstallController.java` registers the Articles application with the WEM Framework:

      – `InstallController.java` creates an application asset named `Articles` (asset type `FW_Application`) in the WebCenter Sites database.

        The **`iconurl`** attribute points to the URL where the icon representing the application is located.

        The **`layouturl`** attribute specifies the URL of the `layout.app` page (implemented by `LayoutController.java`). The `layout.app` page defines the application layout.

        The **`layouttype`** attribute takes the default (and only) value: `layoutrenderer`. Using the `layoutrenderer` value, the UI container is responsible for rendering the application's associated views by using the `layout.app` page, specified by **`layouturl`**.

      – `InstallController.java` creates a view asset named `ArticlesView` (asset type `FW_View`) in the WebCenter Sites database. The association between the view asset and the application asset is made through the `views` attribute in the `FW_Application` asset type.

   b. `InstallController.java` installs the application's asset model and sample assets:

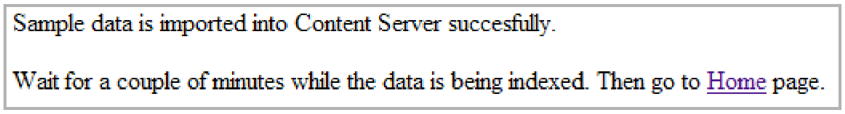
      – Creates the application's `FW_Article` asset type in the WebCenter Sites database. (`FW_Article` is a basic asset type defined in `InstallController.java`.)

      – Enables the `FW_Article` asset type on the site that was specified in the `csSiteName` parameter in `applicationContext.xml` (step 5 of Section 71.2.1, "Building and Deploying the 'Articles' Application").

      – Writes the two sample article assets to the `FW_Article` asset type tables. (The articles' text and images are stored in: `/sample app/articles/src/main/resources/install`)

   c. `InstallController.java` creates an asset type-based index to support searches on assets of type `FW_Article`. (The controller specifies index configuration data.)

4. When the installation process completes successfully, `InstallController.java` displays the following page (at `http://<server>:<port>/articles/install.app`), where **Home** is `home.app`:



Sample data is imported into Content Server succesfully.

Wait for a couple of minutes while the data is being indexed. Then go to Home page.

71.3 Testing the Articles Application

1. Navigate to the `home.app` page:

   ```
   http://<hostname>:<portnumber>/<context_path>/home.app
   ```

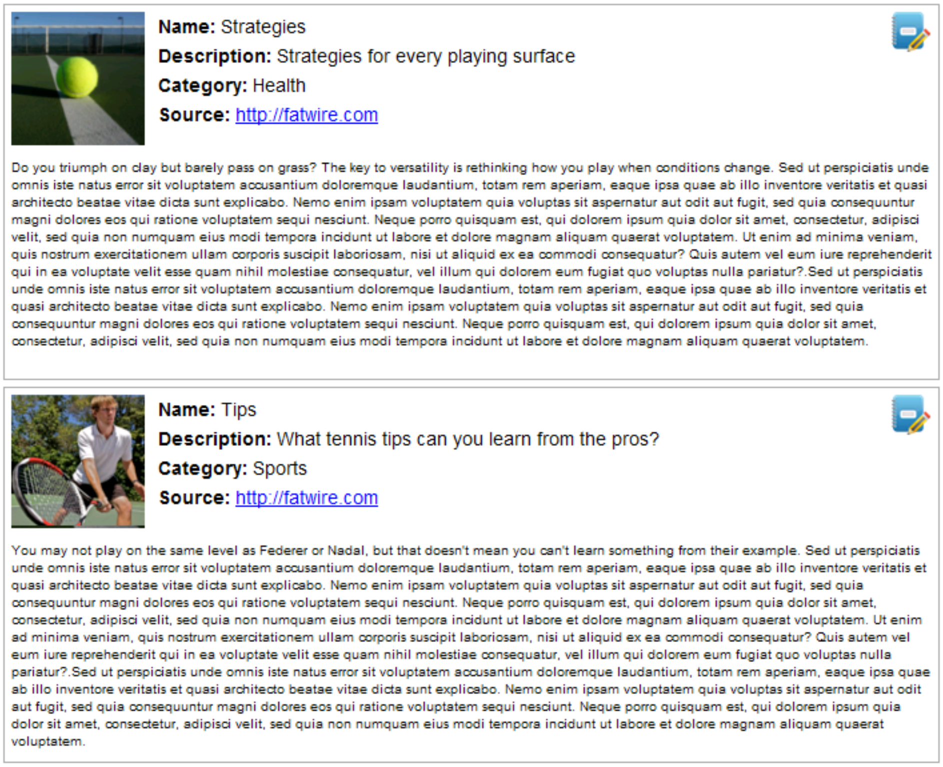
   For example:

   ```
   http://localhost:8080/articles-1.0/home.app
   ```

2. Use any credentials to log in (the application's predefined user, specified by `csUserName` and `csPassword` (see Section 71.2.1, "Building and Deploying the 'Articles' Application"), provides you with permissions to the application. The sample application does not perform authorization checks as it does not use roles.)

   WEM displays the application's home page:

Figure 71–1 Articles Home Page



Name: Strategies

Description: Strategies for every playing surface

Category: Health

Source: http://fatwire.com

Do you triumph on clay but barely pass on grass? The key to versatility is rethinking how you play when conditions change. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?.Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

Name: Tips

Description: What tennis tips can you learn from the pros?

Category: Sports

Source: http://fatwire.com

You may not play on the same level as Federer or Nadal, but that doesn't mean you can't learn something from their example. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?.Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

3. If you wish to experiment with this application (for example assign it to other sites and add users), use WEM Admin. For more information on the Web Experience Management Framework, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

72

WEM Framework: Developing Applications

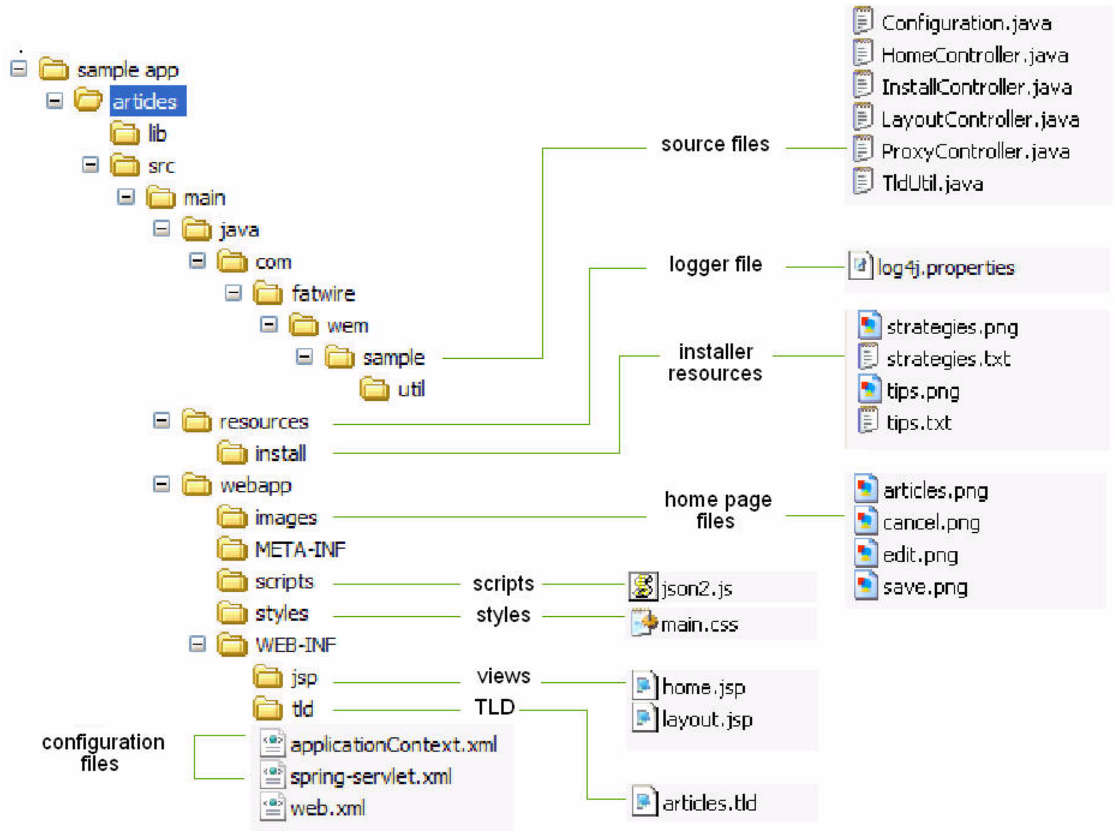
The "Articles" sample application is used throughout this chapter to illustrate the basic architecture of an application that makes REST calls.

This chapter contains the following sections:

72.1 Application Structure

Figure 72–1 shows the source structure of the "Articles" sample application. On deployment, the following directories are copied from source to target: The contents of the `lib` directory are copied to `/WEB-INF/lib/`. The contents of the `resources` directory are copied to `/WEB-INF/classes/`.

Figure 72–1 'Articles' Sample Application Source Structure



"Articles" is a Java Web application developed on Spring MVC. The following pages are available:

- `/install.app` is the "Articles" installation page, which also displays a confirmation message when the application is successfully installed
- `/home.app` is the home page of the "Articles" application (Figure 72–3).

72.2 Configuration Files

- `applicationContext.xml` (in `/WEB-INF/`) holds SSO and application-specific configurations (such as a predefined user and the site on which to enable the data model and assets).
- `spring-servlet.xml` (in `/WEB-INF/`) is the default Spring configuration file. This file stores the Spring configuration and references the following controllers (described in the "Source Files" section):
  - `HomeController`
  - `InstallController`
  - `LayoutController`
  - `ProxyController`
  - `log4j.properties` (in `/resources/`) is the logging configuration file. On application deployment, it is copied from `/resources/` to `/WEB-INF/classes/`.

Source Files

/sample app/articles/src/main/java/

The /sample/ folder contains the source files listed below:

- Configuration.java is populated (by the Spring framework) from the applicationContext.xml file (described in Section 72.2, "Configuration Files").
- HomeController.java is the home page controller, which renders a single home page. This controller reads the list of sample articles from the WebCenter Sites platform using the REST API and displays them on the home page.

  The sample articles consist of images and text, stored in /sample app/articles/src/main/resources/install. The sample articles are installed in the WebCenter Sites database by InstallController.java.
- InstallController.java registers the "Articles" application, and writes the application's asset model and sample assets to the database
- LayoutController.java displays the application's layout page (layout.app) used by the WEM UI framework. LayoutController.java is also used during the application registration procedure.
- ProxyController.java delegates AJAX requests to the WebCenter Sites REST servlet.
- TldUtil.java utility class contains TLD function implementations.

Installer Resources

/sample app/articles/src/main/resources/install

The /install/ folder contains the following resources, used by the InstallController to construct the home page (Figure 72–3):

- strategies.png
- strategies.txt
- tips.png
- tips.txt

Home Page Files

/sample app/articles/src/main/webapp/images

The /images/ folder contains:

- articles.png icon (Figure 72–2), which represents the 'Articles" application in the applications bar
- In Figure 72–3:
  - edit.png is the icon for the **Edit** function
  - save.png is the icon for the **Save** function
  - cancel.png is the icon for the **Cancel** function

Scripts

/sample app/articles/src/main/webapp/scripts

The /scripts/ folder contains the json2.js utility script, used to convert strings to and from JSON objects.

Styles

/sample app/articles/src/main/webapp/styles

The /styles/ folder contains main.css, which specifies CSS styles used by this Web application.

Views

/sample app/articles/src/main/WEB-INF/jsp

The /jsp/ folder contains:

- home.jsp, which is used to render the home page view of the "Articles" application (Figure 72–3)
- layout.jsp, which defines the application layout

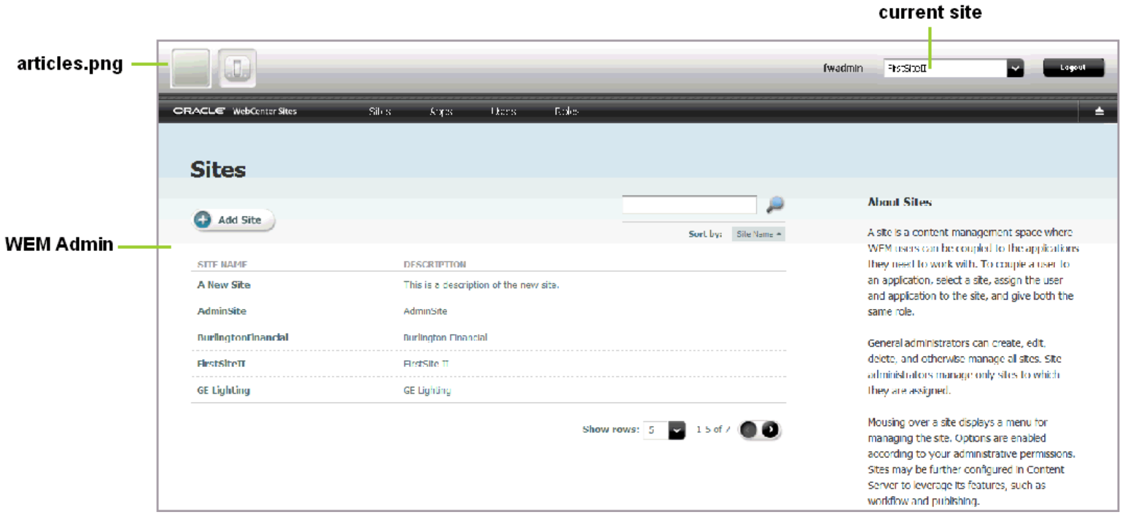
WEB-INF

/sample app/articles/src/main/WEB-INF

The /WEB-INF/ folder contains:

- articles.tld, the TLD declaration file
- spring-servlet.xml, the Spring configuration file
- web.xml, the Web application deployment descriptor

Figure 72–2 'Articles' Icon (`articles.png`)



Sites

SITE NAME	DESCRIPTION
A New Site	This is a description of the new site.
AdminSite	AdminSite
BurlingtonFinancial	Burlington Financial
FirstSiteII	FirstSite II
GE Lighting	GE Lighting

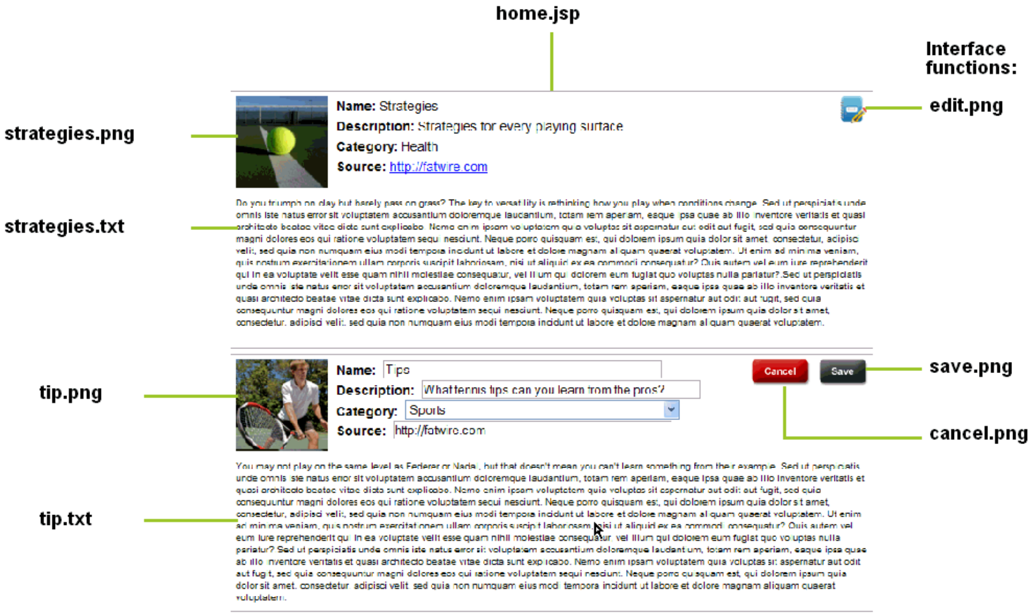
About Sites

A site is a content management space where WEM users can be coupled to the applications they need to work with. To couple a user to an application, select a site, assign the user and application to the site, and give both the same role.

General administrators can create, edit, delete, and otherwise manage all sites. Site administrators manage only sites to which they are assigned.

Mousing over a site displays a menu for managing the site. Options are enabled according to your administrative permissions. Sites may be further configured in Content Server to leverage its features, such as workflow and publishing.

Figure 72–3 'Articles' Home Page



Name: Strategies
Description: Strategies for every playing surface
Category: Health
Source: http://fatwire.com

Do you triumph on clay but barely pass on grass? The key to versatility is rethinking how you play when conditions change. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?.Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

Name: Tips
Description: What tennis tips can you learn from the pros?
Category: Sports
Source: http://fatwire.com

You may not play on the same level as Federer or Nadal, but that doesn't mean you can't learn something from their example. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur? Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

72.3 Making REST Calls

WebCenter Sites REST resources support two types of input and output formats: XML and JSON. To get the desired return formats, you will need to set HTTP headers that specify the MIME type `application/xml` or `application/json`.

For example, when specifying input format to be XML, set `Content-Type` to `application/xml`. When specifying the output format, set `Accept` (the expected format) to `application/xml`. If other output formats are specified, they will be ignored. The default is `XML`, if not specified in `Content-Type` or `Accept` (for sample code, see lines 64 and 66 in Section 72.3.1, "Making REST Calls from JavaScript").

For more detailed information about REST calls, see the following topics in this section:

- Section 72.3.1, "Making REST Calls from JavaScript"
- Section 72.3.2, "Making REST Calls from Java"

72.3.1 Making REST Calls from JavaScript

The following code (in `home.jsp`) performs AJAX calls to the asset REST services to save asset data. Note that the request is actually performed to the proxy controller which redirects the request to the destination REST service.

> **Note:** We use the JSON stringify library (`http://json.org/js.html`) to serialize a JavaScript object as a string. It is much more convenient to write JSON objects instead of strings.

```
1.  // Form the URL pointing to the asset service
2.  // to the proxy controller, which will redirect this request to the CS REST
    servlet.
3.  var idarr = assetId.split(":");
```

```
var assetUrl =
"${pageContext.request.contextPath}/REST/sites/${config.csSiteName}/types/" +
idarr[0] + "/assets/" + idarr[1];

// For the data object to be posted.
var data =
{
 "attribute" :
[
{
   "name" : "source",
  "data" :
 {
    "stringValue" : document.getElementById("source_e_" + assetId).value
}
},
{
  "name" : "cat",
 "data" :
 {
"stringValue" : document.getElementById("cat_e_" + assetId).value
}
}
],
 "name" : document.getElementById("name_e_" + assetId).value,
 "description" : document.getElementById("desc_e_" + assetId).value,
 // This should be removed.
 "publist" : "${config.csSiteName}"
};
// Convert JSON data to string.
var strdata = JSON.stringify(data);

// Perform AJAX request.
var req = getXmlHttpObject();
req.onreadystatechange = function ()
{
if (req.readyState == 4)
{
if (req.status == 200)
 {
 // On successful result
// update the view controls with new values and switch the mode to 'view'.
```

```
  for (c in controls)
{
document.getElementById(controls[c] + "_v_" + assetId).innerHTML =
document.getElementById(controls[c] + "_e_" + assetId).value;
}
switchMode(assetId, false);
}
else
{
// Error happened or the session timed out,
// reload the current page to re-acquire the session.
alert("Failed to call " + assetUrl + ", " + req.status + " " + req.statusText);
window.location.reload( false );
}
}
};
// We put Content-Type and Accept headers
// to tell CS REST API which format we are posting
// and which one we are expecting to get.
req.open("POST", assetUrl, true);
req.setRequestHeader("Content-Type", "application/json;charset=utf-8");
req.setRequestHeader("Content-Length", strdata.length);
req.setRequestHeader("Accept", "application/json");
req.send(strdata);
}
```

72.3.2 Making REST Calls from Java

The code below (in `HomeController.java`) calls the assets search service to list all assets of type `FW_Article`. The code uses the Jersey Client library passing objects from the `rest-api-<version>.jar` library provided by the WEM Framework. This way we leverage strong typing in Java.

It is important to note that a token must be acquired from Java code by calling the `SSOAssertion.get().createToken()` method. It is unnecessary to do so in JavaScript as that side is already authenticated against WEM SSO.

```
// Use Jersey client to query CS assets.
Client client = Client.create();
String url = config.getRestUrl() + "/types/FW_Article/search";
WebResource res = client.resource( url );

// Construct URL and add token (for authentication purposes)
// and fields (specify which fields to retrieve back) parameters.
res = res.queryParam("fields",
URLEncoder.encode("name,description,content,cat,source", "UTF-8"));
res = res.queryParam("ticket",
SSO.getSSOSession().getTicket(res.getURI().toString(), config.getCsUsername(),
config.getCsPassword()));
```

```
// Put Pragma: auth-redirect=false to avoid redirects to the CAS login page.
Builder bld = res.header("Pragma", "auth-redirect=false");

// Make a network call.
AssetsBean assets = bld.get(AssetsBean.class);
```

Note: The custom `Pragma: auth-redirect=false` header instructs the CAS SSO filter not to redirect to the CAS sign-in page, but to return a 403 error instead, when no ticket is supplied or the supplied ticket is invalid.

72.4 Constructing URLs to Serve Binary Data

The "Articles" application leverages the Blob server in WebCenter Sites to serve BLOB data. The following utility function could be used to construct the URL pointing to the binary data for a given attribute in a given asset, where `blobUrl` points to the Blob server (`http://localhost:8080/cs/BlobServer` by default).

```
public String getBlobUrl(String assetType, String assetId, String attrName, String
contentType)
throws Exception
{
String contentTypeEnc = URLEncoder.encode(contentType, "UTF-8");

return blobUrl + "?" +
"blobkey=id" +
"&blobnocache=true" +
"&blobcol=thumbnail" +
"&blobwhere=" + assetId +
"&blobtable=" + assetType +
"&blobheader=" + contentTypeEnc +
"&blobheadername1=content-type" +
"&blobheadervalue1=" + contentTypeEnc;
    }
```

An alternative way to get binary data is to load an asset using the resource `/sites/{sitename}/types/{assettype}/assets/{id}`. When loaded, the asset will contain the URL pointing to the BLOB server.

72.5 Context Object: Accessing Parameters from the WEM Framework

The UI container provides a JavaScript Context object (`WemContext`) to all applications inside the container. The Context object is used by the applications to get details from the WEM Framework about the logged-in user and site (typically, to get the current site's name from the UI container). The Context object also provides various utility methods that the applications will use to share data. The Context Object can be used by applications running in the same domain as WebCenter Sites or in different domains.

Note: The `wemcontext.html` file lists the exposed methods, summarized in Section 72.5.3, "Methods Available in Context Object."

This section contains the following topics:

- Section 72.5.1, "Same Domain Implementations"
- Section 72.5.2, "Cross-Domain Implementations"
- Section 72.5.3, "Methods Available in Context Object"

72.5.1 Same Domain Implementations

To initialize and use Context Object for applications in the WebCenter Sites domain:

1. Include `wemcontext.js` (line 1 in the sample code below; `wemcontext.js` is located in `<cs webapp path>/wemresources/js/WemContext.js`).
2. Retrieve an instance of the `WemContext` object (line 3).
3. Use the methods of `WemContext` (lines 4 and 5).

Example 72–1 Sample Code for Same-Domain Implementations

```
1. <script
   src='http://<csinstalldomain>/<contextpath>/wemresources/js/WemContext.js'></sc
   ript>
2. <script type="text/javascript">
3. var wemContext = WemContext.getInstance(); // Instantiate Context Object
4. var siteName = wemContext.getSiteName(); // Get Site Name
5. var userName = wemContext.getUserName(); // Get UserName
6. </script>
```

72.5.2 Cross-Domain Implementations

To initialize and use Context Object for cross-domain applications:

1. Copy `wemxdm.js, json2.js, and hash.html` (from the `Misc/Samples` folder) to your application.
2. Open the `sample.html` file and make the following changes to perform cross-domain calls:
   a. Change the paths of `wemxdm.js` and `json.js` and `hash.html` to their paths in the application (see lines 1 through 4 in the code below).
   b. Change the path of `wemcontext.html` to its location in WebCenter Sites (`wemcontext.html` is located under `/wemresources/wemcontext.html`. Use the WebCenter Sites host name and context path. See line 14.)
   c. In the interface declaration, specify methods that will be used in the framework (line 15).
   d. Implement those methods in the local scope and invoke the remote method (line 30).

Example 72–2 sample.html for Cross-Domain Calls

```
1. <script type="text/javascript" src="../js/wemxdm.js"></script>
2. <script type="text/javascript">
3. //  Request the use of the JSON object
4. WemXDM.ImportJSON("../js/json2.js");
5. var remote;
6.
```

```
window.onload = function() {
// When the window is finished loading start setting up the interface
remote = WemXDM.Interface(/** The channel configuration */
{
// Register the url to hash.html.
local: "../hash.html",
// Register the url to the remote interface
remote: "http://localhost:8080/cs/wemresources/wemcontext.html"
}, /** The interface configuration */
{
remote: {
getSiteName :{},
...

}
},/**The onReady handler*/ function(){
// This function will be loaded as soon as the page is loaded
populateAttributes();
});
}
</script>

<script type="text/javascript">
/** Define local methods for accessing remote methods  */
function getSiteName(){
remote.getSiteName(function(result){
alert("result = " + result);
});
}
 ...
</script>
```

72.5.3 Methods Available in Context Object

Table 72–1 Methods Available in Context Object

Return Type	Method name and Description
`Object`	`getAttribute`*`(attributename)`* Returns attribute value for the given attribute name.
`Object`	`getAttributeNames()` Returns all the attribute names.

Table 72–1 (Cont.) Methods Available in Context Object

Return Type	Method name and Description
`Object`	`getCookie(name)` Returns cookie value for the given name. Has all restrictions of the normal browser cookie.
`Object`	`getCookies()` Returns all the cookies.
`Object`	`getLocale()` Returns locale.
`Object`	`getSiteId()` Returns the site id.
`Object`	`getSiteName()` Returns the site name.
`Object`	`getUser()` Returns user object.
`Object`	`getUserName()` Returns user name.
`void`	`removeCookie`(name, properties) Removes cookie.
`void`	`setAttribute(attributename, attributevalue)` Sets attribute. These attributes can be accessed in other applications.
`void`	`setCookie(name,value,expiredays,properties)` Sets the cookie.

72.6 Registration Code

Registration exposes applications in the WEM Framework, as explained in Section 70.5, "Authorization Model." Registering an application creates an asset of type `FW_Application` and an asset of type `FW_View` for each view associated with the application. The asset types are enabled on AdminSite. Their attributes are defined in the *Oracle Fusion Middleware WebCenter Sites REST API Bean Reference*. Programmatic registration is the preferred method. For an example of manual registration, see Chapter 78, "WEM Framework: Registering Applications Manually."

This section contains the following topics:

- Section 72.6.1, "Registering Applications with an iframe View"
- Section 72.6.2, "Registering Applications with JavaScript and HTML Views"

72.6.1 Registering Applications with an iframe View

The section uses code from the "Articles" sample application to illustrate the registration process. "Articles" has a single view of type iframe. The same steps apply to JavaScript and HTML views.

To register an application

1. Create or get an icon to represent your application. (The icon will be displayed in the applications bar.)

   (The "Articles" sample application uses the `articles.png` image file located in: `/sample app/articles/src/main/webapp/images/`)

2. Create a file that specifies the layout of the application in HTML, that is, for each view, create a placeholder element to hold the content rendered by the view. Applications and views are related as shown in Figure 72–4.

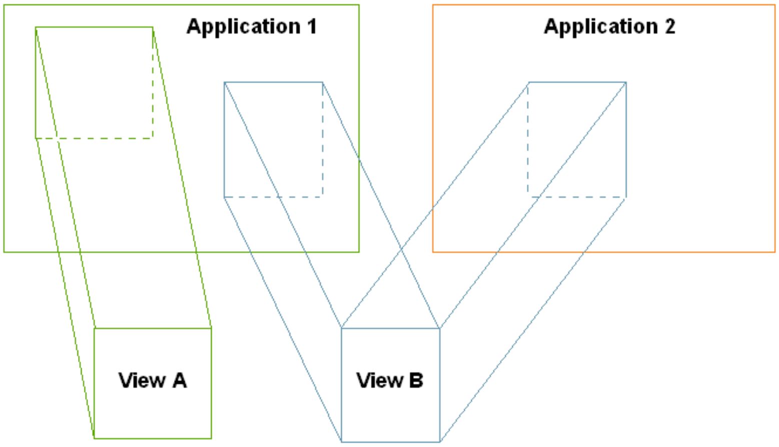
   For example, `layout.jsp` for the "Articles" sample application contains the following line:

```
<div id="articles" style="float:left;height:100%;width:100%"
class="wemholder"></div>
```

The view's content will be rendered within the placeholder element when the application is displayed (`layout.app` renders the application's layout; `home.app` renders the view).

> **Note:** When creating the layout file, specify a unique `id` for the placeholder element. You will specify the same `id` for the `parentnode` attribute when coding the view object. Use `class="wemholder"` for the placeholder elements.

Figure 72–4 Applications and Views



The relationship between applications and views is many-to-many (Figure 72–4). One application can have multiple views and each view can be used by many applications. Only registered views can be shared (through their asset IDs). If the asset ID is omitted, the view will be created within the context of its application. In the basic case, an application has only one view associated with it.

1. Invoke the `PUT wem/applications/{applicationid}` REST service and specify your application bean. Populate the bean with the view asset and application asset.

   For an iframe view, use the code of the "Articles" sample application, i.e., `InstallController.java` (locate the comment lines `// Create a new view object` and `// Create a new application object`). Set the `layouturl` attribute to specify the URL of the application's layout page.

In the "Articles" application, the `layouturl` attribute points to the URL of `layout.app` (implemented by `LayoutController.java`):

```
app.setLayouturl(config.getArticlesUrl() + "/layout.app");
```

You can test the results of your registration process by logging in to the WEM Admin interface as a general administrator and selecting **Apps** on the menu bar. Your application should be listed on that page.

72.6.2 Registering Applications with JavaScript and HTML Views

For applications that use HTML and JavaScript views, follow the steps in the previous section, but use the sample code and attributes listed below:

- Section 72.6.2.1, "JavaScript View"
- Section 72.6.2.2, "HTML View"

72.6.2.1 JavaScript View

> **Note:** JavaScript specified in the view will be rendered (executed) when the application is rendered. Make sure that the JavaScript does not conflict with other views.

Sample code:

```
window.onload = function () {
   if (GBrowserIsCompatible()) {
     var map = new GMap2(document.getElementById("map_canvas"));
     map.setCenter(new GLatLng(37.4419, -122.1419), 13);
     map.setUIToDefault();
   }
}
```

- **Rendering the** JavaScript vi**ew from a source URL**

  Set the following attributes:

  – `name`: Name of the view

  – `parentnode`: ID of the placeholder element (from step 2 in Section 72.6.1, "Registering Applications with an iframe View") .

  – `viewtype`: `fw.wem.framework.ScriptRenderer`, which renders JavaScript into the placeholder element.

  – `sourceurl`: Path of the `.js` file, which provides content for the view. For example: `http://example.com:8080/js/drawTree.js`

- **Rendering the** JavaScript **view from source code**

  Set the following attributes:

  – `name`: Name of the view

  – `parentnode`: ID of the placeholder element (from step 2 in Section 72.6.1, "Registering Applications with an iframe View").

  – `viewtype`: `fw.wem.framework.ScriptRenderer`, which renders JavaScript into the placeholder element

- javascriptcontent: JavaScript code (sample provided above. The code must not contain <script> tags.)

72.6.2.2 HTML View

Note: HTML specified in the view will be rendered (executed) when the application is rendered.

Sample code:

```
<object width="480" height="385">
 <param name="movie" value="http://www.localhost:8080/jspx/flash_slider_
main.swf"></param>
 <param name="allowFullScreen" value="true"></param>
 <embed src=" http://www.localhost:8080/jspx/flash_slider_main.swf"
  type="application/x-shockwave-flash" allowscriptaccess="always"
allowfullscreen="true"
  width="480" height="385">
 </embed>
</object>
```

- **Rendering the HTML view from a source URL**

  Set the following attributes:

  - name: Name of the view
  - parentnode: ID of the placeholder element (from step 2 in Section 72.6.1, "Registering Applications with an iframe View").
  - viewtype: fw.wem.framework.IncludeRenderer, which renders JavaScript into the placeholder element
  - sourceurl: Path to the HTML file that provides content for the view. For example: http://example.com:8080/js/drawTree.jsp

- **Rendering the HTML view from source code**

  Set the following attributes:

  - view: Name of the view
  - parentnode: ID of the placeholder element (from step 2 in Section 72.6.1, "Registering Applications with an iframe View").
  - viewtype: fw.wem.framework.IncludeRenderer, which renders JavaScript into the placeholder element
  - includecontent: HTML content (sample provided above. The code must not contain <html> or <body> tags.

73

WEM Framework: Developing Custom REST Resources

This chapter contains the following sections:

- Section 73.1, "The Recommendations Sample Application"
- Section 73.2, "Creating REST Resources"

73.1 The Recommendations Sample Application

The "Recommendations" sample application demonstrates how to create REST resources for WebCenter Sites and Satellite Server. The application registers a new REST resource `sample/recommendations/<id>` with `GET` and `POST` operations, which allow for retrieval and modification of static list recommendations. The application also demonstrates how it is possible to leverage the Satellite Server caching system.

This section contains the following topics:

- Section 73.1.1, "Building and Deploying the Application"
- Section 73.1.2, "Testing the Application"

73.1.1 Building and Deploying the Application

1. The "Recommendations" sample application is located in the `Misc/Samples` folder under your WebCenter Sites installation directory. Navigate to `recommendations` and edit the `build.properties` file. Specify the correct paths for `cs.webapp.dir` and `ss.webapp.dir` properties.
2. Run Apache `ant` while in the `recommendations` folder. This will build and deploy your sample application.
3. Launch the `catalogmover` application. Use the **Server**, **Connect** menu to connect to WebCenter Sites. Go to **Catalog, then Auto Import Catalog(s)** and select `src\main\schema\elements.zip` file. Append `xceladmin`, `xceleditor` when specifying the list of ACLs.
4. Go to the WebCenter Sites web application folder. Edit the `WEB-INF/classes/custom/RestResource.xml` file. Uncomment `recommendationService`, `recommendationConfig` and `resourceConfigs` beans.
5. Go to the Satellite Server web application folder. Edit `WEB-INF/classes/custom/RestResource.xml` file. Uncomment `recommendationService`, `recommendationConfig`, and `resourceConfigs` beans.
6. Restart both WebCenter Sites and Satellite Server.

73.1.2 Testing the Application

Use the existing static list recommendation id (or create a new recommendation) for the URL `http://<hostname>:<port>/<contextpath>/REST/sample/recommendations/<recommendationid>`. Use the same URL for both WebCenter Sites and Satellite Server installations. For example, use `http://localhost:8080/cs/REST/sample/recommendations/1266874492697`. See the XML response for both WebCenter Sites and Satellite Server.

73.2 Creating REST Resources

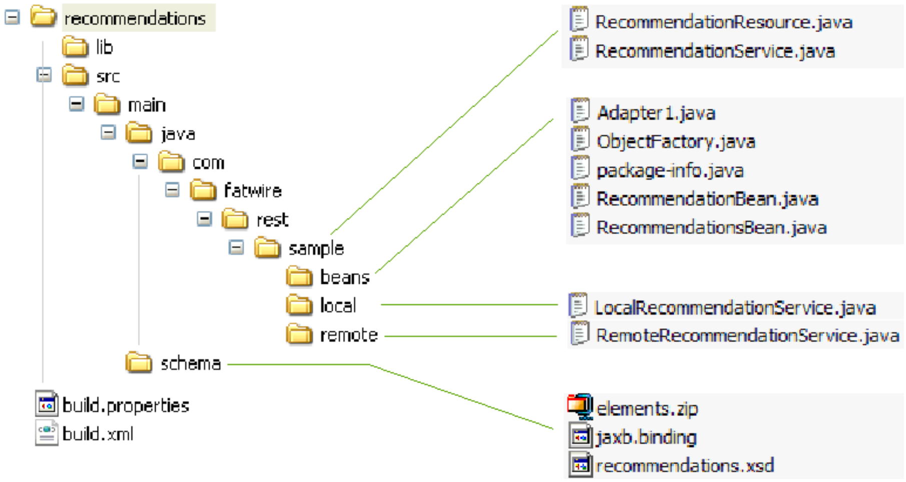
This section contains the following topics:

- Section 73.2.1, "Application Structure"
- Section 73.2.2, "Steps for Implementing Custom REST Resources"

73.2.1 Application Structure

The "Recommendations" sample application was created to guide you through the process of creating your own REST resources.

Figure 73–1 Recommendations" Sample Application



- Schema files: `src/main/schema`
  - `elements.zip` contains a sample element, which is used by Satellite Server for caching purposes.
  - `jaxb.binding` is a customization for the default JAXB bindings used during the bean generation process.
  - `recommendation.xsd` is an XML schema for the `RecommendationService` beans.
- Java source files: `src/main/java/ ... /sample`
  - `RecommendationResource` contains the REST resource implementation. It is used on both WebCenter Sites and Satellite Server.
  - `RecommendationService` is an interface that provides the functionality for the `RecommendationResource` class. It is implemented differently, depending on

where the resource is hosted: locally (on WebCenter Sites) or remotely (on Satellite Server).

- `beans/*` classes are generated using Java `xjc` compiler. They are pre-packaged with the application. If you want to regenerate beans (i.e., when changing the `recommendation.xsd` file) you can run "`generate`" ant's task from `build.xml`.
- `LocalRecommendationService` is a local (WebCenter Sites) implementation for the `RecommendationService` interface.
- `RemoteRecommendationService` is a remote (Satellite Server) implementation for the `RecommendationService` interface.

73.2.2 Steps for Implementing Custom REST Resources

1. Write your XSD file describing your REST service (`recommendations.xsd` file).
2. Generate beans using the `JAXB xjc` utility ("`generate`" ant's task).
3. Create your REST interface, which will be implemented differently for WebCenter Sites and Satellite Server.
4. Implement the REST interface by extending the following classes: `com.fatwire.rest.BaseLocalService` `com.fatwire.rest.BaseRemoteService`
5. This step is optional in case you decide to leverage Satellite Server caching:

   Create elements on the WebCenter Sites side, which load the same assets as the local implementation does.
6. Create your REST resource class by extending the `com.fatwire.rest.BaseResource` class.
7. Register your REST service and configuration in `WEB-INF/classes/custom/RestResources.xml` file on both WebCenter Sites and Satellite Server sides.

   The `custom/RestResources.xml` file contains the following components:

   - The only mandatory bean is the bean with `resourceConfigs` id. The `resourceConfigs` property contains references to all REST configurations used.

     Note: If custom `resourceConfigs` is uncommented, then bean should be referenced. Otherwise, the default REST resource, which is provided with the WEM installation will not be registered.

   - Resource configurations must be of type `com.fatwire.rest.ResourceConfig`. Typically only one instance of this class is registered (multiple services can be registered per configuration.

     Note: For multiple services, create a new configuration for each disjoint group of your REST services, usually identified by separate XSD files.

   - The `resourceClasses` property contains the list of all resources used.
   - `beanPackage` contains the Java package name specified for the output beans when running the `xjc` utility.

- schemaLocation is the xsi:schemaLocation attribute to be put in all output XML files produced by your REST service.

74

WEM Framework: Single Sign-On for Production Sites

Our SSO sample application is driven by a delivery use case. Given that ready-to-use CAS cannot be used to secure applications on production sites, we provide a simple example of how to enable single sign-on and sign-out for applications on live sites.

This chapter contains the following sections:

74.1 Deploying the SSO Sample Application

1. Unpack the `wem-sso-api-cas-sample.war` file (to the `/sso-sample` folder, for example). The application is located in the WebCenter Sites `Misc/Samples/WEM Samples/ WEM Sample applications/` directory.
2. Modify the `applicationContext.xml` file in the `WEB-INF` folder by setting the following properties:
   - `casUrl`: Point to the CAS server base path:

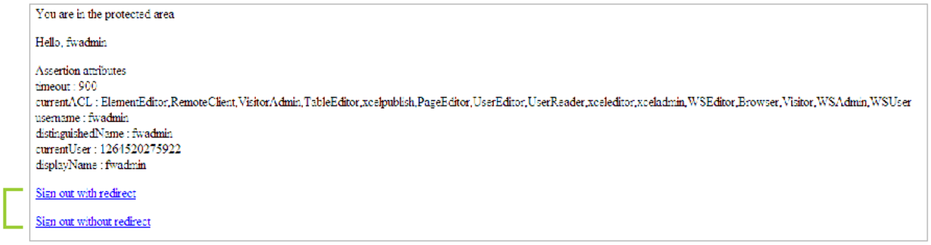
     ```
     http://localhost:8080/cas
     ```
   - `casLoginPath`: Include the login form template hosted by the SSO sample application:

     ```
     /login?wemLoginTemplate=http%3A%2F%2Flocalhost%3A9080%2Fsso-cas-sample%2Fte
     mplate.html
     ```
3. Deploy the modified SSO sample application to your application server.
4. Access the application.

The SSO sample application consists of the following pages:

- **Protected area**: a page that is protected by the WEM SSO filter. This page contains two single sign-out links (Figure 74–1).

Figure 74–1 Protected page with single sign-out links



You are in the protected area

Hello, fwadmin

Assertion attributes
timeout : 900
currentACL : ElementEditor,RemoteClient,VisitorAdmin,TableEditor,xcelpublish,PageEditor,UserEditor,UserReader,xceleditor,xceladmin,WSEditor,Browser,Visitor,WSAdmin,WSUser
username : fwadmin
distinguishedName : fwadmin
currentUser : 1261520275922
displayName : fwadmin

Sign out with redirect

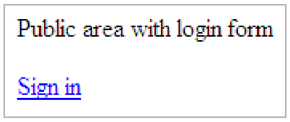
Sign out without redirect

single sign-out links

The first link (single sign-out with redirect) is an HTML link that performs single sign-out on the CAS side and redirects the user back to the home page. The second link (single sign-out without redirect) is also an HTML link that performs single sign-out on the CAS side, but without leaving or reloading the current page.

- **Public area**: a page that is excluded from the protection filter.
- **Public area with login form**: this page is excluded from the protection filter, but has a login form, which allows performing a sign-in operation without leaving or reloading the current page.

Figure 74–2 Public area with "Sign in" link



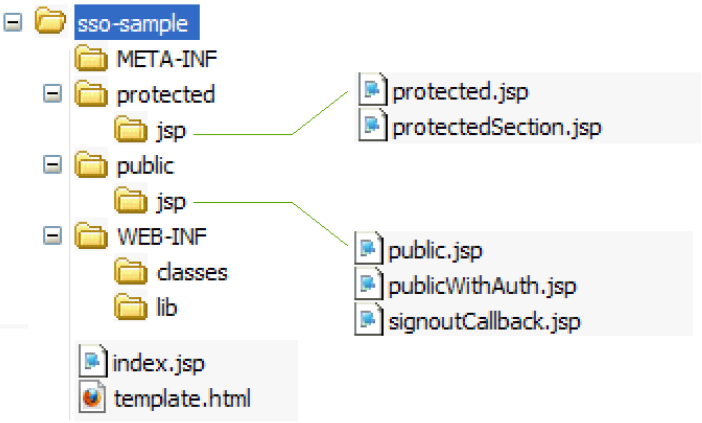
Public area with login form

Sign in

74.2 Application Structure

The SSO sample application provides you with the basic code for utilizing single sign-on and sign-out functionality to protect applications on production sites. The following components provide access to the SSO sample application:

- `index.jsp`: Starting page. This page contains links to the pages described as **Protected area**, **Public area**, and **Public area with login form pages** (see Section 74.1, "Deploying the SSO Sample Application").
- `template.html`: Used to provide a custom sign-in form for CAS. Its path is referenced in the `wemLoginTemplate` parameter in `casLoginPath` in the `applicationContext.xml` file.



Configuration Files: `/sso-sample/WEB-INF`

`WEB-INF` contains the following configuration files:

- `applicationContext.xml`: Spring web application configuration file, which configures the SSO subsystem.
- `web.xml`: Web application deployment descriptor.

Protected Files: `/sso-sample/protected/jsp`

Files in this area are protected by the SSO filter. By default, the following files are included in this folder:

- `protected.jsp`: A page protected by the SSO filter. This page hosts two links for performing single sign-out. The first link leads to the CAS sign-out page with a redirect to the application's home page when sign-out is complete. The second link embeds an iframe into this page, which calls the CAS sign-out page with a redirect to the `signoutCallback.jsp` page. The `protected.jsp` page also prints out all attributes from the `Assertion` object, which describes the current logged in user.
- `protected/jsp/protectedSection.jsp`: Page that is referenced from the `public.jsp` page, when the **Sign in** link is clicked in an embedded iframe. As this page is protected, a login screen is presented in the embedded iframe.

Public Files: `/sso-sample/public/jsp`

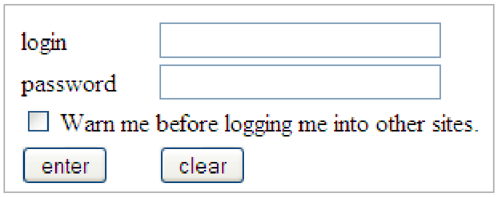
Files in this area are not protected by the SSO filter. By default, the following sample files are included in the `/public/jsp/` folder:

- `public.jsp`: This page not protected by the CAS filter
- `publicWithAuth.jsp`: This page displays the **Sign in** link. Clicking the link embeds an iframe into the `publicWithAuth.jsp` with the iframe pointing to the `protectedSection.jsp` page. As the page is protected, a login screen is presented in the embedded iframe.
- `signoutCallback.jsp`: This page is called from the `protected.jsp` page upon sign-out completion when using iframe.

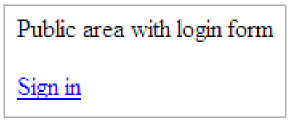
74.3 Implementing Single Sign-On

Implementing single sign-on for a website amounts to implementing a sign-in form. The sign-in form can be presented to site visitors in one of two ways:

- The sign-in form is presented when the visitor tries to access a protected page. This is the default sign-in implementation. This sign in form could be either a default sign-in form shipped with CAS or a custom form provided by an application.



- The sign-in form is embedded into a public page, and the sign-in function is performed without the user leaving the current page. This behavior can be implemented by embedding the iframe that points to a protected page. As the page is being protected, the sign-in form is presented to the visitor.



Public area with login form

Sign in

74.4 Implementing Single Sign-Out

When implementing single sign-out on a web page, you can do one of the following:

- Retrieve the "single sign-out" URL by invoking the following method:

  getSignoutUrl() or getSignoutUrl(String callbackUrl) method of com.fatwire.wem.sso.SSO.getSSOSession() object.

  After performing single sign-out, CAS can optionally redirect to the visitor-supplied URL, which is set in the callbackUrl parameter.

- Use an iframe-embedding technique if the sign-out is to be performed without leaving the current page. This technique involves embedding an iframe with the single sign-out URL as source. When the iframe is loaded, the sign-out URL is called (this is done primarily to avoid cross-domain restrictions in browsers).

75

WEM Framework: Using REST Resources

This chapter contains the following sections:

- Section 75.1, "Authentication for REST Resources"
- Section 75.2, "Configuring CAS"
- Section 75.3, "REST Authorization"
- Section 75.4, "Managing Assets Over REST"

75.1 Authentication for REST Resources

The WEM Framework uses the SSO mechanism built on top of CAS (`http://www.jasig.org/cas`) for authentication purposes. The system behaves differently when the REST API is used from a browser or programmatically.

When accessing the REST API from a browser, the user is redirected to the CAS login page and, upon successful login, back to the original location with the `ticket` parameter, which is validated to establish the user's identity. When accessing the REST API programmatically, the developer must supply either the `ticket` or `multiticket` parameter.

Both the `ticket` and `mu ltiticket` parameters could be acquired by using either the Oracle SSO API if making calls from Java, or simply by using the HTTP protocol if making calls from any other language. The difference between `ticket` and `multicket` is that a ticket is acquired per each REST resource and can be used only once (as the name implies, think of a train or a theater ticket, which is valid for one ride or one play), while a multiticket could be used multiple times for any resource. Both the `ticket` and `multiticket` parameters are limited in time, but the typical usage pattern differs. As a ticket is acquired per each call, there is no need to worry about its expiration time. However, reusing the same multiticket will eventually lead to its expiration and getting an `HTTP 403` error. The application must be able to recognize such behavior and fall back to the multiticket re-acquisition procedure in such a case. The decision to use either `ticket` or `multiticket` is up to the application developer.

This section contains the following topics:

- Section 75.1.1, "Acquiring Tickets from Java Code"
- Section 75.1.2, "Acquiring Tickets from Other Programming Languages (Over HTTP)"
- Section 75.1.3, "Using Tickets and Multitickets"
- Section 75.1.4, "SSO Configuration for Standalone Applications"

75.1.1 Acquiring Tickets from Java Code

The Oracle SSO API is implemented in an authentication provider-independent manner. Users will not be able to register their own SSO authentication providers. Support for a new authentication provider can be implemented only by Oracle. Switching between providers involves only changing the SSO configuration files.

All SSO calls originate at the SSO front-end class SSO. It is used to get the `SSOSession` object. `SSOSession` is acquired per each SSO configuration. It is a single configuration in the web application case, which is loaded using the Spring Web application loader or a configuration loaded from a configuration file in the case of a standalone application.

Web Application

```
SSO.getSession().getTicket(String service, String username, String password)
SSO.getSession().getMultiTicket(String username, String password)
```

Standalone Application

```
SSO.getSession(String configName).getTicket
  (String service, String username, String password)
SSO.getSession(String configName).getMultiTicket
  (String username, String password)
```

75.1.2 Acquiring Tickets from Other Programming Languages (Over HTTP)

The CAS REST API is used to acquire a ticket and/or multiticket in the delivery environment. Two `HTTP POST` calls should be performed to acquire either ticket or multiticket. The difference between ticket and multiticket is that the `service` parameter is " `*` " for multiticket, while it is an actual REST resource you are trying to access for the `ticket` parameter.

The example below demonstrates the calls to be made to the CAS server to get a ticket to the `http://localhost:8080/cs/REST/sites` service with `fwadmin/xceladmin` credentials:

1. Call to get `Ticket Granting Ticket`

   Request

   ```
   POST /cas/v1/tickets HTTP/1.1
   Content-Type: application/x-www-form-urlencoded
   Content-Length: 35

   username=fwadmin&password=xceladmin
   ```

   Response

   ```
   HTTP/1.1 201 Created
   Location:
   http://localhost:8080/cas/v1/tickets/TGT-1-ej2biTUFoCNBwA5X4lJn4PjYLRcLtLYg2QhL
   HclInfQqUk3au0-cas
   Content-Length: 441
   ...
   ```

2. Call to get a Service ticket

   Request

   ```
   POST
   ```

```
/cas/v1/tickets/TGT-1-ej2biTUFoCNBwA5X4lJn4PjYLRcLtLYg2QhLHclInfQqUk3au0-cas
HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 57

service=http%3A%2F%2Flocalhost%3A8080%2Fcs%2FREST%2Fsites
```

Response

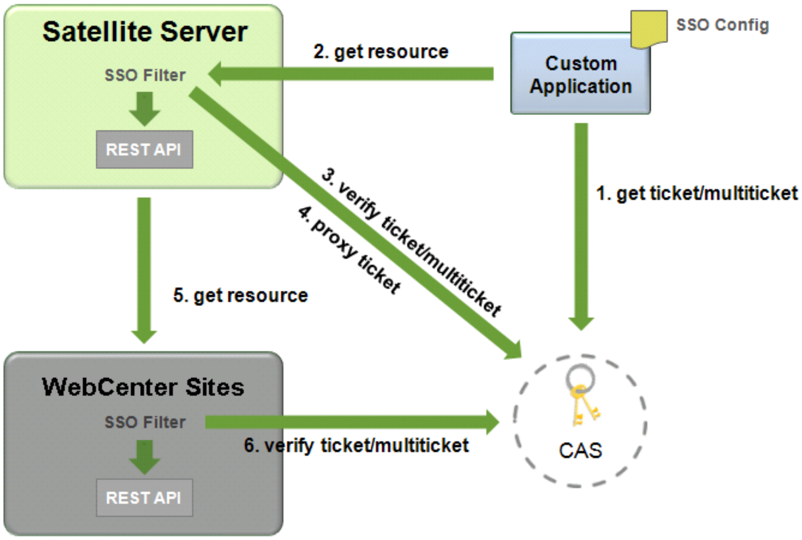
```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 29

ST-1-7xsHEMYR9ZmKdyNuBz6W-cas
```

The protocol is fairly straightforward. First a call to get Ticket Granting Ticket (TGT) is made by passing the username and password parameter in `application/x-www-form-urlencoded POST` request. The Response will contain the `Location` HTTP header, which should be used to issue a second `application/x-www-form-urlencoded POST` request with `service` parameter. The response body will contain the actual ticket.

75.1.3 Using Tickets and Multitickets

To use the generated ticket/multiticket, supply the `ticket/multiticket` URL query parameter. For example:

```
http://localhost:8080/cs/REST/sites?ticket=ST-1-7xsHEMYR9ZmKdyNuBz6W-cas
http://localhost:8080/cs/REST/sites?multiticket=ST-2-Bhen7VnZBERxXcepJZaV-cas
```



1. The application performs a call to get the ticket/multiticket.
   - Input: service, username, password
   - Output: ticket /multiticket
2. The application performs call to Remote Satellite Server to get the resource.

- Input: ticket, resource input data
- Output: resource output data

3. Remote Satellite Server performs a call to validate the resulting 'assertion'. The assertion contains user information. Satellite Server also maintains a time-based cache of multitickets, so that subsequent calls do not incur the cost of validation.
   - Input: ticket/multiticket
   - Output: assertion
4. This step is optional. If the `proxyTickets` parameter in the `SSOConfig.xml` file parameter is set to `true` on the Satellite Server side, it also proxies the ticket.
   - Input: ticket
   - Output: proxied ticket
5. Remote Satellite Server performs a call to WebCenter Sites.
   - Input: assertion (in serialized form), resource input data
   - Output: resource output data
6. This step is optional. If security is enabled on the WebCenter Sites side, it performs a call to validate the ticket.
   - Input: ticket/multiticket
   - Output: assertion

By default the communication channel between WebCenter Sites and Remote Satellite Server is not trusted. The `proxyTickets` parameter in the `SSOConfig.xml` file on Remote Satellite Server is set to `true`, which forces Remote Satellite Server to proxy the ticket supplied by the application that is being accessed.

For optimal performance, the system can be configured for authentication by Satellite Server alone. The security check should be disabled on the WebCenter Sites side by excluding the REST and WebCenter Sites elements used by the REST API from the SSO filter; the `proxyTickets` parameter in the `SSOConfig.xml` file on Remote Satellite Server should be set to `false`. In this mode it is possible to leverage multitickets. Note that the WebCenter Sites installation should be hosted inside a private network in this mode, and the communication channel between WebCenter Sites and Remote Satellite Server should be trusted.

75.1.4 SSO Configuration for Standalone Applications

The single sign-on module relies on the Spring configuration. The only required bean is `ssoprovider`, which references the `ssoconfig` bean.

This section contains the following topics:

- Section 75.1.4.1, "Beans and Properties"
- Section 75.1.4.2, "Query Parameters Processed by SSO Filter"

75.1.4.1 Beans and Properties

```
id="ssolistener", class="com.fatwire.wem.sso.cas.listener.CASListener"
```

Table 75–1 id="ssolistener"

Property	Description
No properties for this bean.	n/a

```
id="ssofilter",
class="com.fatwire.wem.sso.cas.filter.CASFilter"
```

Table 75–2 id="ssofilter"

Property	Description
`config`	Required. SSO configuration reference. Sample Value: `ssoconfig`
`provider`	Required. SSO provider reference. Sample Value: ssoprovider

```
id="provider",
class="com.fatwire.wem.sso.cas.CASProvider"
```

Table 75–3 id="provider"

Property	Description
`config`	SSO configuration reference. Sample Value: `ssoconfig`

```
id="config",
class="com.fatwire.wem.sso.cas.conf.CASConfig"
```

Table 75–4 id="config"

Property	Description
`applicationProxyCallbackPath`	Proxy callback path, relative to `casUrl`. **Sample Value:** `/proxycallback`
`authRedirect`	Use this property to specify the default behavior on unauthenticated access to protected pages. true redirects the user to the CAS login page; false displays a 403 error if users are not unauthenticated. This setting could be overridden by the Pragma: auth-redirect HTTP header. **Sample Value:** `true`
`casLoginPath`	Login page path, relative to casUrl. Can accept additional query parameters: ■ `wemLoginTemplate`, points to the page containing the HTML login template to be used instead of the default template. The template must have two input fields: `username` and `password`. Note, that the HTML `<form>` tag should not be used in the template. ■ `wemLoginCss`, points to the CSS page containing style declarations used on the login form. **Sample Value:** `/login`
`casRESTPath`	CAS REST servlet path, relative to `casUrl`. **Sample Value:** `/v1`

Table 75–4 (Cont.) id="config"

Property	Description
`casSignoutPath`	Logout page path, relative to `casUrl`. **Sample Value:** /logout
`casUrl`	Required property. CAS URL prefix. **Example**: `http://localhost:8080/cas`
`gateway`	If true, the request to protected pages will be redirected to CAS. If a ticket-granting cookie is present, then the user will be implicitly authenticated; if not, the user will be redirected back to the original location. This is used primarily to allow implicit authentication if the user is already logged in to another application.
`gateway` *(continued)*	Be careful when enabling the redirect behavior to occur by default. Make sure that the clients are able to follow the redirects. Otherwise, gateway=false URL query parameter should be used to override the default behavior. For example, while processing wemLoginTemplate and wemLoginCss parameters, CAS does not follow redirects; you will have to prepend gateway=false to URLs when turning this setting on. **Default value:** `false`
`multiticketTimeout`	Multiticket timeout in msecs. **Default value:** `600000`
`protectedMappingExcludes`	List of mappings that should be excluded. Regular expressions are allowed. **Allowed value:** `See protectedMappingIncludes`

`protectedMappingIncludes`	List of protected mappings. Regular expressions are allowed.
	Allowed value: `path?[name=value,#]`
	path is a URL path part. It may contain asterisks (`*` and `**`). The single asterisk `*` symbolizes any character sequence up to the forward slash character (/), while ** applies to the entire path.
	Example
	`/folder1/folder2 matches against /folder1/*, while /folder1/folder2/folder3 does not.`
	`/folder1/folder2 matches against /folder1/**, as well as /folder1/folder2/folder3.`
	?[..] block is optional. Query parameters can be specified inside the block. Parameters are comma separated. The special character # means that the specified parameters are a subset of those from the request; omitting # requires the request parameters to exactly match the specified parameters.
	Parameters may contain only *name*. The match will be done against *name* only, or against *name=value* (i.e., both *name* and *value*). A parameter can take multiple values. In this case, the match test will pass if any of the specified parameter values match the corresponding parameter value from the request.
	Example
	`/file1[size=1|2] matches against /file1?size=2, but not against /file1?size=2&author=admin`
	`/file1[size=1|2,name=file1,#] matches against /file1?size=2 and /file1?size=2&author=admin,but not against /file1?size=3`

Table 75–4 (Cont.) id="config"

Property	Description
`protectedMappingIncludes` *(continued)*	To make custom REST resources in an application available via remote Satellite Server, specify the following value: `/ContentServer?[pagename=rest/<path toCSElement>,#]` **Example** `/ContentServer?[pagename=rest/sample/recommendation,#]` for custom REST resources in Chapter 73.1, "The Recommendations Sample Application."
`proxyTickets`	Specifies whether to proxy tickets. Set this property to false for the last server in the call chain for optimal performance. Set this property to true if you need to call another CAS-protected application from this application on behalf of the currently logged-in user. This results in the ability to call the following method: `SSO.getSSOSession().getTicket(String service, String username, String password)` **Default value:** `true`
`useMultiTickets`	Specifies whether to use multitickets. **Default value:** `true`

75.1.4.2 Query Parameters Processed by SSO Filter

Table 75–5 Query Parameters Processed by SSO Filter

Property Name	Description
ticket	Used to verify user identity. Can be used only during some limited period of time for one resource and only once. **Type:** `<query parameter>` **Value:** `<random string>`
multiticket	Used to verify user identity. Can be used only during some limited period, multiple times for any resource. **Type:** `<query parameter>` **Value:** `<random string>`
`gateway`	If this property is set to true, the request for public pages will be redirected to CAS. If the ticket granting cookie is present, then the user will be implicitly authenticated; if not, the user will be redirected back to the original location. This is primarily to allow implicit authentication if the user is already logged in to another application. **Type:** `<query parameter>` **Value:** `true | false`
`auth-redirect`	Used to specify the default behavior on unauthenticated access to protected pages. If this property is set to true, the user will be redirected to the CAS login page; if false, a `403` error will be presented. **Type:** `<Pragma HTTP header>` **Value:** `true | false`

75.2 Configuring CAS

Information about CAS clustering can be found in the following sources:

- For information about CAS architecture, use the following link:

  `http://www.jasig.org/cas/about`

- For information about configuring CAS clustering during the WebCenter Sites installation, see the *Oracle Fusion Middleware WebCenter Sites Installation Guide*.

- For information about configuring CAS with LDAP providers, use the following link:

  `http://www.jasig.org/cas/server-deployment/authentication-handler`

75.3 REST Authorization

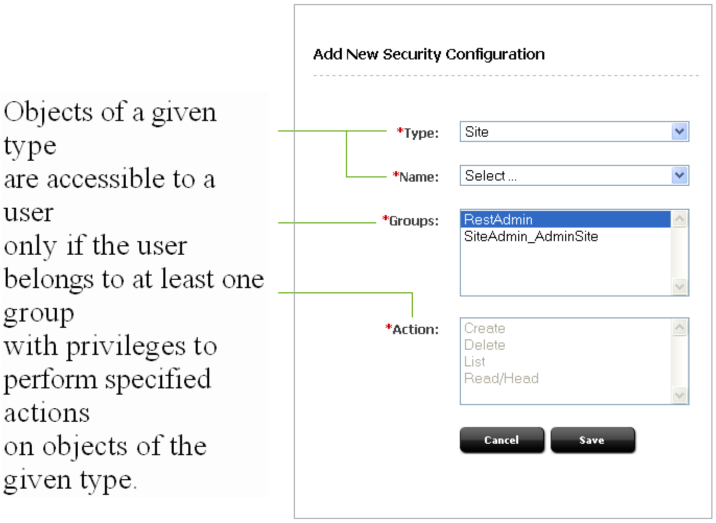
This section is for developers who are interested in administrators' authorization processes. REST authorization is the process of granting privileges to perform REST operations on applications' resources (which map to objects in WebCenter Sites). REST authorization uses the "deny everything by default" model. If a privilege is not explicitly granted to a particular group, that privilege is denied.

This section contains the following topics:

- Section 75.3.1, "Security Model"
- Section 75.3.2, "Using the Security Model to Access REST Resources"
- Section 75.3.3, "Configuring REST Security"
- Section 75.3.4, "Privilege Resolution Algorithm"

75.3.1 Security Model

The WEM security model is based on objects, groups, and actions. Security must be configured per object type in the WebCenter Sites Admin interface:



Objects of a given type
are accessible to a user
only if the user belongs to at least one group
with privileges to perform specified actions
on objects of the given type.

Add New Security Configuration

*Type: Site

*Name: Select ...

*Groups: RestAdmin

- **Object** is a generic term that refers to any entity in the WEM Framework such as a site, a user, or an asset. Protected objects are of the following types:
  - Asset Type

  - Asset
  - Index
  - Site
  - Role
  - User
  - User Locale
  - ACL
  - Application

- **Security groups** are used to gather users for the purpose of managing their permissions (to operate on objects) simultaneously.
- An **action** is a security privilege: `LIST,READ`, `UPDATE`, `CREATE`, `DELETE`. `LIST` provides `GET` permission on services that list objects (such as `/types`), whereas `READ` provides `GET` permission on services that retrieve individual objects in full detail (such as `/types/{assettype}`).

  Privileges are assigned to groups to operate on allowed objects. Some objects, such as ACLs, are read-only (they can be created directly in WebCenter Sites, but not over REST).

A security configuration is an array, such as shown above, which specifies:

- The protected object type and object(s)
- Groups that are able to access the objects
- Actions that groups (and their members) can perform on the objects

For more information on possible security configurations and the Web Experience Management Framework, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

75.3.2 Using the Security Model to Access REST Resources

Object types and objects in WebCenter Sites map to REST resources in the WEM Framework. For example, the `Asset Type` object maps to:

- `<BaseURI>/types/` resource (which lists all asset types in the system)
- `<BaseURI>/types/<assettype>` resource (which displays information about the selected asset type), and so on.

Actions in WebCenter Sites map to REST methods in the WEM Framework. For example, granting the `READ` privilege to group `Editor` to operate on asset type `Content_C` gives users in the `Editor` group permission to use `GET` and `HEAD` methods on the REST resource `/types/Content_C`.

- The `LIST` action allows group members to use `GET` methods on REST resources.
- `The READ` action allows group members to use `GET` and `HEAD` methods on REST resources.
- The `UPDATE` action allows group members to use `POST` methods on REST resources.
- The `CREATE` action allows group members to use `PUT` methods on REST resources.
- The `DELETE` action allows group members to use `DELETE` methods on REST resources.

For comprehensive information, see the *Oracle Fusion Middleware WebCenter Sites REST API Resource Reference*

75.3.3 Configuring REST Security

For more information on configuring REST security and the Web Experience Management Framework, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

75.3.4 Privilege Resolution Algorithm

When configuring a security privilege, you can specify that the privilege applies to all objects of a certain type or a single object of a certain type. For example, granting the privilege to `UPDATE (POST)` any site allows users in the group to modify the details of all sites in the WEM Framework. Granting the privilege to `UPDATE (POST)` the FirstSiteII sample site allows users in the group to modify this site's details in WEM.

The `Asset` object type requires you to specify the site to which the security setting applies, as assets are always accessed from a particular site. The `AssetType` object can be extended by specifying a subtype, which is used to make the security configuration more granular. For example, setting the `DELETE` privilege on asset type `Content_C` in allows a `DELETE` request to be performed on the REST resource `/types/Content_C` (i.e., to delete the `Content_C` asset type from the system).

Because privileges can be granted only to groups, a user's total privileges are not obvious until they are computed across all of the user's groups. The WEM Framework provides a privilege resolution algorithm. Its basic steps are listed below:

1. REST finds the groups in which the user has membership.
2. REST determines which groups can perform which REST operations on which REST resources. If site or subtype is specified, each is taken into account.
3. REST compares the results of steps 1 and 2. If at least one of the groups from step 1 is in the list of groups from step 2, then access is granted. Otherwise, access is denied.

75.4 Managing Assets Over REST

Sample code illustrating management of assets via the WebCenter Sites REST API is available in your WebCenter Sites installation directory, in the following paths:

```
Misc/Samples/WEM Samples/REST API samples/Basic
Assets/com/fatwire/rest/samples/basic/
Misc/Samples/WEM Samples/REST API samples/Basic
Assets/com/fatwire/rest/samples/flex/
```

The subfolders `basic` and `flex` each contain the following set of files:

- `CreateAsset.java`
- `DeleteAsset.java`
- `ReadAsset.java`
- `UpdateAsset.java`.

The code is richly documented with step-by-step instructions. Examples of basic asset management use the HelloAssetWorld sample site. Examples of flex asset management use the FirstSiteII sample site. All information regarding the required asset types and assets can be found in the `java` files.

76

WEM Framework: Customizable Single Sign-On Facility

This chapter contains the following sections:

76.1 Customizing Login Behavior for the WEM Framework

WEM Framework authentication, which is built over the CAS framework, includes a customization layer called the Oracle Customizable Single Sign-On facility, also called *CSSO*. The CSSO facility contains authentication extensions that you can use to create a custom SSO solution, without directly modifying the CAS configuration. Instead, the Spring configuration directs the injection of these extensions into the CAS configuration to implement the desired login behavior.

The CSSO facility provides pre-packaged classes that can be extended to implement a custom SSO solution. It also provides a default Spring configuration file which identifies the classes to Spring for instantiation. Customizing WEM SSO enables you to use a different login screen, require credentials other than a username/password pair, or use an external authentication authority to authenticate WebCenter Sites users. A custom SSO implementation consists of:

- Three Java classes (which extend the default classes)
- A configuration file that exposes the new classes to the framework

The default CSSO classes defer all credential discovery and authentication to the standard WEM SSO implementation. These classes are instantiated by the `customdefaultWEMSSObean.xml` Spring configuration file. Extending the default CSSO classes enables you to define methods which specify the behavior of your custom SSO solution. For example you can create a different authentication for browser access, REST, and/or thick client authentication. When you extend the default CSSO classes, you must create a custom Spring configuration file that identifies the custom classes and exposes them to the WEM Framework.

The CSSO facility provides a complete SSO sample (including Java source files) that replaces the default WEM login behavior with custom login behavior. The sample SSO implementation demonstrates two different types of authentication; username/password pair (with an additional domain field) and external user identifier. The external identifier maps a user authenticated by an external authentication authority to a WebCenter Sites system user.

The rest of this chapter provides information about the default components of the CSSO facility and instructions on implementing a custom SSO solution. If you wish to see an example of a custom SSO solution, the end of this chapter provides information about the CSSO sample, and instructions for running the sample.

76.2 Components of the Default CSSO Implementation

This section provides information about the default components provided by the CSSO facility. These components are your starting point for customizing your own SSO implementation.

The `com.fatwire.wem.sso.cas.custom.basis` package (shown in Table 76–1) contains the default classes that are included in the CSSO facility. The default Spring configuration file (`customdefaultWEMSSObeans.xml`) instantiates these classes to implement the default WEM login behavior.

> **Note:** The CSSO facility provides a complete SSO sample that replaces the default WEM login behavior with custom login behavior. For more information, see Section 76.4, "Running the CSSO Sample Implementation."

Table 76–1 com.fatwire.wem.sso.cas.custom.basis

Class	Description
`CustomAuthenticator.java`	Implements the `CustomAuthentication` interface. This class controls the behavior of the login sequence and handles authentication requests. By default, it returns to the WEM Framework to complete the authentication by displaying the standard WEM login form. For information about extending this class, see Section 76.3.1, "Extending the Default CSSO Classes."
`CustomConfiguration.java`	Provides access to the properties that are set in the default Spring configuration file. You can extend this class when additional properties are required for a custom SSO implementation. For information about extending this class, see Section 76.3.1, "Extending the Default CSSO Classes."
`CustomCredentials.java`	Provides a standard set of credential values for custom authentication. You can extend this class when additional attributes are needed for a custom SSO implementation. For information about extending this class, see Section 76.3.1, "Extending the Default CSSO Classes."

The `com.fatwire.wem.sso.cas.custom.interfaces` package (shown in Table 76–2) defines the custom authentication interfaces.

Table 76–2 com.fatwire.wem.sso.cas.custom.interfaces

Class	Description
`CustomAuthentication.java`	Defines the interfaces that must be implemented by any custom SSO solution. For more information, see Section 76.3.1, "Extending the Default CSSO Classes."
`CustomRestCodec.java`	Defines the interfaces that must be implemented to encode and decode a custom REST authentication token that is not username/password based.

76.3 Configuring and Deploying Custom SSO Behavior

To configure and deploy custom SSO behavior you must first extend the default classes that are included in the CSSO facility. You then identify the new Java classes to Spring by creating a custom Spring configuration file which instantiates the classes, exposing them to the CSSO framework. These are your basic steps:

1. Extend the default CSSO classes: `CustomAuthenticator.java`, `CustomConfiguration.java`, and `CustomCredentials.java` (contained within the `com.fatwire.wem.sso.cas.custom.basis` package):

   a. Create new Java classes that extend the default CSSO classes.

   b. Package the Java classes you created in a `jar` file, then place the `jar` file in the classpath of the CAS servlet (in `cas/WEB-INF/lib`).

   For more information, see Section 76.3.1, "Extending the Default CSSO Classes."

2. Identify your new Java classes to Spring for instantiation:

   a. Create a Spring configuration file that contains all the custom class names and properties for your SSO implementation.

   b. Place the custom Spring configuration file in the `spring-configuration` folder (in `cas/WEB-INF/`).

   c. Remove the `.xml` extension from the default Spring configuration file (`customDefaultWEMSSObeans.xml`).

   For more information, see Section 76.3.2, "Identifying Your Java Classes to Spring for Instantiation."

3. If an external authentication authority is used to authenticate a user, map the external user identifier to the appropriate WebCenter Sites system user name, unique identifier, and ACLs. For instructions, see Section 76.3.3, "Mapping External User Identifiers to WebCenter Sites Credentials."

4. Restart the CAS web application. For more information, see Section 76.3.4, "Restarting the CAS Web Application."

The rest of this section provides detailed information for the steps outlined above.

76.3.1 Extending the Default CSSO Classes

An SSO implementation is a set of called methods that are specified in the default CSSO classes `CustomAuthenticator.java`, `CustomConfiguration.java`, and `CustomCredentials.java`. To replace the default WEM login behavior with custom behavior, you must create new Java classes for it that extend the default CSSO classes. By extending the CSSO classes, the methods specified in the default CSSO classes are replaced by the methods specified in the custom classes for the functionality you wish to change.

The three classes (located in the `com.fatwire.wem.sso.cas.custom.basis` package) that must be extended to implement a custom SSO solution are:

- **CustomConfiguration.java**: Provides access to the externally defined properties that are specified in the default Spring configuration file. By default, this class exists only as a placeholder for injecting properties into the SSO configuration from the Spring configuration file. Extend this class if you wish to include additional properties, such as URLs or other configuration information, that are specific to your custom SSO implementation.
- **CustomCredentials.java**: Provides a standard set of credential values for custom authentication. This class is built and populated by the web-flow handler or the custom REST authenticator. By default, this class defines the standard `UsernamePasswordCredentials` object (provided by CAS), which collects all information required to complete user authentication in the following properties; `username`, `userId`, and `currentACL`. The values of these properties populate the attributes map used by the authenticator (`CustomAuthenticator.java`), to perform the actual user authentication.

  Extend this class if you wish to require additional credentials for your custom SSO solution. For an example of how this class passes user information to the authenticator to complete user authentication, refer to the code of the sample CSSO class `SampleCredentials.java` (located in the `Misc/Samples/WEM/Samples/CustomizableSSO/lib` folder).
- **CustomAuthenticator.java**: Implements the `CustomAuthentication` interface. This class controls the behavior of the login sequence and handles authentication requests. By default, it returns to the WEM Framework to complete the authentication by displaying the standard WEM login form.

  > **Note:** The default `CustomAuthenticator.java` class is the most important class because it contains all the authentication methods for an SSO implementation.

- All authentication decisions and CAS web-flow actions are directed to this class for action. CAS web-flow performs a number of steps, one of which invokes the `performLoginAction` method. This method displays a login form or communicates with an external authentication authority.

  This class also defines the static method `callCsResolverPage` which maps an external user to a WebCenter Sites user. If your custom SSO implementation uses an external authentication authority to authenticate users, the `callCsResolverPage` method must define the unique name for the CSSO authenticator. For more information, see Section 76.3.3, "Mapping External User Identifiers to WebCenter Sites Credentials."

  The following is a complete interface description of the methods this class implements:

```
static final int SUCCESS  = 0;
static final int GOTOWEM  = 1;
static final int FAILURE  = 2;
static final int REDIRECT = 3;
static final int ERROR    = 4;
static final int REPEAT   = 5;
```

```
/**
 * Called from UserAuthentication handler to check for alternate
 * credentials and validate appropriately.
 * @param userCredentials
 * @return
 */
public int authenticate(com.fatwire.wem.sso.cas.custom.basis.CustomCredentials
userCredentials);

/**
 * Called from CSAuthenticationHandler to check for REST user
 * credentials and validate appropriately.
 */
public int authenticateRest(UsernamePasswordCredentials restCredentials);

/**
 * Called from CSAuthenticationHandler to check is username/password
 * combination is detected.
 */
public boolean checkRestCredentials(String token);

/**
 * Called from CSAttributeDAO to check for encoded credentials and
 * if so then return the correct username for DAO processing.
 * @param username
 * @return
 */
public String resolveRestUsername(String username);

/**
 * Called from LoginViewAction to handle login view processing. This
 * method allows the calling of internal CAS methods.
 * @param context
 * @param userAuthentication
 * @param centralAuthenticationService
 * @return
 */
public int performLoginAction(RequestContext context,
CustomAuthentication userAuthentication,
CentralAuthenticationService centralAuthenticationService);

/**
 * Called from casLogoutView to perform sign in cleanup
 * @param request
 * @param response
 */
public void performLogoutAction(HttpServletRequest request, HttpServletResponse
response);
```

76.3.2 Identifying Your Java Classes to Spring for Instantiation

All customization settings for an SSO implementation are specified in a single Spring configuration file, located in the `spring-configuration` folder (in `cas/WEB-INF`).

The rest of this section contains the following topics:

- Section 76.3.2.1, "Creating a Spring Configuration File"
- Section 76.3.2.2, "Placing Your Spring Configuration File"

76.3.2.1 Creating a Spring Configuration File

The classes and properties for the default SSO implementation are defined by the Spring configuration file `customDefaultWEMSSObeans.xml`, which is located in the `spring-configuration` folder (in `cas/WEB-INF`). When customizing CSSO, you can either create a new Spring configuration file or customize the classes and properties referenced in the default Spring configuration file. The rest of this section focuses on the second option.

The default Spring configuration file contains several bean identifiers that reference the classes and properties required for the default SSO implementation. The `customUserConfiguration` bean references the `CustomConfiguration.java` class and the `customUserAuthenticator` bean references the `CustomAuthenticator.java` class. These classes are instantiated by the Spring configuration file, which uses them to create the persistent objects for the SSO implementation's authentication process. To create a custom SSO solution, you must reference your custom Java classes within these beans.

> **Note:** The `CustomCredentials.java` class is **not** referenced by the Spring configuration file. Instead, you provide the code that instantiates this object in the `performLoginAction` method, defined in the default CSSO `CustomAuthenticator.java` class. This method creates a custom credentials object for every login request and passes it into CAS for authentication.

The `customUserConfiguration` bean also identifies the configuration properties which supply system information to the default SSO implementation. These properties are set with values of the environment on which you are deploying the SSO implementation. When you customize the Spring configuration file, you must modify the values of the properties to match the custom SSO implementation's environment, or include additional properties required by the custom SSO implementation.

Extending the `CustomConfiguration.java` class enables you to define additional properties in the Spring configuration file's `customUserConfiguration` bean. For example, if you created a JSP file that provides a custom login form for your SSO implementation, create a property that specifies the location of the JSP file by extending the `CustomConfiguration.java` class.

The rest of this section analyzes the classes and properties that are referenced in the default Spring configuration file (`customDefaultWEMSSObean.xml`).

The default Spring configuration file

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:flow="http://www.springframework.org/schema/webflow-config"
5.        xmlns:p="http://www.springframework.org/schema/p"
6.        xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
7.        http://www.springframework.org/schema/webflow-config
   http://www.springframework.org/schema/webflow-config/spring-webflow-config-1.0.
   xsd">
8. <!-- Custom SSO Bean definitions. This file defines either the default CAS/SSO
   configuration or a special
```

```
         user implementation. No other CAS configuration   files are modified
    for a custom implementation -->
9.  <!-- This bean is never modified. It defines the web-flow controller which
    always passes control into

         the custom authenticator -->
10. <bean id="customUserLoginAction"
    class="com.fatwire.wem.sso.cas.web.CustomLoginViewAction"

11. p:centralAuthenticationService-ref="centralAuthenticationService"

12. p:customAuthentication-ref="customUserAuthenticator"

13. />

14. <!-- This bean is usually not modified. Override it when there needs to be a
    custom encoding for

         information passed between the web-flow and any external component -->
15. <bean id="customRestCoder"
    class="com.fatwire.wem.sso.cas.custom.basis.CustomRestTokenCoding"

16. />

17. <!-- Modify this bean with a custom configuration implementation class when
    additional parameters are

            needed for a custom implementation -->
18. <bean id="customUserConfiguration"
    class="com.fatwire.wem.sso.cas.custom.basis.CustomConfiguration"

19. p:casLoginUrl="http://localhost:7080/cas/login"

20. p:resolverUrl="http://localhost:8080/cs/custom/customCsResolver.jsp"

21. p:resolverUsername="fwadmin"

22. p:resolverPassword="xceladmin"

23. p:traceFlag="false"

24. />

25. <!-- Modify this bean with a customAuthentication class for a custom
    implementation. -->

26. <bean id="customUserAuthenticator"
    class="com.fatwire.wem.sso.cas.custom.basis.CustomAuthenticator"

27. p:customConfiguration-ref="customUserConfiguration"

28. p:customRestCoder-ref="customRestCoder"

29. />

30. </beans>
```

Analyzing the default Spring configuration file

- Line 18 is the `customUserConfiguration` bean, which references the default CSSO `customConfiguration.java` class. For information about this class, see Section 76.3.1, "Extending the Default CSSO Classes." This bean also contains the required properties for the default SSO implementation:
  - Line19 references the `casLoginURL` property. This property specifies the base URL to the CAS login function. The domain and port number settings require modification if the values specified are different from the CAS server installation.
  - Lines 20 through 22 reference the external authentication properties `resolverURL`, `resolverUsername`, and `resolverPassword`. If WebCenter Sites is used to authenticate users, these properties do not need to be referenced. If an external authentication authority is used to authenticate users, these

properties must be referenced. When these properties are referenced they enable you to implement mapping from an external identifier to a WebCenter Sites system user.

  - Line 20 references the `resolverURL` property. If an external authentication authority is used to authenticate a user, this property must specify the full URL to the `customCsResolver` page, located on WebCenter Sites. The `customCsResolver` page obtains a user's external identifier and queries the WebCenter Sites database to retrieve the user's WebCenter Sites credentials. The domain and port number specified in this property, must be modified if the values specified are different from the WebCenter Sites installation.
  - Line 21 references the `resolverUsername` property. If an external authentication authority is used to authenticate a user, this property must specify the username of a WebCenter Sites user who has permissions to read the `SystemUserAttr` table. This username is used when the `customCsResolver` page needs to query the WebCenter Sites database to resolve an external user identifier into a registered WebCenter Sites user.
  - Line 22 references the `resolverPassword` property. If an external authentication authority is used to authenticate a user, this property must specify the password of the user identified by the `resolverUsername` property (line 21).

  For information about implementing mapping, see Section 76.3.3, "Mapping External User Identifiers to WebCenter Sites Credentials."

- Line 23 references the `traceFlag` property. This property specifies whether the trace log, which provides information about the custom SSO layer, is enabled or disabled. This property can either be set to `True` or `False`.

- Line 26 is the `customUserAuthenticator` bean, which references the default CSSO `CustomAuthenticator.java` class. For information about this class, see Section 76.3.1, "Extending the Default CSSO Classes."

76.3.2.2 Placing Your Spring Configuration File

The default Spring configuration file, which specifies the classes and properties for the default WEM login behavior, is located in the `spring-configuration` folder (in `cas/WEB-INF`). Placing your own file into the same location requires deactivating the default file (by removing or changing the file's `.xml` extension). This is because Spring loads all Spring configuration files contained in the `spring-configuration` folder (in `cas/WEB-INF`) and merges those files into a single configuration. As both the custom and the default files specify the same bean identifiers, only one of the files can be recognized by the Spring configuration. Duplicate bean identifiers result in initialization failure.

> **Note:** Avoid deleting `customDefaultWEMSSObeans.xml`. Instead, remove or change the file's `.xml` extension. This way you can restore the file if you wish to return to using the default WEM login screen.

76.3.3 Mapping External User Identifiers to WebCenter Sites Credentials

The CSSO facility enables you to use an external authentication authority to authenticate WebCenter Sites users. When the external authentication authority validates the user's credentials, it associates a unique external identifier with that user. To complete WEM authentication, the user's external identifier must be mapped to the corresponding WebCenter Sites system username, unique identifier, and ACLs by using the method `callCsResolverPage` (defined as a static method in the default CSSO class `CustomAuthenticator.java`).

To map an external identifier to a WebCenter Sites system user, make sure you have set the external authentication properties in the Spring Configuration file (see, "Analyzing the default Spring configuration file" in Section 76.3.2.1, "Creating a Spring Configuration File"). To implement mapping from an external identifier to the appropriate WebCenter Sites system credentials, do the following:

To implement mapping

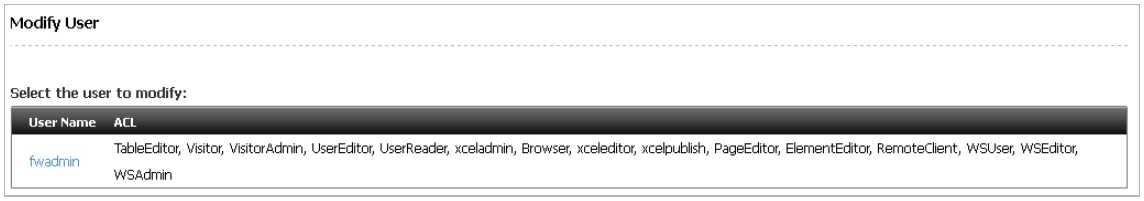
1. Define a unique CSSO authenticator name for the external authentication authority of your custom SSO implementation in the `callCsResolverPage` method (defined in your extended `CustomAuthenticator.java` class).

   For example, the following `callCsResolverPage` method (defined in the Sample CSSO class `SampleAutheticator.java)` defines `"samplesso"` as the unique authenticator name:

   ```
   Map<String,String>csTokens=callCsResolverPage(externalUserId,
   "samplesso")
   ```

2. Access the WebCenter Sites Admin interface as a general administrator (for example, `fwadmin/xceladmin`).
3. In the **Admin** tab, expand the **Management Tools** node and double-click **User**.
4. Select the user whose external identifier you wish to map to WebCenter Sites credentials:
   a. In the "Enter User Name" field, enter the name of the user.
   b. In the "Select Operation" section, select the **Modify User Attributes** option.
   c. Click **OK**.

   The Modify User form is displayed:



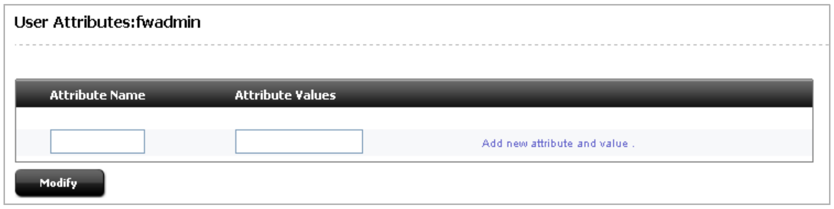
Modify User

Select the user to modify:

User Name	ACL
fwadmin	TableEditor, Visitor, VisitorAdmin, UserEditor, UserReader, xceladmin, Browser, xceleditor, xcelpublish, PageEditor, ElementEditor, RemoteClient, WSUser, WSEditor, WSAdmin

5. In the "User Name" column, click the name of the user whose external identifier you wish to map to WebCenter Sites credentials.

   The following form is displayed:



User Attributes:fwadmin

Attribute Name	Attribute Values	
		Add new attribute and value .

6. In the form, fill in the fields:
   – In the "Attribute Name" field, enter the unique CSSO authenticator name (the name used to identify the external authentication authority). This name must

match the unique name of the CSSO authenticator defined in the callCsResolverPage method (in step 1).

– In the "Attribute Values" field, enter the user's external identifier provided by the external authentication authority.

7. Click **Modify** to store the new attribute and value in the WebCenter Sites SystemUserAttr database table.
8. Repeat steps 3 through 7 for all users associated with an external identifier.

Analyzing the Mapping Process

When the callCsResolverPage method is called to map an external identifier to a WebCenter Sites system user, it defines the unique CSSO authenticator name for your custom SSO implementation. The method uses the external identifier and the unique CSSO authenticator name to map the external user to the WebCenter Sites system user. This map contains the following items, which are placed in the associated properties of the CustomCredentials object:

- username: The user's WebCenter Sites username.
- currentUser: The user's WebCenter Sites unique identifier.
- currentACL: The user's ACLs.

The CustomCredentials object passes the username, currentUser, and currentACL values to the authenticate method, defined in the CustomAuthenticator.java class. The authenticate method uses these values to build the response map, which identifies the WebCenter Sites user.

76.3.4 Restarting the CAS Web Application

To deploy your custom SSO implementation, restart the CAS web application. Once CAS has been restarted, it uses the classes defined in the custom Spring configuration file, located in the spring-configuration folder (in cas/WEB-INF) to provide the custom login behavior.

76.4 Running the CSSO Sample Implementation

The CSSO facility provides a working example of a custom SSO implementation (including Java source files). The sample replaces the default WEM login behavior with custom login behavior, which includes the standard username and password fields, an additional field for a user to specify a domain name, and a field for an external user identifier. This demonstrates two different types of authentication: username/password pair (with an additional domain field) and user authentication through an external authentication authority.

> **Note:** The CSSO sample does not enforce any validation rules that apply to the fields on the login form. Fields are not checked for completeness and incorrect values are not reported. If authentication fails, the form is re-displayed without comment. If you implement this form in a production environment, you must ensure that all rules are enforced with suitable diagnostic messages if an error occurs.

For information about all the sample components included in the CSSO facility, see Section 76.4.3, "Sample CSSO Components."

To run the sample SSO implementation

1. Deploy the `customizable-sso-1.0.jar` (`Misc/Samples/WEM Samples/CustomizableSSO`) by placing it in the CAS classpath (`cas/WEB-INF/lib` folder). This file contains the sample CSSO classes.

   For more information about the sample classes, see Section 76.4.1, "Sample CSSO Classes."

2. Create a `fatwire` folder in the CAS web application context folder. Copy the `SampleLoginform.jsp` file into the `fatwire` folder.

3. Identify the classes contained in the `customizable-sso-1.0.jar` file to Spring for instantiation:

   a. Copy the `customSampleSSObeans.xml` configuration file into the `spring-configuration` folder.

   b. Modify the properties in the `customSampleSSObeans.xml` file to match your operation environment.

   c. Remove the `.xml` extension from the `customDefaultWEMSSObeans.xml` configuration file's name, located in the `spring-configuration` folder.

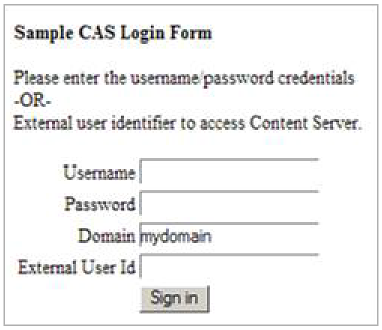
   For more information about the sample Spring configuration file, see Section 76.4.2, "Sample Spring Configuration File."

4. If you wish to use the external identifier credentials to validate users, define the mapping relationship between the external user identifier and the user's WebCenter Sites system credentials by adding the appropriate entry to the `SystemUserAttr` table.

   For instructions, see Section 76.3.3, "Mapping External User Identifiers to WebCenter Sites Credentials."

5. Restart the CAS web application.

   The sample login form looks as follows:



Sample CAS Login Form

Please enter the username/password credentials
-OR-
External user identifier to access Content Server.

Username
Password
Domain mydomain
External User Id

76.4.1 Sample CSSO Classes

The CSSO sample contains three Java classes which extend the default CSSO classes, providing the methods for the sample SSO implementation's login behavior:

- **SampleConfiguration.java**: This class extends the default CSSO `CustomConfiguration.java` class to include a domain property (`sampleDomain`) which will be validated by an external authentication authority when a user provides a value for this field on the login form. The `sampleDomain` property is injected into the CSSO configuration by Spring.

This class also includes the sampleFormURL property which defines the sample login form that is called to retrieve a user's credentials. Standard and custom properties for this class are supplied through the sample Spring configuration file.

- **SampleCredentials.java** : This class extends the default CSSO CustomCredentials.java class and collects all information required to complete user authentication. The SampleAuthenticator class uses the UsernamePasswordCredentials object when a user supplies a username and password on the login form. If a user supplies an external identifier on the login form instead of username and password credentials, the SampleCredentials object is created to provide that information to the authenticator (in this example, sample SSO class SampleAuthenticator.java).

  In CAS, the type of credentials object that is created controls which authenticator is used (either standard or custom). If username and password credentials are supplied on the login form, the standard WEM username and password authenticator is used automatically. If an external identifier is supplied on the login form, the custom authenticator is called to authenticate the SampleCredentials object.

- **SampleAuthenticator.java**: This class extends the default CSSO CustomAuthenticator.java class and contains all the authentication methods that are called by the CSSO framework. When the sample is deployed, all authentication decisions and web-flow actions, during CAS authentication, are directed to this class for action.

  The performLoginAction method (extended by this class) displays the sample login form. When a user submits his credentials on the form, CAS returns to this method to process the input fields. Depending on the credentials that require verification, the method creates either a UsernamePasswordCredentials object or a SampleCredentials object, populated with the user's assigned credentials. The credentials object is then inserted into the CAS context (provided by CAS) and a TGT is requested. The TGT request triggers authentication of the credentials object. If authentication is denied, a ticket exception results in the login form being redisplayed. If the authentication is successful, the next action in the web-flow occurs. For example, acquire a ticket, append the ticket to the original service URL (the WebCenter Sites URL), and redirect back to the original service.

  There are two authentication methods in this class. One handles authentication using SampleCredentials and the other authenticates REST requests, which are usually username/password based. The sample introduces the sampleDomain value as a new value to be authenticated. In this case, the performLoginAction method encodes the username, password, and sampleDomain values provided by the user and passes the encoded values to the UsernamePasswordCredentials object. The default WEM authentication handler detects the sampleDomain value and passes that credential to the authenticationRest method. This method decodes the sampleDomain value from the other values and verifies that the correct domain has been specified. If the value is incorrect, authentication fails. If the value is correct, this method encodes the username and password back into the credentials object, and the default WEM authentication handler validates the username and password.

76.4.2 Sample Spring Configuration File

The classes and properties for the sample SSO implementation are defined by the sample Spring configuration file customSampleSSObeans.xml (located in Misc/Samples/WEM Samples/CustomizableSSO/src/main/webapp/WEB-INF/spring-configuration).

The rest of this section contains the following topics:

- Section 76.4.2.1, "Analyzing the Sample Spring Configuration File"
- Section 76.4.2.2, "Placing the Sample Spring Configuration File"

76.4.2.1 Analyzing the Sample Spring Configuration File

The sample Spring configuration file contains the same bean identifiers as the default Spring configuration file (see, Section 76.3.2.1, "Creating a Spring Configuration File"). However, the property values are modified to implement the sample login behavior. For example, the `customUserConfiguration` bean references the `SampleConfiguration.java` class and the `customUserAuthenticator` bean references the `SampleAuthenticator.java` class.

The `customUserConfiguration` bean also identifies the configuration properties which supply system information to the sample SSO implementation. For example, since the `SampleLoginForm.jsp` file provides the browser form that is used by the sample to obtain a user's credentials, the `SampleConfiguration.java` class is extended to include the `sampleFormURL` property. This property specifies the full URL of the login page for the sample SSO implementation. The domain name and port number match the CAS server installation, and the path points to where this page was placed during set up.

The following is the sample Spring configuration file's code. For more information about the properties referenced by this file, see "Analyzing the default Spring configuration file" in Section 76.3.2.1, "Creating a Spring Configuration File".

The sample Spring configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:flow="http://www.springframework.org/schema/webflow-config"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://www.springframework.org/schema/webflow-config
http://www.springframework.org/schema/webflow-config/spring-webflow-config-1.0.xsd
">

<!-- Custom SSO Bean definitions. This file defines either the
default CAS/SSO configuration or a special user implementation.
No other CAS configuration files are modified for a custom implementation
-->

<!-- This bean is never modified. It defines the web-flow controller which always
passes control into the custom authenticator -->
<bean id="customUserLoginAction"
class="com.fatwire.wem.sso.cas.web.CustomLoginViewAction"
p:centralAuthenticationService-ref="centralAuthenticationService"
p:customAuthentication-ref="customUserAuthenticator"
/>
<!-- This bean is usually not modified. Override it when there needs to be a
custom encoding for information passed between the web-flow and any external
component -->
<bean id="customRestCoder" class="com.fatwire.wem.sso.cas.custom.basis.
CustomRestTokenCoding"
/>
<!-- Modify this bean with a custom configuration class when additional parameters
are needed for a custom implementation -->
<bean id="customUserConfiguration"
```

```
class="com.fatwire.wem.sso.cas.sample.SampleConfiguration"
p:casLoginUrl="http://localhost:7080/cas/login"
p:resolverUrl="http://localhost:8080/cs/custom/customCsResolver.jsp"
p:resolverUsername="fwadmin"
p:resolverPassword="xceladmin"
p:traceFlag="false"
p:sampleDomain="mydomain"
p:sampleFormUrl="http://localhost:7080/cas/SampleLoginForm.jsp"
/>
<!-- Modify this bean with a customAuthentication class for a custom
implementation. -->
<bean id="customUserAuthenticator"
class="com.fatwire.wem.sso.cas.sample.SampleAuthenticator"
p:customConfiguration-ref="customUserConfiguration"
p:customRestCoder-ref="customRestCoder"
/>
</beans>
```

76.4.2.2 Placing the Sample Spring Configuration File

To instantiate the sample classes, place the sample Spring configuration file in the spring-configuration folder (in cas/WEB-INF) and remove the .xml extension from the default Spring configuration file. For more information, see Section 76.3.2.2, "Placing Your Spring Configuration File."

76.4.3 Sample CSSO Components

The sample CSSO implementation's components are located in the /WEM Samples/CustomizableSSO folder. The following folders are included with the sample CSSO implementation:

Table 76–3 Sample CSSO Components

Folder	Description
Misc/Samples/WEM Samples/CustomizableSSO	Contains the customizable-sso-1.0.jar file. This jar file provides the classes of the executable code for the sample. If you wish to deploy the sample SSO implementation, place this jar file in the CAS classpath (cas/WEB-INF/lib folder).
Misc/Samples/WEM Samples/CustomizableSSO/lib	Contains all the third-party jar files required to compile the Java source files for the sample SSO implementation.
Misc/Samples/WEM Samples/CustomizableSSO/src/main/dist	Contains Word documents that explain the individual source components and operations of the sample implementation. **Note:** We recommend reviewing these documents before viewing the sample's source code.
Misc/Samples/WEM Samples/CustomizableSSO/src/main/java	The root folder for the Java source files.

Table 76–3 (Cont.) Sample CSSO Components

Folder	Description
`Misc/Samples/WEM Samples/CustomizableSSO/src/main/webapp/fatwire`	Contains `SampleLoginForm.jsp`. The JSP provides the browser form that is used by the sample to obtain a user's login credentials. Implementing the sample requires creating a `fatwire` folder in the CAS application context folder and copying the `SampleLoginForm.jsp` to that folder.
`Misc/Samples/WEM Samples/CustomizableSSO/src/main/webapp/WEB-INF/spring-configuration`	Contains the sample Spring configuration file `customSampleSSObeans.xml`, which defines the Spring bean definitions required by the sample SSO implementation. This file must be placed in the `spring-configuration` folder (in `cas/WEB-INF`). The file that exists in the `spring-configuration` folder (`customDefaultWEMSSObeans.xml`) must be given an extension other than `.xml` or removed. **Note:** Save a copy of the `customDefaultWEMSSObeans.xml` file so it can be restored when you wish to return to the standard WEM login screen.

77

WEM Framework: Buffering

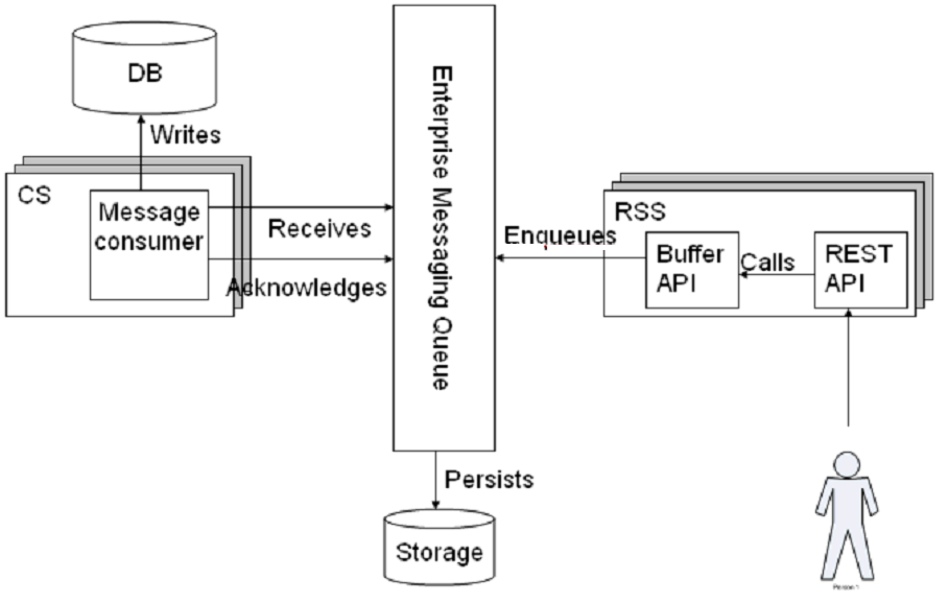
Asset create, update, and delete operations are much slower than the read operation. Sometimes, it is acceptable to delay these operations to occur at a future time with the guarantee of eventual consistency. That is, if a delayed (buffered) operation was performed, it is guaranteed that the WebCenter Sites platform will receive the change at some finite, undetermined period of time. Although buffering operations are extremely fast, they do not speed up the total time that is needed to create, update, and delete assets in the platform.

This chapter contains the following sections:

- Section 77.1, "Architecture"
- Section 77.2, "Using Buffering"

77.1 Architecture

The current implementation of buffering subsystem relies on Java Messaging Service (JMS) technology.



Buffering consists of the following components:

- Buffering Producer, which produces messages and puts them into the Messaging Queue (MQ).

- Buffering Consumer, which picks messages from MQ and persists them in the platform.

The buffering producer can be used on both WebCenter Sites and Remote Satellite Server, where the asset REST service `<BaseURI>/sites/<sitename>/types/<assettype>/assets/<id>` is available. When used on Remote Satellite Server, the buffering producer does not communicate with WebCenter Sites, which ensures linear scalability of the entire system.

> **Note:** The buffering consumer is available only on WebCenter Sites. We recommend enabling the buffering consumer only on the primary cluster member. Enabling on multiple cluster members cannot guarantee that the sequence of CRUD operations will be preserved.

77.2 Using Buffering

1. Install the JMS provider if one is not already available. (For supported providers, see the *Oracle Fusion Middleware WebCenter Sites Certification Matrix* available from the Oracle Technology Network at `http://otn.oracle.com`.
2. Configure `BufferingConfig.xml` on WebCenter Sites and optionally on Remote Satellite Server.

```
id="bufferingManager" class=
  "com.fatwire.cs.core.buffering.jms.JmsBufferingManager"
```

Table 77–1 Properties in BufferingConfig.xml

Property name	Description
`jmsConnectionFactory`	Required. Instance of `javax.jms.ConnectionFactory`
`jmsDestination`	Required. Instance of `javax.jms.Destination`
`messageConsumers`	List of `com.fatwire.cs.core.buffering.IMessageConsumer` implementations.

> **Note:** When you configure `BufferConfig.xml`, add `activemq-all.jar` to the WebCenter Sites web application's classpath (for example, `WEB-INF/lib`).

3. Specify `buffer=true` when invoking the REST asset service `<BaseURI>/sites/<sitename>/types/<assettype>/assets/<id>`.

> **Note:** Buffering does not return the result of `PUT` and `POST` operations in the response. Instead, an empty payload is sent. Developers should be aware of this behavior when coding the client application.

The default `BufferingConfig.xml` file, provided with WebCenter Sites, contains the sample configuration for Apache ActiveMQ. The `BufferingConfig.xml` file is similar for both WebCenter Sites and Remote Satellite Server, except that the list of message consumers for Remote Satellite Server is empty.

78

WEM Framework: Registering Applications Manually

Registration exposes applications in the WEM Framework, as described in Chapter 70, "WEM Framework: Understanding the Framework and Services." Registering an application manually requires using the WebCenter Sites Admin interface to create an asset for the application, create an asset for each of its views, and associate the view assets with the application asset. The registration asset types `FW_Application` and `FW_View` are enabled on AdminSite.

This chapter contains the following sections:

- Section 78.1, "Registration Steps"
- Section 78.2, "Reference: Registration Asset Types"

78.1 Registration Steps

To manually register an application and view

The section uses code from the "Articles" sample application to illustrate the registration process. "Articles" has a single view of type iframe. The same steps apply to JavaScript and HTML views.

1. Create or get an icon to represent your application. (The icon will be displayed in the applications bar.)

   The "Articles" sample application uses the `articles.png` image file located in: `/sample app/articles/src/main/webapp/images/`

2. Create a file that specifies the layout of the application in HTML, i.e., for each view, create a placeholder element to hold the content rendered by the view. Applications and views are related as shown in Figure 72–4.

   For example, `layout.jsp` (for the "Articles" sample application) contains the following line:

   ```
   <div id="articles" style="float:left;height:100%;width:100%"
     class="wemholder"></div>
   ```

   The view's content will be rendered within the placeholder element when the application is displayed (`layout.app` renders the application's layout; `home.app` renders the view).

Note: When creating the layout file, specify a unique `id` for the placeholder element. You will specify the same `id` for the Parent Node attribute when creating the view asset. Use `class="wemholder"` for the placeholder elements.

---

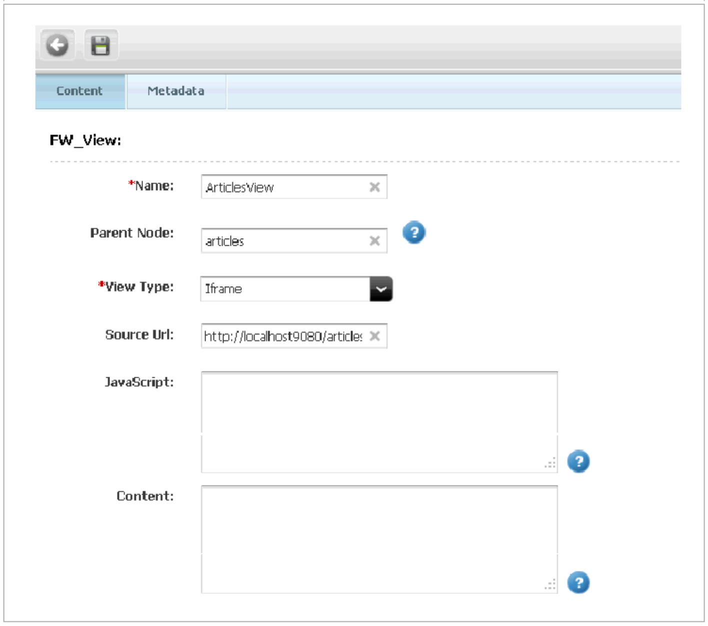
3. Register the view and application.

   a. Log in to the WebCenter Sites Admin interface as a general administrator, navigate to the AdminSite and click the **Admin** tab, where the `FW_View` and `FW_Application` asset types are enabled.

      (We assume you will create the view and application assets in the same session, in which case both assets will be listed on the **History** tab. When creating the application asset, you will select the view asset from the **History** tab and associate it with the application asset. The **History** tab is volatile; it is cleared at the end of the user's session. Assets can be permanently placed on the **Active List** tab. For instructions, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.)

   b. Create an instance of the `FW_View` asset type:

      Click **New,** select **New FW_View**, and set attributes as shown below this figure. (This figure displays attribute values for the view asset of the "Articles" sample application.)



FW_View:

*Name: ArticlesView

Parent Node: articles

*View Type: Iframe

Source Url: http://localhost9080/article

JavaScript:

Content:

Name: Enter a short descriptive name for this view asset.

Parent Node: Enter the `id` of the placeholder element (defined in step 2 of Section 78.1, "Registration Steps") that will hold the content rendered by the view.

View Type: Select one of the following options to specify how the view's content should be rendered in the placeholder:

– `Iframe`: Renders the view in an iframe into the placeholder element

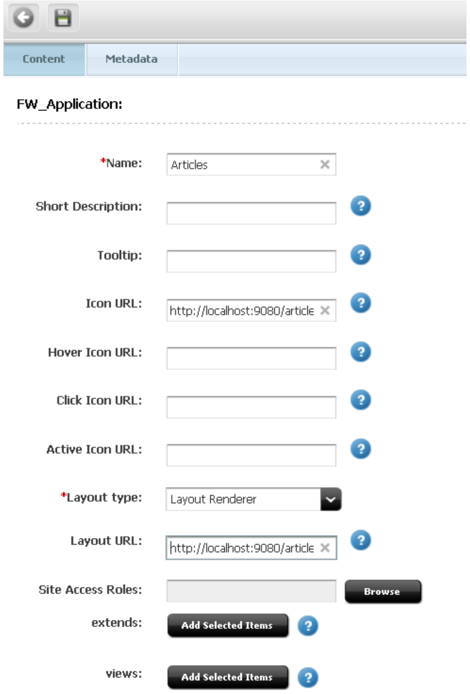
– `IncludeHTML`: Renders HTML into the placeholder element

– `IncludeJavaScript`: Renders JavaScript into the placeholder element

Source URL: Enter the URL that provides content for the view. For example, Source URL for the "Articles" sample application takes the following value: `http://localhost:9080/articles-1.0/home.app`

c. Create an instance of the `FW_Application` asset type:

Click **New**, select **New FW_Application**, and set attributes as shown below this figure. (This figure displays attribute values for the application asset of the "Articles" sample application.)



FW_Application:

*Name: Articles

Short Description:

Tooltip:

Icon URL: http://localhost:9080/article

Hover Icon URL:

Click Icon URL:

Active Icon URL:

*Layout type: Layout Renderer

Layout URL: http://localhost:9080/article

Site Access Roles:

extends:

views:

Name: Enter a short descriptive name for this application asset.

ToolTip: Enter the text that will be displayed over the application's icon when users mouse over the icon.

Icon URL: Enter the URL of the icon that represents the application. The icon will be displayed on the login page and at the top of the WEM interface. For example, the Icon URL for the "Articles" sample application takes the following value: `http://localhost:9080/articles-1.0/images/articles.png`

Hover Icon URL: Enter the URL of the icon that represents the application when users mouse over the icon.

Click Icon URL: Enter the URL of the icon that represents the application when users click on the icon.

Active Icon URL: Enter the URL of the icon that represents the application when it is in use.

Layout Type: `LayoutRenderer` (the default and only value). Layout Type is used by the UI container to render the application's views by using the application's layout page (specified below in the Layout URL attribute).

Layout URL: Enter the URL of the page that displays the application's layout. The layout page has only HTML placeholder elements (such as `div`) for placing the view(s).

For example, Layout URL for the "Articles" sample application takes the value: `"http://localhost:9080/articles-1.0/layout.app"` rather than `".../layout.jsp"`, given the Spring MVC framework.

Related: Associated FW_View: views: Select the view asset created in step 3 of Chapter 78.1, "Registration Steps" (click the **History** tab, select the view asset, and click **Add Selected Items**).

78.2 Reference: Registration Asset Types

This section contains the following topics:

- Section 78.2.1, "FW_View Asset Type"
- Section 78.2.2, "FW_Application Asset Type"

78.2.1 FW_View Asset Type

This asset type is used to register the views of an application. For each view, create an instance of `FW_View`. Attributes of `FW_View` are listed below as they appear in the WebCenter Sites Admin interface and in the *Oracle Fusion Middleware WebCenter Sites REST API Bean Reference*. Shading indicates a required attribute. This asset type is enabled on the site named 'AdminSite.'

Table 78–1 ***`FW_View` Asset Type Attributes***

Attribute: WebCenter Sites Interface	Attribute: REST API	Description
Name	name	Short descriptive name for this view asset.
Description	description	Description of this view asset.
Parent Node	parentnode	ID of the placeholder element in the application's layout file. The placeholder element will hold the content rendered by the view. The layout file has only HTML placeholder elements (such as `div`) for placing the views.
View Type	viewtype	How the view should be rendered. The following view types are available: Iframe: renders the view in an iframe into the placeholder element IncludeHTML: renders HTML into the placeholder element IncludeJavaScript: renders JavaScript into the placeholder element
Source URL	sourceurl	URL that provides content for the view.

Table 78–1 (Cont.) `FW_View` Asset Type Attributes

Attribute: WebCenter Sites Interface	Attribute: REST API	Description
JavaScript	javascriptcontent	Required if IncludeJavaScript is the view type and Source URL is not specified. The content specified by this attribute is included in a script tag if IncludeJavaScript is specified as the view type. If `IncludeJavaScript` is specified, either `Source URL` must be specified, or code must be provided for the `JavaScript` attribute.
Content	includecontent	Required if IncludeHTML is the view type and Source URL is not specified. The content specified by this attribute is included in the placeholder element tag if IncludeHTML is specified as the view type. If `IncludeHTML` is specified, either the `Source URL` must be specified or code must be provided for the `Content` attribute.

78.2.2 FW_Application Asset Type

This asset type is used to register the application. The asset type is enabled on AdminSite. Attributes of `FW_Application` are listed below as they appear in the WebCenter Sites Admin interface and in the *Oracle Fusion Middleware WebCenter Sites REST API Bean Reference*. Shading indicates a required attribute.

Table 78–2 `FW_Application` Asset Type Attributes

Attribute: WebCenter Sites Interface	Attribute: REST API	Description
Name	name	Short descriptive name for this application asset.
Description	description	Description of this application asset.
Tooltip	tooltip	Text that will be displayed on the application's icon when users mouse over the icon.
Icon URL	iconurl	URL of the icon that represents the application in the WEM Framework.
Hover Icon URL	iconurlhover	URL of the icon that represents the application when users mouse over the icon.
Click Icon URL	clickiconurl	URL of the icon that represents the application when users click on the icon.

Table 78–2 (Cont.) `FW_Application` Asset Type Attributes

Attribute: WebCenter Sites Interface	Attribute: REST API	Description
Active Icon URL	iconurlactive	URL of the icon that represents the application while it is in use.
Layout Type	layouttype	Type of layout. The value is `LayoutRenderer`. Layout Type is responsible for rendering the application's views by using the application's layout page (specified in the Layout URL attribute, below).
Layout URL	layouturl	URL of the page where the application's layout is displayed. This page has only HTML placeholder elements (such as `div`) for placing the views.
Related: Associated FW_Application: extends	parentnode	Parent application which the current application extends.
Related: Associated FW_View: views	views	List of view assets used in this application.

Part VI

Developing the Community-Gadgets Application

This part begins with an overview of the Community-Gadgets application and its developers. It continues to the process of integrating the Community-Gadgets application with Facebook, Twitter, Google, and Janrain. It also provides information about customizing Community-Gadgets, as well as translating its functionality into various languages. Finally, this part includes guidelines and tips for maintaining the Community-Gadgets website presence.

This part contains the following chapters:

79

About Oracle WebCenter Sites: Community-Gadgets

Oracle WebCenter Sites: Community-Gadgets is a social computing web application designed to gather visitors' comments, reviews, and ratings on web site content. Community-Gadgets also enables administrators and moderators to create and manage polls that can be used to survey site visitors about desired topics.

This chapter contains the following sections:

- Section 79.1, "Introduction to Oracle WebCenter Sites: Community-Gadgets"
- Section 79.2, "Community-Gadgets Management and Production Components"
- Section 79.3, "Technical Overview of Oracle WebCenter Sites: Community-Gadgets"
- Section 79.4, "Prerequisites"

79.1 Introduction to Oracle WebCenter Sites: Community-Gadgets

Community-Gadgets is a set of JavaEE web applications that integrates with Oracle WebCenter Sites and works in a distributed environment as a social computing application. Community-Gadgets provides it users with two Web Experience Management (WEM) applications:

- **Community WEM application**: Provides the Community interface, which is used to configure widgets for collecting visitors' comments, reviews, ratings, and votes at polls. The Community WEM application is also used to moderate user-generated content.
- **Gadgets WEM application**: Provides the Gadgets administrator interface (Global Catalog) and the Gadgets user interface. Which interface is displayed to the user depends on the user's roles. Both interfaces are used for gadgets management, which includes registering, sharing, and configuring gadgets and the dashboard.

For more information about the WEM Framework and WEM applications, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide* and the *Oracle Fusion Middleware WebCenter Sites Developer's Guide*.

79.2 Community-Gadgets Management and Production Components

Community-Gadgets consists of two parts: a management set of applications, and a production (delivery) set of applications.

> **Note:** Management Community-Gadgets and production Community-Gadgets must be deployed on separate application server instances. Deployment of both on a single application server is not supported.

- Management Community-Gadgets contains two web applications:
  - Community-Gadgets web application
  - Shindig web application
- Production Community-Gadgets contains three web applications:
  - Community-Gadgets web application
  - Shindig web application
  - CAS web application

Management Community-Gadgets is used for the administration of user-generated content (UGC), registering gadgets, and similar functions. For security reasons, it is typically accessible only internally. Production Community-Gadgets is accessed by visitors through widgets and gadget dashboards deployed onto web pages. It manages visitor authorization through Visitors CAS.

Both management and production Community-Gadgets use Shindig OpenSocial container for rendering gadgets. Production Community-Gadgets must be externally accessible. More information about the Shindig web application is available at the following URL:

`http://shindig.apache.org/`

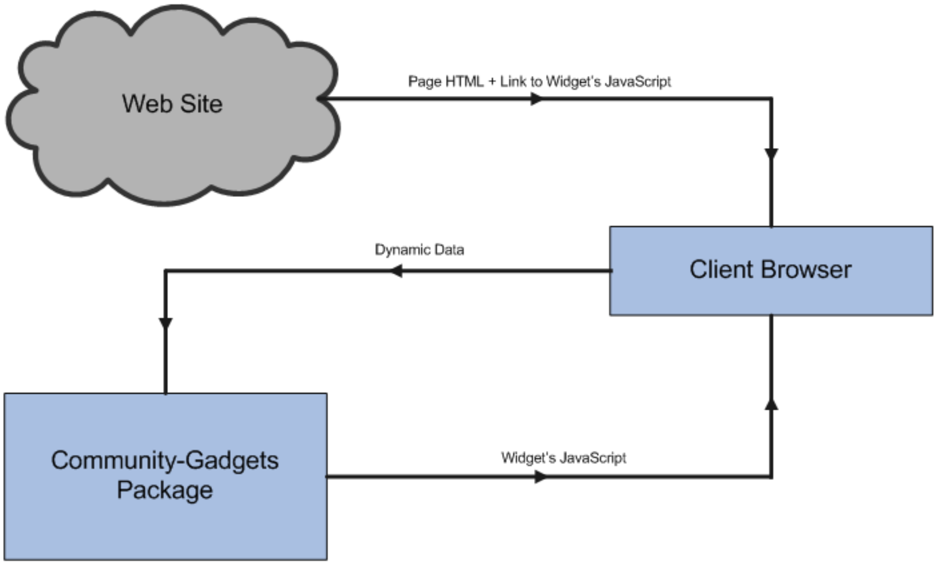
79.3 Technical Overview of Oracle WebCenter Sites: Community-Gadgets

Community-Gadgets functionality such as comments, reviews, and polls is developed as separate modules called *widgets*. Each widget can be deployed on a web site by using its JavaScript code snippet, which renders the widget in a browser. The ability to deploy widgets in a browser makes deployment simple and platform-independent. The widget deployment approach also facilitates high scale maintenance and upgrades.

Community-Gadgets widgets are compatible with most server-side and backend technologies, and therefore, there are no major deployment prerequisites that developers must know of. Since Community-Gadgets is an add-on for WebCenter Sites, it can be developed separately from WebCenter Sites.

The use of JavaScript enables developers to create easy-to-use and feature-rich widget interfaces. Developers can build dynamic screens that are responsive to user actions and do not require the whole web page to be re-loaded. As soon as a widget is deployed on a web page, it starts rendering and loading dynamic content such as lists of comments, reviews, or particular poll information from the data repository. The data for a widget is loaded using JavaScript, with the help of a remote scripting technology similar to AJAX. This technology also facilitates the deployment of the Community-Gadgets web application on a domain other than the original web site domain. Figure 79–1 shows the post-deployment web site environment.

Figure 79–1 Client Web Site Environment After Deploying Community-Gadgets Functionality


Web Site
Page HTML + Link to Widget's JavaScript
Dynamic Data
Client Browser
Community-Gadgets Package
Widget's JavaScript

To become familiar with the Community-Gadgets object data model on which the Community-Gadgets widgets are based, see Section 81.1, "Overview of Community-Gadgets Data Model."

79.4 Prerequisites

Developers must understand how the Community-Gadgets web application is installed and know the Community interface and the Gadgets interface. To effectively use this guide, developers must have experience with the following:

- WEM Framework, REST API, and WEM Admin interface.
- Oracle WebCenter Sites asset model and Oracle WebCenter Sites: Satellite Server for caching.
- Central Authentication Service (CAS) used for Web Single Sign-On.
- Deployment of web applications on Oracle WebLogic Server, WebSphere Application Server or Tomcat application server.

80

Community-Gadgets: Integrating with Social Networking Services

The Community-Gadgets widgets are designed to leverage the authentication APIs of Facebook, Twitter, and Janrain on your web site. Community-Gadgets widgets enable the visitors of your web site (or blogs) to use their existing social networking profiles to provide feedback on your company's content and to share the same content on their social networking profiles. This chapter describes how to integrate the Community-Gadgets web application with Facebook, Twitter, Google, Janrain, and Oracle Mobile and Social Access Service (OMSAS).

This chapter contains the following sections:

80.1 About Authentication Plug-Ins

Developers can leverage third-party authentication providers in the following ways:

- Use the built-in support of Facebook, Twitter, and Google, which is available ready-to-use with the Community-Gadgets web application.
- Use the authentication hub, Janrain, to enable access to most prevalent providers available online. Janrain is a SaaS solution that provides integration services with a number of online authentication providers such as Facebook, Twitter, Google, and so on. The more services supported, the more chances that the site visitor has an identity in one of these services, which they may use to log in.
- Use the authentication hub, Oracle Mobile and Social Access Service (OMSAS), to enable access to prevalent providers available online. OMSAS is a solution that provides integration services with a number of online authentication providers such as Facebook, Twitter, Google, and so on.

Note: Community-Gadgets uses only social functionality from OMSAS. Mobile functionality is not used.

80.2 Integrating with Facebook

On a web site, visitors' Facebook identities are authenticated via a trusted bridge, which is established between the Community-Gadgets web application and Facebook at each login attempt. To enable this bridge, you must first create a Facebook application using the Community-Gadgets production site URL. Then, you will configure the Facebook authentication properties on the application server on the production system of the Community-Gadgets web application. This section describes how to:

- Section 80.2.1, "Create a Facebook Application for Community-Gadgets"
- Section 80.2.2, "Configure Facebook Application Authentication Settings on the Community-Gadgets Web Application"

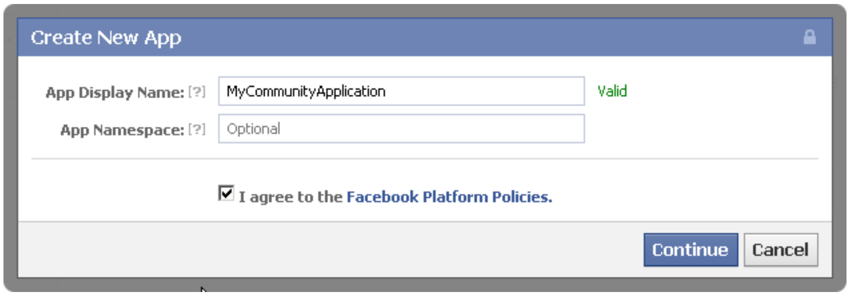
80.2.1 Create a Facebook Application for Community-Gadgets

Note: Steps and screens shown in this section may not match the Facebook interface at the time when you create an application. If you see new fields and require help, contact product support.

To create a Facebook application for Community-Gadgets:

1. Log in to Facebook to access the Apps page: `https://developers.facebook.com/apps`
2. On the top right corner of the page, click **Create New App** to display the New App dialog box.
3. In the **App Display Name** field, enter a name for the Community-Gadgets web application.
4. Select the **I agree to the Facebook Platform Policies** check box, then click **Continue**.

Figure 80–1 New App: Facebook



Create New App

App Display Name: [?] MyCommunityApplication Valid

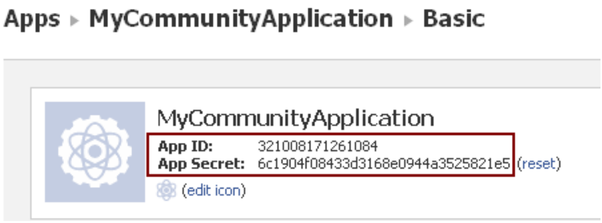
App Namespace: [?] Optional

☑ I agree to the Facebook Platform Policies.

5. On the Security Check Required page, in the **Text in the box** field, enter the displayed CAPTCHA text. Then, click **Submit**.

6. Under **Basic**, copy the values of **App ID** and **App Secret** parameters (Figure 80–2). You will need these values while configuring the Facebook authentication properties on the application server.

Figure 80–2 App ID And App Secret



Apps ▸ MyCommunityApplication ▸ Basic

MyCommunityApplication

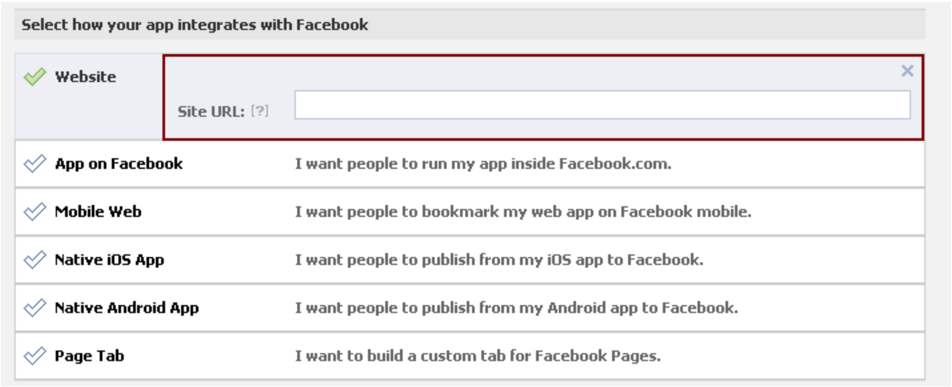
App ID: 321008171261084

App Secret: 6c1904f08433d3168e0944a3525821e5 (reset)

(edit icon)

7. Under **Select how your app integrates with Facebook**, click the gray text next to **Website** to display the **Site URL** input field, as shown in Figure 80–3.

Figure 80–3 Select How Your App Integrates With Facebook



Select how your app integrates with Facebook

✓ Website	Site URL: [?]
✓ App on Facebook	I want people to run my app inside Facebook.com.
✓ Mobile Web	I want people to bookmark my web app on Facebook mobile.
✓ Native iOS App	I want people to publish from my iOS app to Facebook.
✓ Native Android App	I want people to publish from my Android app to Facebook.
✓ Page Tab	I want to build a custom tab for Facebook Pages.

8. In the **Site URL** field, enter the production address for the Community-Gadgets web application, and click **Save Changes**.

80.2.2 Configure Facebook Application Authentication Settings on the Community-Gadgets Web Application

To configure Facebook application authentication properties on the Community-Gadgets application server:

1. In the application server home on the production system, navigate to the standalone_node folders. Default management and production locations are:

```
<cg_install_dir>/deploy/management/management_node1
<cg_install_dir>/deploy/production/production_node1
```

2. Open the `setup_auth.properties` file, then scroll down to the Facebook Application Settings section.

3. For `widgets.external_auth.facebook.attrs.client_id` and `widgets.external_auth.facebook.attrs.client_secret` configuration properties, enter the application ID and application secret values you copied at the time of creating the Facebook application.

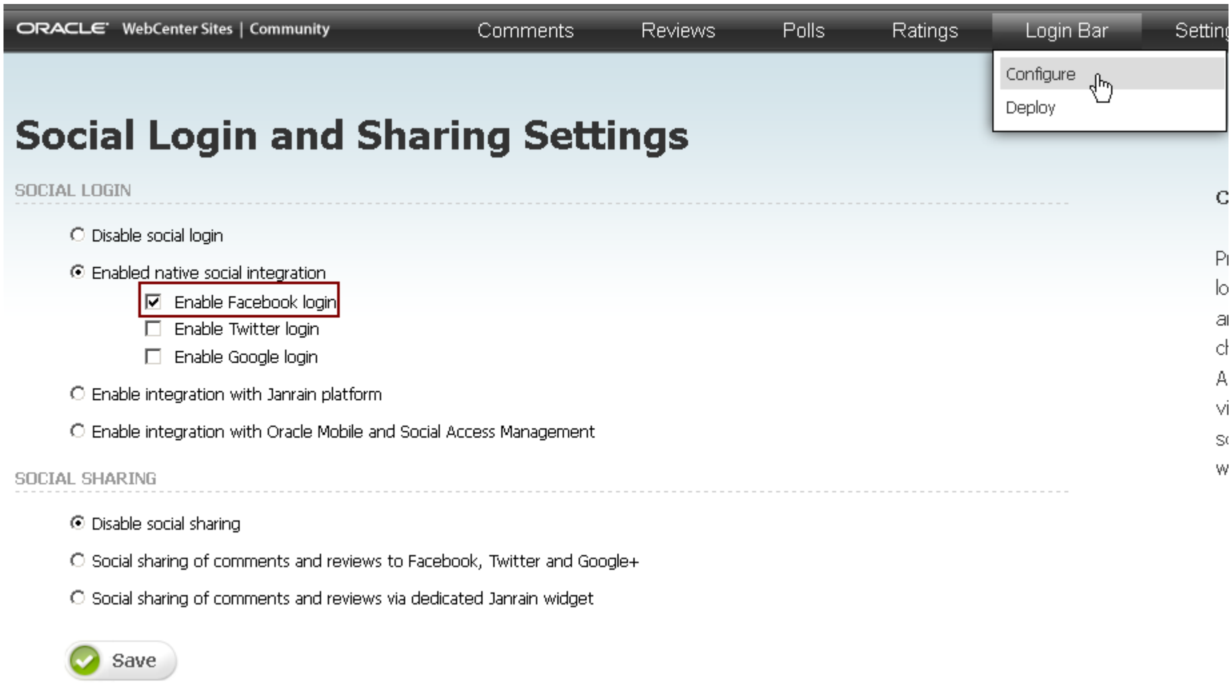
   The Facebook Application Settings section should look like this:

```
## Facebook Application Settings
```

```
#######################################
#
# Facebook application id
#
widgets.external_auth.facebook.attrs.client_id=1111111111111111111
#
# Facebook application secret
#
widgets.external_auth.facebook.attrs.client_secret=571bf600e8d23
```

4. After saving the configuration file, restart the application server.
5. Open the Community interface or the Gadgets interface on the management side as a general administrator: `http://`*`host:port`*`/cs/login`.
6. Choose the desired site.
7. From the **Login Bar** menu, choose **Configure**.
8. On the Social Login and Sharing Settings page, in the **Social Login** section, under **Enabled native social integration**, select the **Enable Facebook login** check box (Figure 80–4), then click **Save**.

Figure 80–4 Enable Facebook Login



Social Login and Sharing Settings

SOCIAL LOGIN

○ Disable social login

● Enabled native social integration

☑ Enable Facebook login

□ Enable Twitter login

□ Enable Google login

○ Enable integration with Janrain platform

○ Enable integration with Oracle Mobile and Social Access Management

SOCIAL SHARING

● Disable social sharing

○ Social sharing of comments and reviews to Facebook, Twitter and Google+

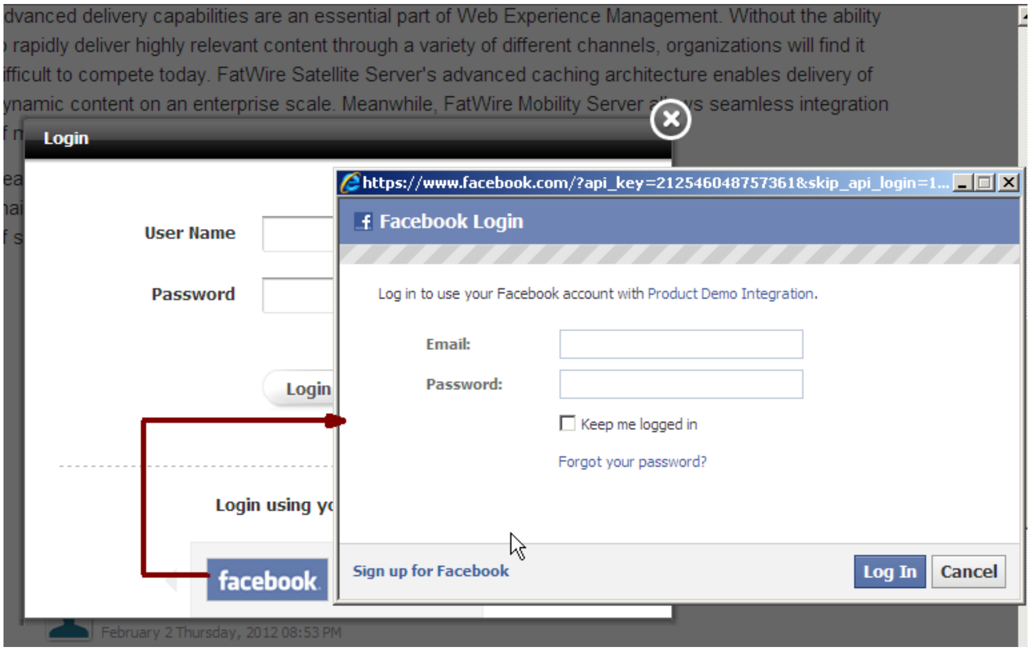
○ Social sharing of comments and reviews via dedicated Janrain widget

Save

9. Restart the management and production environments.
10. Access the web site on which the widget is deployed.

    The Facebook login button is displayed (Figure 80–5) in the Community-Gadgets widget's Login screen.

Figure 80–5 Facebook Login Button


dvanced delivery capabilities are an essential part of Web Experience Management. Without the ability
 rapidly deliver highly relevant content through a variety of different channels, organizations will find it
ifficult to compete today. FatWire Satellite Server's advanced caching architecture enables delivery of
ynamic content on an enterprise scale. Meanwhile, FatWire Mobility Server al s seamless integration

Log in to use your Facebook account with Product Demo Integration.

Note: If your Community-Gadgets web application is running on Oracle WebLogic Server or Tomcat application server, then the procedures described in Section 80.2.1, "Create a Facebook Application for Community-Gadgets" and Section 80.2.2, "Configure Facebook Application Authentication Settings on the Community-Gadgets Web Application" are all you require to integrate Community-Gadgets with Facebook. However, on a WebSphere Application Server instance, you must also perform the export/import procedures described in Section 80.7, "Enabling Social Networking Services on WebSphere Application Server" to enable communication between Community-Gadgets and this social networking service.

80.3 Integrating with Twitter

You can integrate the Community-Gadgets web application with Twitter just like you integrated it with Facebook. That is, first create a Twitter application using the production site URL, then update the authentication properties on the application server to include the Twitter application's required information.

This section describes how to:

- Section 80.3.1, "Create a Twitter Application for Community-Gadgets"
- Section 80.3.2, "Configure Twitter Application Authentication Settings on Community-Gadgets"

80.3.1 Create a Twitter Application for Community-Gadgets

To create a Twitter application:

1. Log in to Twitter's developers central at `https://dev.twitter.com/`.
2. Under **Create applications that integrate Twitter**, click **Create an app**.

3. Under **Application Details**, enter the required information as follows:

   a. In the **Name** and **Description** fields, enter a name for the Community-Gadgets web application and a description.

   b. In the **WebSite** field, enter the Community-Gadgets production server URL. For example, `http://production.example.com`.

   c. In the **Callback URL** field, enter the production host location of the Community-Gadgets web application, which is the same as the URL in step b.

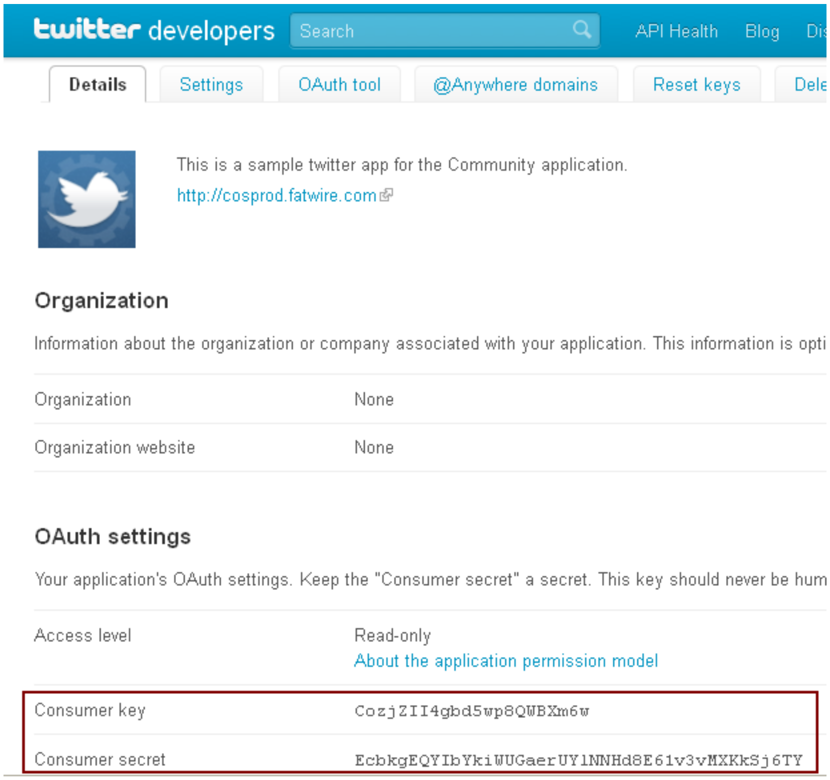
   d. Select the **Yes, I agree** check box.

   e. Under **CAPTCHA**, enter the words you see on the screen.

   f. Click **Create your Twitter application**.

4. From the Details page, copy the values of **Consumer key** and **Consumer secret** properties displayed in the **OAuth settings** section (Figure 80–6). You will need these values to configure the authentication properties on the application server.

Figure 80–6 OAuth Settings



This is a sample twitter app for the Community application.

http://cosprod.fatwire.com

Organization

Information about the organization or company associated with your application. This information is opti

Organization	None
Organization website	None

OAuth settings

Your application's OAuth settings. Keep the "Consumer secret" a secret. This key should never be hum

Access level	Read-only About the application permission model
Consumer key	CozjZII4gbd5wp8QWBXm6w
Consumer secret	EcbkgEQYIbYkiWUGaerUY1NNHd8E61v3vMXKkSj6TY

80.3.2 Configure Twitter Application Authentication Settings on Community-Gadgets

To configure the authentication properties on Community-Gadgets:

1. In the application server home on the production system, navigate to the standalone_node folders. Default management and production locations are:

```
<cg_install_dir>/deploy/management/management_node1
<cg_install_dir>/deploy/production/production_node1
```

2. Open the `setup_auth.properties` file, then scroll down to the Twitter Application Settings section.

3. For widgets.external_auth.twitter.attrs.consumer_key and widgets.external_auth.twitter.attrs.consumer_secret configuration properties, enter the consumer key and consumer secret values you copied earlier.

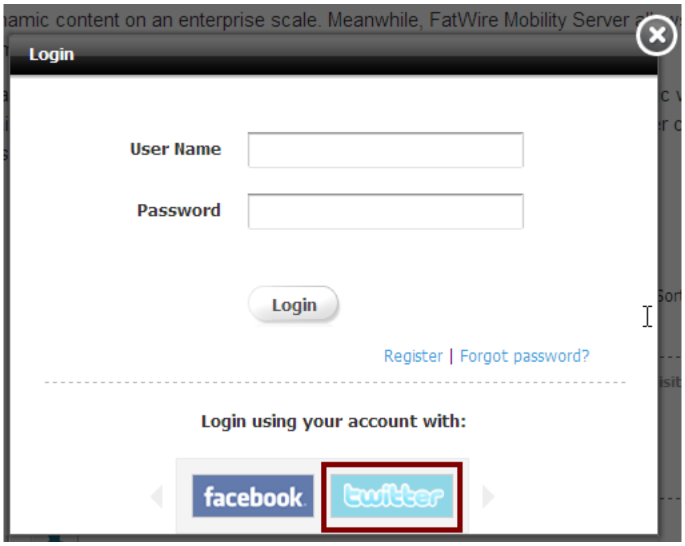
   The Twitter Application Settings section should look like this:

   ```
   ## Twitter Application Settings
   # Twitter consumer key
   #
   widgets.external_auth.twitter.attrs.consumer_key=uh5nvtwCg5WjBD9UHM29eQ
   #
   # Twitter consumer secret
   #
   widgets.external_auth.twitter.attrs.consumer_secret=CaYaLDk9IZBConpQVkxk
   ```

4. After saving the configuration file, restart the application server.
5. Open the Community interface or the Gadgets interface on the management side as a general administrator: http://*host*:*port*/cs/login.
6. Choose the desired site.
7. From the **Login Bar** menu, choose **Configure**.
8. On the Social Login and Sharing Settings page, in the **Social Login** section, under **Enabled native social integration**, select the **Enable Twitter login** check box, then click **Save**.
9. Restart the management and production environments.
10. Access the web site on which the widget is deployed.

    The Twitter login button is displayed (Figure 80–7) in the Community-Gadgets widget's Login screen on the web site on which the widget is deployed.

Figure 80–7 Twitter Login Button


amic content on an enterprise scale. Meanwhile, FatWire Mobility Server al
Login
User Name
Password
Login
Register | Forgot password?
Login using your account with:
facebook
twitter

Note: If your Community-Gadgets web application is running on Oracle WebLogic Server or Tomcat application server, then the procedures described in Section 80.3.1, "Create a Twitter Application for Community-Gadgets" and Section 80.3.2, "Configure Twitter Application Authentication Settings on Community-Gadgets" are all you require to integrate Community-Gadgets with Twitter. However, on a WebSphere Application Server instance, you must also perform the export/import procedures described in Section 80.7, "Enabling Social Networking Services on WebSphere Application Server" to enable communication between Community-Gadgets and this social networking service.

80.4 Integrating with Google

On a web site, visitors' Google identities are authenticated via a trusted bridge, which is established between the Community-Gadgets web application and Google at each login attempt. To enable this bridge, you must first create a Google application using the Community-Gadgets production site URL. Then, configure the Google authentication properties on the application server on the production system of the Community-Gadgets web application.

This section describes how to:

- Section 80.4.1, "Create a Google Application for Community-Gadgets"
- Section 80.4.2, "Configure Google Application Authentication Settings on Community-Gadgets"

80.4.1 Create a Google Application for Community-Gadgets

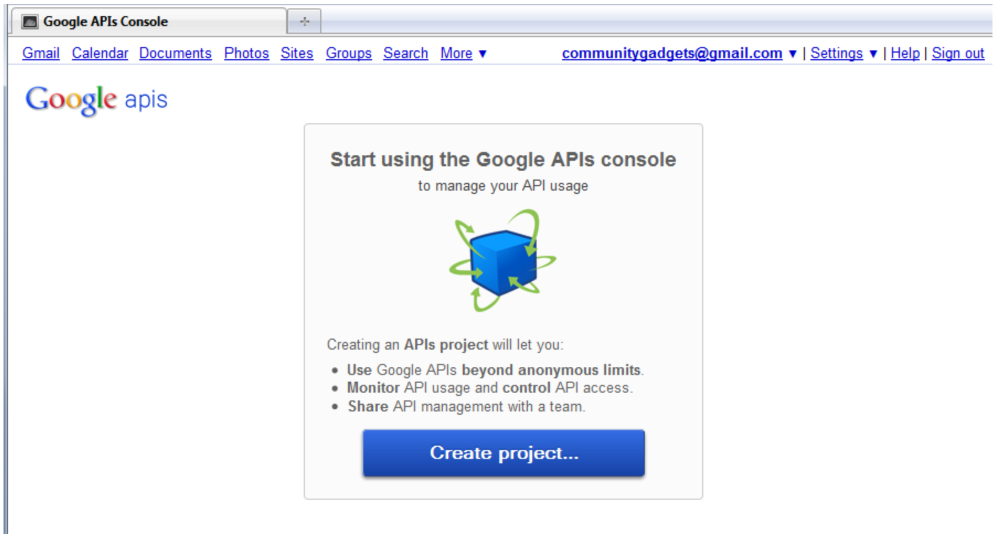
To create a Google application:

1. Open the Google APIs Console by signing in to the following URL:

   ```
   https://code.google.com/apis/console
   ```

2. Click **Create project....** as shown in Figure 80–8.

Figure 80–8 Create Project



Google APIs Console
Gmail Calendar Documents Photos Sites Groups Search More ▾
communitygadgets@gmail.com ▾ | Settings ▾ | Help | Sign out
Google apis

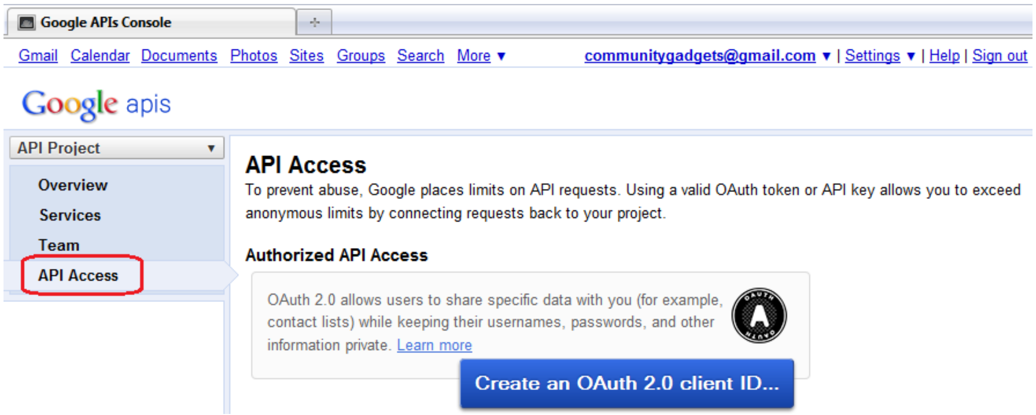
Start using the Google APIs console
to manage your API usage

Creating an **APIs project** will let you:

- **Use** Google APIs **beyond anonymous limits**.
- **Monitor** API usage and **control** API access.
- **Share** API management with a team.

3. On the All Services page, on the left pane, click **API Access**, as shown in Figure 80–9.

Figure 80–9 API Access


Google APIs Console
Gmail Calendar Documents Photos Sites Groups Search More ▾
communitygadgets@gmail.com ▾ | Settings ▾ | Help | Sign out
Google apis
API Project ▾
Overview
Services
Team
API Access
Create an OAuth 2.0 client ID...

API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allows you to exceed anonymous limits by connecting requests back to your project.

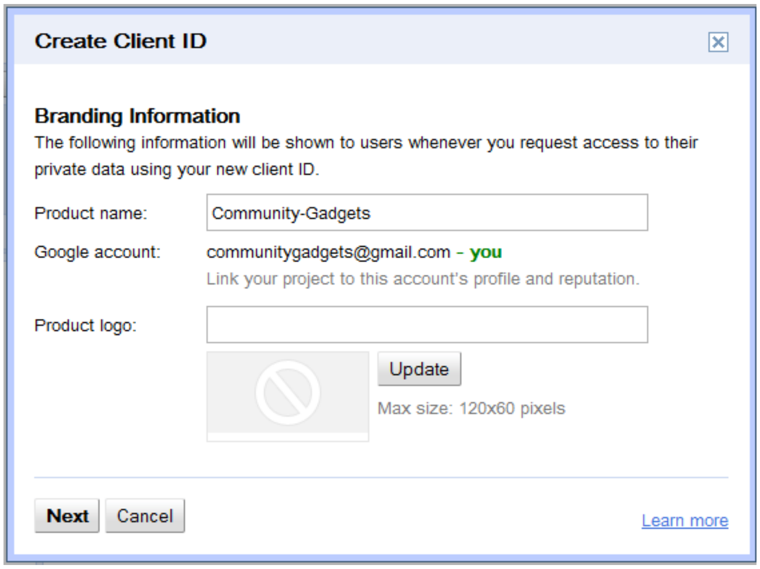
Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. Learn more

4. To create an application, on the API Access page, click **Create an OAuth 2.0 client ID** (Figure 80–9) and do the following:

   a. In the Create Client ID dialog box, in the **Product name** field, enter a meaningful name for your application. For example, `Community-Gadgets`, as shown in Figure 80–10.

Figure 80–10 Create Client ID



Branding Information

The following information will be shown to users whenever you request access to their private data using your new client ID.

Product name: Community-Gadgets

Google account: communitygadgets@gmail.com – you

Link your project to this account's profile and reputation.

Product logo:

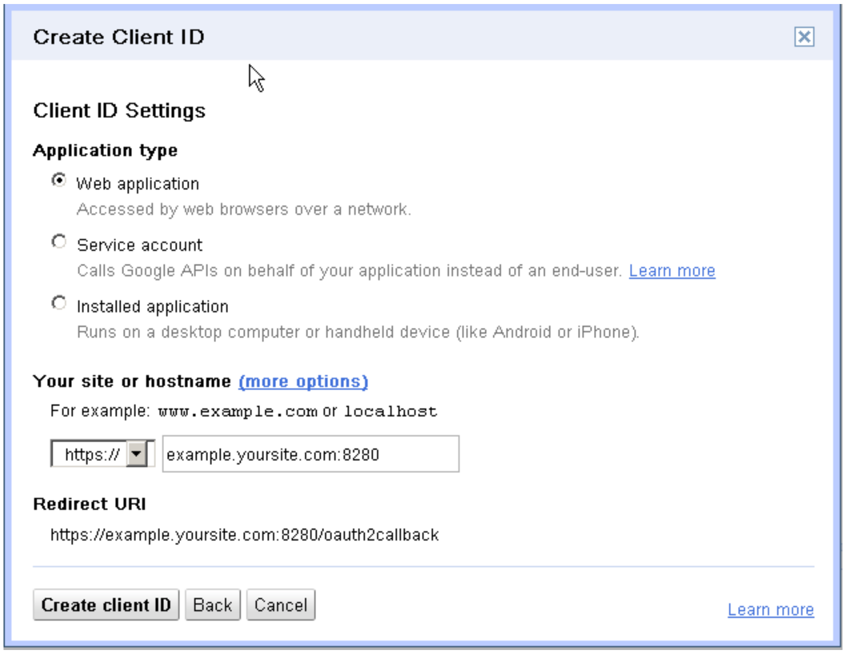
Max size: 120x60 pixels

   b. Click **Next**.

   c. On the Client ID Settings screen, for **Application type**, choose **Web application** (Figure 80–11), if it is not selected by default.

   d. Under **Your site or hostname**, enter the URL of your Community-Gadgets production server. For example, `example.yoursite.com:8280`, as shown in Figure 80–11.

Figure 80–11 Client ID Settings



Create Client ID

Client ID Settings

Application type

● Web application
Accessed by web browsers over a network.

○ Service account
Calls Google APIs on behalf of your application instead of an end-user. Learn more

○ Installed application
Runs on a desktop computer or handheld device (like Android or iPhone).

Your site or hostname (more options)

For example: www.example.com or localhost

https:// ▾ example.yoursite.com:8280

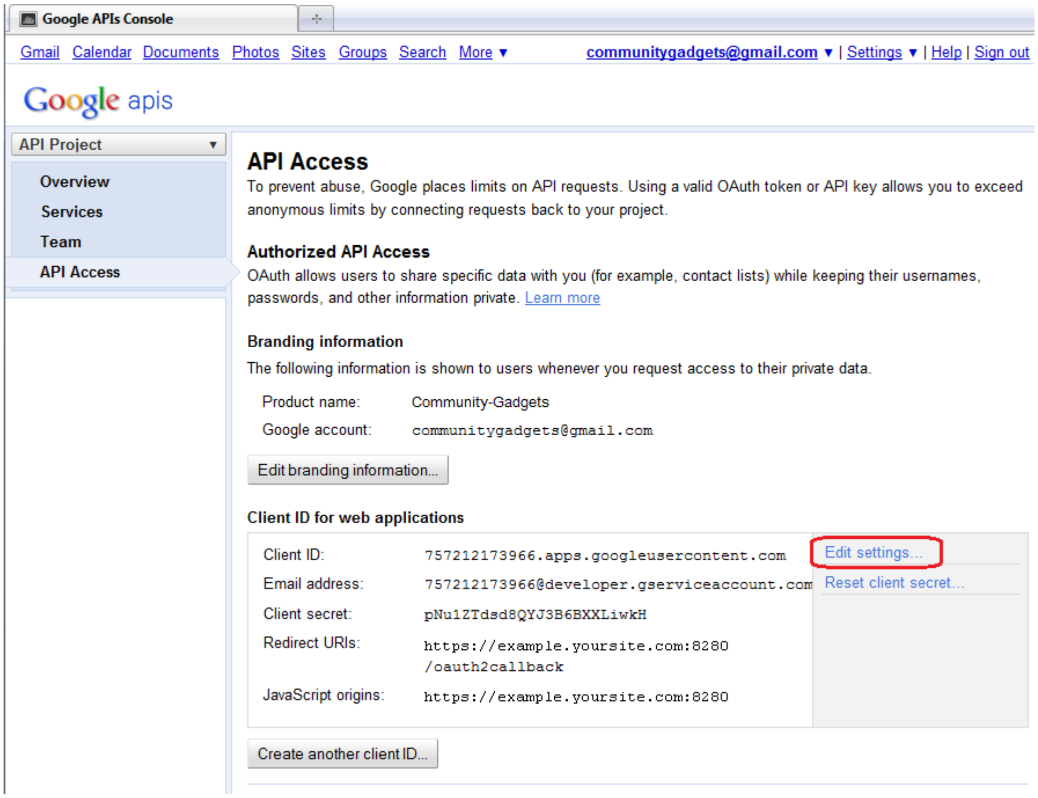
Redirect URI

https://example.yoursite.com:8280/oauth2callback

e. Click **Create client ID**.

5. To generate client ID and client secret for your application, in the **Client ID for web applications** section, click **Edit settings** (Figure 80–12) and do the following:

Figure 80–12 Edit Settings



Google APIs Console
Gmail Calendar Documents Photos Sites Groups Search More ▾
communitygadgets@gmail.com ▾ | Settings ▾ | Help | Sign out
Google apis
API Project ▾
Overview
Services
Team
API Access

API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allows you to exceed anonymous limits by connecting requests back to your project.

Authorized API Access

OAuth allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. Learn more

Branding information

The following information is shown to users whenever you request access to their private data.

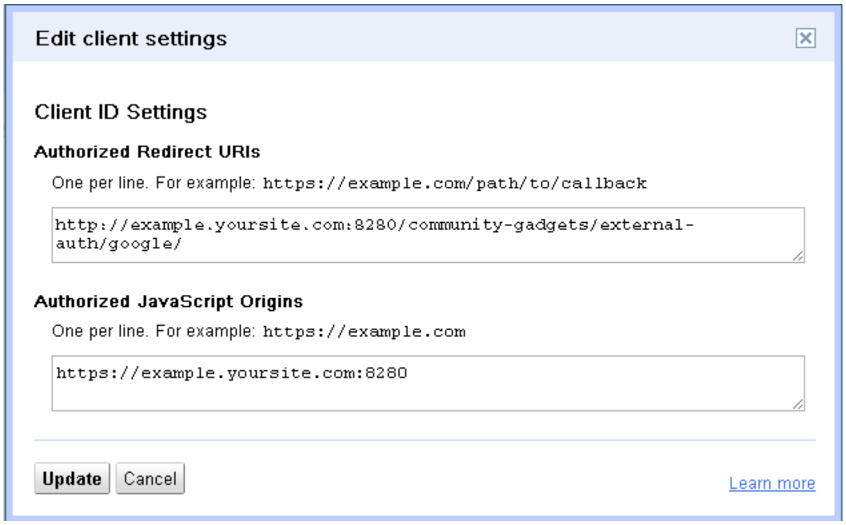
Product name: Community-Gadgets
Google account: communitygadgets@gmail.com

Client ID for web applications

Client ID: 757212173966.apps.googleusercontent.com
Email address: 757212173966@developer.gserviceaccount.com
Client secret: pNu1ZTdsd8QYJ3B6BXXLiwkH
Redirect URIs: https://example.yoursite.com:8280/oauth2callback
JavaScript origins: https://example.yoursite.com:8280

a. In the Edit client settings dialog box, in the **Authorized JavaScript Origins** field, enter `http://<cg_production_hostname>:<cg_production_port>/<context-root>/external-auth/google/`. For example, `http://example.yoursite.com:8280/community-gadgets/external-auth/google/`, as shown in Figure 80–13.

Figure 80–13 Client ID Settings



Edit client settings

Client ID Settings

Authorized Redirect URIs

One per line. For example: `https://example.com/path/to/callback`

`http://example.yoursite.com:8280/community-gadgets/external-auth/google/`

Authorized JavaScript Origins

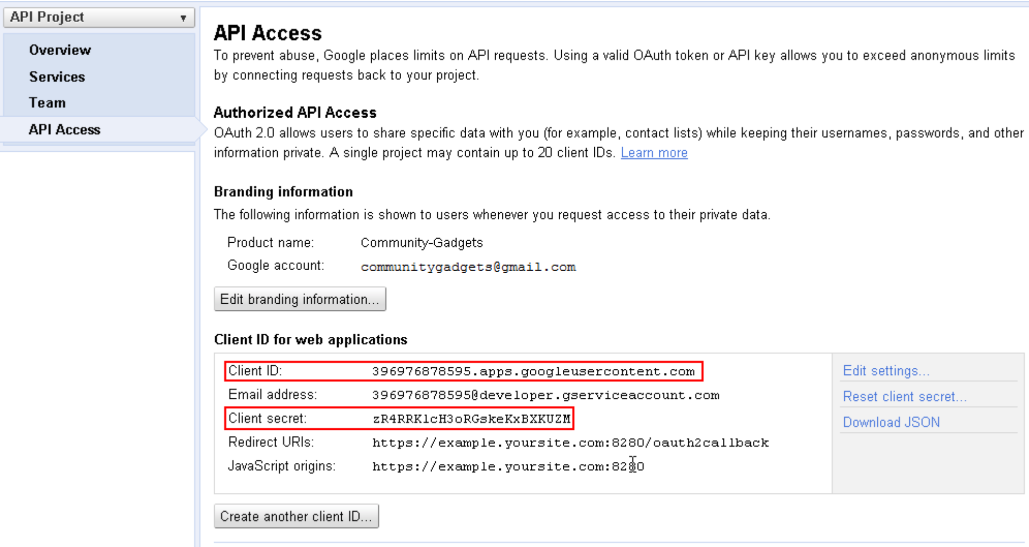
One per line. For example: `https://example.com`

`https://example.yoursite.com:8280`

b. Click **Update**.

You will receive `Client ID` and `Client secret` similar to those displayed in Figure 80–14 in your email.

Figure 80–14 Client ID for Web Application



API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allows you to exceed anonymous limits by connecting requests back to your project.

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. A single project may contain up to 20 client IDs. Learn more

Branding information

The following information is shown to users whenever you request access to their private data.

Product name: Community-Gadgets
Google account: communitygadgets@gmail.com

Client ID for web applications

Client ID: 396976878595.apps.googleusercontent.com
Email address: 396976878595@developer.gserviceaccount.com
Client secret: zR4RRKlcH3oRGskeKxBXKUZM
Redirect URIs: https://example.yoursite.com:8280/oauth2callback
JavaScript origins: https://example.yoursite.com:8280

These parameters are added to the `setup_auth.properties` file, as described in Section 80.4.2, "Configure Google Application Authentication Settings on Community-Gadgets."

80.4.2 Configure Google Application Authentication Settings on Community-Gadgets

To configure Google application authentication properties on the Community-Gadgets application server:

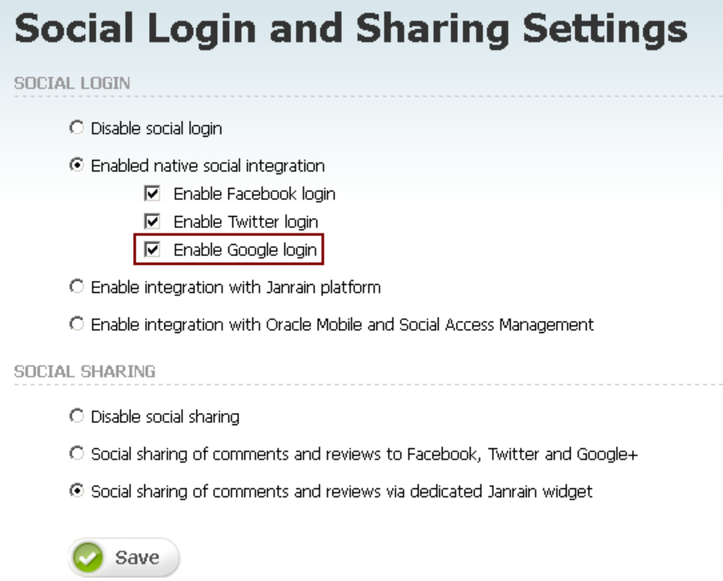
1. In the application server home on the production system, navigate to the standalone_node folders. Default location are: `<cg_install_dir>/deploy/management/management_node1` and `<cg_install_dir>/deploy/production/production_node1`.
2. Open the `setup_auth.properties` file, then scroll down to the Google Application Settings section.
3. For `widgets.external_auth.google.attrs.client_id` and `widgets.external_auth.google.attrs.client_secret` configuration properties, enter the client ID and client secret values you copied at the time of creating the Google application.

   The Google Application Settings section should look like this:

   ```
   ## Google+ Application Settings
   #######################################
   #
   # Google+ application id
   #
   widgets.external_auth.google.attrs.client_id=
   757212173966.apps.googleusercontent.com
   #
   # Google+ application secret
   #
   widgets.external_auth.google.attrs.client_secret= pNu1ZTdsd8QYJ3B6BXXLiwkH
   ```

4. After saving the configuration file, restart the application server.
5. Open the Community interface or the Gadgets interface on the management side as a general administrator: `http://`*`host:port`*`/cs/login`.
6. Choose the desired site.
7. From the **Login Bar** menu, choose **Configure**.
8. On the Social Login and Sharing Settings page, in the **Social Login** section, under **Enabled native social integration**, select the **Enable Google login** check box (Figure 80–15), then click **Save**.

Figure 80–15 Enable Google Login



Social Login and Sharing Settings

SOCIAL LOGIN

○ Disable social login

● Enabled native social integration

☑ Enable Facebook login

☑ Enable Twitter login

☑ Enable Google login

○ Enable integration with Janrain platform

○ Enable integration with Oracle Mobile and Social Access Management

SOCIAL SHARING

○ Disable social sharing

○ Social sharing of comments and reviews to Facebook, Twitter and Google+

● Social sharing of comments and reviews via dedicated Janrain widget

9. Restart the management and production environments.
10. Access the web site on which the widget is deployed.

    The Google login button is displayed in the Community-Gadgets widget's Login screen on the web site on which the widget is deployed.

> **Note:** If your Community-Gadgets web application is running on Oracle WebLogic Server or Tomcat application server, then the procedures described in Section 80.4.1, "Create a Google Application for Community-Gadgets" and Section 80.4.2, "Configure Google Application Authentication Settings on Community-Gadgets" are all you require to integrate Community-Gadgets with Google. However, on a WebSphere Application Server instance, you must also perform the export/import procedures described in Section 80.7, "Enabling Social Networking Services on WebSphere Application Server" to enable communication between Community-Gadgets and this social networking service.

80.5 Integrating with Janrain

The Community-Gadgets web application integrates with Janrain, and Janrain integrates with a number of third-party services. From this list of services, visitors can choose a service that they already use to log in to your site.

To configure Janrain support:

1. Create an account on Janrain's web site.
2. Create a Janrain Application for Community-Gadgets.
3. Configure Janrain Application Authentication Settings on Community-Gadgets.

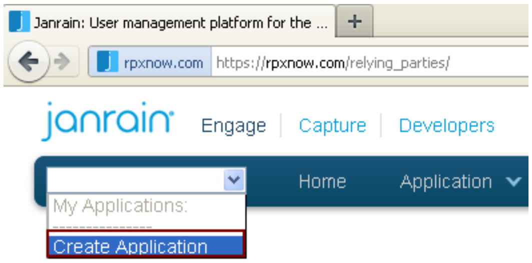
80.5.1 Create a Janrain Application for Community-Gadgets

To perform the steps described in this section, you must have a Janrain account. Register your company on the Janrain web site by choosing the Enterprise subscription plan.

To create a Janrain application:

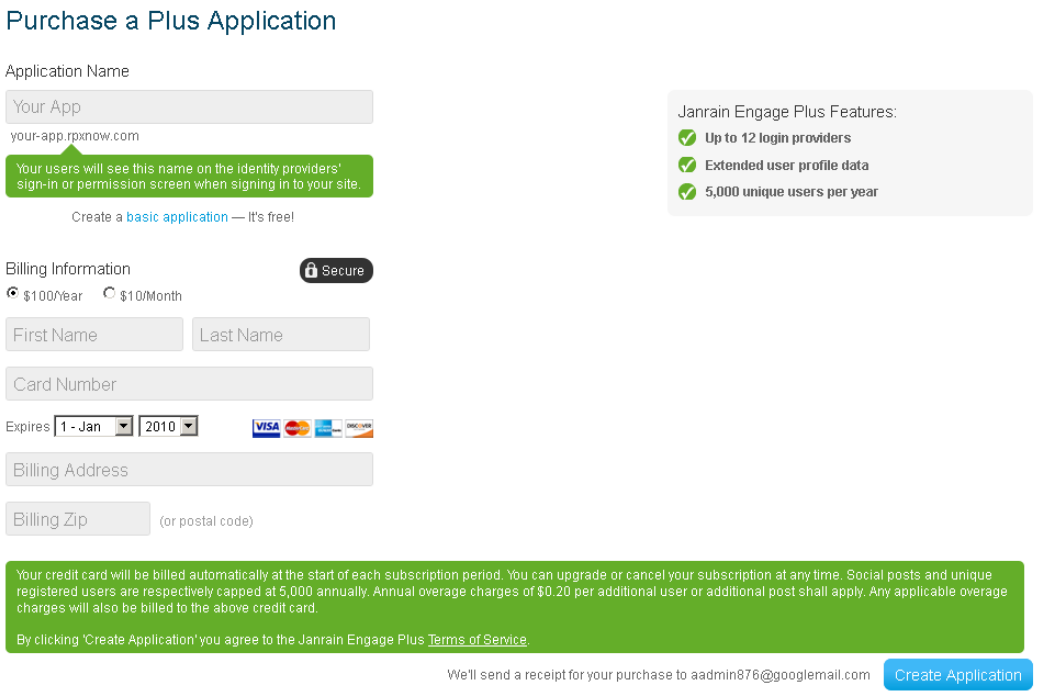
1. Go to `rpxnow.com`, then from the drop-down list on the toolbar, as shown in Figure 80–16, select **Create Application**.

Figure 80–16 Create Janrain Application



2. Fill in the required fields, then click **Create Application**.

Figure 80–17 Purchase a Plus Application



Purchase a Plus Application

Application Name

Your App

your-app.rpxnow.com

Your users will see this name on the identity providers' sign-in or permission screen when signing in to your site.

Create a basic application — It's free!

Janrain Engage Plus Features:
- ✓ Up to 12 login providers
- ✓ Extended user profile data
- ✓ 5,000 unique users per year

Billing Information

Secure

● $100/Year ○ $10/Month

First Name

Last Name

Card Number

Expires 1 - Jan 2010

Billing Address

Billing Zip (or postal code)

Your credit card will be billed automatically at the start of each subscription period. You can upgrade or cancel your subscription at any time. Social posts and unique registered users are respectively capped at 5,000 annually. Annual overage charges of $0.20 per additional user or additional post shall apply. Any applicable overage charges will also be billed to the above credit card.

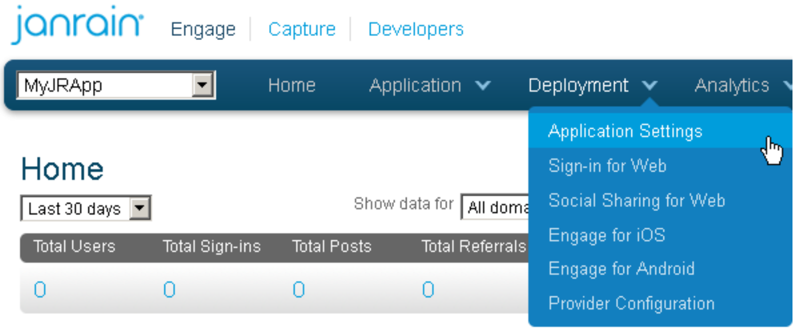
By clicking 'Create Application' you agree to the Janrain Engage Plus Terms of Service.

We'll send a receipt for your purchase to aadmin876@googlemail.com

Create Application

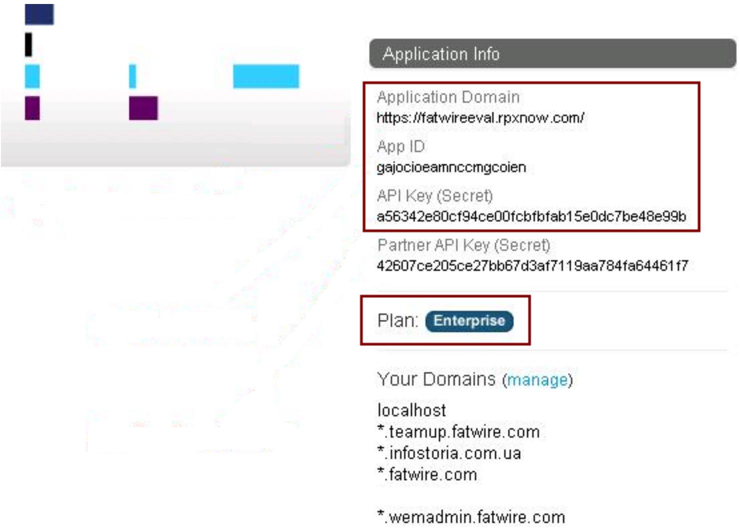
3. From the Deployment menu, choose **Application Settings**.

Figure 80–18 Application Settings



4. In the Application Info section, unique values for your newly created application are displayed under Application Domain, App ID, and API Key (Secret). Copy these values as you will need them when you configure Janrain support on the Community-Gadgets production server.

Figure 80–19 Required Application Parameters



Application Info

Application Domain
https://fatwireeval.rpxnow.com/

App ID
gajocioeamnccmgcoien

API Key (Secret)
a56342e80cf94ce00fcbfbfab15e0dc7be48e99b

Partner API Key (Secret)
42607ce205ce27bb67d3af7119aa784fa64461f7

Plan: Enterprise

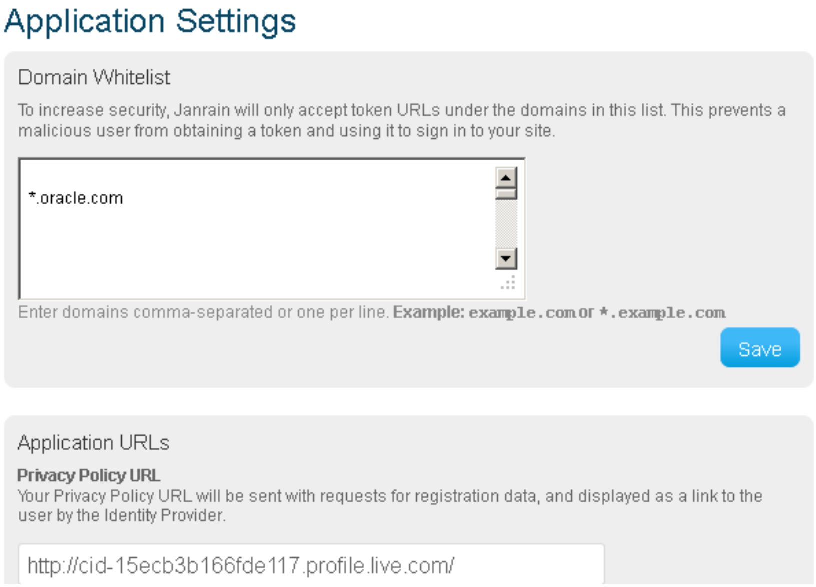
Your Domains (manage)

localhost
*.teamup.fatwire.com
*.infostoria.com.ua
*.fatwire.com

*.wemadmin.fatwire.com

5. In the Domain Whitelist box, specify the web site domains on which the Community-Gadgets widgets will be deployed (Figure 80–20).

Figure 80–20 Domain Whitelist



Application Settings

Domain Whitelist

To increase security, Janrain will only accept token URLs under the domains in this list. This prevents a malicious user from obtaining a token and using it to sign in to your site.

*.oracle.com

Enter domains comma-separated or one per line. **Example:** `example.com` or `*.example.com`

Application URLs

Privacy Policy URL

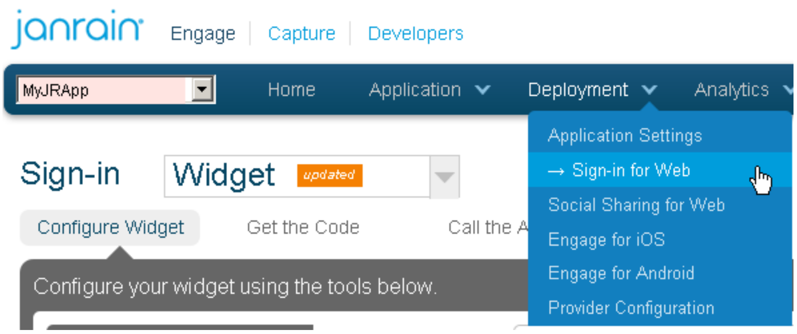
Your Privacy Policy URL will be sent with requests for registration data, and displayed as a link to the user by the Identity Provider.

http://cid-15ecb3b166fde117.profile.live.com/

6. In this step, you will enable the identity providers to be listed on the web site on which Community-Gadgets widgets will be deployed. Therefore, visitors will be able to use one of their existing online identities to add comments and ratings on your web site content.

   a. From the Deployment menu, choose **Sign-in for Web**.

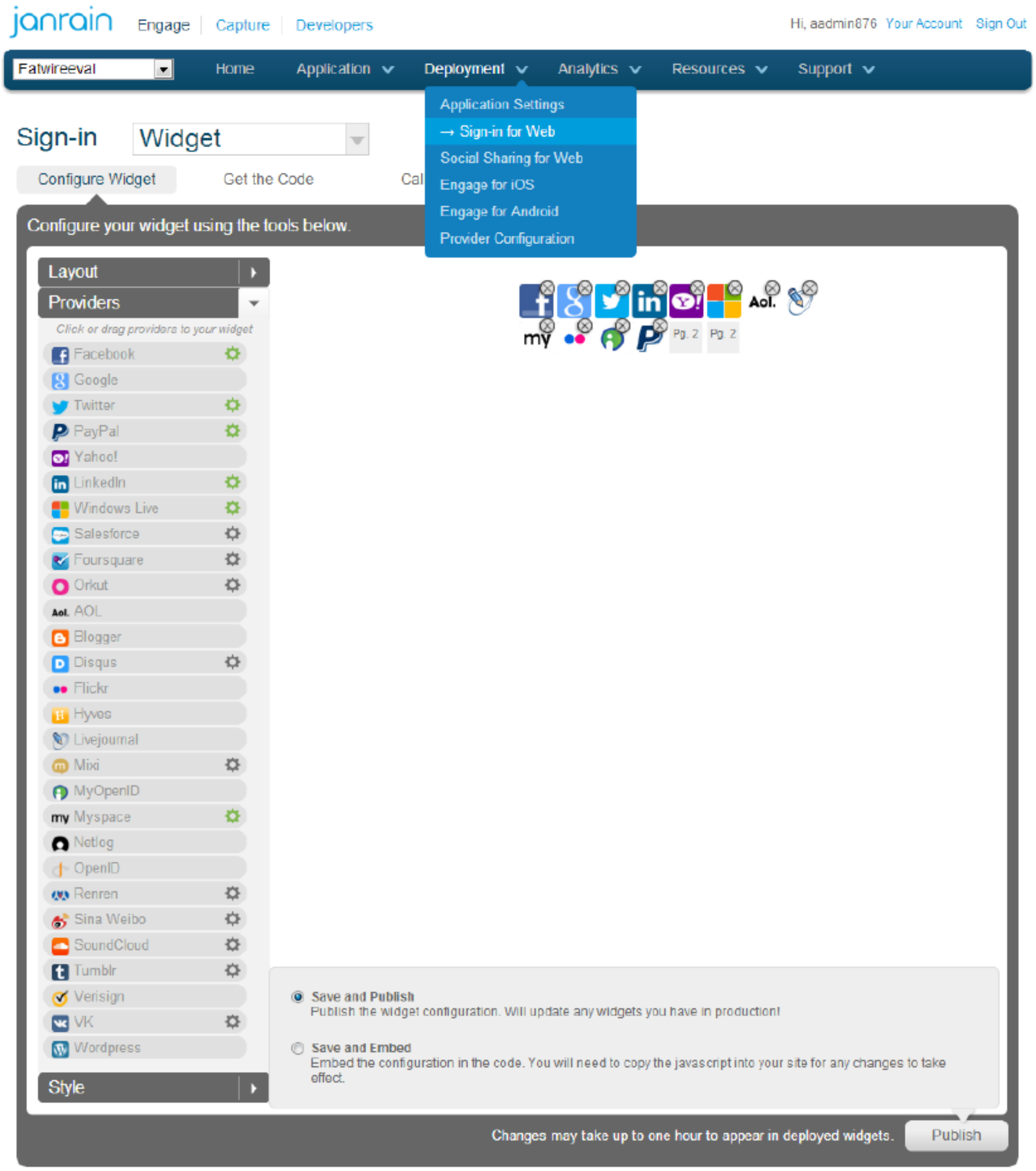
Figure 80–21 Sign-in for Web



Configure your widget using the tools below.

   b. Click the **Choose Providers** link, and drag each provider to Your Widget.

Figure 80–22 Configuration of Identity Providers



Configure your widget using the tools below.

Click or drag providers to your widget

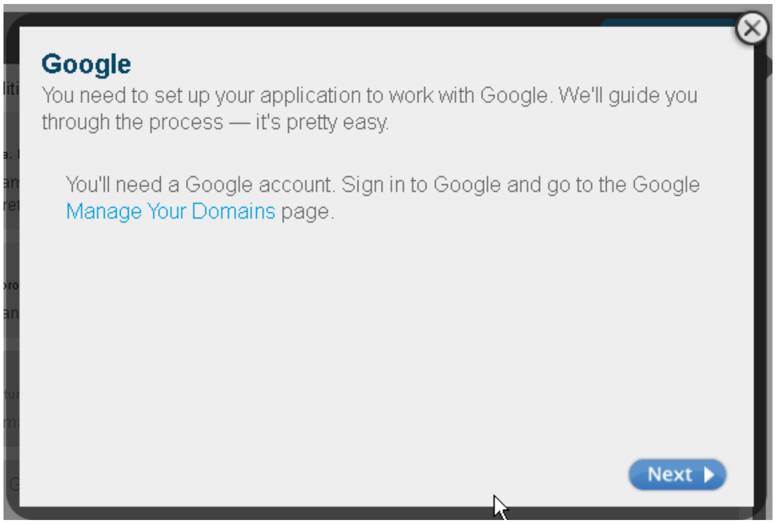
● **Save and Publish**
Publish the widget configuration. Will update any widgets you have in production!

○ **Save and Embed**
Embed the configuration in the code. You will need to copy the javascript into your site for any changes to take effect.

Changes may take up to one hour to appear in deployed widgets.

7. Configuring identity providers will allow site visitors to choose a provider for which they already have login credentials. When an identity provider is enabled, its configuration wizard is displayed. For example, Figure 80–23 shows the configuration wizard for Google. Complete the configuration steps for each enabled provider.

Figure 80–23 Google Configuration via Janrain



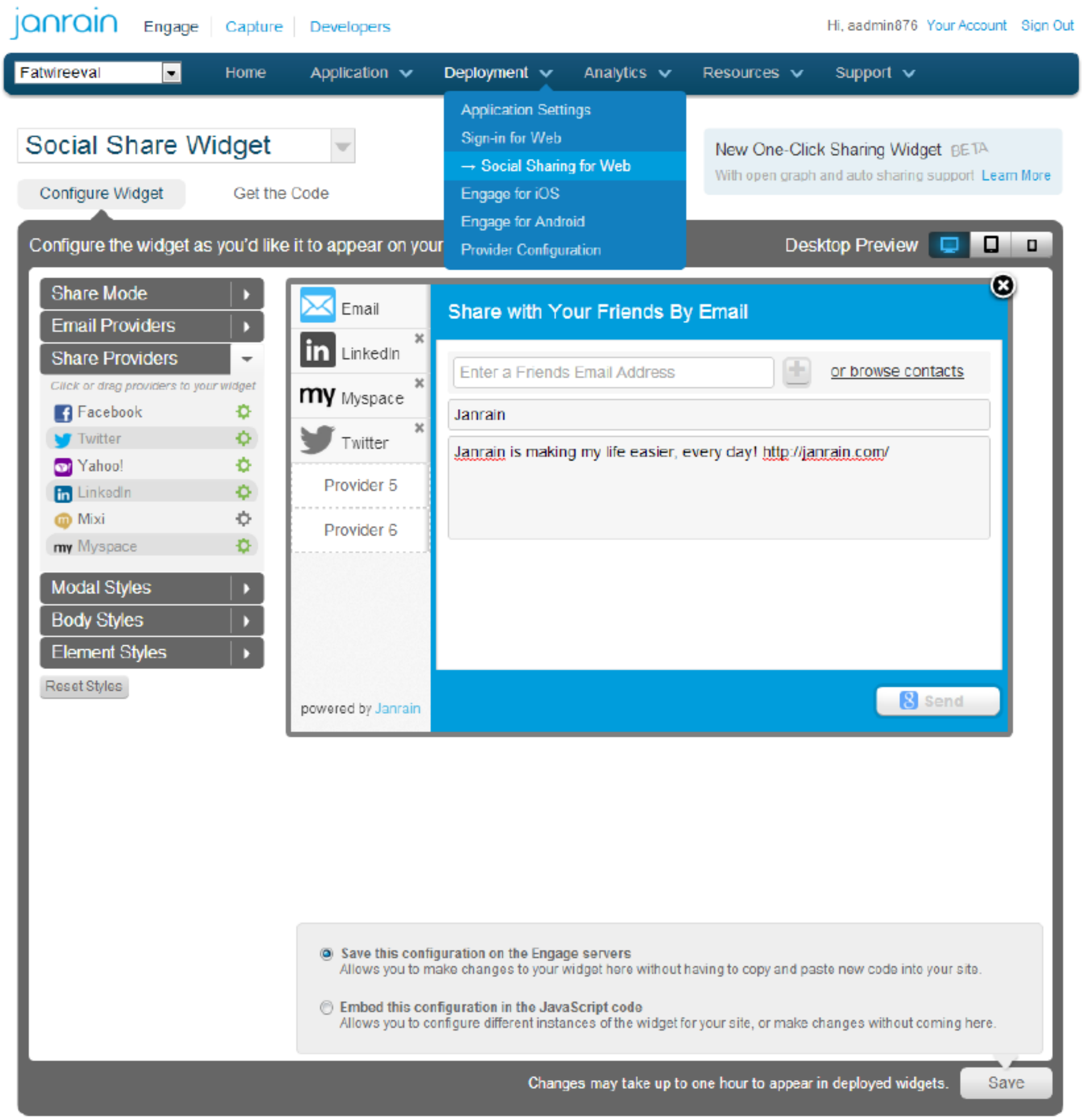
Google

You need to set up your application to work with Google. We'll guide you through the process — it's pretty easy.

You'll need a Google account. Sign in to Google and go to the Google Manage Your Domains page.

8. Configure social sharing with Janrain. This step will enable your web site visitors to share selected content via their profiles on social networking sites. From the Deployment menu, choose **Social Sharing for Web**.

Figure 80–24 Configuration of Social Networking Services Providers



Configure the widget as you'd like it to appear on your

Janrain

Janrain is making my life easier, every day! http://janrain.com/

● Save this configuration on the Engage servers
Allows you to make changes to your widget here without having to copy and paste new code into your site.

○ Embed this configuration in the JavaScript code
Allows you to configure different instances of the widget for your site, or make changes without coming here.

Changes may take up to one hour to appear in deployed widgets.

80.5.2 Configure Janrain Application Authentication Settings on Community-Gadgets

After configuring the Janrain application, configure Janrain authentication settings on the production Community-Gadgets web application.

1. In the application server on the production system, navigate to the standalone_node folders. Default management and production locations are:

```
<cg_install_dir>/deploy/management/management_node1
<cg_install_dir>/deploy/production/production_node1
```

2. Open the `setup_auth.properties` file, then scroll down to the Janrain Application Settings section.

3. Enable Janrain support by setting the `widgets.external_auth.janrain.attrs.enabled` property to `true`.

4. For `widgets.external_auth.janrain.attrs.app_name`, `widgets.external_auth.janrain.attrs.app_id`, and `widgets.external_auth.janrain.attrs.app_`

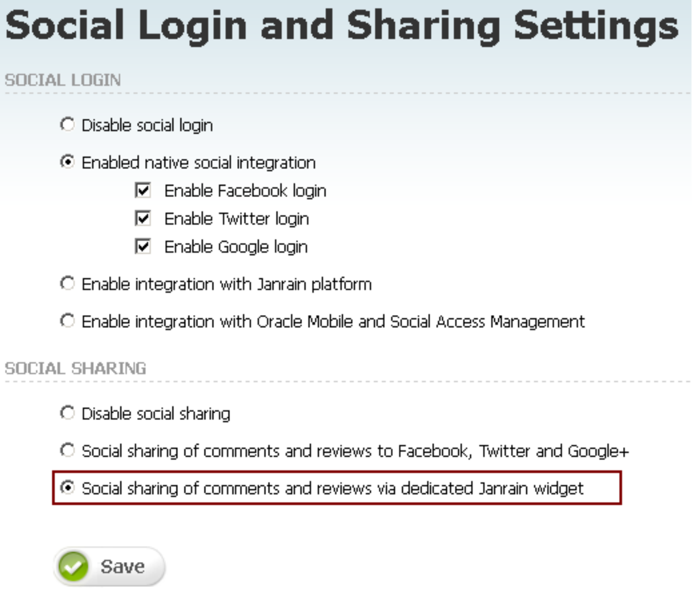
secret configuration properties, enter the application name, application domain, application ID, and API key you copied earlier.

The Janrain Application Settings section should look like this:

```
## Janrain Application Settings
########################################
#
# Enabling Janrain support
# Default is "false"
#
widgets.external_auth.janrain.attrs.enabled=true

#
# Jainrain application name
#
widgets.external_auth.janrain.attrs.app_name=fw

## Janrain application domain
# widgets.external_auth.janrain.attrs.app_domain=https://fw.rpxnow.com/

#
# Jainrain application id
#
widgets.external_auth.janrain.attrs.app_id=gajo1d134dfk3tgcoien

#
# Jainrain application secret
#
widgets.external_auth.janrain.attrs.app_secret=5asdf234jasdfk531
```

5. After saving the configuration file, restart the application server.
6. Open the Community interface or Gadgets interface, then go to **Login Bar** then **Configure**.
7. On the Social Login and Sharing Settings page, in the **Social Sharing** section, select the **Social sharing of comments and reviews via dedicated Janrain widget** option (Figure 80–25) if you configured the sharing option on the Janrain site, then click **Save**.

Figure 80–25 Janrain Support Enabled



Social Login and Sharing Settings

SOCIAL LOGIN

○ Disable social login

● Enabled native social integration

☑ Enable Facebook login

☑ Enable Twitter login

☑ Enable Google login

○ Enable integration with Janrain platform

○ Enable integration with Oracle Mobile and Social Access Management

SOCIAL SHARING

○ Disable social sharing

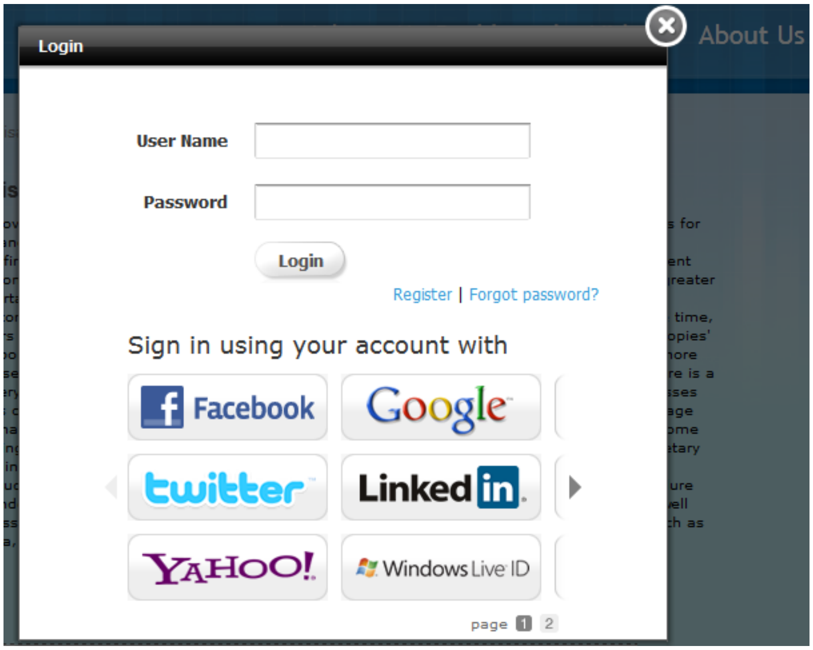
○ Social sharing of comments and reviews to Facebook, Twitter and Google+

● Social sharing of comments and reviews via dedicated Janrain widget

8. Restart the management and production environments.

9. On the web site on which the Community-Gadgets widgets are deployed, verify that the configured identity providers are displayed in the Community-Gadgets widget's Login screen, as shown in Figure 80–26.

Figure 80–26 Login Screen with Identity Provider Links



Note: If your Community-Gadgets web application is running on Oracle WebLogic Server or Tomcat application server, then the procedures described in Section 80.5.1, "Create a Janrain Application for Community-Gadgets" and Section 80.5.2, "Configure Janrain Application Authentication Settings on Community-Gadgets" are all you require to integrate Community-Gadgets with Janrain. However, on a WebSphere Application Server instance, you must also perform the export/import procedures described in Section 80.7, "Enabling Social Networking Services on WebSphere Application Server" to enable communication between Community-Gadgets and this social networking service.

80.6 Integrating with Oracle Internet Access Service

This section describes how to configure WebLogic Server to support Internet identity providers, then how to enable Mobile and Social service and define Internet identity providers and create for them a new Mobile and Social application profile, and finally, how to enable integration of OMSAS with Community-Gadgets for secure transactions through the configured identity providers.

This section includes the following topics:

- Section 80.6.1, "Configuring WebLogic Server to Support Mobile and Social Identity Providers"
- Section 80.6.2, "Enabling Mobile and Social Service on OAM"
- Section 80.6.3, "Defining Internet Identity Providers for OMSAS"
- Section 80.6.4, "Creating a New Mobile and Social Application Profile"
- Section 80.6.5, "Enabling the Integration of OMSAS with Community-Gadgets"

80.6.1 Configuring WebLogic Server to Support Mobile and Social Identity Providers

Before you can enable Mobile and Social service and configure identity providers on OAM, you must configure the OAM's WebLogic managed server instance for the required identity providers.

For information about configuring WebLogic Server for Facebook compatibility, see the section *Troubleshooting Internet Identity Providers* at `http://docs.oracle.com/cd/E27559_01/admin.1112/e27239/oicconfiginetidentitysrvcs.htm#AIAAG8292`

80.6.2 Enabling Mobile and Social Service on OAM

This section describes the procedure to enable Mobile and Social service on OAM so that in the next steps you can configure the required identity providers (such as Facebook, Twitter, and so on) with this service, as well as Community-Gadgets and the identity providers with OAM through an application profile.

To enable the Mobile and Social service on OAM:

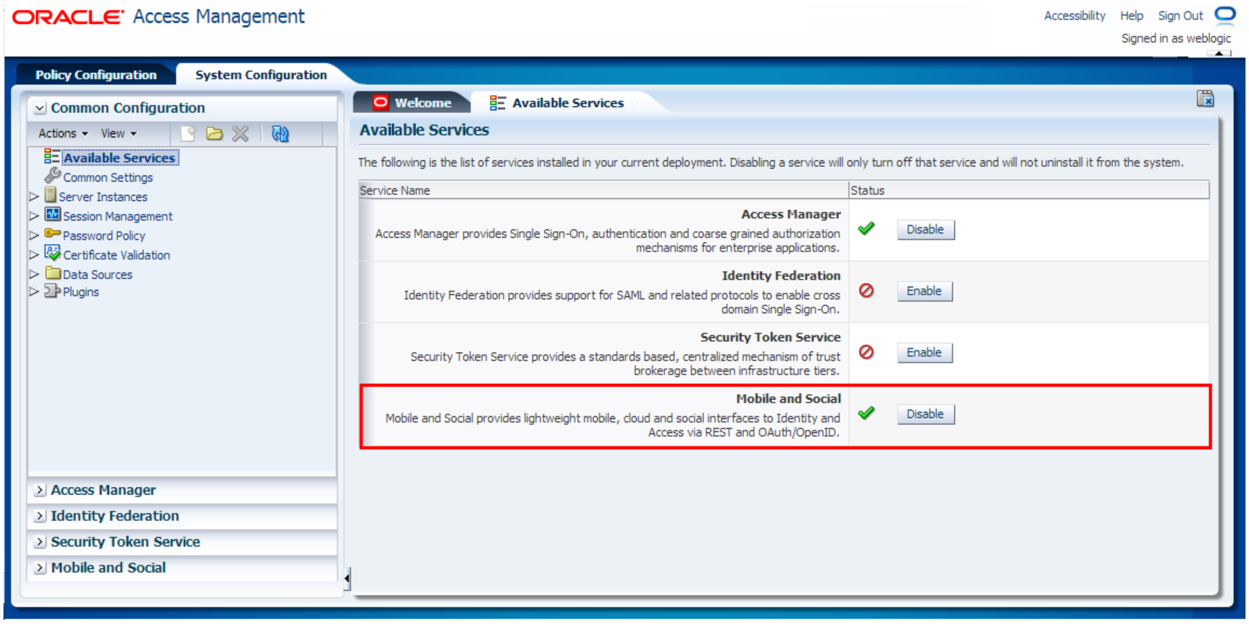
1. Log in to the OAM Console application:

   ```
   http://<weblogic_host>:<weblogic_admin_port>/oamconsole
   ```

2. On the **System Configuration** tab, expand the **Available Services** node.
3. On the **Available Services** tab, click **Enable** for the **Mobile and Social** service.

In Figure 80–27, the Mobile and Social service has already been enabled, and therefore, the **Disable** option is displayed.

Figure 80–27 Mobile and Social Service Enabled



Available Services

The following is the list of services installed in your current deployment. Disabling a service will only turn off that service and will not uninstall it from the system.

Service Name	Status	
Access Manager Access Manager provides Single Sign-On, authentication and coarse grained authorization mechanisms for enterprise applications.	✓	Disable
Identity Federation Identity Federation provides support for SAML and related protocols to enable cross domain Single Sign-On.	⊘	Enable
Security Token Service Security Token Service provides a standards based, centralized mechanism of trust brokerage between infrastructure tiers.	⊘	Enable
Mobile and Social Mobile and Social provides lightweight mobile, cloud and social interfaces to Identity and Access via REST and OAuth/OpenID.	✓	Disable

80.6.3 Defining Internet Identity Providers for OMSAS

After enabling the Mobile and Social service on OAM, define the identity providers (such as Facebook, Twitter, LinkedIn, and so on) to be supported through this service. For information, see the section *Defining Internet Identity Providers* at `http://docs.oracle.com/cd/E27559_01/admin.1112/e27239/oicconfiginetidentitysrvcs.htm#autoId5`

80.6.4 Creating a New Mobile and Social Application Profile

The next step after defining Internet identity providers is to create a new Mobile and Social application profile for the production instance of Community-Gadgets. While creating this application profile, you will also enable the required identity providers to be supported by it.

You can create a new Mobile and Social Application profile in the following ways:

- Section 80.6.4.1, "Create a Mobile and Social Application Profile using the OAM Console Application"
- Section 80.6.4.2, "Create an Application Profile using WebLogic Scripting Tool (WLST) Script"

80.6.4.1 Create a Mobile and Social Application Profile using the OAM Console Application

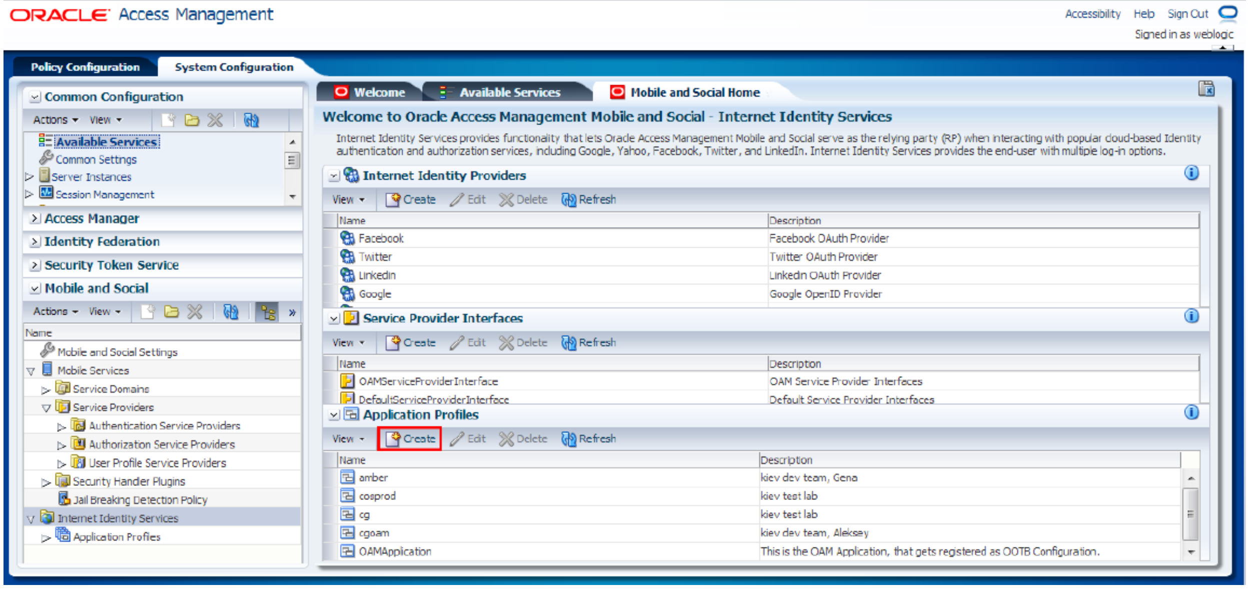
To create the new application profile using the OAM Console:

1. Log in to the OAM Console application, then click the **System Configuration** tab (Figure 80–28).
2. On the left pane, click **Mobile and Social** (Figure 80–28).

3. On the left pane, expand **Internet Identity Services**, then click **Application Profiles** (Figure 80–28).

4. On the **Mobile and Social Home** tab, under the **Application Profiles** section, click **Create** (Figure 80–28).

Figure 80–28 OAM - System Configuration - Mobile and Social



Welcome to Oracle Access Management Mobile and Social - Internet Identity Services

Internet Identity Services provides functionality that lets Oracle Access Management Mobile and Social serve as the relying party (RP) when interacting with popular cloud-based Identity authentication and authorization services, including Google, Yahoo, Facebook, Twitter, and LinkedIn. Internet Identity Services provides the end-user with multiple log-in options.

Internet Identity Providers

Name	Description
Facebook	Facebook OAuth Provider
Twitter	Twitter OAuth Provider
LinkedIn	LinkedIn OAuth Provider
Google	Google OpenID Provider

Service Provider Interfaces

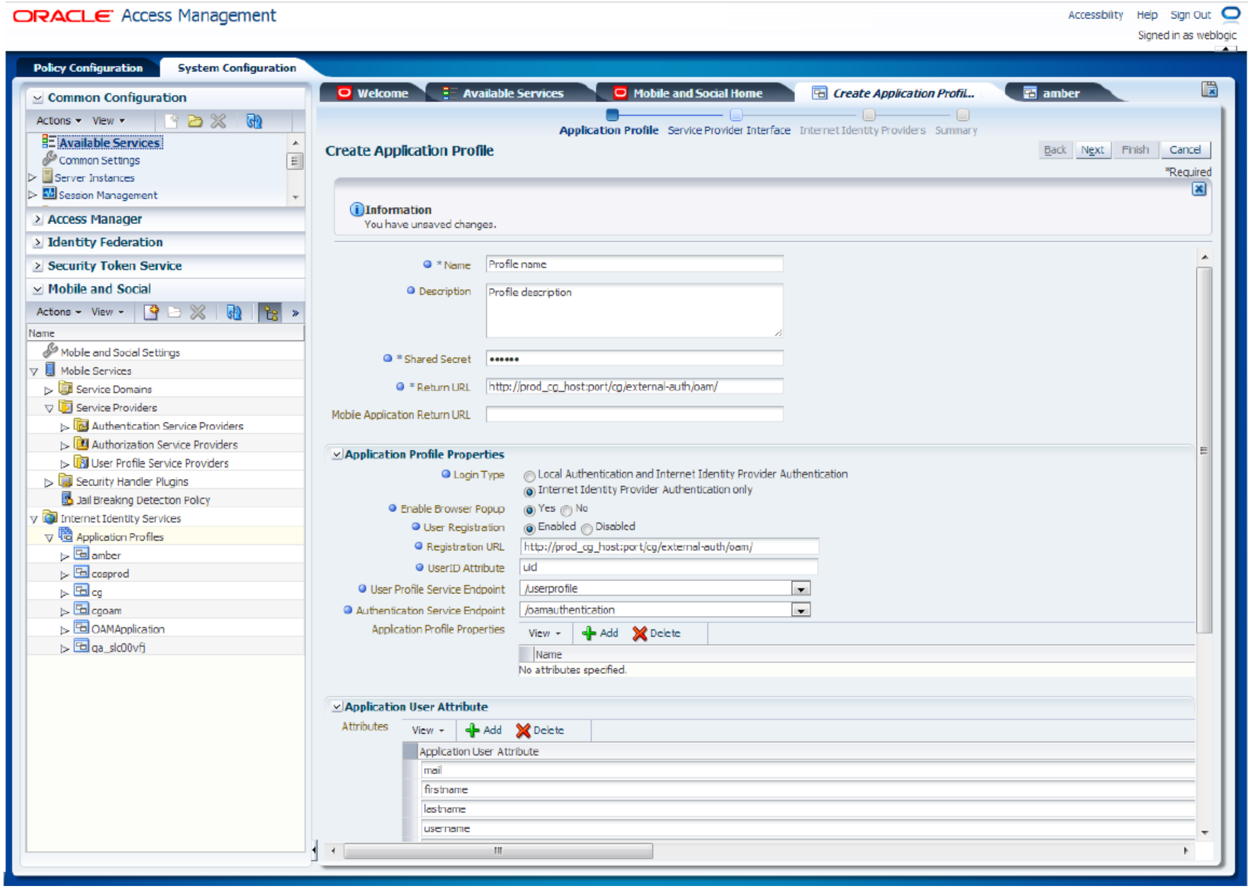
Name	Description
OAMServiceProviderInterface	OAM Service Provider Interfaces
DefaultServiceProviderInterface	Default Service Provider Interfaces

Application Profiles

Name	Description
amber	kiev dev team, Gena
cosprod	kiev test lab
cg	kiev test lab
cgoam	kiev dev team, Aleksey
OAMApplication	This is the OAM Application, that gets registered as OOTB Configuration.

5. In the **Create Application Profile** form, on the **Application Profile** page, specify the required information, as shown in Figure 80–29 and as described in Table 80–1, then click **Next**.

Figure 80–29 Create Application Profile Form



Create Application Profile

*Required

Information
You have unsaved changes.

* Name: Profile name

Description: Profile description

* Shared Secret: ••••••

* Return URL: http://prod_cg_host:port/cg/external-auth/oam/

Mobile Application Return URL:

Application Profile Properties

Login Type: ○ Local Authentication and Internet Identity Provider Authentication
● Internet Identity Provider Authentication only

Enable Browser Popup: ● Yes ○ No

User Registration: ● Enabled ○ Disabled

Registration URL: http://prod_cg_host:port/cg/external-auth/oam/

UserID Attribute: uid

User Profile Service Endpoint: /userprofile

Authentication Service Endpoint: /oamauthentication

Application Profile Properties: View Add Delete

Name
No attributes specified.

Application User Attribute

Attributes: View Add Delete

Application User Attribute
mail
firstname
lastname
username

Table 80–1 Application Profile Form Fields

Field Name	Description
Name	Name of the new application profile.
Description	Description of the new application profile.
Shared Secret	Password for the application profile.
Return URL	URL that Mobile and Social should use to return authentication responses to web applications. URL format: `<production_cg_protocol>://<production_cg_host>:<production_cg_port>/<production_cg_context>/external-auth/oam` where: - `<production_cg_protocol>`: protocol used for Community-Gadgets production system. Can be `http` or `https` - `<production_cg_host>`: host server of the production system - `<production_cg_port>`: port of the production system - `<production_cg_context>`: context root where application is deployed Example: `http://prodcg.example.com:8080/cg/external-auth/oam`

Table 80–1 (Cont.) Application Profile Form Fields

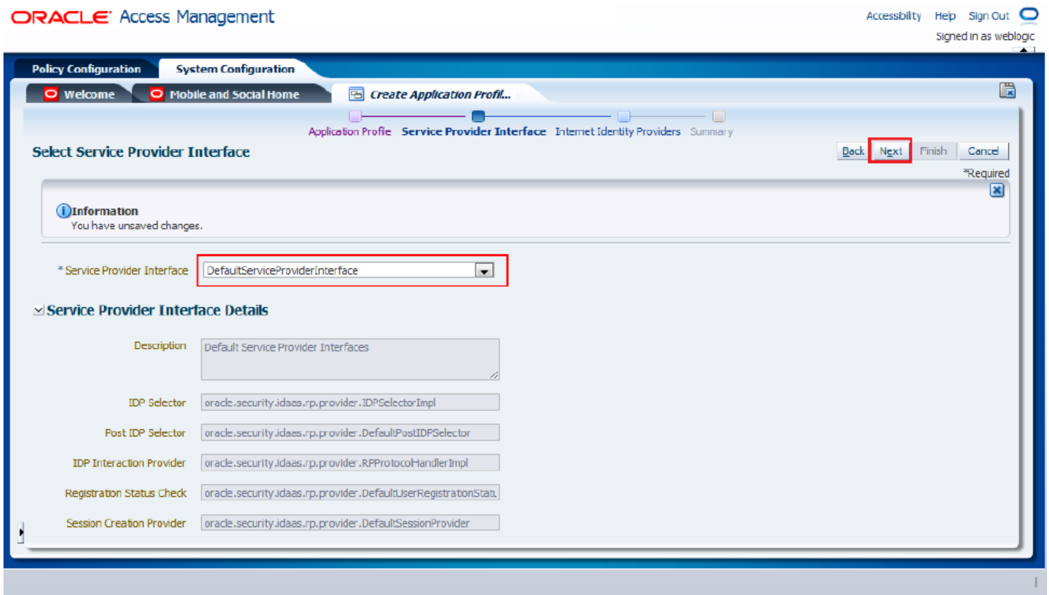
Field Name	Description
Mobile Application Return URL	URL that Mobile and Social should use to return authentication responses to mobile applications.
Application Profile Properties	
Login Type	Local authentication is not supported. Therefore, select the **Internet Identity Provider Authentication Only** option.
Enable Browser Popup	For the login dialog box to pop up in a new window choose **Yes**.
User Registration	You must choose **Enabled** to create appropriate user account in Community-Gadgets.
Registration URL	URL that Mobile and Social should use to return authentication responses to web applications with credentials from the identity provider (For example, avatar, user name, and so on.) URL format: `<production_cg_protocol>://<production_cg_host>:<production_cg_port>/<production_cg_context>/external-auth/oam` Token description is described in previous entries.
UserID Attribute	Unique user attribute: `uid`.
User Profile Service Endpoint	It must be `/userprofile`.
Authentication Service Endpoint	It must be `/internetidentityauthentication`.
Application Profile Properties	Not required
Application User Attributes	Add user attributes such as `mail`, `firstname`, `lastname`, `username`, `image`, `user_id`, `displayname`.
Registration Service Details with Application User Attribute Mapping	Mapping of registration service attributes with user attribute display names For example:
uid	Login ID
mail	Email ID
firstname	First Name
lastname	Last Name
displayname	Display Name
name	Common Name
jpegphoto	User Image

6. The Service Provider Interface page refers to the set of rules that govern the authentication flow for the specified application profile. Mobile and Social provides the following service provider interfaces (Figure 80–30):

   - **DefaultServiceProviderInterface**: provides support for web applications that run on Java-compliant application servers.
   - **OAMServiceProviderInterface**: provides support for web applications that run on the Access Manager service.

Choose the suitable option.

Figure 80–30 Selection of Service Provider Interface



Select Service Provider Interface

Information
You have unsaved changes.

* Service Provider Interface: DefaultServiceProviderInterface

Service Provider Interface Details

Description: Default Service Provider Interfaces

IDP Selector: oracle.security.idaas.rp.provider.IDPSelectorImpl

Post IDP Selector: oracle.security.idaas.rp.provider.DefaultPostIDPSelector

IDP Interaction Provider: oracle.security.idaas.rp.provider.RPProtocolHandlerImpl

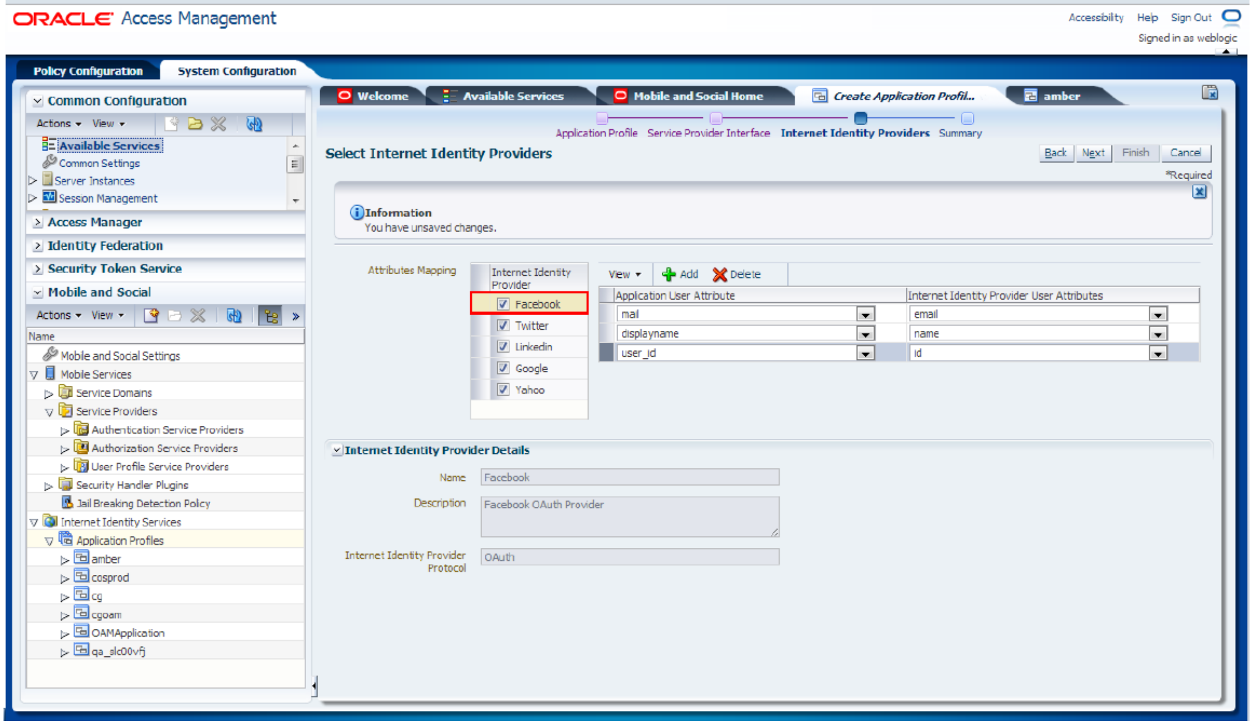
Registration Status Check: oracle.security.idaas.rp.provider.DefaultUserRegistrationStatu

Session Creation Provider: oracle.security.idaas.rp.provider.DefaultSessionProvider

7. Configure identity providers supported by this application profile, as follows, then click **Next**.

   - To configure the Facebook identity provider, select the **Facebook** check box in the **Attribute Mapping** section and specify the attributes of the application user and the Internet identity provider user, as shown in Figure 80–31.

Figure 80–31 Configuration of the Facebook Identity Provider



Select Internet Identity Providers

Information
You have unsaved changes.

Attributes Mapping

Internet Identity Provider
☑ Facebook
☑ Twitter
☑ LinkedIn
☑ Google
☑ Yahoo

Application User Attribute	Internet Identity Provider User Attributes
mail	email
displayname	name
user_id	id

Internet Identity Provider Details

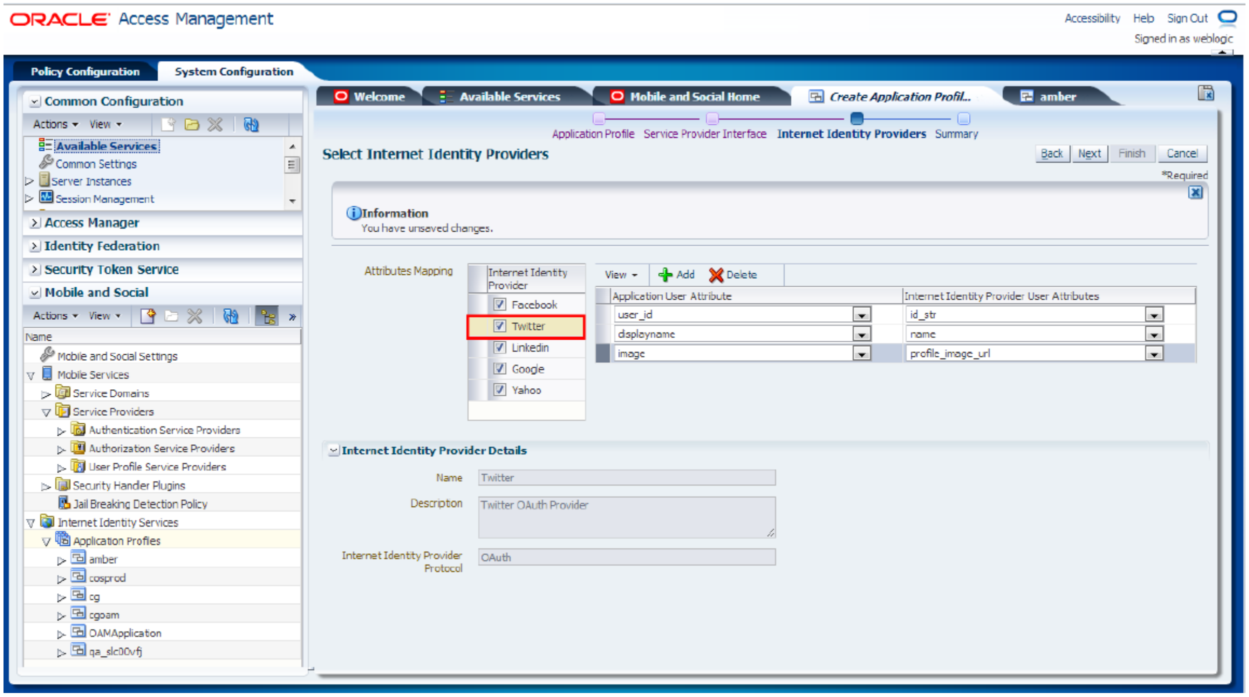
Name: Facebook

Description: Facebook OAuth Provider

Internet Identity Provider Protocol: OAuth

- To configure the Twitter identity provider, select the **Twitter** check box in the **Attribute Mapping** section and specify the attributes of the application user and the Internet identity provider user, as shown in Figure 80–32.

Figure 80–32 Configuration of the Twitter Identity Provider



Select Internet Identity Providers

*Required

Information
You have unsaved changes.

Attributes Mapping

Internet Identity Provider
☑ Facebook
☑ Twitter
☑ Linkedin
☑ Google
☑ Yahoo

Application User Attribute	Internet Identity Provider User Attributes
user_id	id_str
displayname	name
image	profile_image_url

Internet Identity Provider Details

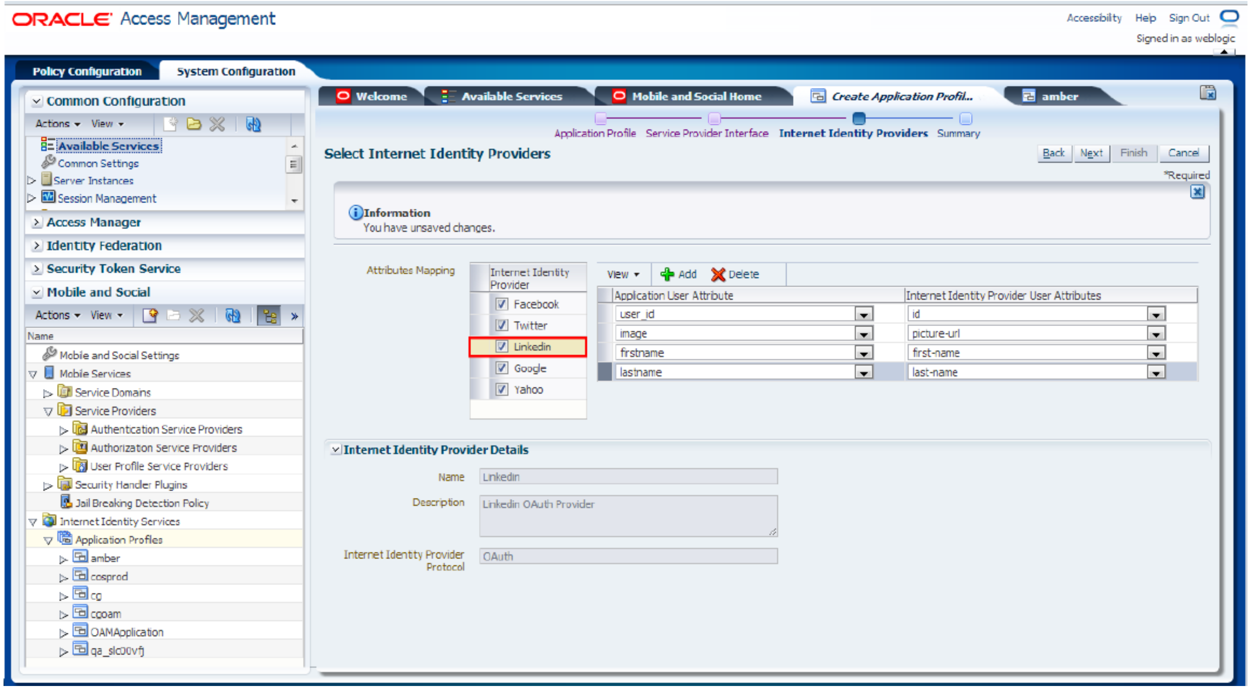
Name: Twitter

Description: Twitter OAuth Provider

Internet Identity Provider Protocol: OAuth

- To configure the **LinkedIn** identity provider, select the **LinkedIn** check box in the **Attribute Mapping** section and specify the attributes of the application user and the Internet identity provider user, as shown in Figure 80–33.

Figure 80–33 Configuration of the LinkedIn Identify Provider



Select Internet Identity Providers

Information
You have unsaved changes.

Attributes Mapping

Internet Identity Provider
☑ Facebook
☑ Twitter
☑ LinkedIn
☑ Google
☑ Yahoo

Application User Attribute	Internet Identity Provider User Attributes
user_id	id
image	picture-url
firstname	first-name
lastname	last-name

Internet Identity Provider Details

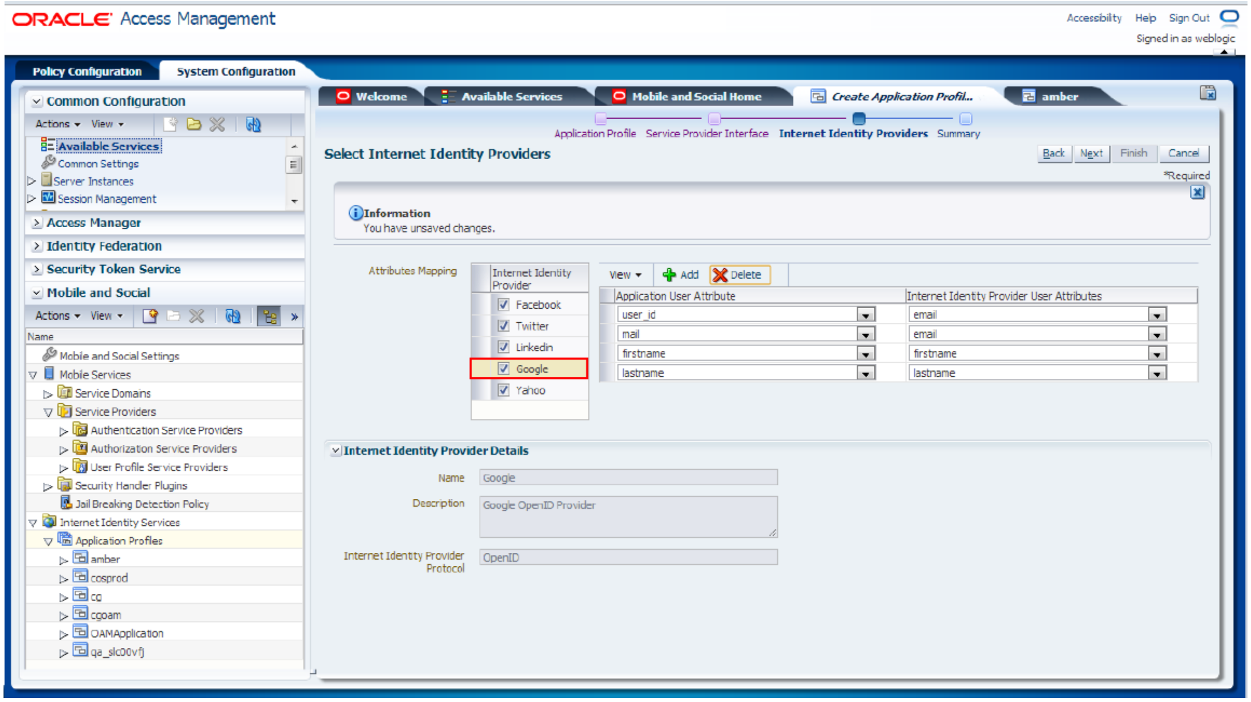
Name: LinkedIn

Description: LinkedIn OAuth Provider

Internet Identity Provider Protocol: OAuth

- To configure the Google identity provider, select the **Google** check box in the **Attribute Mapping** section and specify the attributes of the application user and the Internet identity provider user, as shown in Figure 80–34.

Figure 80–34 Configuration of the Google Identity Provider



Select Internet Identity Providers

Information
You have unsaved changes.

Attributes Mapping

Internet Identity Provider
☑ Facebook
☑ Twitter
☑ LinkedIn
☑ Google
☑ Yahoo

Application User Attribute	Internet Identity Provider User Attributes
user_id	email
mail	email
firstname	firstname
lastname	lastname

Internet Identity Provider Details

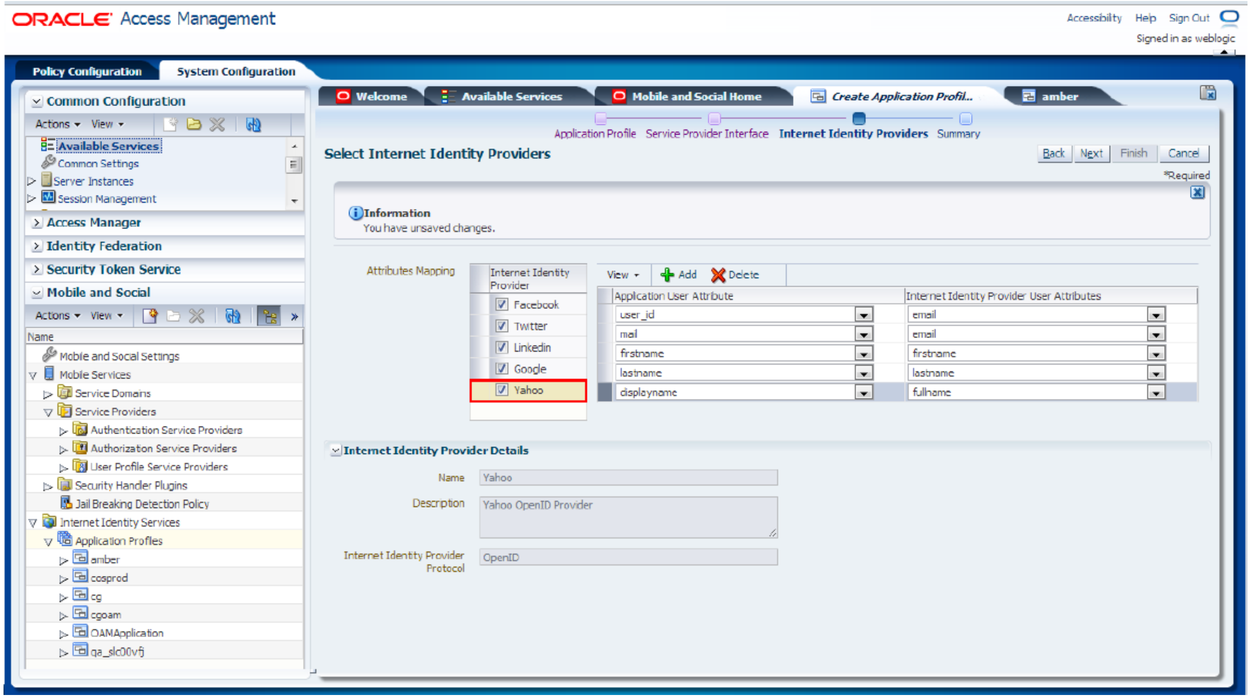
Name: Google

Description: Google OpenID Provider

Internet Identity Provider Protocol: OpenID

- To configure the Yahoo identity provider, select the **Yahoo** check box in the **Attribute Mapping** section and specify the attributes of the application user and the Internet identity provider user, as shown in Figure 80–35.

Figure 80–35 Configuration of the Yahoo Identity Provider



Select Internet Identity Providers

Information
You have unsaved changes.

Attributes Mapping

Internet Identity Provider
☑ Facebook
☑ Twitter
☑ LinkedIn
☑ Google
☑ Yahoo

Application User Attribute	Internet Identity Provider User Attributes
user_id	email
mail	email
firstname	firstname
lastname	lastname
displayname	fullname

Internet Identity Provider Details

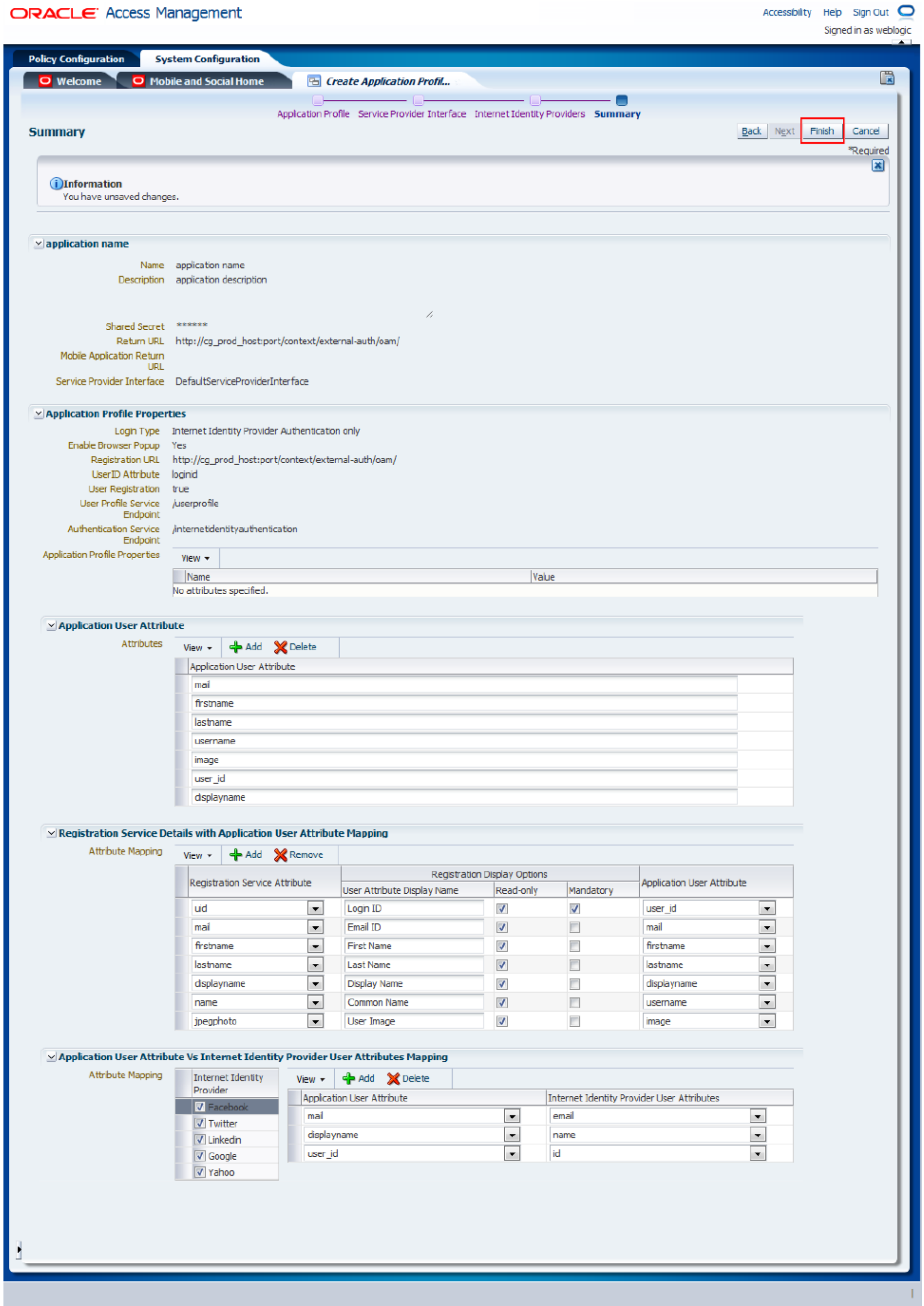
Name: Yahoo

Description: Yahoo OpenID Provider

Internet Identity Provider Protocol: OpenID

8. In the **Summary** page, on which all defined values are displayed (Figure 80–36), click **Finish** to complete creation process.

Figure 80–36 Summary Screen



Summary

Information
You have unsaved changes.

application name

Name	application name
Description	application description
Shared Secret	******
Return URL	http://cg_prod_host:port/context/external-auth/oam/
Mobile Application Return URL	
Service Provider Interface	DefaultServiceProviderInterface

Application Profile Properties

Login Type	Internet Identity Provider Authenticaton only
Enable Browser Popup	Yes
Registration URL	http://cg_prod_host:port/context/external-auth/oam/
UserID Attribute	loginid
User Registration	true
User Profile Service Endpoint	/userprofile
Authentication Service Endpoint	/internetidentityauthentication

Application Profile Properties — View ▾

Name	Value
No attributes specified.	

Application User Attribute

Attributes — View ▾ Add Delete

Application User Attribute
mail
firstname
lastname
username
image
user_id
displayname

Registration Service Details with Application User Attribute Mapping

Attribute Mapping — View ▾ Add Remove

Registration Service Attribute	Registration Display Options			Application User Attribute
	User Attribute Display Name	Read-only	Mandatory	
uid	Login ID	☑	☑	user_id
mail	Email ID	☑	☐	mail
firstname	First Name	☑	☐	firstname
lastname	Last Name	☑	☐	lastname
displayname	Display Name	☑	☐	displayname
name	Common Name	☑	☐	username
jpegphoto	User Image	☑	☐	image

Application User Attribute Vs Internet Identity Provider User Attributes Mapping

Attribute Mapping

Internet Identity Provider
☑ Facebook
☑ Twitter
☑ Linkedin
☑ Google
☑ Yahoo

View ▾ Add Delete

Application User Attribute	Internet Identity Provider User Attributes
mail	email
displayname	name
user_id	id

80.6.4.2 Create an Application Profile using WebLogic Scripting Tool (WLST) Script

To create a new application profile using WLST:

1. Launch WLST. For detailed steps, see the *Oracle Fusion Middleware Oracle WebLogic Scripting Tool*.

2. To connect to the WebLogic Server managed server on which OAM is running, run the `connect();` command.

3. To connect to the root of the domain-wide runtime management objects, run the `domainRuntime()` command.

4. Replace tokens in the following script and launch it. Table 80–2 describes these tokens and Example 80–1 includes is a sample script.

```
createRPApplication('Facebook,Twitter,Linkedin,Google,Yahoo',
   '<application_profile_shared_secret>',
   '<production_cg_protocol>://<production_cg_host>:<production_cg_
    port>/<production_cg_context>/external-auth/oam/',
   'DefaultServiceProviderInterface',
   '[{app.logintype:idp},
   {app.login.popup:true},
   {app.user.registration.enabled:true},
   {app.registration.url: "<production_cg_protocol>://<production_cg_
   host>:<production_cg_port>/<production_cg_context>/external-auth/oam/"},
   {app.userid.attribute.name:uid},
   {idaas.rest.crudservice:"/userprofile"},
   {idaas.rest.tokenservice:"/internetidentityauthentication"},
   {app.user.provisioning.mapping:"uid:Login ID:user_id:true:true;mail:Email
   ID:mail:true:false;firstname:First Name:firstname:true:false;lastname:Last
   Name:lastname:true:false;displayname:Display
   Name:displayname:true:false;name:Common
   Name:username:true:false;jpegphoto:User Image:image:true:false;"}]',
   '[{Facebook:[{user_id:id},{mail:email},{displayname:name}]},
   {Twitter:[{user_id:id_str},{displayname:name},{image:profile_image_url}]},
   {Linkedin:[{user_
   id:id},{image:picture-url},{firstname:first-name},{lastname:last-name}]},
   {Google:[{user_
   id:email},{mail:email},{firstname:firstname},{lastname:lastname}]},
   {Yahoo:[{user_
   id:email},{mail:email},{firstname:firstname},{lastname:lastname},
   {displayname:fullname}]}]',
   '[{user_id:uid},{mail:mail},{firstname:firstname},{lastname:lastname},
   {displayname:displayname},
   {username:name},{image:jpegphoto}]', '',
   '<mobile_and_social_application_profile_name>',
   '<mobile_and_social_application_profile_description>');
```

Table 80–2 Descriptions and Examples of Tokens

Token	Description	Example
`<application_profile_shared_secret>`	OAM application profile shared secret	password
`<production_cg_host>`	Host of the Community-Gadgets application server	prodcg.example.com
`<production_cg_port>`	Port of the Community-Gadgets application server	8080
`<production_cg_context>`	Context root of the Community-Gadgets application server	cg
`<mobile_and_social_application_profile_name>`	Name of the OAM application profile	Community-Gadgets profile
`<mobile_and_social_application_profile_description>`	Description of the OAM application profile	Description of the Community-Gadgets profile

Example 80–1 Sample Script

```
createRPApplication('Facebook,Twitter,Linkedin,Google,Yahoo', 'password',
     'http://prodcg.example.com:8080/cg/external-auth/oam/',
'DefaultServiceProviderInterface',
     '[{app.logintype:idp}, {app.login.popup:true},
     {app.user.registration.enabled:true}, {app.registration.url:
     "http://prodcg.example.com:8080/cg/external-auth/oam/"}, {
      app.userid.attribute.name:uid},
     {idaas.rest.crudservice:"/userprofile"},
     {idaas.rest.tokenservice:"/internetidentityauthentication"},
     {app.user.provisioning.mapping:"uid:Login ID:user_id:true:true;mail:Email
     ID:mail:true:false;firstname:First Name:firstname:true:false;lastname:Last
     Name:lastname:true:false;displayname:Display
     Name:displayname:true:false;name:Common
     Name:username:true:false;jpegphoto:User Image:image:true:false;"}]',
     '[{Facebook:[{user_id:id}, {mail:email}, {displayname:name}]},
     {Twitter:[{user_id:id_str}, {displayname:name}, {image:profile_image_url}]},
     {Linkedin:[{user_id:id}, {image:picture-url}, {firstname:first-name},
     {lastname:last-name}]}, {Google:[{user_id:email}, {mail:email},
     {firstname:firstname}, {lastname:lastname}]}, {Yahoo:[{user_id:email},
     {mail:email}, {firstname:firstname}, {lastname:lastname},
     {displayname:fullname}]}]', '[{user_id:uid}, {mail:mail},
     {firstname:firstname}, {lastname:lastname}, {displayname:displayname},
     {username:name}, {image:jpegphoto}]', '', 'Community Gadgets profile',
     'Community Gadgets profile description');
```

To complete the integration process, perform the steps described in Section 80.6.5, "Enabling the Integration of OMSAS with Community-Gadgets."

80.6.5 Enabling the Integration of OMSAS with Community-Gadgets

The last step after defining Internet identity providers for the Mobile and Social service and creating a Mobile and Social application profile for the Community-Gadgets application is to enable the integration of OMSAS with Community-Gadgets.

In the Community-Gadgets instance on the production system:

1. Navigate to the `<cg_install_dir>/deploy/production/production_node1` folder (or the folder you created during installation. See the *Oracle Fusion Middleware WebCenter Sites Installation Guide* for details).
2. Open the `setup_auth.properties` file and modify **OAM - Social Settings** section as follows:
   a. Enable Mobile and Social support by set the `widgets.external_auth.oam.attrs.enabled` parameter to true:

   ```
   ## OAM - Social Settings
   ########################################
   #
   # Enabling OAM - Social support
   # Default is "false"
   #
   widgets.external_auth.oam.attrs.enabled=true
   ```

   b. Specify the OAM managed server URL in the `widgets.external_auth.oam.attrs.hosturl` parameter:

   ```
   ## OAM - Social server URL
   widgets.external_auth.oam.attrs.hosturl=<oam_protocol>://<oam_host>:
   ```

```
<oam_port>
```

c. Specify the OMSAS application profile name in the `widgets.external_auth.oam.attrs.application_name` parameter:

```
# OAM - Social application name
#
widgets.external_auth.oam.attrs.application_name=<mobile_and_social_
application_profile_name>
```

d. Specify the OMSAS application profile secret in the `widgets.external_auth.oam.attrs.sharedsecret` parameter:

```
# OAM - Social application shared secret
#
widgets.external_auth.oam.attrs.sharedsecret=<application_profile_shared_
secret>
```

e. Save the file.

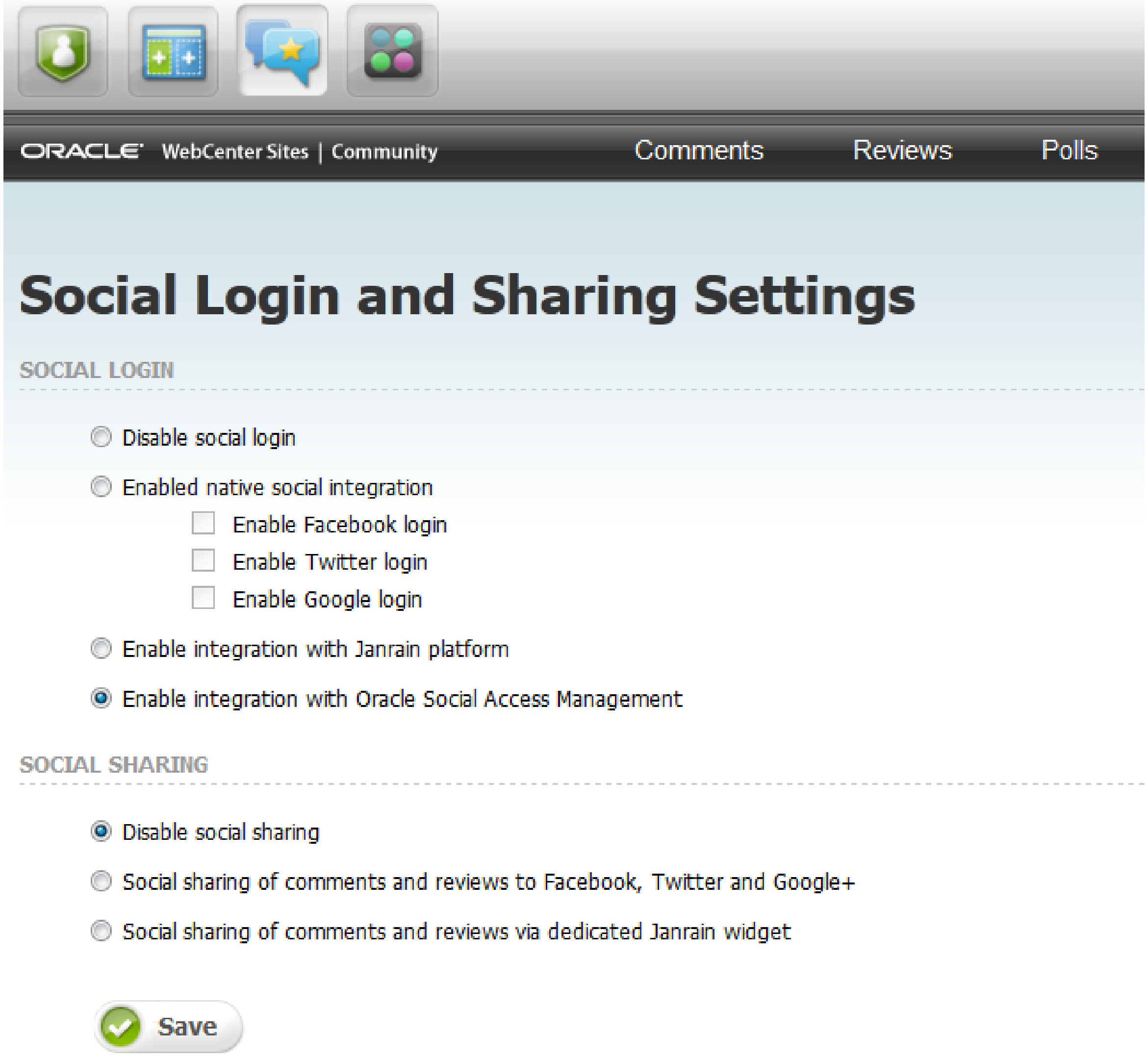
3. Restart Community-Gadgets.

4. Open the Community interface or the Gadgets interface with the credentials of a general administrator.

   a. From the **Login Bar** menu, choose the **Configure** option.

   b. On the **Social Login and Sharing Settings** screen, choose the **Enable integration with Oracle Mobile and Social Access Management** option, as shown in Figure 80–37.

Figure 80–37 Social Login and Sharing Settings - Oracle Mobile and Social Access Management



Social Login and Sharing Settings

SOCIAL LOGIN

○ Disable social login

○ Enabled native social integration

□ Enable Facebook login

□ Enable Twitter login

□ Enable Google login

○ Enable integration with Janrain platform

● Enable integration with Oracle Social Access Management

SOCIAL SHARING

● Disable social sharing

○ Social sharing of comments and reviews to Facebook, Twitter and Google+

○ Social sharing of comments and reviews via dedicated Janrain widget

c. Click **Save** to apply this setting.

80.7 Enabling Social Networking Services on WebSphere Application Server

If you integrated the Community-Gadgets web application with Facebook, Twitter, Google, or Janrain on a WAS instance, then you must also perform the procedures described in this section to enable communication between Community-Gadgets and these social networking sites.

This section includes the following:

- Section 80.7.1, "Export Security Certificate"
- Section 80.7.2, "Import Security Certificates into WebSphere"

80.7.1 Export Security Certificate

If you enabled native social integration and the login option in your Community-Gadgets web application (for reference, see Figure 80–25), then export the application certificates as you will need them to enable integration on WebSphere.

> **Tip:** The native login can be disabled via the Community interface and Gadgets interface by selecting **Login Bar** then **Configure** and choosing the **Disable social login** option. If you disable this feature, then the Login screen displayed on the web site will require visitors' local credentials (as used on the web site).

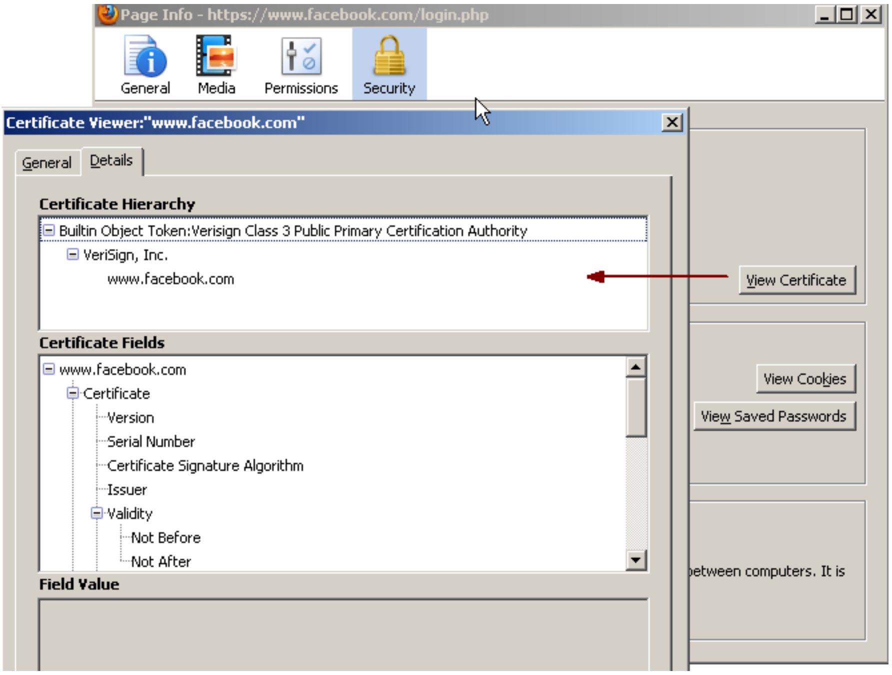
1. To get the security certificates, open a provider's URL (see Table 80–3) in FireFox browser. For example:
   `https://developers.facebook.com/docs/reference/api`

Table 80–3 Providers' URLs

Provider	URL
Facebook	`https://developers.facebook.com/docs/reference/api`
Twitter	`https://api.twitter.com`
Janrain	`https://rpxnow.com/`
Google	`https://accounts.google.com` and `https://www.googleapis.com`

2. From the **Tools** menu, choose **Page Info**.
3. On the **Security** tab, click **View Certificate** (Figure 80–38) to display the Certificate Viewer dialog box.

Figure 80–38 Facebook Security Certificate



4. Choose the lowermost node.

5. On the **Details** tab, click **Export**.
6. From the **Save in** drop-down list, choose a directory in which you want to save this certificate.
7. In the Save Certificate to File dialog box, from the **Save as type** field, choose **X.509 Certificate (DER)**.
8. Click **Save**.
9. Repeat steps 5 through 8 for the parent of the lowermost node.

> **Note:** Facebook additionally requires the `Entrust.net` secure server certificate, which you can download from your Internet Explorer by choosing **Internet Options**, then selecting **Content**, then **Certificates**, and then**Entrust.net Secure Server Certification Authority**.

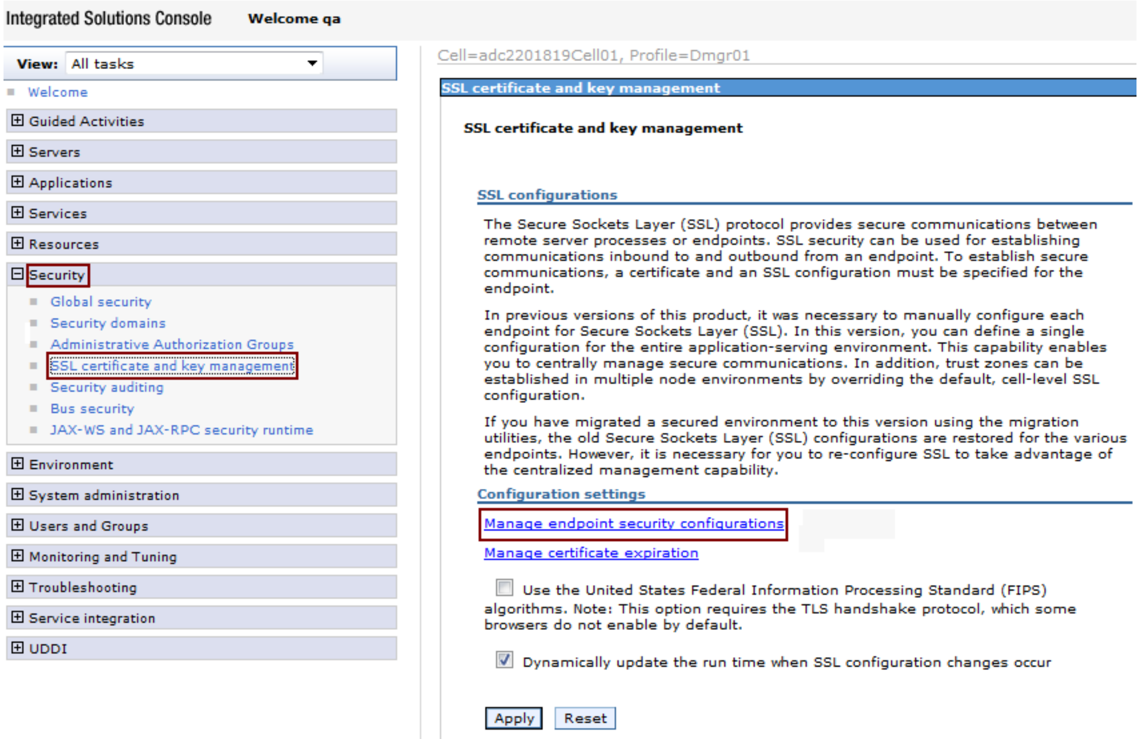
10. Repeat steps 3 through 9 for each Provider URL from Table 80–3.

80.7.2 Import Security Certificates into WebSphere

To import security certificates into WebSphere:

1. Log in to the WAS administrative console.
2. Expand **Security**, then click **SSL certificate and key management**.
3. Under **Configuration settings**, click **Manage endpoint security configurations**, as shown in Figure 80–39.

Figure 80–39 Configuration Settings



SSL certificate and key management

SSL configurations

The Secure Sockets Layer (SSL) protocol provides secure communications between remote server processes or endpoints. SSL security can be used for establishing communications inbound to and outbound from an endpoint. To establish secure communications, a certificate and an SSL configuration must be specified for the endpoint.

In previous versions of this product, it was necessary to manually configure each endpoint for Secure Sockets Layer (SSL). In this version, you can define a single configuration for the entire application-serving environment. This capability enables you to centrally manage secure communications. In addition, trust zones can be established in multiple node environments by overriding the default, cell-level SSL configuration.

If you have migrated a secured environment to this version using the migration utilities, the old Secure Sockets Layer (SSL) configurations are restored for the various endpoints. However, it is necessary for you to re-configure SSL to take advantage of the centralized management capability.

Configuration settings

Manage endpoint security configurations

Manage certificate expiration

☐ Use the United States Federal Information Processing Standard (FIPS) algorithms. Note: This option requires the TLS handshake protocol, which some browsers do not enable by default.

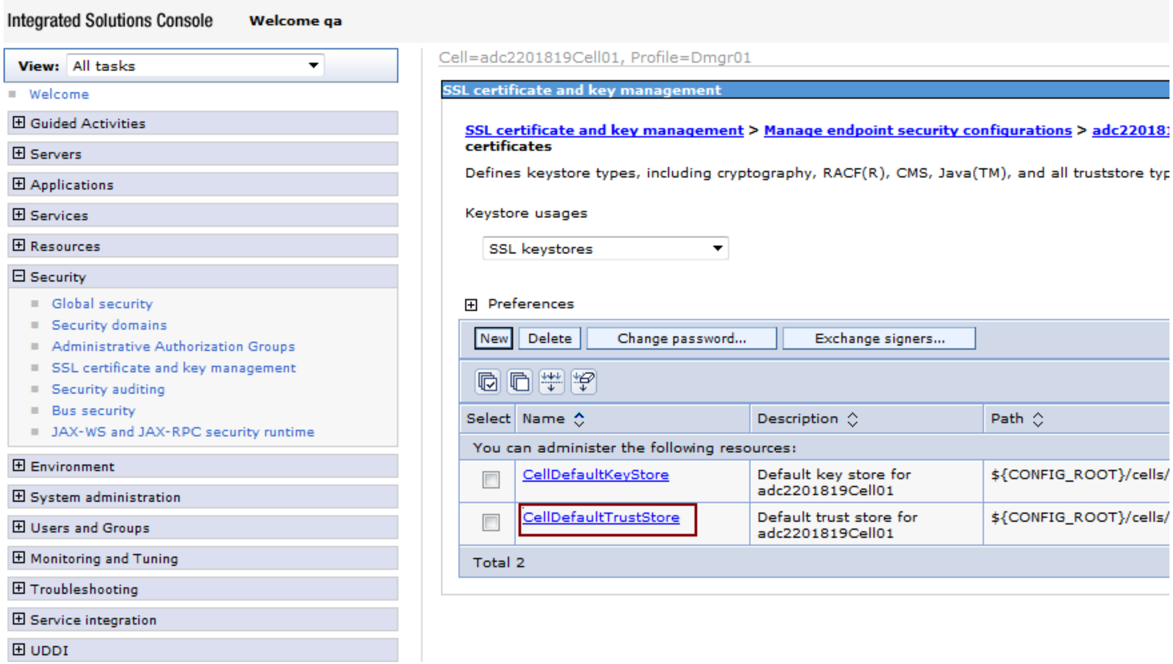
☑ Dynamically update the run time when SSL configuration changes occur

4. Under the **Local Topology** tab, expand **Outbound**, then select the appropriate outbound configuration to get to the (cell).

5. Under **Related Items** on the right side, click **Key stores and certificates**.

6. Under **Preferences**, select the **CellDefaultTrustStore** key store, as shown in Figure 80–40.

Figure 80–40 Preferences: CellDefaultTrustStore



Cell=adc2201819Cell01, Profile=Dmgr01

SSL certificate and key management

SSL certificate and key management > Manage endpoint security configurations > adc22018
certificates

Defines keystore types, including cryptography, RACF(R), CMS, Java(TM), and all truststore typ

Keystore usages: SSL keystores

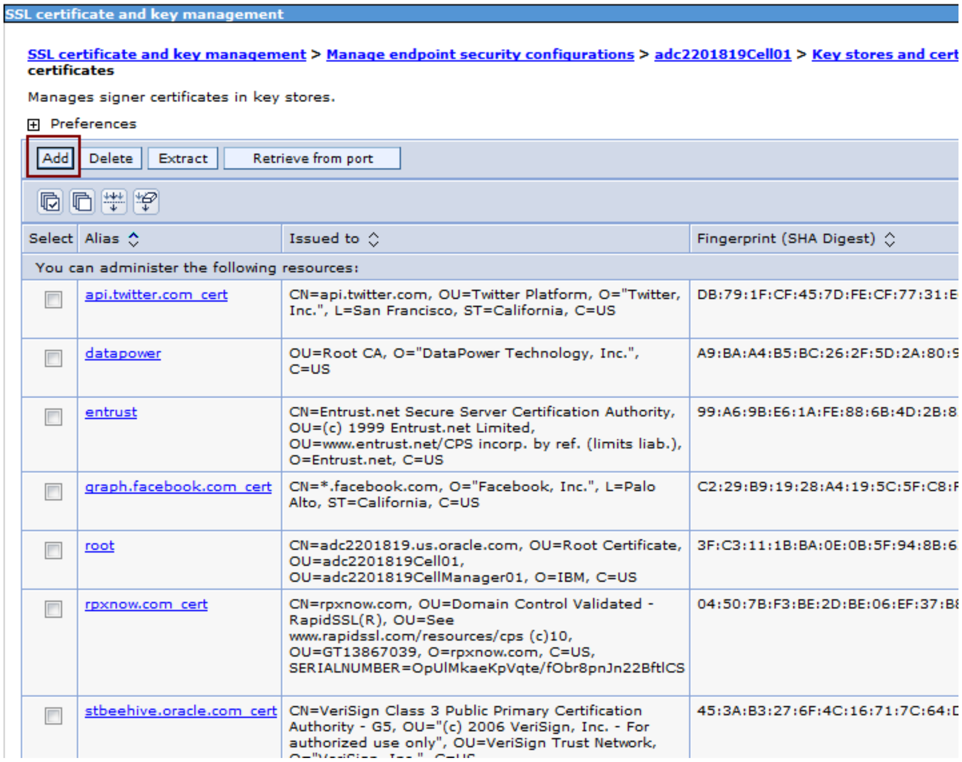
Preferences

Select	Name	Description	Path
You can administer the following resources:			
□	CellDefaultKeyStore	Default key store for adc2201819Cell01	${CONFIG_ROOT}/cells/
□	CellDefaultTrustStore	Default trust store for adc2201819Cell01	${CONFIG_ROOT}/cells/
Total 2			

7. Under **Additional Properties** on the right side, click **Signer certificates**.

8. Under **Preferences**, click **Add** (Figure 80–41).

Figure 80–41 Preferences: Add



SSL certificate and key management

SSL certificate and key management > Manage endpoint security configurations > adc2201819Cell01 > Key stores and cert
certificates

Manages signer certificates in key stores.

⊞ Preferences

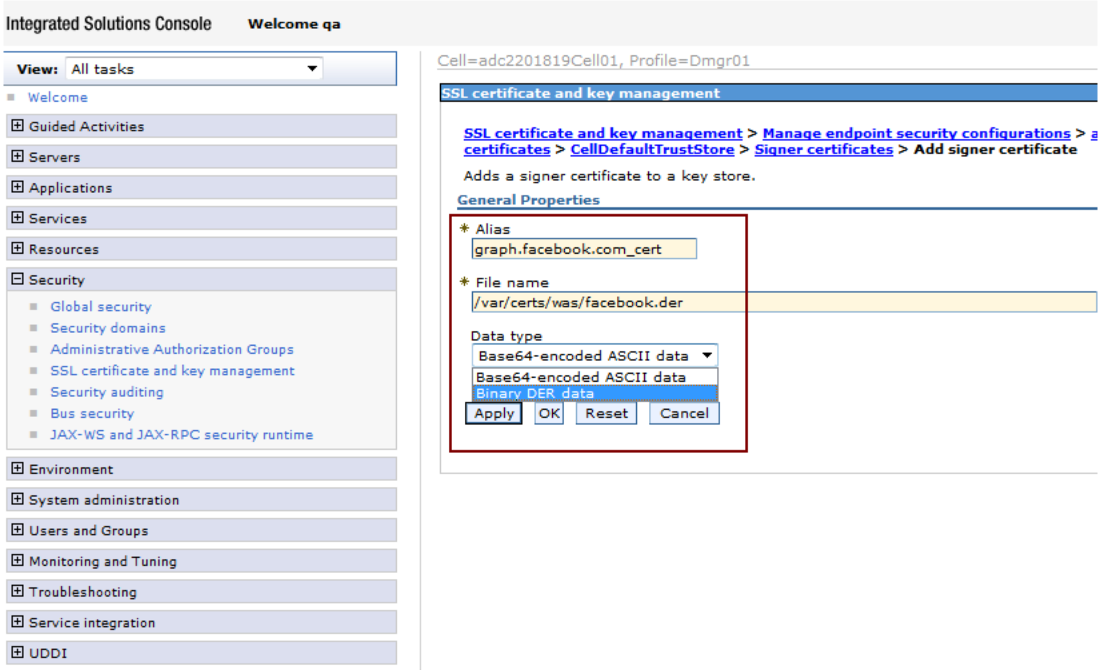
Select	Alias	Issued to	Fingerprint (SHA Digest)
You can administer the following resources:			
□	api.twitter.com_cert	CN=api.twitter.com, OU=Twitter Platform, O="Twitter, Inc.", L=San Francisco, ST=California, C=US	DB:79:1F:CF:45:7D:FE:CF:77:31:E
□	datapower	OU=Root CA, O="DataPower Technology, Inc.", C=US	A9:BA:A4:B5:BC:26:2F:5D:2A:80:9
□	entrust	CN=Entrust.net Secure Server Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/CPS incorp. by ref. (limits liab.), O=Entrust.net, C=US	99:A6:9B:E6:1A:FE:88:6B:4D:2B:8
□	graph.facebook.com_cert	CN=*.facebook.com, O="Facebook, Inc.", L=Palo Alto, ST=California, C=US	C2:29:B9:19:28:A4:19:5C:5F:C8:F
□	root	CN=adc2201819.us.oracle.com, OU=Root Certificate, OU=adc2201819Cell01, OU=adc2201819CellManager01, O=IBM, C=US	3F:C3:11:1B:BA:0E:0B:5F:94:8B:6
□	rpxnow.com_cert	CN=rpxnow.com, OU=Domain Control Validated - RapidSSL(R), OU=See www.rapidssl.com/resources/cps (c)10, OU=GT13867039, O=rpxnow.com, C=US, SERIALNUMBER=OpUlMkaeKpVqte/fObr8pnJn22BftlCS	04:50:7B:F3:BE:2D:BE:06:EF:37:B8
□	stbeehive.oracle.com_cert	CN=VeriSign Class 3 Public Primary Certification Authority - G5, OU="(c) 2006 VeriSign, Inc. - For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US	45:3A:B3:27:6F:4C:16:71:7C:64:D

9. In the **Alias** field, enter the URL names depending on the certificates you are adding, as shown in Figure 80–42. For example, `graph.facebook.com_cert`, `api.twitter.com_ cert`, or `rpxnow.com_cert`.

10. In the **File name** field, enter the path to the saved certificates, as shown in Figure 80–42.

11. From **Data type**, choose **Binary DER data**, as shown in Figure 80–42.

Figure 80–42 SSL Certificate and Key Management: Data Type



Integrated Solutions Console
Welcome qa
View: All tasks
Welcome
⊞ Guided Activities
⊞ Servers
⊞ Applications
⊞ Services
⊞ Resources
⊟ Security
Global security
Security domains
Administrative Authorization Groups
SSL certificate and key management
Security auditing
Bus security
JAX-WS and JAX-RPC security runtime
⊞ Environment
⊞ System administration
⊞ Users and Groups
⊞ Monitoring and Tuning
⊞ Troubleshooting
⊞ Service integration
⊞ UDDI
Cell=adc2201819Cell01, Profile=Dmgr01
SSL certificate and key management
SSL certificate and key management > Manage endpoint security configurations > a
certificates > CellDefaultTrustStore > Signer certificates > Add signer certificate
General Properties
Base64-encoded ASCII data
Binary DER data
Apply
OK
Reset
Cancel

Adds a signer certificate to a key store.

* Alias
graph.facebook.com_cert

* File name
/var/certs/was/facebook.der

Data type
Base64-encoded ASCII data

12. Click **Apply**.

81

Community-Gadgets: Customizing Its Functionality

The Community-Gadgets web application provides a number of customizability and extension points, such as CSS, templates, and Search Engine Optimization (SEO), which you can use to alter the appearance and functionality of any Community-Gadgets widget. However, using customizability and extension points may not always meet particular customers' project requirements because of certain web site design or usage related strategies applied to the widgets. In such scenarios, consider working with WebCenter Sites data structures directly and rendering them separately in a custom way in page templates.

This chapter describes WebCenter Sites data structures in details. It contains the following sections:

- Section 81.1, "Overview of Community-Gadgets Data Model"
- Section 81.2, "Customizing CSS and Widget Templates"
- Section 81.3, "Creating a Custom Word Filter"
- Section 81.4, "Creating a CAPTCHA Generator"
- Section 81.5, "Customizing Dashboard Themes"
- Section 81.6, "Community-Gadgets Events Handling"

81.1 Overview of Community-Gadgets Data Model

Community-Gadgets uses WebCenter Sites as its data repository and communicates with it through REST APIs provided by Web Experience Management (WEM) Framework. Storing Community-Gadgets data structures in the WebCenter Sites repository leverages the WebCenter Sites asset model and caching system.

The object data model of the Community-Gadgets web application consists of the following categories and their objects:

Section 81.1.1, "Comments"

- Object: CommentFeed
- Object: CommentRecord

Section 81.1.2, "Reviews"

- Object: ReviewFeed
- Object: ReviewRecord

Section 81.1.3, "Ratings"

- Object: RatingFeed
- Object: RatingRecord

Section 81.1.4, "Polls"

Section 81.1.5, "Topics"

Section 81.1.6, "Visitors"

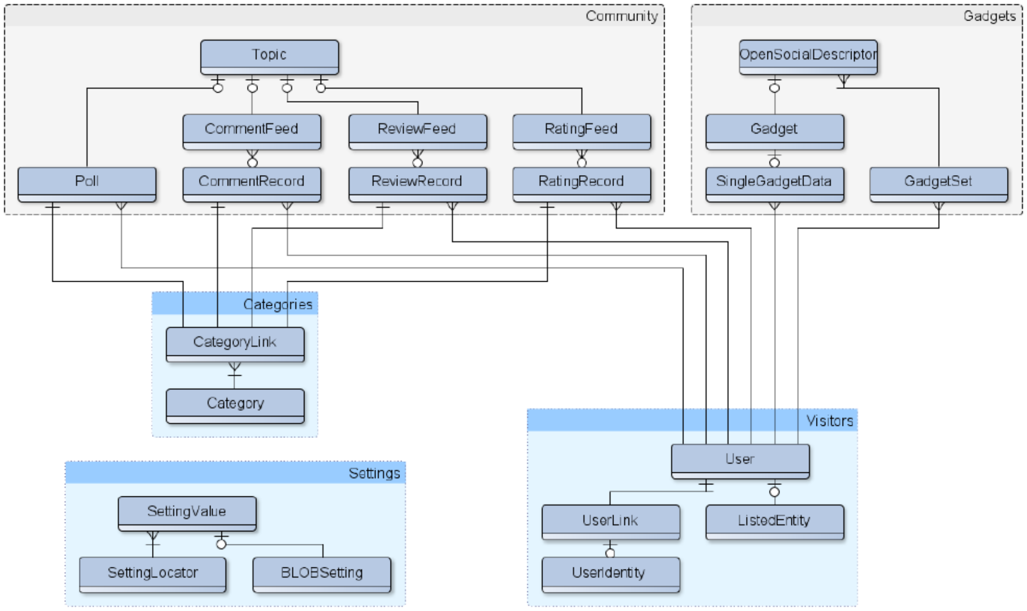
- Object: User
- Object: UserIdentity
- Object: UserLink

Section 81.1.7, "Gadgets and Dashboard"

- Object: Gadget
- Object: OpenSocialDescriptor
- Object: SingleGadgetData
- Object: GadgetSet

Figure 81–1 provides a graphical overview of these categories and objects. The User-Generated Content section shows that business objects are related to Community widgets, such as comments and reviews. Each user-generated content (UGC) entry, if posted by a registered user, has a set of user objects associated with it (the Visitors section).

Figure 81–1 Categories and Object Layout



Objects shown in Figure 81–1 are stored in the WebCenter Sites database tables, as specified in Table 81–1.

Table 81–1 Database Table Names of Objects

Object	Database Table Name
`CommentFeed`	`cg_comment_feed`

Table 81–1 (Cont.) Database Table Names of Objects

Object	Database Table Name
CommentRecord	cg_comment_record
ReviewFeed	cg_review_feed
ReviewRecord	cg_review_record
RatingFeed	cg_rating_feed
RatingRecord	cg_rating_record
Polls	cg_poll
Topics	cg_topic
User	cg_user
UserIdentity	cg_user_id
UserLink	cg_user_link

81.1.1 Comments

The following types of primary objects are associated with comments:

- Section 81.1.1.1, "CommentFeed"
- Section 81.1.1.2, "CommentRecord"

81.1.1.1 CommentFeed

This object represents the computed summary information about the web page on which the widget is deployed. The CommentFeed object is created in the database when the page on which the Comments widget is deployed is accessed for the first time.

To determine the connection between comments and the page on which they should be displayed, the Community-Gadgets web application uses the web page URL, which is the application-generated key in the widget deployment tag, or Resource ID, which administrators can specify during widget deployment. If the web page URL is available as the key, then the hash generated from the page URL is recorded in the CommentFeed object. If the resource ID is specified, it is recorded in the CommentFeed object. The CommentFeed object stores comments, and it has a one-to-many association with the CommentRecord objects that represent comments.

The CommentFeed objects are stored as a `cg_comment_feed` basic asset in WebCenter Sites and correspondingly as a `cg_comment_feed` table in the database. Table 81–2 describes the structure of this asset.

Table 81–2 Structure of the "cg_comment_feed" AssetType

Property	Type	Description
id	BIGINT	Unique asset identifier or primary key of AssetType.

Table 81–2 (Cont.) Structure of the "cg_comment_feed" AssetType

Property	Type	Description
`cos_resource_id`	VARCHAR(256)	Identifier of the discussion. It is a logical link that Community-Gadgets uses to associate a list of comments with the page on which visitors posted comments. This link is established during widget deployment by specifying the value of the **Resource ID** field on the deployment page or the `resource_id` attribute in the widget code snippet. If it is not specified, Community-Gadgets uses the hash of the page URL as a resource ID. It is recommended that you come up with your own strategy of `resource_id` generation. For example, to make it easier to find this object in the database, you can use the page ID as the resource ID.
`cos_url`	VARCHAR(256)	URL that hosts the comment discussion thread.
`cos_date_created`	DATE	Date when the feed object was created.
`cos_date_modified`	DATE	Latest modification date of a feed.
`cos_modified_by`	VARCHAR(256)	ID of the user who last modified the feed.
`cos_owner`	VARCHAR(256)	ID of the user who created the feed object. (The feed object is created during the first page load when a widget is just deployed. Typically it is done using a guest session, so `-1` is a typical value.)
`cos_approved_count`	BIGINT	Number of comments that successfully passed moderation and are displayed to visitors.
`cos_pending_count`	BIGINT	Number of comments that have not passed moderation yet.
`cos_ext_type`	VARCHAR(256)	ID of the content category. It describes the type of content to which comments are attached: file, article, blog, and so on. (Specified as the `Resource Type` parameter on the deployment page or the `resource_type` attribute in the widget deployment code snippet.)

Table 81–2 (Cont.) Structure of the "cg_comment_feed" AssetType

Property	Type	Description
`cos_resource_title`	VARCHAR(256)	Title of the page on which the Comments widget is deployed. By default, the value of the window title is recorded here. Page designers may customize this value by specifying the `resource_title` attribute in the deployment code snippet.
`cos_posting_status`	VARCHAR(256)	State of posting status which is defined from the Admin interface by Topic Page. Possible values: `open`, `closed`

81.1.1.2 CommentRecord

This object represents a comment posted by a visitor on a site page. It is logically linked to the CommentFeed object through a many-to-one relationship via the `cos_root_id` field. In other words, many CommentRecord objects can be linked to one CommentFeed object.

Table 81–3 Structure of the cg_comment_record assetType

Property	Type	Description
`id`	BIGINT	Identifier of a comment.
`cos_text`	VARCHAR(4000)	Content of a posted comment. Its VARCHAR(4000) data type is translated into the most effective type of storage (for example, CLOB), depending on the database on which WebCenter Sites is installed.
`cos_root_id`	BIGINT	Association to the CommentFeed object representing the page on which the conversation is happening. The value of the corresponding `cg_comment_feed.id` item is stored here. Use this column to query all the comments posted on a page.

Table 81–3 (Cont.) Structure of the cg_comment_record assetType

Property	Type	Description
`cos_state_value`	VARCHAR(450)	State of a comment reflecting the stage of comment moderation. If manual moderation is enabled, the value may be: - `pending.new` (newly posted) - `pending.modified` (manual moderation) If a comment has either passed moderation or is auto-published, the value will be: - `approved.all` If a comment has not passed through either automatic filters or moderator, then the values are: - `inappropriate.robotdetected` - `inappropriate.humandetected` To simplify navigation for moderators, the Community interface includes filters based on these categories.
`cos_owner`	BIGINT	ID of the visitor who posted the comment, or `-1` if it is a guest entry.
`cos_owner_ip`	VARCHAR(256)	Client's IP address from which the comment was posted.
`cos_guest_name`	VARCHAR(450)	Display name of a registered user or a guest who posted the comment entry. This field is used in the Community interface to display and sort comments by author names without posting extra requests to database.
`cos_guest_email`	VARCHAR(450)	If guest (unauthenticated visitors) posts are enabled in commenting permissions, and visitors are required to specify their email ID, then the email ID is saved in this column.
`cos_level`	BIGINT	Level of a comment in the comment thread, if replies to comments are enabled. The bigger the number, the deeper the comment is posted in the tree. The initial value is `1`.

Table 81–3 (Cont.) Structure of the cg_comment_record assetType

Property	Type	Description
`cos_parentid`	BIGINT	ID of the immediate parent comment to which a reply is posted.
`cos_parent0id` `cos_parent9id`	BIGINT	Chain of comment parents in the hierarchy. The path from the leaf to the root.
`cos_flagged`	VARCHAR(32)	`true` or `false` based on whether the comment has been flagged and reported by other visitors.
`cos_flagged_count`	BIGINT	The number of times a comment is flagged.
`cos_record_rank`	VARCHAR(450)	Counts of helpfulness reports ("Yes" and "No") made on a comment, separated by comma, starting with zero. The initial value is `0,0`.
`cos_record_rank_calculated`	INTEGER	Pre-calculated rank of a comment according to helpfulness reported by other visitors. This is calculated by subtracting the count of "No" from the count of "Yes".
`cos_date_created`	DATE	Date when the comment was posted.
`cos_date_modified`	DATE	Latest modification date of a comment (registered users are allowed to modify their comments, or moderator may modify a comment.).
`cos_reply_count`	BIGINT	If the discussion thread is enabled and replies can be posted to comments, then the number of replies to the current comment are recorded here. This count is not recursive, and therefore, only immediate child comments are considered (the children of children are excluded).

Table 81–3 (Cont.) Structure of the cg_comment_record assetType

Property	Type	Description
`cos_thread_order`	VARCHAR(450)	String generated in a special format. It enables ordering of comments by thread hierarchy using a simple WEM REST query and alphabetic ordering. The generation strategy is as follows: - If it is a root level comment (that is, it has not been posted as a reply to another comment), the value of the `cos_date_created` field in the hexadecimal format is recorded. - If a comment has a parent comment, the concatenation value of parent's `cg_comment_record.cos_thread_order` field and the `underscore "_"` field, as well as its `cos_date_created` field in the hexadecimal format.

81.1.2 Reviews

The following types of primary objects are associated with reviews. These objects are similar to those of comments:

- Section 81.1.2.1, "ReviewFeed"
- Section 81.1.2.2, "ReviewRecord"

81.1.2.1 ReviewFeed

This object represents a list of reviews posted on a site page. The ReviewFeed object is very similar to the CommentFeed object described in Section 81.1.1.1, "CommentFeed." Community-Gadgets uses the same code infrastructure to handle these objects. However, to simplify maintenance on the database schema level, the data for CommentFeed and ReviewFeed is stored in two separate tables. The only difference between CommentFeed and ReviewFeed is that the latter includes additional information pertaining to the average rank calculated across all posted reviews.

The ReviewFeed object is created in the database when the page on which the Reviews widget is deployed is accessed for the first time.

To determine the connection between reviews and the page on which they should be displayed, Community-Gadgets uses the web page URL, which is the application-generated key in the widget deployment tag, or Resource ID, which administrators can specify during widget deployment. If the web page URL is available as the key, then the hash generated from the page URL is recorded in the ReviewFeed object. If the resource ID is specified, it is recorded in the ReviewFeed object. The ReviewFeed object stores reviews, and it has a one-to-many association with the ReviewRecord objects that represent reviews.

The review feed objects are stored as a cg_review_feed basic asset in WebCenter Sites and correspondingly as a cg_review_feed table in the database. Table 81–4 describes the structure of this asset.

Table 81–4 Structure of cg_review_feed AssetType

Property	Type	Description
id	BIGINT	Unique asset identifier or primary key of the AssetType.
cos_resource_id	VARCHAR(256)	Identifier of a discussion. It is a logical link that Community-Gadgets uses to associate a list of reviews with the page on which the reviewed topic is posted. This link is established during widget deployment by specifying the value of the **Resource ID** field on the deployment page or the resource_id attribute in the widget code snippet. If it is not specified, Community-Gadgets uses the hash of the page URL as a resource ID. It is recommended that you devise your own strategy of resource_id generation. For example, to make it easier to find this object in the database, you can use the page ID as the resource ID.
cos_url	VARCHAR(256)	URL hosting the discussion to which reviews are being posted.
cos_date_created	DATE	Date when the feed object was created.
cos_date_modified	DATE	Latest modification date of a feed.
cos_modified_by	VARCHAR(256)	ID of the user who last modified the feed.
cos_owner	VARCHAR(256)	ID of the user who created the feed object. (The feed object is created during the first page load when a widget is just deployed. Typically it is done using a guest session, so -1 is a typical value.)
cos_approved_count	BIGINT	Number of reviews that successfully passed moderation and are displayed to visitors.
cos_pending_count	BIGINT	Number of reviews that have not pass moderation yet.

Table 81–4 (Cont.) Structure of cg_review_feed AssetType

Property	Type	Description
`cos_rank`	FLOAT	Calculated as a mean, the average of all ranks that have been given to the reviews posted on a page. For example, if two reviews were posted on a page, out of which one has been given three stars and another five stars, then the value in this field will be 3+5/2 = 4.
`cos_rank_precalculation`	VARCHAR(450)	Comma separated list of counts of ranks posted. This list contains five items in total. Each position (index) in the list holds the count of the corresponding number of stars given to a review. For example, if there are three reviews, out of which one has been given three stars and the others five stars, 0-s will be given for missing positions, and therefore, the list will contain `0,0,1,0,2`. Note the counts of reviews posted at corresponding positions. From concrete position, you can know the number of votes/stars.
`cos_thumbs_up_rank`	VARCHAR(450)	If the review type is set to thumbs up/down, the votes are recorded in this field. This is a denormalized field that holds a count of thumbs up and thumbs down separated by comma. For example, if out of three reviews, two have got thumb up and one thumb down, then the value of this field will be `2,1,`.
`cos_ext_type`	VARCHAR(256)	ID of the content category describing the type of content to which the reviews are attached, for example, file, article, blog, and so on. (Specified as the `Resource Type` parameter on the deployment page or the `resource_type` attribute in the widget deployment code snippet.)
`cos_resource_title`	VARCHAR(256)	Title of the page on which the Reviews widget is deployed. By default, the value of the window title is recorded here. Page designers may customize this value by specifying the `resource_title` attribute in the deployment code snippet.

Table 81–4 (Cont.) Structure of cg_review_feed AssetType

Property	Type	Description
`cos_posting_status`	VARCHAR(256)	State of posting status which defined from admin interface by Topic Page. Possible values: open, closed

81.1.2.2 ReviewRecord

This object represents a review posted by a visitor on a site page. It is logically linked to the ReviewFeed object through a many-to-one relationship via the `cos_root_id` field.

Table 81–5 Structure of the cg_review_record AssetType

Property	Type	Description
`id`	BIGINT	Identifier of a review.
`cos_text`	VARCHAR(4000)	Content of a review posted. Its VARCHAR(4000) data type is translated into the most effective type of storage (for example, CLOB), depending on the database on which WebCenter Sites is installed.
`cos_title`	VARCHAR(450)	Title of a review posted.
`cos_rank`	FLOAT	Number of stars (rank) given to a review by a visitor. If the rating type is thumbs up/down, then `5` is the value for a thumbs up and `1` for a thumbs down.
`cos_thumbs_up`	INTEGER	If the rating type is configured to be thumbs up/down, the rank value stored here is: `1` for thumbs up and `-1` for thumbs down.
`cos_rating_type`	INTEGER	Type of a given review rank. Possible value codes are: 0: Stars 1: Thumbs up/down
`cos_root_id`	BIGINT	Association to the ReviewFeed object representing the page on which reviews are submitted. The value of the corresponding `cg_review_feed.id` item is stored here. Use this column to query all the reviews posted on a page.

Table 81–5 (Cont.) Structure of the cg_review_record AssetType

Property	Type	Description
`cos_state_value`	VARCHAR(450)	State of a review reflecting the stage of content moderation. If manual moderation is enabled, the value may be: - `pending.new` (newly posted) - `pending.modified` (manual moderation happened) If a review has either passed moderation or was auto-published, the value will be: - `approved.all` If a review has not passed through either automatic filters or moderator, values will be: - `inappropriate.robotdetected` - `inappropriate.humandetected` To simplify navigation for moderators, the Community interface includes filters based on these categories.
`cos_owner`	BIGINT	ID of the visitor who posted the review, or `-1` if it is a guest entry.
`cos_owner_ip`	VARCHAR(256)	Client's IP address from which the review was posted.
`cos_guest_name`	VARCHAR(450)	Display name of a registered user or a guest who posted the review entry. This field is used to display and sort reviews by author names in the Community interface without making extra requests to database.
`cos_guest_email`	VARCHAR(450)	If guest (unauthenticated visitors) posts are enabled in reviewing permissions and visitors are required to specify their email ID, the email ID value is saved in this column.
`cos_flagged`	VARCHAR(32)	`true` or `false` based on whether a review has been flagged and reported by other visitors.
`cos_flagged_count`	BIGINT	The number of times a review has been flagged by others.

Table 81–5 (Cont.) Structure of the cg_review_record AssetType

Property	Type	Description
`cos_record_rank`	VARCHAR(450)	Counts of helpfulness reports ("Yes" and "No") made on a review, separated by comma, starting with zero. The initial value is `0,0`.
`cos_record_rank_calculated`	INTEGER	Pre-calculated rank of a review according to helpfulness reported by other visitors. It is the count of "No" subtracted from the count of "Yes".
`cos_date_created`	DATE	Date when the review was posted.
`cos_date_modified`	DATE	Latest modification date of a review. (Only registered users are allowed to modify their reviews. Moderator can also modify a review, if needed.)

81.1.3 Ratings

This object contains the list of ratings that visitors give to topics or any posts on a page. This is very similar to the ReviewFeed object.

The Ratings object is created in the database when a web page is accessed for the first time after the deployment of the Ratings widget on a web site.

To determine the connection between ratings and the page on which they should be displayed, Community-Gadgets uses the web page URL, which is the application-generated key in the widget deployment tag, or "Resource ID", which administrators can specify during widget deployment. If the web page URL is available as the key, then the hash generated from the page URL is recorded in the RatingFeed object. If the resource ID is specified, it is recorded in the RatingFeed object. The RatingFeed object stores ratings, and it has a one-to-many association with the RatingRecord objects that represent ratings.

The rating feed objects are stored as the *cg_rating_feed* basic asset in the WebCenter Sites repository and correspondingly as the *cg_rating_feed* table in the database.

The following types of primary objects are associated with ratings:

- Section 81.1.3.1, "RatingFeed"
- Section 81.1.3.2, "RatingRecord"

Table 81–6 and Table 81–7 describe the structure of the `cg_rating_feed` assetType.

81.1.3.1 RatingFeed

Table 81–6 Structure of the cg_rating_feed AssetType

Property	Type	Description
`id`	BIGINT	Identifier of an object which holds a list of ratings posted on a page.

Table 81–6 (Cont.) Structure of the cg_rating_feed AssetType

Property	Type	Description
`cos_resource_id`	VARCHAR(256)	Identifier of the ratings on a particular web page. It is a logical link which enables Community-Gadgets to associate a list of ratings with the relevant page. This link is established during widget deployment by specifying the value of the **Resource ID** field on the deployment page or the `resource_id` attribute in the widget code snippet. This value can be generated or filled manually. If this value is not specified, Community-Gadgets uses the hash of page URL as a resource ID. It is recommended that you create your own strategy to generate `resource_id`. For example, to make the database search of this object easier, you can use the page ID as the resource ID.
`cos_url`	VARCHAR(256)	URL of the web page on which visitors leave their ratings.
`cos_date_created`	DATE	Date when the feed object is created.
`cos_date_modified`	DATE	Last modified date of a feed.
`cos_owner`	VARCHAR(256)	ID of the user who created the feed object initially. (This happens during the first page load when a widget is just deployed. Typically, the feed object is created via a guest session, so `-1` is a typical value.)
`cos_approved_count`	BIGINT	Number of ratings that successfully passed moderation and are counted in the general rating calculation.
`cos_pending_count`	BIGINT	Number of ratings that have not passed moderation yet.
`cos_stars_rank`	FLOAT	Average (mean) of all star type ratings posted by visitors. For example, if out of the two ratings, one is a three-star rank and another is a five-star rank, then the value is calculated as 3+5/2 = 4.

Table 81–6 (Cont.) Structure of the cg_rating_feed AssetType

Property	Type	Description
cos_stars_rank_precalculation	VARCHAR(450)	Comma separated list of counts of ratings. This list contains five items in total. Each position (index) in the list holds the count of the corresponding number of stars given. For example, if out of the three ratings given, one is a three-star and the other two are five-star, then 0-s in this list signify missing positions. So the list will include: 0,0,1,0,2. Note the counts of ratings left at corresponding positions. From concrete positions you can know the number of votes/stars.
cos_thumbs_up_rank	VARCHAR(450)	If the rating type is set to thumbs up/down, then the ratings are recorded in this field. This is a denormalized field to hold a comma separated count of thumbs up and thumbs down. For example, if out of the three ratings, the two are thumbs up and one is thumbs down, then the value stored will be equal to `2,1`.
cos_likeit_count	BIGINT	If the `like it` rating type is deployed, then the number of likes are recorded in this field.
cos_recommend_count	BIGINT	If the `recommend` rating type is deployed, then the number of recommendations is recorded in this field.
cos_ext_type	VARCHAR(256)	ID of a content category. It describes the type of content to which the ratings are attached: file, article, blog, and so on. These are specified as the value of the "Resource Type" parameter on the deployment page or the `resource_type` attribute in the widget deployment code snippet.
cos_resource_title	VARCHAR(256)	Title of the page on which the Ratings widget is deployed. By default the value of the window title is recorded as the page title. Page designers can customize this value by specifying the `resource_title` attribute in the deployment code snippet.

81.1.3.2 RatingRecord

This object represents a rating posted by a visitor on a site page. It is logically linked to the RatingFeed object via the "cos_root_id" field. This object is based on the many-to-one relationship model.

Table 81–7 Structure of the cg_rating_record AssetType

Property	Type	Description
`id`	BIGINT	Identifier of a rating.
`cos_thumbs_up`	INTEGER	If the rating type is configured to be thumbs up/down, then the rank value recorded here is: 1 if it is thumbs up and `-1` if it is thumbs down.
`cos_rating_type_value`	INTEGER	Type of rating rank. Possible value codes are: `0`: stars `1`: thumbs up/down `2`: like it `3`: recommend
`cos_root_id`	BIGINT	Association to the RatingFeed object representing the page on which ratings are submitted. The value of the corresponding `cg_rating_feed.id` item is recorded here. Use this column to query all the ratings posted on a page.
`cos_state_value`	VARCHAR(450)	State of ratings reflecting the stage of content moderation. If manual moderation is enabled, the value may be `pending.new` (newly posted) or `pending.modified` (manual moderation). If a rating has either passed moderation or was auto-published, the value will be `approved.all`. If a rating has not pass through either automatic filters or moderator, then the value will be `inappropriate.robotdetected` or `inappropriate.humandetected`. To simplify navigation for moderators, the Community interface provides filters for these categories on the right-side panel.
`cos_owner`	BIGINT	ID of the visitor who posted the rating. Or `-1` if it is a guest entry.

Table 81–7 (Cont.) Structure of the cg_rating_record AssetType

Property	Type	Description
`cos_owner_ip`	VARCHAR(256)	Client's IP address from which the rating is posted.
`cos_guest_name`	VARCHAR(450)	Display name of a registered user or a guest who gave the rating. This field is used to display and sort ratings by author names in the Community interface without making extra requests to the database.

81.1.4 Polls

The Polls functionality is represented in a single table in the database. When an administrator creates a poll, the corresponding row is inserted into this table (Table 81–8).

Table 81–8 Structure of the cg_poll AssetType

Property	Type	Description
`id`	BIGINT	Identifier of a poll instance.
`cos_uid`	VARCHAR(256)	Unique string identifier of the poll instance. It is embedded into the deployment code snippet along with the regular ID so that if the integer ID is lost during data migration, the system can look up this poll by UID, thus preventing existing deployments from breaking.
`cos_chart_type`	INTEGER	Type of the chart in which poll results are to be displayed. Possible values are: ■ 0: Pie chart ■ 1: Bar chart ■ 2: Flat Results (percents are shown for options)
`cos_theme`	INTEGER	ID of the theme to be applied to Poll on a web site. Possible values are: ■ 0: Basic ■ 1: Advanced ■ 2: No design ■ 3: Open design
`cos_start_date`	DATE	Date when the poll is started and opened for votes. The start date of the poll voting campaign.

Table 81–8 (Cont.) Structure of the cg_poll AssetType

Property	Type	Description
`cos_finish_date`	DATE	Date when the poll is closed and no more voting is allowed. The end date of poll voting campaign.
`cos_result_required`	VARCHAR(32)	Boolean value (`true` or `false`) that specifies whether to show results after voting.
`cos_results_view`	INTEGER	Determines how poll results are shown: ▪ 0: In a context menu ▪ 1: In place ▪ Inside the poll widget
`cos_results_width`	INTEGER	Widget for poll results context menu in pixels.
`cos_title`	VARCHAR(450)	Title of the poll.
`cos_question`	VARCHAR(450)	Main poll question.
`cos_options`	VARCHAR(4000)	List of poll options to be available for voting in the JSON format. The metadata includes ID of the vote used by Community-Gadgets internally, the option title, color, and the number of votes left. For example: `[{"id":"n0","count":1,"color":"#1751a7","value":"Avatar"},{"id":"n1","count":0,"color":"#8aa717","value":"Matrix"}]`
`cos_thankyou_note`	VARCHAR(450)	Message to be shown to visitors after their vote.
`cos_disclaimer_required`	VARCHAR(32)	Boolean value (`true` or `false`) that specifies whether to show the disclaimer text for poll or not.
`cos_disclaimer`	VARCHAR(450)	Disclaimer text to be shown at the bottom of the Poll widget.
`cos_votes`	INTEGER	Total number of votes on the current poll campaign.

81.1.5 Topics

The Topics functionality allows to gather and precalculate statistics, such as review counts, rating values, and so on, across the Community widgets so that the collected information can be easily and efficiently queried later. The precalculation and optimization of visitors' feedback is important because this type of content may appear on the home page of a site or on the visitor's dashboard that lists most discussed/reviewed/rated articles.

Web pages on which the Community widgets are deployed and activities performed on those pages become the primary focus of the Topics functionality. Therefore, statistics are aggregated across tables such as CommentFeed, ReviewFeed, and RatingFeed.

Table 81–9 Structure of the cg_topic AssetType

Property	Type	Description
`id`	BIGINT	Identifier of a topic instance.
`cos_url`	VARCHAR(450)	URL of the page on which widgets are deployed.
`cos_title`	VARCHAR(450)	HTML title of the page with which this topic is associated.
`cos_resource_id`	VARCHAR(450)	Resource ID of the page that links the UGC content to the page on which widgets are deployed.
`cos_date_created`	DATE	Date when the topic is created. That is, when the page on which widgets are deployed is accessed for the first time using a browser.
`cos_comments_feed_id`	BIGINT	Relation between the corresponding CommentFeed object and the `CommentFeed.id` field. This is a source object from which the statistics are aggregated.
`cos_comment_count`	BIGINT	Number of comments posted on the page with which a particular topic is associated.
`cos_comments_resource_type`	VARCHAR(450)	Content category (such as a blog or article) assigned to the Comments widget deployed on the page. It might be either the default category value or the customized category uploaded on the Appearance settings page.
`cos_comments_date_modified`	DATE	Date when the CommentFeed object is last updated with statistic values (the number of comments posted, and so on).
`cos_reviews_feed_id`	BIGINT	Relation to the ReviewFeed object with which a topic is associated.
`cos_review_count`	BIGINT	Number of reviews posted on a page with which a topic is associated.
`cos_reviews_resource_type`	VARCHAR(450)	Content category (such as a blog or article) assigned to the Reviews widget deployed on a page. It might be either a default category or the customized category uploaded on the Appearance settings page.

Table 81–9 (Cont.) Structure of the cg_topic AssetType

Property	Type	Description
`cos_reviews_date_modified`	DATE	Date when the ReviewFeed object is last updated with statistic values (the number of reviews posted, ranks, and so on).
`cos_ratings_feed_id`	BIGINT	Relation to the RatingFeed object with which a particular topic is associated.
`cos_rating_count`	BIGINT	Number of ratings posted on a page with which this topic is associated.
`cos_ratings_resource_type`	VARCHAR(450)	Content category (such as blog, article, and so on) assigned to the Ratings widget deployed on a page. It might be either a default category or the customized category uploaded on the Appearance settings page.
`cos_ratings_date_modified`	DATE	Date when the RatingFeed object is last updated with statistic values (the number of ratings posted, their ranks, and so on).
`cos_rank`	FLOAT	Rank of the current topic among other topics. The current ranking schema is based on the frequency statistics and is calculated as a sum of the following fields: `cos_review_count` + `cos_comment_count` + `cos_rating_count`.

81.1.6 Visitors

The following types of primary objects are associated with visitors:

- Section 81.1.6.1, "User"
- Section 81.1.6.2, "UserIdentity"
- Section 81.1.6.3, "UserLink"

81.1.6.1 User

The User table represents a visitor profile maintained by Community-Gadgets, and it contains visitor-sensitive data such as display name, email, or avatar picture.

Table 81–10 Structure of the cg_user AssetType

Property	Type	Description
`id`	BIGINT	Identifier of a user profile.

Table 81–10 (Cont.) Structure of the cg_user AssetType

Property	Type	Description
`cos_email`	VARCHAR(450)	Email of a web site visitor. May not be populated if visitors logged in using Facebook, Twitter, Google, or through Janrain. The visitors that registered locally may use it to recover their passwords when needed.
`cos_display_name`	VARCHAR(450)	Display name of a user. Either provided by a visitor during registration or taken from the social networking service using which the user logged in.
`cos_profile_type_code`	INTEGER	Type of the user profile. Possible values are: ■ 0: Regular user or visitor ■ 1: Editorial or management user (for example, content moderator) ■ 2: System user or the user used by the system to open connections internally.
`packed_identities`	VARCHAR(4000)	Denormalized list of visitor identities in the JSON format. These identities are used during authentication, and therefore, they are linked to this profile. For example: [{"username":"john"}] The **username** field is related to a corresponding UserIdentity entry via the `UserIdentity.cos_username` field. For the visitor profile of the user who logged in using a social network service via Janrain, the user name will include the "jr:" prefix: `[{"username":"jr:http://twitter.com/account/profile?user_id=3333333"}]`
`cos_picture_blob`	BINARY	For a profile of a local user who registered locally and did not use a social networking service, the customized avatar is stored in this field.
`cos_picture_url`	VARCHAR(450)	For a profile of a user who used a social networking service to log in, the URL of the avatar picture is stored here.

81.1.6.2 UserIdentity

The two options for storing visitor credentials are LDAP and the WebCenter Sites database. The database option is set by default.

When the database is used, credentials of a user are stored in the UserIdentity table and associated with the corresponding visitor profile using the UserLink table.

Table 81–11 Structure of the cg_user_id AssetType

Property	Type	Description
`id`	BIGINT	Identifier of a user's credentials.
`cos_username`	VARCHAR(256)	User name used for authentication.
`cos_email`	VARCHAR(256)	Email of a user for the purpose of recovering user's password when needed.
`cos_provider_id`	VARCHAR(256)	Type of identity provider used for authenticating a visitor. Possible values are: ■ `none`: if this is the identity of a system user who opened connections internally. ■ `ldap`: if visitor authenticates via LDAP ■ `wem-db`: if visitor authenticates via the Community-Gadgets plug-in that uses a WebCenter Sites assetTypes.
`cos_encryption_type`	VARCHAR(256)	Encryption algorithm applied to secure the password of a user identity. The algorithm used in the current version of the system is `SHA-256`.
`cos_password`	VARCHAR(256)	Encrypted value of a password used for authentication.

81.1.6.3 UserLink

Table 81–12 links UserIdentity to User so that a visitor's identities (that have credentials for authentication) are properly associated with the visitor's user profile. This is a denormalized table so that some of the fields from the UserIdentity table are cached here as well.

Table 81–12 Structure of the cg_user_link AssetType

Property	Type	Description
`id`	BIGINT	Identifier of a user link.
`cos_username`	VARCHAR(256)	User name from the UserIdentity object with which this link associates the User object.

Table 81–12 (Cont.) Structure of the cg_user_link AssetType

Property	Type	Description
cos_email	VARCHAR(256)	Email from the UserIdentity object with which this link associates the User object.
cos_provider_id	VARCHAR(256)	Identity provider code from the UserIdentity object with which this link associates the User object. Possible values are: ■ ext_auth: if visitor comes from social networks like Facebook or Twitter, or through Janrain. ■ none: if this is the identity of a system user who opens connections internally ■ ldap: if visitor authenticates via LDAP ■ wem-db: if visitor authenticates via the Community-Gadgets plug-in which uses WebCenter Sites assets.
cos_account_id	BIGINT	Identifier of a User object with which this link associates the UserIdentity object.

81.1.7 Gadgets and Dashboard

The following types of primary objects are associated with gadgets and dashboard fuctionality:

- Section 81.1.7.1, "Gadget"
- Section 81.1.7.2, "OpenSocialDescriptor"
- Section 81.1.7.3, "SingleGadgetData"
- Section 81.1.7.4, "GadgetSet"

81.1.7.1 Gadget

The Gadget table represents a gadgets user settings maintained by Community-Gadgets.

Table 81–13 Structure of the cg_gadget AssetType

Property	Type	Description
id	BIGINT	Identifier of a gadget.
original_gadget_id	BIGINT	Relation to the Gadget object from which a particular gadget is inherited.
title	VARCHAR(256)	Title of a gadget.
gadget_owner_id	BIGINT	Relation to the User object with which a particular Gadget is associated.

Table 81–13 (Cont.) Structure of the cg_gadget AssetType

Property	Type	Description
packed_type	VARCHAR(256)	Type of a Gadget object. Possible values: ■ user_dash - a gadget linked to dashboard by end side visitor. ■ site_dash - a gadget linked to dashboard by admin in the Gadgets interface. ■ site_registry - a gadget object added to site Gadget catalog. ■ global_registry - a gadget object added to global Gadget catalog.
opensocial_descriptor_url	VARCHAR(4000)	URL to OpenSocial XML descriptor from which gadget is registered in the Gadgets interface.
gadgetset_id	BIGINT	Relation to the GadgetSet object with which a particular Gadget is associated.
packed_attributes	VARCHAR(4000)	Community-Gadgets specific gadget attributes of a particular Gadget object in the JSON format.
packed_categories	VARCHAR(4000)	List of space separated categories that assigned to a particular Gadget object.
packed_preferences	VARCHAR(4000)	Settings of a particular Gadget object in the JSON format.
gadget_iconid	BIGINT	Relation to the Blob object in which gadget icon is stored.
gadget_previewid	BIGINT	Relation to the Blob object in which gadget preview is stored.
gadget_thumbnailid	BIGINT	Relation to the Blob object in which gadget thumbnail is stored.

81.1.7.2 OpenSocialDescriptor

The OpenSocialDescriptor table represents OpenSocial gadgets XML descriptor which is used for the gadget registration in the Gadgets interface.

Table 81–14 Structure of the cg_open_social_descriptor AssetType

Property	Type	Description
id	BIGINT	Identifier of a OpenSocialDescriptor.
source_gadget_id	BIGINT	Relation to the Gadget object with which a particular OpenSocial gadget descriptor is associated.
site_id	BIGINT	Site name on which descriptor is created.
gadget_links	VARCHAR(4000)	De-normalized list of links to Gadget objects with a particular OpenSocial gadget descriptor in the JSON format. For example:[{"siteId":"FirstSiteII", "gadgetId":1337101669171, "siteRegistry":false}, ...]
packed_info	VARCHAR(4000)	De-normalized list of gadget descriptor settings in the JSON format. The settings are read from gadget XML descriptor.

81.1.7.3 SingleGadgetData

The SingleGadgetData table represents authorized user settings for separately deployed gadget.

Table 81–15 Structure of the cg_single_gadget_data AssetType

Property	Type	Description
`Id`	BIGINT	Identifier of a `SingleGadgetData`.
`gadget_id`	BIGINT	Relation to the Gadget object with which a particular data is associated.
`user_id`	BIGINT	Relation to the `User` object with which a particular data is associated.
`resource_id`	BIGINT	Identifier of the gadget. It is a logical link that Community-Gadgets uses to associate a deployed gadget with the page on which visitors posted comments. This link is established during widget deployment by specifying the value of the Resource ID field on the deployment page or the `resource_id` attribute in the widget code snippet. It is recommended that you come up with your own strategy of `resource_id` generation. For example, to make it easier to find this object in the database, you can use the page ID as the resource ID.
`data_value`	VARCHAR(4000)	De-normalized list of gadget descriptor settings in the JSON format. The settings are read from gadget XML descriptor.

81.1.7.4 GadgetSet

The `GadgetSet` table represents dashboard settings.

Table 81–16 Dashboard Properties

Property	Type	Description
`id`	BIGINT	Identifier of a `GadgetSet`.
`user_id`	BIGINT	Relation to the User object with which a particular `GadgetSet` is associated.
`packed_type`	VARCHAR(256)	Type of a GadgetSet Possible values: ■ `user_dash` - a end side user dashboard ■ `site_dash` - a admin side user dashboard
`packed_ui_settings`	VARCHAR(4000)	Dashboard interface settings that are stored as the URL parameter.

81.2 Customizing CSS and Widget Templates

You can customize Community widgets in the following ways:

- **Color Schema and Skinning via CSS**. The CSS is customized when a widget's look-and-feel must match that of the web site on which the widget is deployed.
- **Redesign via Widget Templates.** Widget templates are customized when a widget requires significant changes, such as a new layout or enhancements to functionality.

The following sections describe how to perform these customizations:

- Section 81.2.1, "Customizing CSS: Color Schema and Skinning"
- Section 81.2.2, "Customizing a Widget Template"

81.2.1 Customizing CSS: Color Schema and Skinning

The general steps for customizing the CSS are:

1. Download the standard CSS skin of the widget.
2. Modify CSS styles for the interface elements that require customization.
3. Apply the customized CSS.

While the overall process is the same, customizing CSS styles for Comments and Reviews widgets is different from customizing other widgets.

This section includes the following:

- Section 81.2.1.1, "Customizing Comments and Reviews Widgets"
- Section 81.2.1.2, "Customizing Other Widgets"

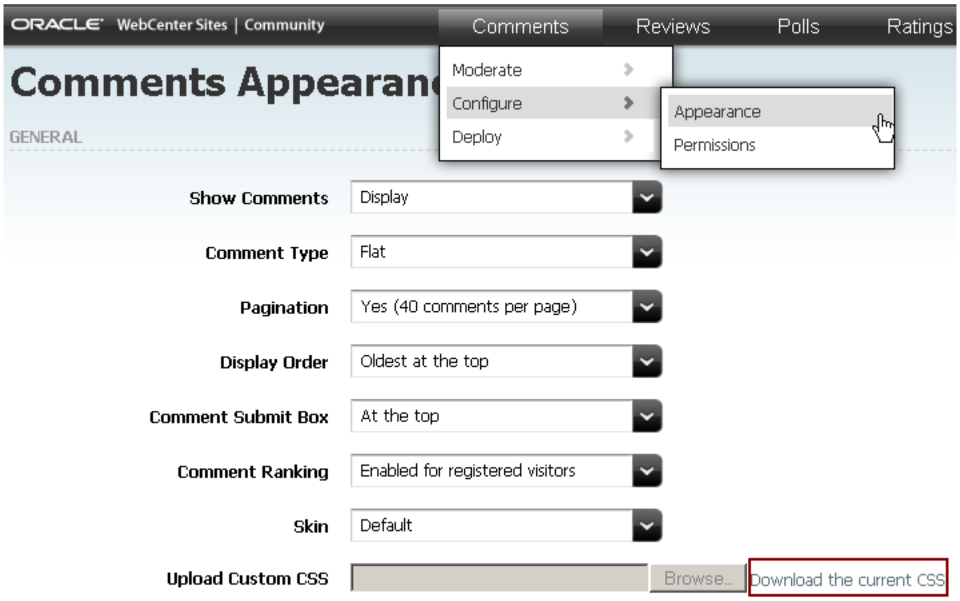
81.2.1.1 Customizing Comments and Reviews Widgets

For Comments and Reviews, you can download the default CSS from the widget's Appearance page in the Community interface, modify it, and then upload it back. Community-Gadgets continues to host the CSS while it is being customized, so you can change it anytime you like.

To customize the Comments CSS:

1. Log in to the Community interface as an administrator.
2. From the Comments menu, choose **Configure**, then **Appearance** to display the Comments Appearance page.
3. In the General section, click **Download the current CSS** next to the **Upload custom CSS** field.

Figure 81–2 Download the Current CSS



Comments Appearance

Show Comments: Display

Comment Type: Flat

Pagination: Yes (40 comments per page)

Display Order: Oldest at the top

Comment Submit Box: At the top

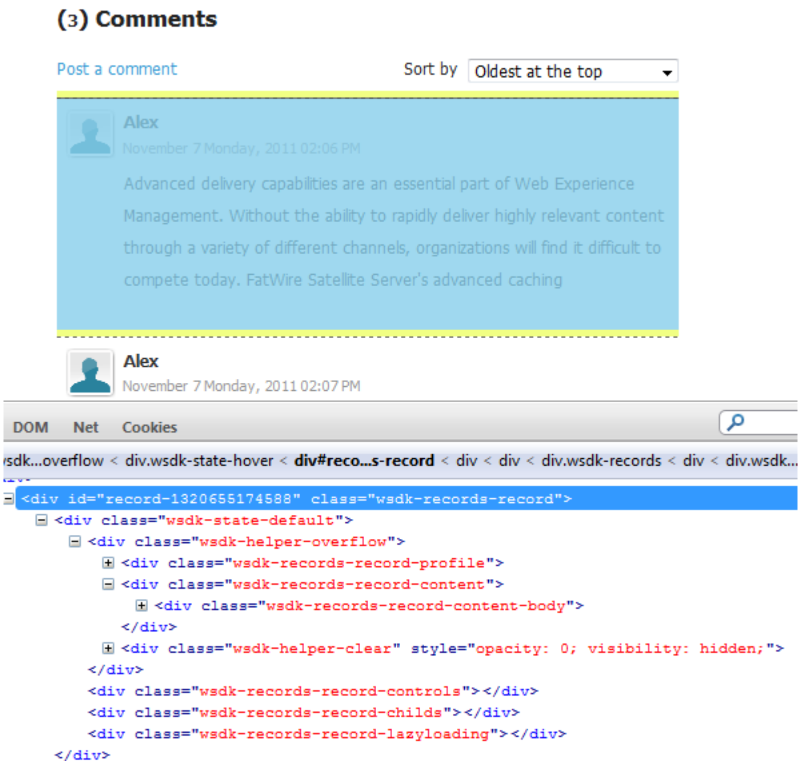
Comment Ranking: Enabled for registered visitors

Skin: Default

Upload Custom CSS

4. Open the downloaded CSS in FireFox for editing.
5. From the View menu, choose **Firebug**.
6. From the FireBug menu, choose **Inspect Element**. The CSS should look like the CSS in Figure 81–3.

Figure 81–3 CSS Opened in FireBug



Alex

November 7 Monday, 2011 02:06 PM

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content through a variety of different channels, organizations will find it difficult to compete today. FatWire Satellite Server's advanced caching

Alex

November 7 Monday, 2011 02:07 PM

sdk...overflow < div.wsdk-state-hover < **div#reco...s-record** < div < div < div.wsdk-records < div < div.wsdk...

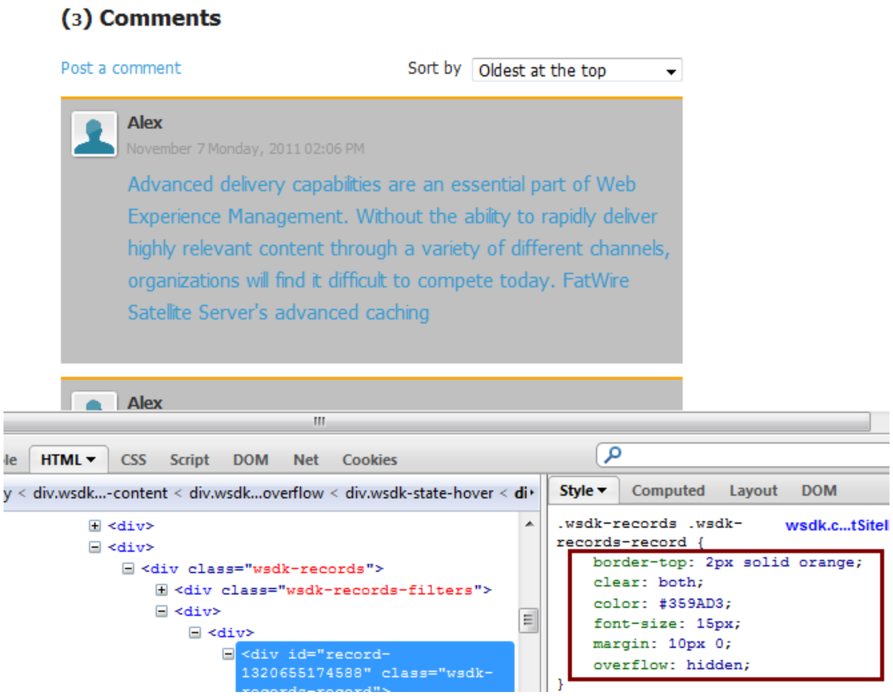
```
<div id="record-1320655174588" class="wsdk-records-record">
  <div class="wsdk-state-default">
    <div class="wsdk-helper-overflow">
      <div class="wsdk-records-record-profile">
      <div class="wsdk-records-record-content">
        <div class="wsdk-records-record-content-body">
      </div>
      <div class="wsdk-helper-clear" style="opacity: 0; visibility: hidden;">
    </div>
    <div class="wsdk-records-record-controls"></div>
    <div class="wsdk-records-record-childs"></div>
    <div class="wsdk-records-record-lazyloading"></div>
  </div>
```

7. The DIV element, which contains the entire Comments code, includes the `wsdk-records-record` class. Search for this class in the default CSS previously downloaded by finding the following:

```
.wsdk-records .wsdk-records-record
{
border-top: 1px dashed #666;
clear: both;
overflow: hidden;
margin: 5px 0px;
}
Now, let's apply some customizations, for example change the default text
 color and size, and border:
.wsdk-records .wsdk-records-record
{
border-top: 2px solid orange;
color: #359AD3;
font-size:15px;
clear: both;
overflow: hidden;
margin: 10px 0px;
}
We can also find that background color is defined in the next div inside this
 container div, marked with CSS class "wsdk-state-default". Let's also define
 a custom background color:
.wsdk-state-default
{
background: silver;
}
```

8. Customize the `wsdk-records-record` class as required (Figure 81–4), then save your changes.

Figure 81–4 Modified CSS



(3) Comments

Post a comment

Sort by Oldest at the top

Alex

November 7 Monday, 2011 02:06 PM

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content through a variety of different channels, organizations will find it difficult to compete today. FatWire Satellite Server's advanced caching

Alex

le
HTML
CSS
Script
DOM
Net
Cookies
y < div.wsdk...-content < div.wsdk...overflow < div.wsdk-state-hover < di
Style
Computed
Layout
DOM
wsdk.c...tSitel

```
⊞ <div>
⊟ <div>
  ⊟ <div class="wsdk-records">
    ⊞ <div class="wsdk-records-filters">
    ⊟ <div>
      ⊟ <div>
        ⊟ <div id="record-
          1320655174588" class="wsdk-
```

```
.wsdk-records .wsdk-
records-record {
    border-top: 2px solid orange;
    clear: both;
    color: #359AD3;
    font-size: 15px;
    margin: 10px 0;
    overflow: hidden;
}
```

9. To upload the customized CSS, go back to the Community interface.
10. From the **Comments** menu, choose **Configure**, then **Appearance** to display the Comments Appearance page.
11. In the General section, change Skin from Default to Custom, click **Browse** next to the **Upload Custom CSS** field, and select the CSS file you just customized.
12. Click **Save**.
13. Refresh the web page to refresh the widget, and verify if the changes you made to the CSS reflect in the widget interface.

To customize the CSS for the Reviews widget, go to **Reviews**, select **Configure**, and then **Appearance**. Then follow the procedure described above (from step 3 through step 12).

81.2.1.2 Customizing Other Widgets

For customizing widgets other than Comments and Reviews, you can use the general approach described in this section. Depending on your use cases, you can even customize Comments and Reviews widgets using the approach explained here.

The main difference between the approach that will be discussed in this section and what is described for Comments and Reviews widgets is that the customized CSS for other widgets is no longer hosted on the Community-Gadgets web application. Typically, the customized CSS is hosted on the web site on which comments are deployed.

Customize the Top Ranked Topics widget, which is an add-on to the Reviews functionality y following these steps:

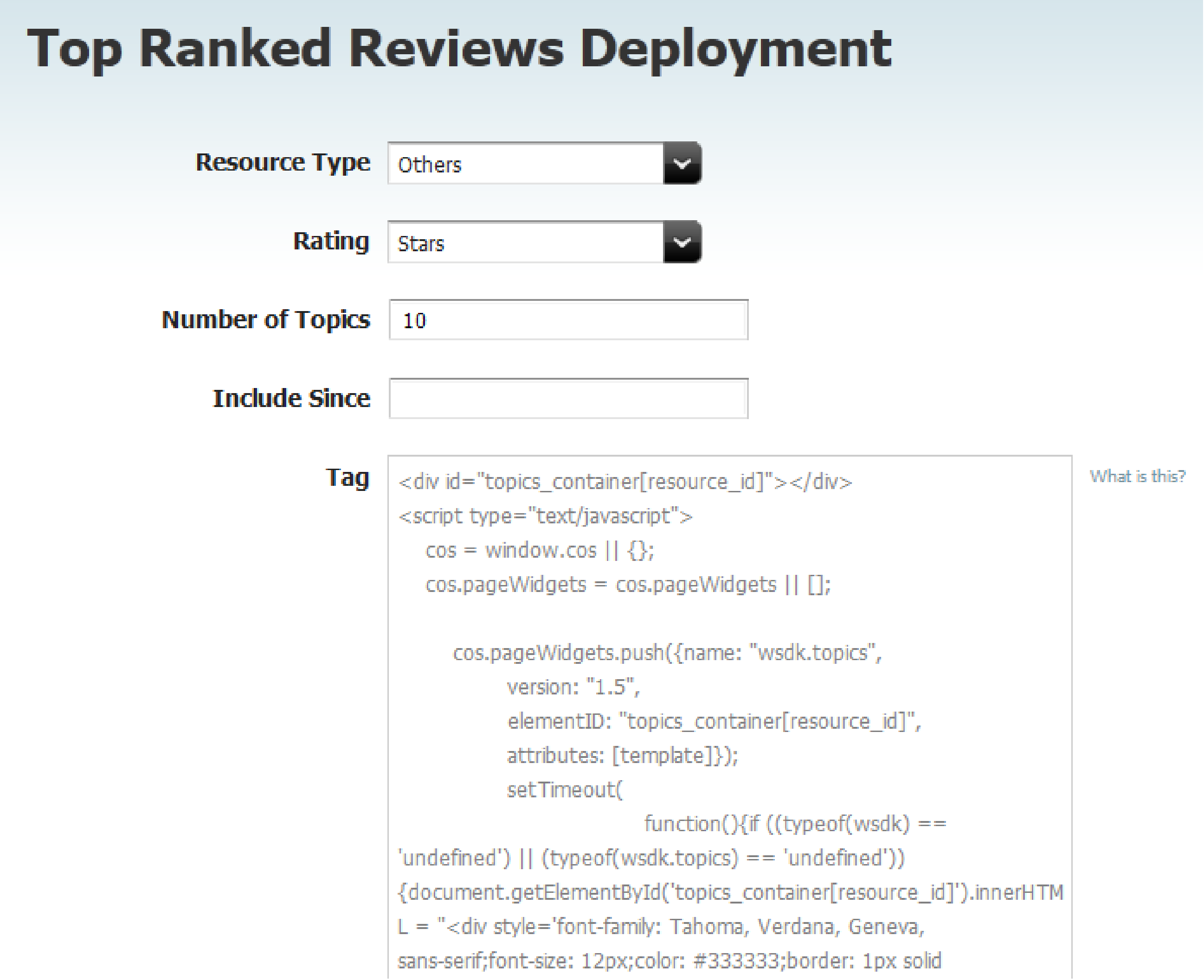
- Copy the Deployment and CSS Tags Into the Page Template
- Customize the CSS

Copy the Deployment and CSS Tags Into the Page Template

1. Log in to the Community-Gadgets as an administrator.
2. From the Reviews menu, choose **Deploy**, then **Top Ranked Topics**.

   The Top Ranked Reviews Deployment form is displayed(Figure 81–5).

Figure 81–5 Top Ranked Reviews Deployment



Top Ranked Reviews Deployment

Resource Type: Others

Rating: Stars

Number of Topics: 10

Include Since:

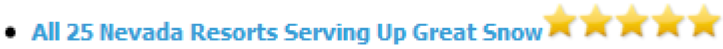
Tag:

```
<div id="topics_container[resource_id]"></div>
<script type="text/javascript">
   cos = window.cos || {};
   cos.pageWidgets = cos.pageWidgets || [];

      cos.pageWidgets.push({name: "wsdk.topics",
            version: "1.5",
            elementID: "topics_container[resource_id]",
            attributes: [template]});
            setTimeout(
                           function(){if ((typeof(wsdk) ==
'undefined') || (typeof(wsdk.topics) == 'undefined'))
{document.getElementById('topics_container[resource_id]').innerHTM
L = "<div style='font-family: Tahoma, Verdana, Geneva,
sans-serif;font-size: 12px;color: #333333;border: 1px solid
```

3. Enter the necessary information for the **Resource Type**, **Rating**, **Number of Topics**, and **Include Since** fields. When the field for **Include Since** is selected, a calendar displays to select the initial date.
4. Copy and paste into the web page template the contents of both **Tag** and **CSS Tag** fields. The CSS will update based on the information added to the fields above. Insert the contents of the **Tag** field into the `<body>` section of the template. Insert the contents of the **CSS Tag** field into the `<head>` section of page template.
5. Deploy the code snippets.

   After the page is rendered, the default look-and-feel of the top ranked topics will be similar to Figure 81–6.

Figure 81–6 Top Ranked Topic After the Tags are Placed in the Template



- All 25 Nevada Resorts Serving Up Great Snow ★★★★★

Customize the CSS

First, take a look at the ID and HREF attributes in the contents of the **CSS Tag** field that you just deployed. The value of the ID attribute must remain unaffected from customization. The HREF value will change as you will learn in the steps described in this section.

```
<link
id="cos_css"
type="text/css"
rel="stylesheet" href="http://localhost:8280/cg/wsdk/skin/wsdk.topics.css?site_
id=FirstSiteII&gateway=true"
/>
```

Before you customize the CSS, download its contents by accessing the HREF location via an Internet browser.

1. In the downloaded CSS, locate CSS classes for topic links. Examine the structure of the widget.
2. To customize the topic headline and topic link colors, locate the respective code snippets. Search for `.fw_topics .headline` and `.fw_topics .topic`.
3. Modify the following code based on web site design specifications:

   ```
   .fw_topics .headline h1 {
   color: #359AD3;
   font-size: 16px;
   }
   .fw_topics .topic {
   font-size: 14px;
   padding: 5px;
   font-weight: bold;
   text-decoration: none;
   }
   .fw_topics .topic a {
       color: orange;
       font-size: 12px;
       text-decoration: none;
   }
   ```

4. To apply the CSS to the web site:
   a. Copy it to the folder in which web site styles are stored.
   b. In the CSS you copied to the page template earlier, replace the default location with the new URL from which the customized CSS is accessible. For example, if you copied it to the `skins` folder, then the HREF element should look like this:

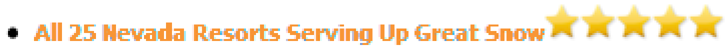
   ```
   <link
   id="cos_css"
   type="text/css"
   rel="stylesheet" href="http://localhost:8180/cs/skins/wsdk.topics.css"
   />
   ```

Note: The ID attribute must remain equal to `cos_css` because when the Community-Gadgets web application loads any widget, it searches on the page for the `<link>` element with the `cos_css` ID. If this ID exists, the application continues to render the widget. If this ID does not exist, the application makes an explicit request for the default CSS skin and applies it to the page. Therefore, using `cos_css` as the ID attribute value prevents Community-Gadgets from overwriting the applied customization.

---

5. Reload the page which uses the widget whose CSS you just modified, and verify that your customizations are applied (Figure 81–7).

Figure 81–7 Customization Applied to a Widget



- All 25 Nevada Resorts Serving Up Great Snow ★★★★★

Tip: When deploying Community widgets on the same page, optimize the number of network calls by loading all of the widgets' CSS files in a single network call. Community-Gadgets supports this option by allowing you to specify a colon-separated list of widget names in the `<link/>` tag. That is, widget names do not need to be in separate `<link/>` elements. For example:

```
<link
id="cos_css"
type="text/css"
rel="stylesheet" href=" http://localhost:8280/cg/wsdk/skin/<widget
name #1>:<widget name #2>:<widget name #3>.css?site_
id=FirstSiteII&gateway=true "
/>
```

Examples of widget names are `wsdk.topics` and `comments-summary`. You can find out the actual values for a particular widget in the **CSS Tag** field of the corresponding deployment page.

81.2.2 Customizing a Widget Template

In the case of dramatic changes (such as changing the position of action links from bottom to top in the Comments widget), widget templates can be customized according to project requirements.

Widget templates are rendered dynamically using JavaScript on the browser side. These templates are based on the Google Closure Templates technology.

There is a set of template directives that can be used in customizations. For information, visit the Google documentation web site at:
`http://code.google.com/closure/templates/docs/commands.html`

Ensure that you have read and understood the `print`, `if/else`, `for`, and `foreach` statements in Google documentation before you apply the information discussed in this section.

This section includes the following:

- Section 81.2.2.1, "Understanding Community Widgets Templates"

- Section 81.2.2.2, "Creating a Sample Template"
- Section 81.2.2.3, "Loading Custom Data Sets"

81.2.2.1 Understanding Community Widgets Templates

This section describes widget template technology and syntax. It explains how you can override attach points by translating those declared in the Community-Gadgets templates into corresponding WebCenter Sites template names. This section provides information about attach points available for customization in Community-Gadgets, and it explains how to navigate to the widget structure and locate the necessary attach points when customizing a template.

This section includes the following:

- Section 81.2.2.1.1, "Context Variable Access Points"
- Section 81.2.2.1.2, "Dynamic Scripting"
- Section 81.2.2.1.3, "Widget Sources and Templates"
- Section 81.2.2.1.4, "Model-View-Controller Pattern"
- Section 81.2.2.1.5, "Model-View-Controller Regions"
- Section 81.2.2.1.6, "Nested Templates"
- Section 81.2.2.1.7, "Customization Workflow"
- Section 81.2.2.1.8, "Attach Points in the Widget Template Structure"

81.2.2.1.1 Context Variable Access Points To modify templates, developers need to work with visitors' permissions (configured in the Community interface) and variables. Each Community-Gadgets template includes a standard `$stack` variable that you can use to discover these context variables and permissions. For example, you can use the following tag to insert localizable resources into a template:

```
<h1>Welcome to {$stack.resources.get('label.comments')} Widget </h1>
```

The `$stack.resources` variable is an access point to all the resources declared in `cg.war/WEB-INF/classes/i18n_resources/widgets/cos.commons` and `cg.war/WEB-INF/classes/i18n_resources/widgets/<widget name>`.

81.2.2.1.2 Dynamic Scripting You can build the dynamic logic in the template by using JavaScript's `{script}` directive:

```
{script}
var lastIndex = 10;
{/script}
```

Then, variables defined inside `{script}` are available in other closure directives with `$$` mark; for example, `$$lastIndex`:

```
{for $index in range($$lastIndex)}
For loop: Iteration #{$index + 1} of {$$lastIndex} in total
<br />
{/for}
```

In a scenario when the Google closure template variable needs to be accessed in the `{script}` tag, the regular `{$var}` syntax can be used:

```
{for $index in range(2)}
   {script}
          alert({$index});
```

```
    {/script}
{/for}
```

The data produced inside `{script}` can be displayed on a web page in the following ways:

- By closing the `{script}` tag and printing the data previously declared using the `{$$variable}}` syntax.
- By using the system output variable provided by Community-Gadgets to print results immediately from JavaScript using `output.append(variable)` inside the `{script}` statement:

  ```
  {script}
         var message= "Hello World!";
         output.append(message);
  {/script}
  ```

81.2.2.1.3 Widget Sources and Templates The list of widget sources shipped with the Community-Gadgets web application is located in the `cg.war/js/widgets` directory. For example, the Comments-Gadgets widget's code is stored in the `cg.war/js/widgets/wsdk.comments` directory. Each widget directory contains a `.shtml` file, the widget layout definition, and the main entry point to the process of rendering the widget interface. For example, the `comments_layout_view.shtml` file is the main entry point for the Comments widget.

81.2.2.1.4 Model-View-Controller Pattern To build a widget interface, the Community-Gadgets uses the Model-View-Controller (MVC) pattern. Therefore, each interface region includes three items corresponding to Model, View, and Controller. For example:

```
comments_layout_action.js (Controller)
comments_layout_model.js (Model)
comments_layout_view.shtml (View)
```

81.2.2.1.5 Model-View-Controller Regions To build a widget interface, Community-Gadgets works with MVC definitions instead of template (`.shtml`) definitions.

For example, the Comments layout's MVC region can be identified by the `/comments_layout` ID in the `cg.war/js/widgets/<WidgetName>/...` directory. The `"/"` special character signifies a relative path in the widget folder. The `comments_layout` MVC prefix is used with suffixes such as `_view.shtml` (`view` contains HTML pages with bindings) for viewing and discovering other corresponding MVC entities. So, the MVC definition of a particular interface region is identified by its relative path in its widget folder and the prefix of MVC entities that are used to discover a particular model, view, or controller.

Understanding such MVC IDs is essential in building widgets interfaces because these IDs are heavily used across templates.

81.2.2.1.6 Nested Templates When an MVC entity is processed and its template is rendered, another MVC entity can be recursively included in the template so that complex interfaces can be built by aggregating smaller interface pieces.

To enable nesting of templates, you can add the following tag to a template in which you want to nest other templates. For example, nesting the Login Bar widget in the Reviews widget.

```
<div attachPoint="/sample"></div>
```

When you include this tag in your template, Community-Gadgets searches for the `sample_action.js` and `sample_view.shtml` files in the widget's root folder: `cg.war/js/widgets/<WidgetName>/`.

In addition to the `attachPoint=""` syntax, the `bindMvc=""` directive also includes nested templates, for example `bindMvc="wsdk.ui.input"`. However, this directive calls the templates that are located outside of a particular widget folder hierarchy, and therefore, does not include templates from the current widget folder hierarchy. This enables the reuse of components across multiple widgets in Community-Gadgets.

81.2.2.1.7 Customization Workflow The previous section discussed how you can use attach points to nest templates. This section describes how to customize or override attach points via templates.

When Community-Gadgets compiles a JavaScript bundle for a widget, it downloads from WebCenter Sites any widget template customizations that are defined as a WebCenter Sites template, compiles them into JavaScript, and applies them to the widget. A contract for template names, which allows to make an association between Community-Gadgets and WebCenter Sites, is specified at the following location: `http://{host}:{port}/cs/ContentServer?pagename={site name}/cg/{widget}/{attach point}_view.shtml&ft_ss=true`. It also enables Community-Gadgets to discover specific templates.

To customize a template, for example the `/sample` template, you must first create the WebCenter Sites template asset on the production WebCenter Sites system on which the Community-Gadgets is configured.

To translate an attach point into a WebCenter Sites template name, use the following format:

```
http://{host}:{port}/cs/ContentServer
?pagename={site name}/cg/{widget}/{attach point}_view.shtml
&ft_ss=true
```

For instance, in one of the templates of the "hello-world" widget, the `attachPoint="/sample"` attach point is translated into the following HTTP request to WebCenter Sites:

```
http://{host}:{port}/cs/ContentServer
?pagename={site name}/cg/hello-world/sample_view.shtml
&ft_ss=true
```

For a custom Community-Gadgets template, you must create a corresponding template in WebCenter Sites by following the WebCenter Sites naming convention: "`cos/hello-world/sample_view.shtml`".

The `bindMvc` is a set of pre-defined attach points that can be potentially used for nesting templates. Table 81–3 lists the templates that you can nest using the `bindMvc=""` attribute syntax in the template you just created. You can find the default templates in the corresponding subfolders in the `cg.war/js/widgets.commons` directory in which reusable widget code is stored.

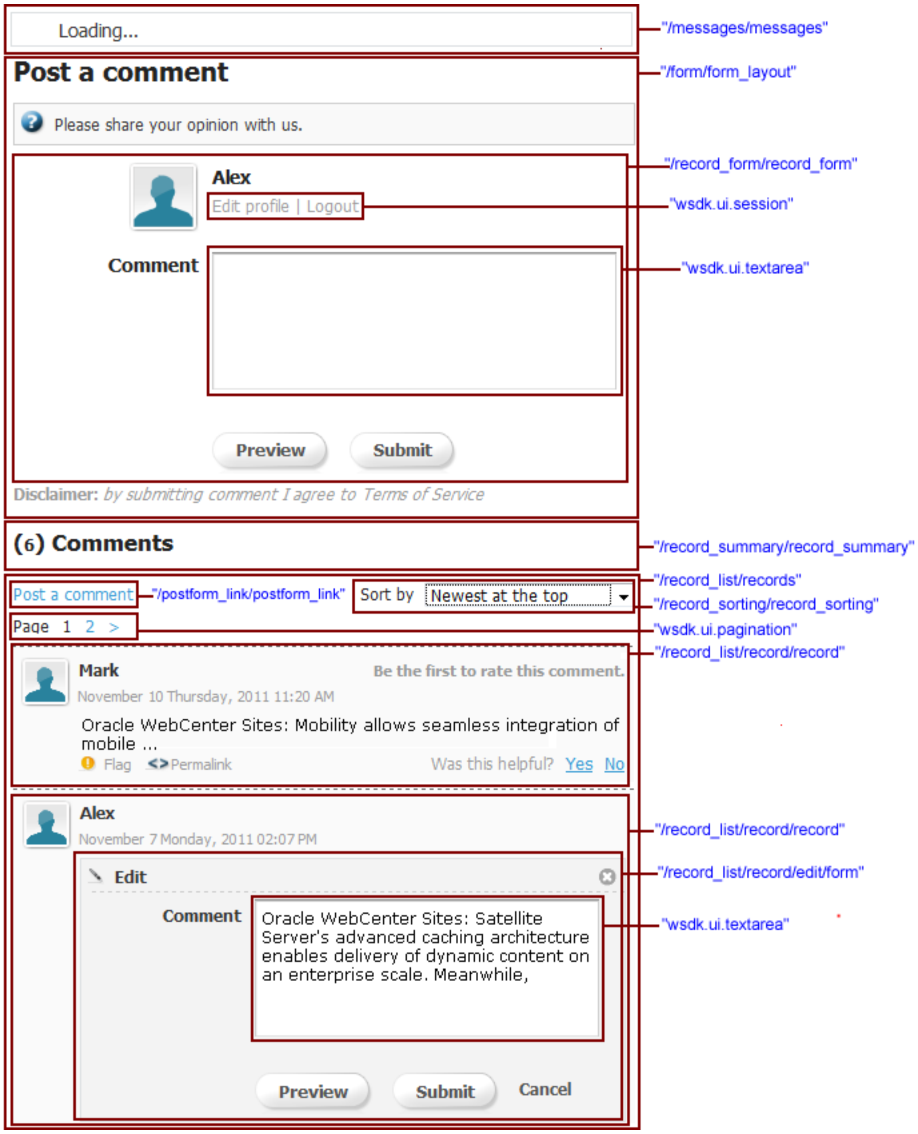
To override a component, use the mapping (Table 81–17) for binding the MVC names to the names of templates that you will create in WebCenter Sites.

Table 81–17 Mapping for Binding MVC Names To Template Names

Bound MVC	Template Name to Override
`wsdk.ui.stars`	`cg/wsdk.ui.stars/stars_view.shtml`
`wsdk.ui.thumbs`	`cg/wsdk.ui.thumbs/thumbs_view.shtml`
`wsdk.ui.input`	`cg/wsdk.ui.input/input_view.shtml`
`wsdk.ui.textarea`	`cg/wsdk.ui.textarea/textarea_view.shtml`
`wsdk.ui.pagination`	`cg/wsdk.ui.pagination/pagination_view.shtml`
`wsdk.ui.session`	`cg/wsdk.ui.session/session_view.shtml`

81.2.2.1.8 Attach Points in the Widget Template Structure The interfaces and functionality of Community widgets are built with the help of templates structured in a certain way. Figure 81–8 shows the structure of the Comments widget. You can treat this structure as a navigation map while you are working with customizations.

Figure 81–8 Structure of the Comments Widget



Loading...

Post a comment

Please share your opinion with us.

Alex

Edit profile | Logout

Comment

Preview Submit

Disclaimer: *by submitting comment I agree to Terms of Service*

(6) Comments

Post a comment

Sort by Newest at the top

Page 1 2 >

Mark

Be the first to rate this comment.

November 10 Thursday, 2011 11:20 AM

Oracle WebCenter Sites: Mobility allows seamless integration of mobile ...

Flag <>Permalink Was this helpful? Yes No

Alex

November 7 Monday, 2011 02:07 PM

Edit

Comment: Oracle WebCenter Sites: Satellite Server's advanced caching architecture enables delivery of dynamic content on an enterprise scale. Meanwhile,

Preview Submit Cancel

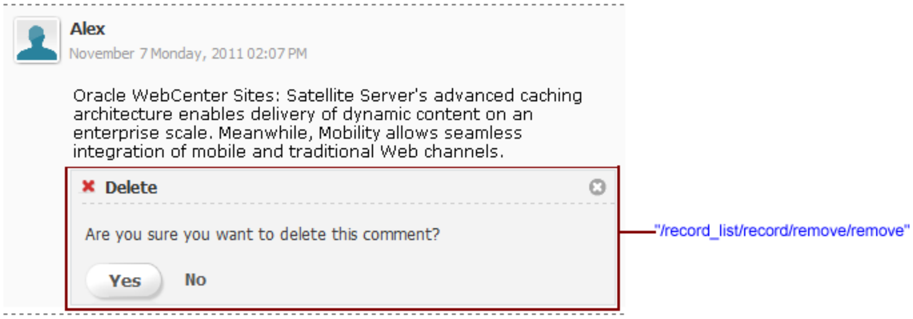
"/messages/messages"
"/form/form_layout"
"/record_form/record_form"
"wsdk.ui.session"
"wsdk.ui.textarea"
"/record_summary/record_summary"
"/postform_link/postform_link"
"/record_list/records"
"/record_sorting/record_sorting"
"wsdk.ui.pagination"
"/record_list/record/record"
"/record_list/record/record"
"/record_list/record/edit/form"
"wsdk.ui.textarea"

The attach points that are invoked and rendered dynamically by following a visitor's action are not referenced directly in templates. For example, the attach points that are invoked at the time when a visitor clicks **edit** or **delete** links on a comment. The dialog boxes for these events cannot be rendered during the initial widget load phase, and therefore, they are loaded programmatically later.

Some examples of attach points for such dynamic dialog boxes are:

- The Delete dialog box (Figure 81–9) is displayed when a visitor clicks the **delete** link on a comment.

Figure 81–9 Delete Dialog Box



Alex

November 7 Monday, 2011 02:07 PM

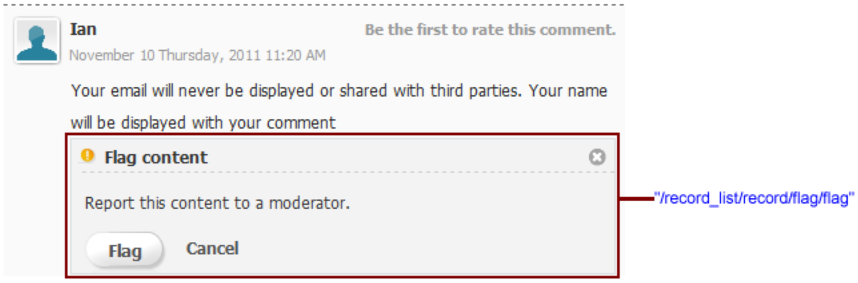
Oracle WebCenter Sites: Satellite Server's advanced caching architecture enables delivery of dynamic content on an enterprise scale. Meanwhile, Mobility allows seamless integration of mobile and traditional Web channels.

Delete

Are you sure you want to delete this comment?

- The Flag Content dialog box (Figure 81–10) is displayed when a comment is flagged.

Figure 81–10 Flag Content Dialog Box



Ian

Be the first to rate this comment.

November 10 Thursday, 2011 11:20 AM

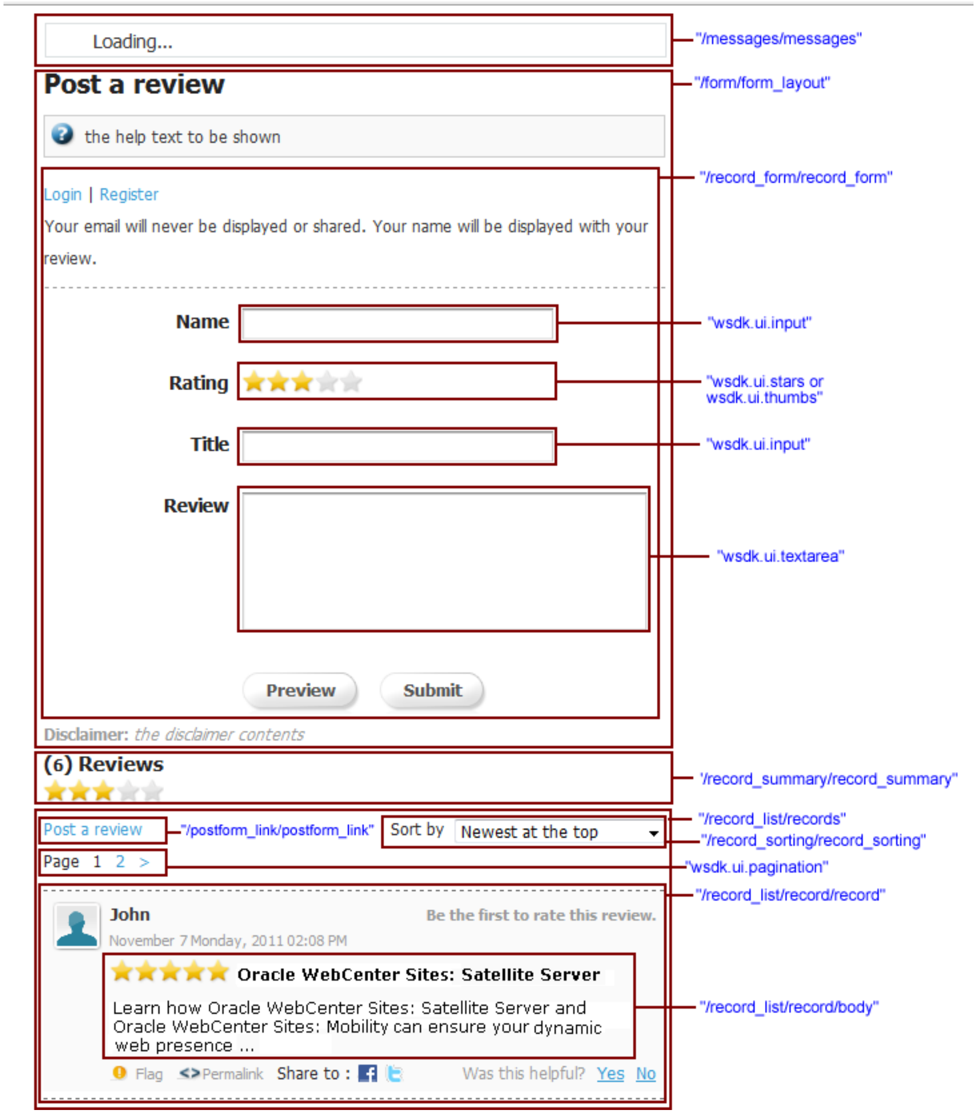
Your email will never be displayed or shared with third parties. Your name will be displayed with your comment

Flag content

Report this content to a moderator.

For the Reviews widget, the templates and their attach points are structured as shown in Figure 81–11. The attach points for flagging, editing, and removing a review are the same as for comments: `/record_list/record/flag/flag`, `/record_list/record/edit/form`, and `/record_list/record/remove/remove` respectively.

Figure 81–11 Post a Review Page



Loading...

Post a review

the help text to be shown

Login | Register

Your email will never be displayed or shared. Your name will be displayed with your review.

Disclaimer: *the disclaimer contents*

(6) Reviews

Sort by Newest at the top

Page 1 2 >

John

Be the first to rate this review.

November 7 Monday, 2011 02:08 PM

Oracle WebCenter Sites: Satellite Server

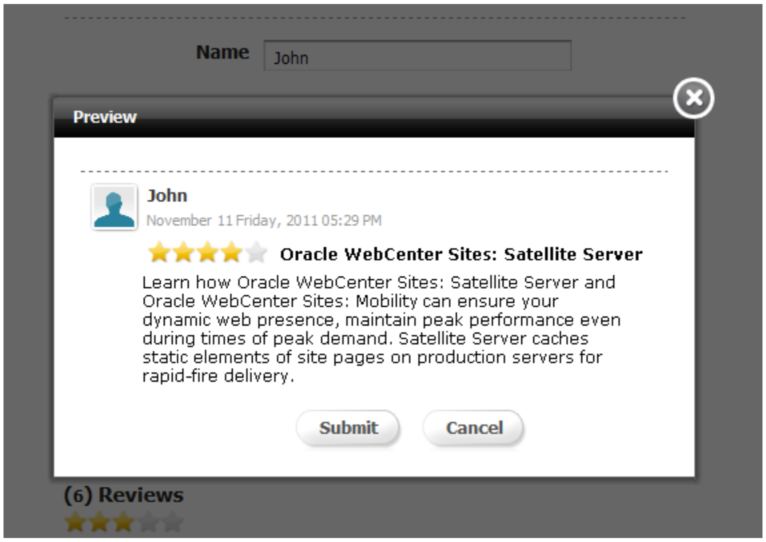
Learn how Oracle WebCenter Sites: Satellite Server and Oracle WebCenter Sites: Mobility can ensure your dynamic web presence ...

Was this helpful? Yes No

"/messages/messages"
"/form/form_layout"
"/record_form/record_form"
Name
"wsdk.ui.input"
Rating
"wsdk.ui.stars or wsdk.ui.thumbs"
Title
"wsdk.ui.input"
Review
"wsdk.ui.textarea"
Preview
Submit
"/record_summary/record_summary"
Post a review
"/postform_link/postform_link"
"/record_list/records"
"/record_sorting/record_sorting"
"wsdk.ui.pagination"
"/record_list/record/record"
"/record_list/record/body"
Flag
<>Permalink
Share to :

Forms for posting comments and reviews contain the **Preview** button. Clicking this button displays a context menu with the rendered comment or preview (Figure 81–12).

Figure 81–12 Preview Pop-Up Dialog Box



John
November 11 Friday, 2011 05:29 PM
Oracle WebCenter Sites: Satellite Server
Learn how Oracle WebCenter Sites: Satellite Server and Oracle WebCenter Sites: Mobility can ensure your dynamic web presence, maintain peak performance even during times of peak demand. Satellite Server caches static elements of site pages on production servers for rapid-fire delivery.

For both, comments and reviews, the `/record_list/record/preview/preview` attach point creates this pop-up functionality.

> **Tip:** The Community-Gadgets web application loads customized templates from the production WebCenter Sites instance. However, the production instance may not have editorial interface to manage these templates. Therefore, it is recommended to start with the development environment first, and point production and management Community-Gadgets server instances to the single instance of WebCenter Sites that has the editorial interface. Using this approach, you can compose templates and verify their look-and-feel in Community-Gadgets. Once you have created and tested your templates, you can either export/import them from development environment to production or publish your custom templates to the desired location.

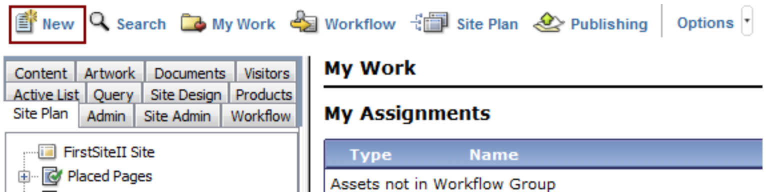
81.2.2.2 Creating a Sample Template

To create a sample template using the WebCenter Sites Advanced interface:

1. Stop all Community-Gadgets servers, including those on production and management environments.
2. Start the production WebCenter Sites instance.

   In the following steps, it is assumed that this server is running on the local host at port 8080.
3. Log in to WebCenter Sites at `http://localhost:8080/cs/login`, then select the site in which templates need to be customized.
4. Launch the Advanced interface application on that site.
5. To create a customizable template, click **New**.

Figure 81–13 WebCenter Sites Advanced Interface: New



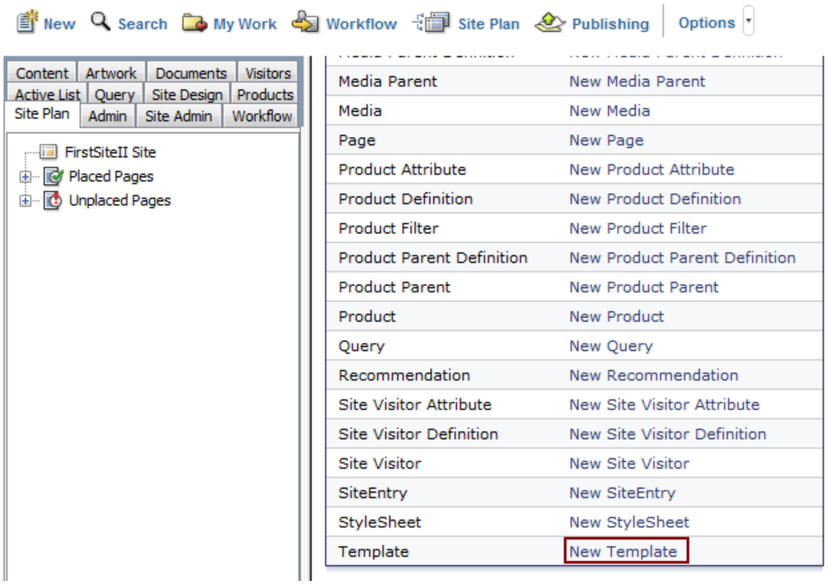
My Work

My Assignments

Type	Name
Assets not in Workflow Group	

6. In the table on the right side, click the **New Template** link next to Template.

Figure 81–14 New Template Link



Media Parent	New Media Parent
Media	New Media
Page	New Page
Product Attribute	New Product Attribute
Product Definition	New Product Definition
Product Filter	New Product Filter
Product Parent Definition	New Product Parent Definition
Product Parent	New Product Parent
Product	New Product
Query	New Query
Recommendation	New Recommendation
Site Visitor Attribute	New Site Visitor Attribute
Site Visitor Definition	New Site Visitor Definition
Site Visitor	New Site Visitor
SiteEntry	New SiteEntry
StyleSheet	New StyleSheet
Template	New Template

7. Set Assignee to any value as this value is not relevant here, then click **Continue**.
8. In the **Name** field, specify the attach point to be overridden in the form `cg/{widget}/{attach point}_view.shtml`.
9. In the **For Asset Type** field, choose **can apply to various asset types**, then click **Continue**.
10. In the **Usage** field, choose **Element defines a whole HTML page and can be called externally**.
11. In the **Create Template Element?** field, click **JSP**.
12. In the **Element Logic** field, just before the `</cs:ftcs>` closing tag, enter the content of the template. The template content must have the following statement, as well as its essentials for invalidating templates cached in Community-Gadgets:

```
<%-- Record dependencies for the Template --%>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep
cid='<%=ics.GetVar("tid")%>' c="Template"/></ics:then></ics:if>
```

13. For the content of the template, navigate to the Community-Gadgets file system and copy the default content available at `cg.war/js/widgets/{widget}/{attach point}_view.shtml`.
14. Click **Continue**.
15. On the Site Entry page, click **Save**.

16. To verify that the newly created template is available to Community-Gadgets over HTTP, enter the template's URL in a browser in the following format:

    ```
    http://{host}:{port}/cs/ContentServer?pagename={site}/cg{attach point}_
    view.shtml&ft_ss=true
    ```

17. Start Community-Gadgets servers, and verify that the customization has been applied to the deployed widgets.

81.2.2.3 Loading Custom Data Sets

In some scenarios, it is necessary to load the customized project-specific data in the existing templates. You can achieve this by exposing the data to Community-Gadgets as a REST service, in the JSONP format.

Once you have created a REST service, invoke it from the template using the system loader variable. To ensure that the custom data is loaded before the template is rendered, you must include the invocation in the `{script.preload}` section at the beginning of the page.

```
{script.preload}
       loader.loadData(url, params, key);
{/script.preload}
```

- `url`: URL address of the JSONP endpoint that provides data.
- `params`: Parameters to be passed as GET parameters to the JSONP endpoint, if the query needs to be parameterized.
- `key`: Result set returned with data are included in this variable so that this variable is accessible at `$stack.key`.

1. As an example, create a `data.jsp` file which you will use as the JSONP REST service endpoint. Include the following contents in this file:

    ```
    <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
    <%
    out.write(request.getParameter("cosrestcallback"));
    %>({ responseCode : "OK", data : ["foo","bar"]});
    ```

    In this example, the response is provided in the JSON format, containing sample data as a response code and the data array. The JSON response is wrapped up by the callback name according to the JSONP protocol used by Community-Gadgets:

2. Deploy the `data.jsp` file to the Community-Gadgets `webapp` folder.
3. Community-Gadgets has a sample widget that you can use as a sandbox for experimenting with the widget technology. Its source is located in the `cg.war/js/widget/sample.widget` code. To create custom template in WebCenter Sites with the `cg/sample.widget/layout_view.shtml` name, follow the procedure described in Section 81.2.2.2, "Creating a Sample Template."
4. In the customized template, download the data given in the `script.preload` section and then print it on a page.
5. Replace the Community-Gadgets URL with your location.

    Figure 81–1 shows the code for sample data after it is loaded.

Example 81–1 Sample Data Loaded

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
```

```
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/commercecontext.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="listobject" uri="futuretense_cs/listobject.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="siteplan" uri="futuretense_cs/siteplan.tld"
%><%@ taglib prefix="searchstate" uri="futuretense_cs/searchstate.tld"
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   COM.FutureTense.Util.ftErrors"
%><cs:ftcs><%-- /cos/sample.widget/layout_view.shtml
--%>
<%-- Record dependencies for the Template --%>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep
cid='<%=ics.GetVar("tid")%>' c="Template"/></ics:then></ics:if>
        {script.preload}
             loader.loadData("http://localhost:8280/cg/data.jsp",{},
"customData");
        {/script.preload}
        <div>
<h1>Welcome to Sample Widget. Data Load Demo</h1>
<br />
Data loaded with response code = "{$stack.customData.responseCode}"
<br />
Custom data loaded:&nbsp;&nbsp;
{script}
    var data = {$stack.customData.data};
    for(var dataIdx in data)
    {
    var item = data[dataIdx];
    {/script}
    {$$item},&nbsp;&nbsp;&nbsp;
    {script}
    }
{/script}
</cs:ftcs>
```

81.3 Creating a Custom Word Filter

Community-Gadgets enables visitors to post their feedback on a web site on which the Community widgets are deployed. To prevent publishing of objectionable content, spam, or any fraudulent information on the web site, Community-Gadgets provides a User-Generated Content (UGC) filtering sub-system for content filtration.

The default configuration of the filtering system relies on a word filter file, a list of prohibited words, that can be uploaded via the Community interface by selecting **Settings** then **Restricted Words**. Once the list of prohibited words is uploaded and auto-moderation is enabled via the Community interface by selecting **Settings** and then **Moderation**, Community-Gadgets examines the content posted by visitors and compares each word with those in the list. If a posted comment or review contains any of the prohibited words, then it is marked "Inappropriate" and is not published on the web site.

> **Note:** To ensure that prohibited words containing special characters are marked "Inappropriate" and not "Approved," you must create a custom word filter with the syntax shown in Example 81–2.

There may be a need to customize the behavior of the default content filter. For example, to integrate with third-party libraries that provide a functionality for statistical inferences and advanced techniques for spam detection and prevention, Community-Gadgets provides a pluggable word filter model. It lets you create a custom word filter from scratch.

To create a custom word filter, you must first create a Java class containing the word filter logic, compile this class, and then assemble it as a JAR file so it can be plugged into the Community-Gadgets deployment.

1. Create a new Java project in your IDE.
2. Add a JAR library, which contains the `com.fatwire.cos.moderation.filter.WordFilter` interface, to your project's classpath. You can copy this library JAR from the Community-Gadgets web application at `cg.war/WEB-INF/lib/cos-api-11.1.1.8.0.jar`.

   Example 81–2 provides the WordFilter interface.

Example 81–2 New Java Project

```
package com.fatwire.cos.moderation.filter;
import java.util.Set;
/**
 * The word filter interface that needs to be implemented
 * when developing and plugging-in a custom word filter
 * to Community-Gadgets Server.
 * The content is assigned to "Inappropriate" category and isn't
 * shown on web site once any of the filters registered in the system
 * reports an abuse.
 * @author e_shevchenko
 *
 * May 16, 2011
 */
public interface WordFilter
{
    /**
     * The filter method makes a decision whether content
     * specified can be shown on web site or not
     * @param text the content submitted by visitor
     * @param profanityWords the list of prohibited words uploaded on
     * "Moderation" page in the Community interface
     * @return true if an abuse detected and false if content
     * can be shown on web site
     */
    boolean filter(StringBuffer text, Set<String> profanityWords);
}
```

3. Create a new Java class in your project, for example, `cos.demo.DemoFilter`.
4. Declare that this class implements the WordFilter interface:

```
package cos.demo;

import java.util.Set;
import com.fatwire.cos.moderation.filter.WordFilter;
```

```
public class DemoFilter
     implements WordFilter
{
     @Override
    public boolean filter(StringBuffer text,
                          Set<String> profanityWords)
    {
       // Add filter logic here
       return false;
   }
}
```

5. Add your custom implementation inside the `filter` method body.

   The `filter` method takes two parameters: the source text submitted by visitors and the list of prohibited words uploaded as a text file via the Community interface. Based on your requirements, you can decide whether to use the existing list of prohibited words or use a third-party database of prohibited words.

   The following example shows how to verify whether user-generated content contains the word "demo":

```
package cos.demo;
import java.util.Set;
import com.fatwire.cos.moderation.filter.WordFilter;
public class DemoFilter
    implements WordFilter
{
    @Override
    public boolean filter(StringBuffer text, Set<String> profanityWords)
    {
        boolean result = false;
        if(null != text )
        {
            result = text.toString().contains("demo");
        }
        return result;
    }
}
```

6. Implement and then compile the contents of the filter class.
7. Package the compiled class as a JAR file, `filter-demo.jar`.
8. To make this custom filter available to Community-Gadgets web application's class loader, copy the `filter-demo.jar` file to the `cg.war/WEB-INF/lib` directory in both management and production environments.
9. To enable the Community-Gadgets web application to discover the custom filter, plug this filter into the `cos_word_filters.xml` configuration file located in the `cg.war/WEB-INF/classes` directory:

```
<?xml version="1.0" encoding="UTF-8" ?>
<word-filters>
<word-filter
className="com.fatwire.cos.records.moderation.filter.DefaultWordFilter"
/>
</word-filters>
```

10. Replace the default word filter. On both management and production environments, remove the existing `<word-filter/>` element from the `cos word`

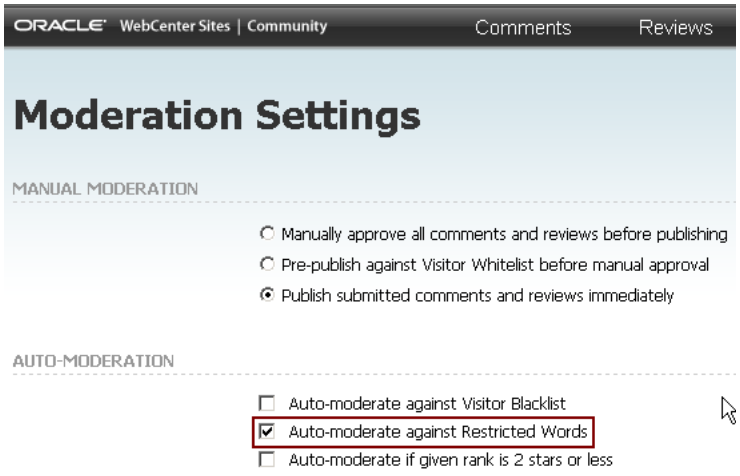
filters.xml files and add a new one that contains a reference to the class you created. The contents of the new cos word filters.xml files will look like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<word-filters>
    <word-filter className="cos.demo.DemoFilter"/>
</word-filters>
```

11. Restart the Community-Gadgets servers on both, management and production sytems.

12. To enable automatic word filtering, go to the Community interface and select **Settings**, then **Moderation** and then **Auto-moderate against Restricted Words** in the "AUTO-MODERATION"section (Figure 81–15).

Figure 81–15 Auto Moderation Against Restricted Words



Moderation Settings

MANUAL MODERATION

○ Manually approve all comments and reviews before publishing
○ Pre-publish against Visitor Whitelist before manual approval
● Publish submitted comments and reviews immediately

AUTO-MODERATION

□ Auto-moderate against Visitor Blacklist
☑ Auto-moderate against Restricted Words
□ Auto-moderate if given rank is 2 stars or less

13. On the web site, post a comment or review which includes the word "demo". You will notice that your post will be assigned to the "Inappropriate" category, and therefore, it will not be published.

81.4 Creating a CAPTCHA Generator

CAPTCHA technology helps identify robots of automated spamming systems that may attack a web site. Community-Gadgets provides a plug-in model for custom implementations of CAPTCHA.

This section describes the procedure for creating a simple CAPTCHA generator and plugging it into Community-Gadgets. This procedure is similar to the procedure for creating a custom word filter. To create a CAPTCHA generator, you will create a new class which implements the required interface, compile this class, package it as a JAR file, and then deploy it to Community-Gadgets server.

1. Create a new Java project, "demo-captcha" in your IDE.

2. Add a JAR library for your project to your project's classpath. You can copy the required JAR from the Community web application directory available here: `cg.war/WEB-INF/lib/cos-api-11.1.1.8.0.jar`.

3. From the `cos-api-11.1.1.8.0.jar` file, extract the interface `com.fatwire.cos.captcha.CaptchaGenerator` to be implemented. Example 81–3 shows the contents of this interface.

Example 81–3 com.fatwire.cos.captcha.CaptchaGenerator Interface

```
package com.fatwire.cos.captcha;
/**
 * The interface that needs to be implemented when
 * creating plug-in that generates CAPTCHA to be shown to
 * visitors when UGC is submitted.
 *
 * It's a factory that generates complex Captcha objects that are managed
 * by Community-Gadgets Server and utilized when CAPTCHA is rendered for visitors
 * and when it's validated on content submission
 *
 * @author alex
 *
 * Oct 17, 2011
 */
public interface CaptchaGenerator
{
    /**
     * Factory method that creates a new captcha object
     * that consists of challenge text
     * and challenge image to shown to visitors
     * @return
     */
    public Captcha generate();
}
```

4. In the "demo-captcha" project, create a new Java class called `cos.demo.DemoGenerator` (Example 81–4).

Example 81–4 demo-captcha Project

```
package cos.demo;
import com.fatwire.cos.captcha.Captcha;
import com.fatwire.cos.captcha.CaptchaGenerator;
public class DemoGenerator
    implements CaptchaGenerator
{
    @Override
    public Captcha generate()
    {
        // Add implementation
        return null;
    }
}
```

5. Implement the `CaptchaGenerator` interface.

6. Implement the `generate` method, as shown in Example 81–5.

   First, generate the challenge text, and then create the image displaying the generated text. Once text and image are ready, package them into the Captcha object with the help of the `CaptchaFactory` class. It is highly recommended that you use this factory instead of creating a custom implementation of the Captcha object. This is because the Captcha object built by the factory is already serializable and can be safely shared across cluster members. If you choose to create a custom implementation, then make the Captcha object serializable.

Example 81–5 Generate Method

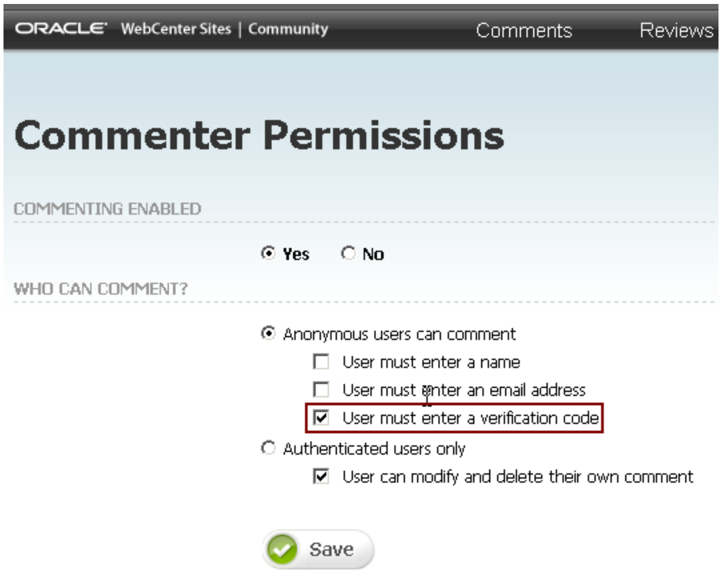
```
package cos.demo;
import com.fatwire.cos.captcha.Captcha;
import com.fatwire.cos.captcha.CaptchaFactory;
import com.fatwire.cos.captcha.CaptchaGenerator;
public class DemoGenerator
    implements CaptchaGenerator
{
    @Override
    public Captcha generate()
    {
        String challenge = "foo_bar";// Add generation
        byte[] image = generateImage(challenge); // Generate image
        return CaptchaFactory.create(challenge, image);
    }
}
```

7. Package the compiled class into a JAR file, `captcha-demo.jar`.
8. To make the generator available to the Community-Gadgets class loader, copy the `captcha-demo.jar` file to the `cg.war/WEB-INF/lib` directory on both management and production environments.
9. To enable Community-Gadgets to discover the generator, plug it into the configuration file. Create the `cos_captcha.xml` file in the `cg.war/WEB-INF/classes` directory with the following contents:

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <captcha-generator className="cos.demo.DemoGenerator"/>
   ```

10. Restart the Community-Gadgets servers on both, management and production systems.
11. To enable CAPTCHA for comments, go to the Community interface, select **Comments**, then select **Configure**, then **Permissions**, and then **User must enter a verification code** in the "Who Can Comment?" section (Figure 81–16).

Figure 81–16 Enable CAPTCHA



Commenter Permissions

COMMENTING ENABLED

● Yes ○ No

WHO CAN COMMENT?

● Anonymous users can comment

□ User must enter a name

□ User must enter an email address

☑ User must enter a verification code

○ Authenticated users only

☑ User can modify and delete their own comment

12. Repeat the previous step to enable CAPTCHA for reviews.
13. Deploy Comments or Reviews widgets with login bar support.
14. To verify that the field for the CAPTCHA challenge is displayed and the CAPTCHA image is rendered, click the **Register** link on the web site. You can also try to post a comment or review as a guest. CAPTCHA will be displayed in the form.

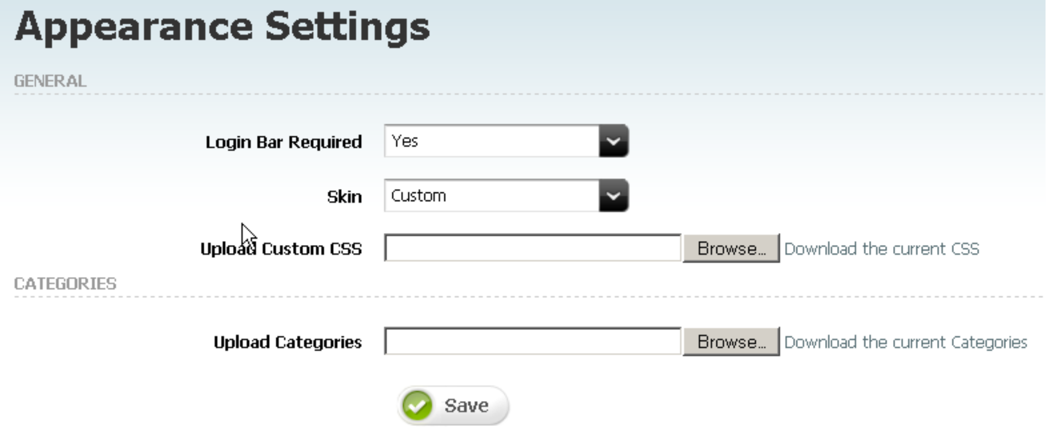
81.5 Customizing Dashboard Themes

As an administrator you can change the existent theme(s) in dashboard using CSS file.

To customize dashboard themes:

1. Open the Gadgets Interface.
2. From the **Settings** menu, choose **Appearance Setting**, as shown in Figure 81–17.

Figure 81–17 Gadgets Interface - Appearance Settngs Page



Appearance Settings

GENERAL

Login Bar Required: Yes

Skin: Custom

Upload Custom CSS

CATEGORIES

Upload Categories

3. On the Appearance Settings page, in the **General** section, next to the **Upload Custom CSS** field, click **Download the current CSS** link, then save the `gadgets_dashboard.css` file on your computer.

4. In this CSS file, modify the dashboard theme's CSS styles in the downloaded file. For example, to modify the Blue theme, find the following lines and change them as required.

   Common dashboard background color:

   ```
   .gas.gas_blue{
   background-color: #d7e9f2;
   }
   ```

   Gadget header's left, right, and content picture:

   ```
   .gas.gas_blue .gas_gadgetHeader_left {
   background-image: url(../../admin-gadgets/images/BlueLeftTop.png);
   }

   .gas.gas_blue .gas_gadgetHeader_right {
   background-image: url(../../admin-gadgets/images/BlueTopRight.png);
   }

   .gas.gas_blue .gas_gadgetHeader {
   background-image: url(../../admin-gadgets/images/BlueStrip.png);
   }
   ```

   Gadget header action background:

   ```
   .gas.gas_blue .gas_gadgetActions {
   background-image: url(../../admin-gadgets/images/BlueStrip.png);
   }
   ```

   Gadget header action buttons:

   ```
   .gas.gas_blue .gas_gadgetHeader_content a.gas_close {
   background-image: url(../../admin-gadgets/images/BlueDeleteOff.png);
   }

   .gas.gas_blue .gas_gadgetHeader_content a.gas_expand {
   background-image: url(../../admin-gadgets/images/BlueWSOpenOff.png);
   }

   .gas.gas_blue .gas_gadgetHeader_content a.gas_collapse {
   background-image: url(../../admin-gadgets/images/BlueWSCollapsedOff.png);
   }
   User toolbar collors (gadget settings)

   .gas.gas_blue .gas_UserPrefs_content {
   background-color: #d7e9f2;
   color: #ffffff;
   }
   ```

   Panel theme's icon:

   ```
   .gas .gas_setup .gas_setupIcon_theme_blue {
   background-image: url(../../admin-gadgets/images/gasThemeBlue.png);
   }

   * html .gas .gas_setup .gas_setupIcon_theme_blue {
   background-image: url(../../admin-gadgets/images/gasThemeBlue.gif);
   ```

```
}
```

5. Save the modified file.
6. From the Skin drop-down list, choose **Custom**.

   This enables the field through which you can upload the customized `gadgets_dashboard.css` file, as shown in Figure 81–17.
7. Next to the **Upload Custom CSS** field, click **Browse**.
8. In the File Upload dialog box, choose the modified `gadgets_dashboard.css` file, then click **Open**.
9. On the Appearance Settings page, click **Save**.

81.6 Community-Gadgets Events Handling

In Community-Gadgets, all events are inherited from the `CoSEvent` class. Examples of events that can be used for customer monitoring are `CommentEvent`, `ReviewEvent`, `RatingEvent` and `ExternalAuthEvent`.

Each event has a major type and a minor type. These types help identify specific events. Major types show the part of the system each event belongs to. This is represented with `CoSEventMajorType`. Possible values for are `ALL`, `COMMENT`, `REVIEW`, and `RATING`. `ALL` refers to all possible major types.

Event minor types show performed actions which lead to events appearing in and represented with `CoSEventMinorType`. Possible values are `ALL`, `CREATED`, `MODIFIED`, `DELETED`, and `REPORTED`. `REPORTED` appears when comments and reviews are flagged. `ALL` refers to all possible minor types.

Events are handled (processed) by listeners. All listeners are inherited from the `CoSEventListener` class. The listener should know which event (for example, `CommentEvent` or `ReviewEvent`) it handles.

This section contains the following topics:

- Section 81.6.1, "Event Entities"
- Section 81.6.2, "Event Listeners"

81.6.1 Event Entities

The parent class for all event entities is `CoSEvent`; the full class name is `com.fatwire.cos.events.CoSEvent`. This class is located in the `cg.war/WEB-INF/libs/cos-api-11.1.1.8.0.jar` library.

This section contains the following topics:

- Section 81.6.1.1, "CommentEvent Entity"
- Section 81.6.1.2, "ReviewEvent Entity"
- Section 81.6.1.3, "RatingEvent Entity"
- Section 81.6.1.4, "ExternalAuthEvent Entity"

81.6.1.1 CommentEvent Entity

The CommunityEvent entity is used with comment creation, modification and deletion. The full class name is `com.fatwire.cos.records.events.CommentEvent`. This class is located in the `cg.war/WEB-INF/libs/cos-shared-11.1.1.8.0.jar` library.

- **Supported major types**: `ALL`, `COMMENT`
- **Supported minor types**: `ALL`, `CREATED`, `MODIFIED`, `DELETED`, `REPORTED`

Table 81–18 CommunityEvent Methods

Method Declaration	Method Definition
`Long getFeedId()`	Link to CommentsFeed object
`String getSiteId()`	Name of site on which the event happens

`ModerateContentState getRecordState()`	Shows moderation state. Possible values:
	`RECORD_PENDING_NEW`
	`RECORD_PENDING_MODIFIED`
	`RECORD_PENDING`
	`RECORD_APPROVED`
	`RECORD_INAPPROPRIATE_ROBOT_DETECTED`
	`RECORD_INAPPROPRIATE_HUMAN_DETECTED`
	`RECORD_INAPPROPRIATE`

81.6.1.2 ReviewEvent Entity

The ReviewEvent entity appears on review creation, modification and deletion. The full class name is `com.fatwire.cos.records.events.ReviewEvent`. This class is located in the `cg.war/WEB-INF/libs/cos-shared-11.1.1.8.0.jar` library.

- **Supported major types**: `ALL`, `COMMENT`
- **Supported minor types**: `ALL`, `CREATED`, `MODIFIED`, `DELETED`, `REPORTED`

Table 81–19 ReviewEvent Methods

Method Declaration	Method Definition
`Long getFeedId()`	Link to ReviewsFeed object
`String getSiteId()`	Name of site on which the event happens

`ModerateContentState getRecordState()`	Shows moderation state. Possible values:
	`RECORD_PENDING_NEW`
	`RECORD_PENDING_MODIFIED`
	`RECORD_PENDING`
	`RECORD_APPROVED`
	`RECORD_INAPPROPRIATE_ROBOT_DETECTED`
	`RECORD_INAPPROPRIATE_HUMAN_DETECTED`
	`RECORD_INAPPROPRIATE`

81.6.1.3 RatingEvent Entity

The RatingEvent entity appears on ratings creation and deletion. The full class name is `com.fatwire.cos.records.events.RatingEvent`. This class is located in the `cg.war/WEB-INF/libs/cos-shared-11.1.1.8.0.jar` library.

- **Supported major types**: `ALL`, `RATING`
- **Supported minor types**: `ALL`, `CREATED`, `DELETED`

Table 81–20 RatingEvent Methods

Method Declaration	Method Definition
`Long getFeedId()`	Link to RatingFeed object
`String getSiteId()`	Name of site on which the event happens
`ModerateContentState getRecordState()`	Shows moderation state. Possible values: ■ `RECORD_APPROVED` ■ `RECORD_INAPPROPRIATE_ROBOT_DETECTED` ■ `RECORD_INAPPROPRIATE_HUMAN_DETECTED` ■ `RECORD_INAPPROPRIATE`

81.6.1.4 ExternalAuthEvent Entity

The ExternalAuthEvent entity appears on successful authentication with an external authentication provider. The full class name is `com.fatwire.cos.events.api.ExternalAuthEvent`. The class is located in the `cg.war/WEB-INF/libs/cos-api-11.1.1.8.0.jar` library.

- **Supported major types**: `ALL`
- **Supported minor types**: `ALL`

Table 81–21 ExternalAuthEventEvent Methods

Method Declaration	Method Definition
`String getAuthenticatorId()`	Identifier of external authentication provider
`JSONObject getRawData()`	Data received from external authentication provider in `JSONObject` class. This depends on the provider (for example, Facebook or Twitter). It should be parsed in provider-specific way.

81.6.2 Event Listeners

Event listeners are used for listening to all events and logging the activity. All listeners are inherited from the `CoSEventListener` class. The listener should know which event (for example, `CommentEvent` or `ReviewEvent`) it handles.

The full class name is `com.fatwire.cos.events.CoSEventListener`. This class is found in the `cg.war/WEB-INF/libs/cos-api-11.1.1.8.0.jar` library.

This section contains the following topics:

- Section 81.6.2.1, "Creating a Simple Listener"
- Section 81.6.2.2, "Developing Event Notification Listeners - Process Method"
- Section 81.6.2.3, "Developing Event Notification Listeners - Sending Email"

81.6.2.1 Creating a Simple Listener

This example shows the creation of a listener that logs all events. The `cos-api-11.1.1.8.0.jar` library is required.

To create a simple listener

1. Create a `SampleListener` class by extending abstract class `CoSEventListener`

```
public class SampleListener extends com.fatwire.cos.events.CoSEventListener
```

2. Choose the major and minor types which the listener will record (in this example, `ALL`) and set them in the class constructor.

```
public SampleListener () {
   setMajorType(CoSEventMajorType.ALL);
   setMinorType(CoSEventMinorType.ALL);
}
```

3. Implement two functions - `void process (CoSEvent event)` and `int getTaskCount()` - which are not implemented in the parent abstract class.

```
private final static Logger LOG = Logger.getLogger(SampleListener.class);
private AtomicInteger counter = new AtomicInteger();

@Override
public void process(CoSEvent event) {
    counter.incrementAndGet();
    try{
        LOG.info("Process event: " + event.getClass() + ": " + event);
    } finally {
        counter.decrementAndGet();
    }
}

@Override
public int getTaskCount(){
    return counter.get();
}
```

Once completed, the full class file will look similar to this:

```
package com.example.listeners;

import java.util.concurrent.atomic.AtomicInteger;

import org.apache.log4j.Logger;

import com.fatwire.cos.events.CoSEvent;
import com.fatwire.cos.events.CoSEventListener;
import com.fatwire.cos.events.CoSEventMajorType
import com.fatwire.cos.events.CoSEventMinorType

public class SampleListener extends CoSEventListener
{
    private final static Logger LOG = Logger.getLogger(SampleListener.class);
    private AtomicInteger counter = new AtomicInteger();

    public SampleListener () {
       setMajorType(CoSEventMajorType.ALL );
       setMinorType(CoSEventMinorType.ALL );
    }

    @Override
    public void process(CoSEvent event) {
        counter.incrementAndGet();
        try{
            LOG.info("Process event: " + event.getClass() + ": " + event);
        } finally {
            counter.decrementAndGet();
        }
    }

    @Override
    public int getTaskCount(){
        return counter.get();
    }
}
```

4. Add the listener to Community-Gadgets configuration.

   Open `listeners.xml` file in the `cg.war/WEB-INF/classes` folder and add the bean with listener class.

   ```
   <bean id="sampleListener" class="com.example.listeners.SampleListener" />
   ```

81.6.2.2 Developing Event Notification Listeners - Process Method

The process method can filter events if the event type is set to `ALL`.

```
@Override
public void process(CoSEvent event) {
    if(event instanceof CommentEvent) {
        sendCommentNotification(event);
    } else if (event instanceof ReviewEvent) {
        sendReviewsNotification(event);
    }
}
```

81.6.2.3 Developing Event Notification Listeners - Sending Email

The listener can send a MIME email to an address or group of addresses with the notification text. One way this can be done by developing a sender using the JavaMail API. However, it is possible send email using the Community-Gadgets Email Facility by including a pair of lines in the class file (This requires `cos-core-11.1.1.8.0.jar` in the classpath).

```
...
import com.fatwire.cos.core.CoS;
...
CoS.instance().getEmailFacility().send(mimeMessage);
...
```

For generating `mimeMessage`, code similar to the following would be used:

```
private MimeMessage generateEmail(CoSEvent event, String emails) throws
  AddressException, MessagingException {
        Session session = null;
        MimeMessage msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress("noreply@samplenotify.example.com"));
        msg.setSentDate(new java.util.Date());
        msg.setRecipients(javax.mail.Message.RecipientType.TO,
            InternetAddress.parse(emails));
        msg.setSubject("==Notification subject==");

        Multipart multipart = new MimeMultipart("alternative");
        BodyPart messageBodyPart = new MimeBodyPart();
        messageBodyPart.setContent("<SOME NOTIFY TEXT>",
                                   "text/plain;charset=\"UTF-8\"");
        multipart.addBodyPart(messageBodyPart);
```

```
        msg.setContent(multipart);
        msg.saveChanges();

        return msg;
    }
```

Note: Velocity library, or another template engine, can be used for message body content generation. Generated template must be used instead of *"<SOME NOTIFY TEXT>"* token above.

When the listener is created, the full class file may look similar to the following example. The class file requires `cos-api-11.1.1.8.0.jar`, `cos-core-11.1.1.8.0.jar`, and `cos-shared-11.1.1.8.0.jar`.

```
package com.fatwire.cos.shared;

import java.util.concurrent.atomic.AtomicInteger;

import javax.mail.BodyPart;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.Session;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

import com.fatwire.cos.core.CoS;
import com.fatwire.cos.events.CoSEvent;
import com.fatwire.cos.events.CoSEventListener;
import com.fatwire.cos.events.CoSEventMajorType;
import com.fatwire.cos.events.CoSEventMinorType;
import com.fatwire.cos.records.events.CommentEvent;
import com.fatwire.cos.records.events.ReviewEvent;

public class NotificationListener extends CoSEventListener {

    private AtomicInteger counter = new AtomicInteger();

    public NotificationListener () {
       setMajorType(CoSEventMajorType.ALL );
       setMinorType(CoSEventMinorType.ALL );
    }

    @Override
    public void process(CoSEvent event) {
        counter.incrementAndGet();
        try {
            if(event instanceof CommentEvent) {
                sendCommentNotification((CommentEvent)event);
            } else if (event instanceof ReviewEvent) {
                sendReviewsNotification((ReviewEvent)event);
            }
        } finally {
            counter.decrementAndGet();
        }
    }
    @Override
    public int getTaskCount() {
        return 0;
    }

    private void sendCommentNotification(CommentEvent event) {
        MimeMessage mimeMessage = null;
        String site = event.getSiteId();
        try {
            mimeMessage = generateEmail(event, site ,
                "moderator_comments@samplenotify.example.com");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        sendEmail(mimeMessage);
    }

    private void sendReviewsNotification(ReviewEvent event) {
        // send notification only for creating review event
        if (event.getMinorType().equals(CoSEventMinorType.CREATED)) {
            MimeMessage mimeMessage = null;
            String site = event.getSiteId();
            try {
                mimeMessage = generateEmail(event, site,
                    "moderator_reviews@samplenotify.example.com");
            } catch (Throwable e) {
                e.printStackTrace();
            }
            sendEmail(mimeMessage);
        }
    }

    private void sendEmail(MimeMessage mimeMessage) {
        CoS.instance().getEmailFacility().send(mimeMessage);
    }

    private MimeMessage generateEmail(CoSEvent event, String site,
        String emails) throws AddressException, MessagingException {
        Session session = null;
        MimeMessage msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress("noreply@samplenotify.example.com"));
        msg.setSentDate(new java.util.Date());
        msg.setRecipients(javax.mail.Message.RecipientType.TO,
                          InternetAddress.parse(emails));
        msg.setSubject("==Notification subject== from " + site + " site");

        Multipart multipart = new MimeMultipart("alternative");
        BodyPart messageBodyPart = new MimeBodyPart();
        messageBodyPart.setContent("<SOME NOTIFY TEXT>",
                                   "text/plain;charset=\"UTF-8\"");
        multipart.addBodyPart(messageBodyPart);

        msg.setContent(multipart);
        msg.saveChanges();

        return msg;
    }

}
```

After the file is created, add the following bean to `listentes.xml`:

```
<bean id="sampleListener" class="com.example.listeners.NotificationListener" />
```

82

Community-Gadgets: Localizing Its Functionality

This chapter describes how to enable translation of Community-Gadgets functionality into multiple languages.

This chapter contains the following sections:

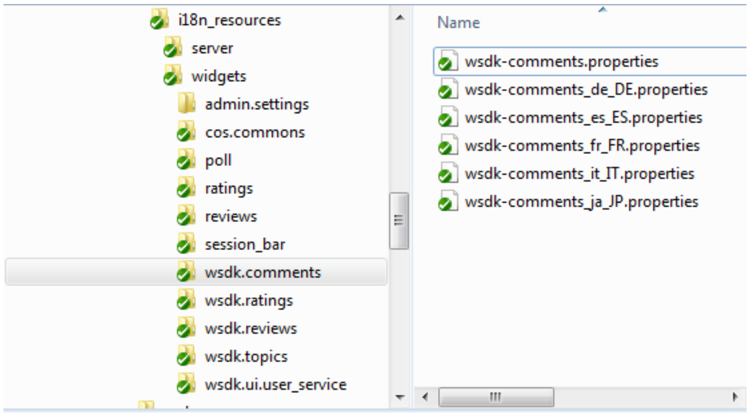
- Section 82.1, "About Localization"
- Section 82.2, "Adding a New Language to Community-Gadgets"
- Section 82.3, "Registering the New Language in Community-Gadgets"

82.1 About Localization

The Community-Gadgets user interface can be translated, by default, into the following languages: English, French, German, Italian, Japanese, Korean, Portuguese, Spanish, Simplified Chinese, and Traditional Chinese. Moreover, you can also customize existing labels and translations to new languages on demand.

The language pack is located in the file system of the Community-Gadgets web application at `cg.war/WEB-INF/classes/i18n_resources`.

Figure 82–1 Language Pack



The `i18n_resources` directory contains two subfolders, `server` and `widgets` that include resources for the Community-Gadgets interface and for client-side widgets respectively.

The `server` directory contains a list of Java property resource bundles that are separated on the basis of the Community application sub-system. It contains localizations for the following sub-systems: `core`, `cos-resources`, `gadgets`, `gadgets-exception, service`, `shared`, and `users`.

The `widgets` directory contains a list of folders in which each folder corresponds to a particular widget. For example, the `wsdk.comments` folder corresponds to the Comments widget and the `wsdk.reviews` folder to the Reviews widget.

The `cos.commons` directory contains resources that are shared across all the widgets.

The logic of picking up a required language is different in the Community interface and in the widgets deployed on a web site. The following sections discuss both types of logic in order of priority.

- Section 82.1.1, "Language Detection for the Community-Gadgets Interface"
- Section 82.1.2, "Language Detection for Community-Gadgets"

82.1.1 Language Detection for the Community-Gadgets Interface

1. If a language is customized in the Web Experience Management (WEM) profile for the logged-in business user, apply it. If not, then proceed to the next step.
2. If the locale of the browser is supported, apply this setting. If not, proceed to the next step.
3. Apply the default `English` locale.

82.1.2 Language Detection for Community-Gadgets

Language detection logic described in this section applies to all Community-Gadgets end side widgets.

The following first and second steps are based on the premise that the web site may already have some language selection mechanism to display site content in different languages. For example, most sites have a language icon on the top or bottom of a site page for this purpose. If this mechanism uses cookies or JavaScript variables for language selection, then the Community widgets can also use them after client's consent.

Site developers customize the language selection mechanism by enabling the creation of the `cos_language` cookie or the JavaScript variable with a language value. Once this is done, the Community widgets can use the client's language selection mechanism to display site content in different languages.

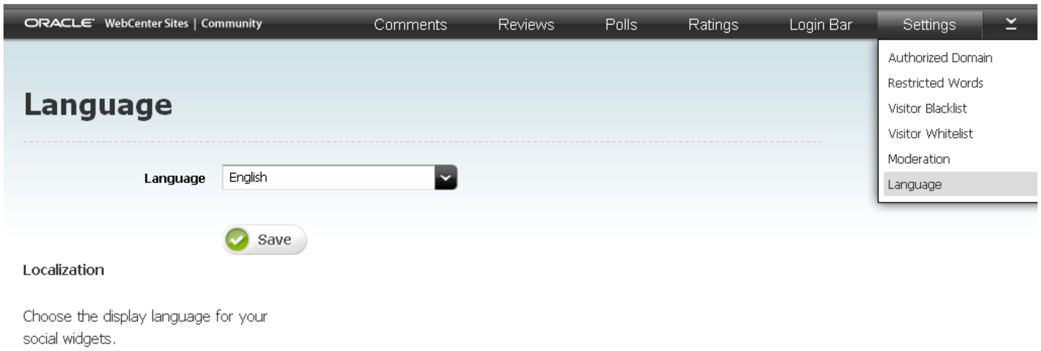
1. Use the language parameter given in a `cos_language` cookie that site developers specify manually.

   If no cookie is set, proceed to the next step.
2. Use the language parameter of a global JavaScript variable, `cos_language` that site developers or integrators can specify manually.

   If no variable is set, proceed to the next step.
3. Use the default language setting chosen by the administrator via the Community interface by selecting **Settings** then **Language** (Figure 82–2).

   This setting applies to all types of widgets deployed on the site.

Figure 82–2 Language Setting in Community-Gadgets



Language

Language: English

Localization

Choose the display language for your social widgets.

82.2 Adding a New Language to Community-Gadgets

To add a new language, complete the following steps:

1. List the new language in the Community-Gadgets configuration (see Section 82.3, "Registering the New Language in Community-Gadgets"), so that it displays in the default language selector in the Community interface.
2. Upload translations for all the resources located in `cg.war/WEB-INF/classes/i18n_resources/<subfolders containing translation files>`. That is, translate all `*_en.properties` files located in this directory and copy back the translated files (for example, for Russian language the file is `*_ru.properties`).

82.3 Registering the New Language in Community-Gadgets

To register a new language in the Community-Gadgets configuration:

1. Navigate to the `cg.war/WEB-INF/lib` directory.
2. Open the `cos-shared-11.1.1.8.0.jar` file, then extract the `cos_core_metadata.xml` file.
3. Search for the `schema::cos::commons:permissions:language` string. The declaration is as follows:

```
<bean id="language"
class="com.fatwire.cos.metadata.core.model.MetadataDescriptorImpl">
     <property name="uid"
                        value="schema::cos::commons:permissions:language"/>
     <property name="name" value="label.language"/>
     <property name="defaultValue" value="en_US"/>
      <property name="dataOptionsValue" value="[
{ name: 'label.language.english', value:'en_US'},
{ name: 'label.language.brazilian_portuguese', value:'pt_BR'},
{ name: 'label.language.chinese_simplified', value:'zh_CN'},
{ name: 'label.language.chinese_traditional', value:'zh_TW'},
{ name: 'label.language.french', value:'fr'},
{ name: 'label.language.german', value:'de'},
{ name: 'label.language.italian', value:'it'},
{ name: 'label.language.japanese', value:'ja'},
{ name: 'label.language.korean', value:'ko'},
{ name: 'label.language.spanish', value:'es'},
<ADD COMMA AND INSERT NEW LANGUAGE HERE>
]"/>
```

...

```
</bean>
```

4. To the `dataOptionsValue` property, add a new JSON object for the new language. For example, for Russian, the entry can be the following:

   ```
   { name: 'label.language.russian', value:'ru'}
   ```

   If the entry for Russian is added, the property value will be:

   ```
   <property name="dataOptionsValue" value="[
   { name: 'label.language.english', value:'en_US'},
   { name: 'label.language.brazilian_portuguese', value:'pt_BR'},
   { name: 'label.language.chinese_simplified', value:'zh_CN'},
   { name: 'label.language.chinese_traditional', value:'zh_TW'},
   { name: 'label.language.french', value:'fr'},
   { name: 'label.language.german', value:'de'},
   { name: 'label.language.italian', value:'it'},
   { name: 'label.language.japanese', value:'ja'},
   { name: 'label.language.korean', value:'ko'},
   { name: 'label.language.spanish', value:'es'}
   { name: 'label.language.russian', value:'ru_RU'} ]"/>
   ```

5. Save the `cos_core_metadata.xml` file and include this revised file in the `cos-shared-11.1.1.8.0.jar` file.

6. Navigate to the `i18n_resources/server/cos-resources_<LANG>.properties` resource bundle, then add the `label.langauge.russian` key with value translation for each supported language.

   Each of the `cos-resources` files (`cos-resources.properties`, `cos-resources_de.properties`, `cos-resources_es.properties`, `cos-resources_fr.properties`, `cos-resources_it.properties`, and so on) contains a set of language labels. The following is an example for `cos-resources_en.properties`:

   ```
   label.language.english = English
   label.language.brazilian_portuguese = Brazilian Portuguese
   label.language.chinese_simplified = Simplified Chinese
   label.language.chinese_traditional = Traditional Chinese
   label.language.french = French
   label.language.german = German
   label.language.italian = Italian
   label.language.japanese = Japanese
   label.language.korean = Korean
   label.language.spanish = Spanish
   ```

   Add a new line containing the parameter name (for example, `label.langauge.russian`) and the parameter value (for example, `Russian for cos-resources_en.properties`):

   ```
   label.langauge.russian=Russian
   ```

   The `cos-resources_en.properties` value for `label.langauge.russian` is different (`Russische` in `cos-resources_de.properties` or `Russe` in `cos-resources_fr.properties`).

   Therefore,

   ```
   label.langauge.<lang_id>=<value_translation>
   ```

7. For all resource bundles that can be found recursively in the `i18n_resources` directory, upload translation files in the same folder with the corresponding language suffix. For example, in the directory named `i18n_`

`resources/server/users.properties` you can find a Russian translation for the English pack. So, upload the translation file for `Russian: users_ru_RU.properties` in the `i18n_resources/server/` directory.

8. Restart the application server for the Community management application.
9. To verify that the new language is displayed, log in to Community as an administrator, then choose **Language** from the Settings menu.

83 Community-Gadgets: Monitoring Its Performance

This chapter provides an overview of caching in WebCenter Sites. It explains how content is fetched and displayed on a web site when cache is enabled or disabled. This chapter also explains different cache modes and how they affect the performance and data consistency of the Community-Gadgets web application. It describes how to configure cache and optimize the handling of user-generated content.

This chapter contains the following sections:

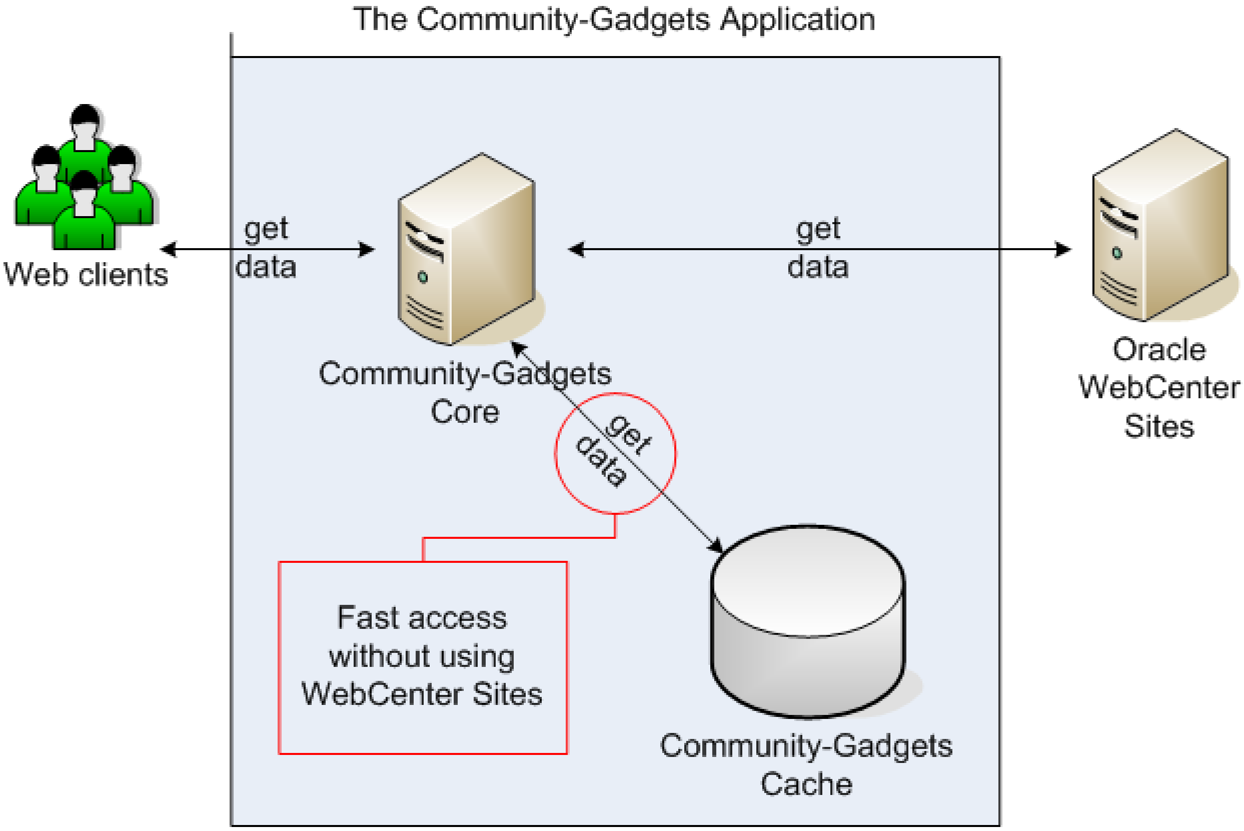
83.1 About Caching

When a system receives a request, it routes it to different sub-systems that perform the required functions to serve this request, as shown in Figure 83–1 and described later in this section. The results of the request are compiled and then served to the client that made the request. These results could be in the form of complete comments for a page, comments sorted in a certain way, reviews, and so on. To improve the application's performance, these results can be stored in a such way that when the system receives a similar request, it can immediately locate and reuse the stored information to serve the new request.

> **Tip:** The Community-Gadgets web application uses the WebCenter Sites data repository and communicates with it over REST. WebCenter Sites, in turn, translates the REST request into a database query.

When Community-Gadgets loads content from its database, it caches that content in the in-memory cache, which is built with the help of the inCache technology provided by WebCenter Sites. Cache is an intermediate storage (RAM) in which data for the most popular and frequent queries is stored. The access to cache is very fast as RAM is used as storage.

Figure 83–1 Cache Process Flow



Caching helps save network communication and data serialization overheads. To improve system performance and throughput, requests should be served from cache.

Two types of events take place when cache technology is used to serve requests: cache hit and cache miss. A *cache hit* occurs when the requested data already exists in the cache, and therefore, it is immediately served from the cache to a client. A *cache miss* occurs when the requested data does not exist in the cache. In this case, first the data from the WebCenter Sites data repository is copied into the cache, and then subsequent requests are served. For example, if a user requests the page on which comments/reviews were posted for the first time (that is, this page was not requested before, or cache was flushed), Community-Gadgets saves it to the cache. When another user requests this page, Community-Gadgets retrieves it from the cache. As a result of interaction between users and Community-Gadgets, the application caches all the data loaded from WebCenter Sites. This cache is an in-memory cache.

The following section explains how a system behaves with caching.

83.2 Community-Gadgets With Cache

This section contains the following topics:

- Section 83.2.1, "Understanding Community-Gadgets"
- Section 83.2.2, "Regular Caching"
- Section 83.2.3, "Stale Caching"
- Section 83.2.4, "Caching Dependencies"

83.2.1 Understanding Community-Gadgets

When a visitor creates, updates, or deletes an item such as a comment or review on a web site page, Community-Gadgets:

- modifies all the related data in WebCenter Sites
- invalidates all the related cache entries

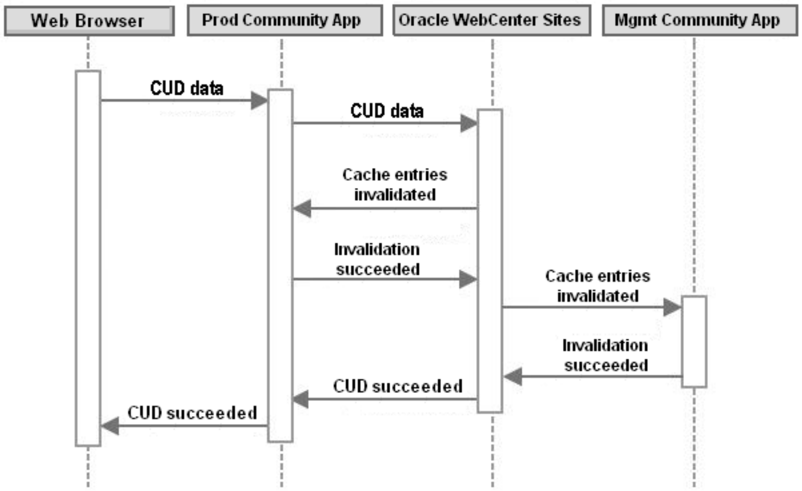
This way, Community-Gadgets indicates to the whole system that cached entries are stale or no longer valid, and therefore, they should be updated based on the user's action. For example, when you edit an existing comment, this particular item and the entire data collection associated with the discussion whose comment you edited needs to be invalidated in the cache. The update of invalidated values happens in the background for the future access.

WebCenter Sites supports remote server in which cached content is kept up-to-date by sending invalidation requests over the REST protocol. To enable this, register the Community-Gadgets web application as a remote server in the `SystemSatellite` table. For Community-Gadgets installation and configuration details, see the *Oracle Fusion Middleware WebCenter Sites Installation Guide*.

Once you have registered Community-Gadgets as a satellite server, it starts receiving notifications about the modified data and invalidates its cache accordingly. When the data is modified, a notification is sent to all satellites, including the Community-Gadgets production and management systems. Figure 83–2 shows the process flow.

Note: Notifications are broadcasted in a synchronous manner, and therefore, a modification request is not completed unless all satellites acknowledge receiving this invalidation.

Figure 83–2 Communication Between WebCenter Sites and Community-Gadgets Satellite Servers



As shown in the process flow, Community-Gadgets caches results for all requests locally in the memory and receives invalidations to keep its cache consistent. For example, if Community-Gadgets collects, stores, and sends results for a request, and then a similar request comes, the application responds with cached data. However, if a visitor makes any changes between the time frame of the first and second requests, and an invalidation request is made in the same time frame, then the second request is served from the WebCenter Sites repository and not from the cache. See also, Figure 83–3, Figure 83–4, and Figure 83–5.

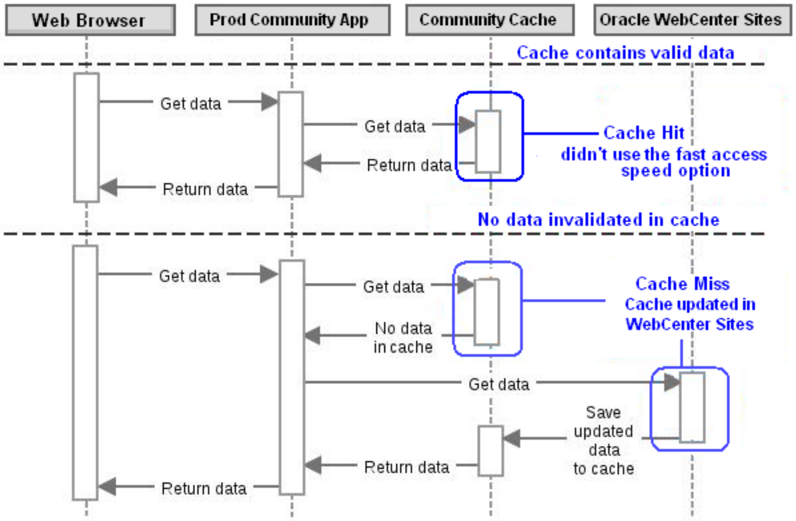
The two caching modes of Community-Gadgets are: *regular caching* and *stale caching*. Both modes share the same approach to data reads. If the required data is in the cache, it is served immediately. When the data is not in the cache, it is loaded from the WebCenter Sites repository synchronously. Therefore, results in the user interface are displayed for the user only after the `load` operation is completed and the data is cached. However, data invalidations and cache consistency work differently in regular caching and stale caching.

83.2.2 Regular Caching

When Community-Gadgets receives an invalidation request for a particular piece of data, it invalidates the cached data immediately so that all subsequent requests for that data get cache misses and the fresh data is loaded from the WebCenter Sites repository.

Figure 83–3 shows the process flow for the cache hit and cache miss events in the regular caching mode.

Figure 83–3 Cache Hit and Cache Miss in Regular Caching



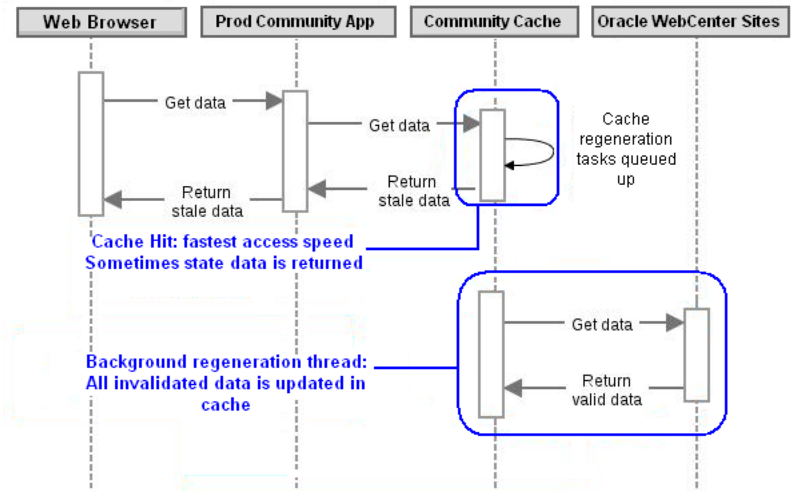
In the regular caching mode the data consistency level is high, and all the changes are immediately reflected in the interface. For example, when a user posts a new comment and refreshes the page, the new comment is shown on the page immediately. Similarly, content deleted from a page disappears immediately after the page is refreshed.

83.2.3 Stale Caching

When Community-Gadgets receives an invalidation request, it does not invalidate the data and purge it immediately. This is in contrast with regular caching in which the cached data is invalidated immediately. In the stale caching mode, Community-Gadgets just marks the data that is no longer valid with a special invalidation mark. However, the data still remains in the cache so it can still be served from cache memory. When a request for the invalidated data is received, a thread is launched to update the invalidated data. This process is called *regeneration cycle*.

Figure 83–4 shows how invalidated data is regenerated in the background and regenerated content is served from cache.

Figure 83–4 Regeneration Cycle in Stale Caching


Web Browser
Prod Community App
Community Cache
Oracle WebCenter Sites
Get data
Get data
Cache regeneration tasks queued up
Return stale data
Return stale data
Cache Hit: fastest access speed
Sometimes state data is returned
Get data
Background regeneration thread:
All invalidated data is updated in cache
Return valid data

In the stale caching mode, the performance level is high because data loads happen seamlessly in the background. However, due to background data loads, the data consistency level is lower than that in the regular caching mode, so visitors do not see the actual results immediately. For example, when a comment is posted, it takes a couple of page refreshes before the comment is displayed on the page. The following process explains what happens in the stale caching mode:

1. Visitor posts a comment: Community-Gadgets marks the comment list with a special invalidation mark.
2. Visitor refreshes the page for the first time: Community-Gadgets still serves old data without the new comment from the cache, but detects the special mark and starts the background update process.
3. Visitor refreshes the page for the second time: the background update process is completed, the cache is updated, the new comment list containing the newly posted entry is returned to the client.

   The background update process starts with the first refresh. The second refresh displays the updated data because cache is updated by the end of the first refresh. However, it depends on network overheads and the volume of changes, so potentially it may take more than two refreshes.

By default, Community-Gadgets uses a regular caching schema. To boost performance, you can enable the stale caching in the `wsdk_facilities.properties` configuration file in the standalone folders (default locations are `<cg_install_dir>/deploy/management/management_node1` and `<cg_install_dir>/deploy/production/production_node1`) by setting the `app_cache.stale` property to `true`, as shown in Example 83–1.

Example 83–1 Stale Cache Mode Enabled

```
#
# Application cache facility configuration
# + stale parameter enables usage 'old' data on production side
#
app_cache.stale=false
```

When the `app_cache.stale` property is set to `false`, the regular caching schema is used. However, certain areas in the product always need a high level of consistency, and therefore, Community-Gadgets overrides the stale cache mode for the following areas:

- The Community interface and the Gadgets interface. Data needs to be always consistent with production system for easy management.
- Operations on visitors' profiles. Especially during registration when the application needs to determine if a user with such user name already exists.

You can manage cache through the Cache Management Console, which is accessible at `http://<cg_host>:<cg_port>/<cg_context>/cache` on each Community-Gadgets instance. For example, `http://mycosprod.example.com:8180/cg/cache/`. For more information about cache management and Community-Gadgets, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.

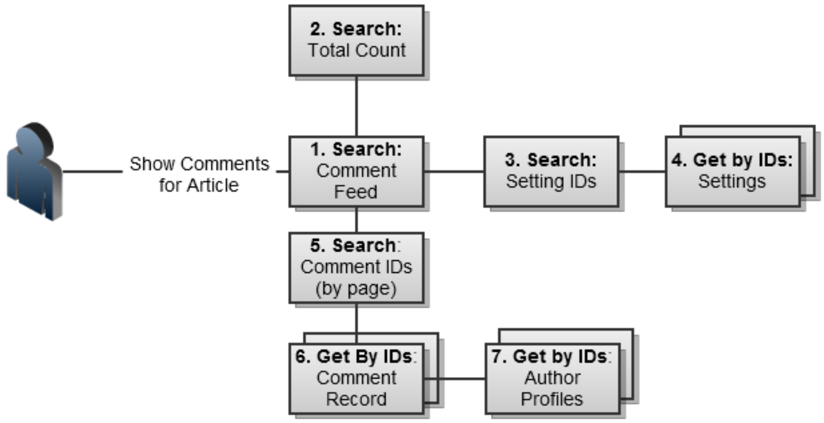
83.2.4 Caching Dependencies

The following types of dependencies are associated with caching:

- **Dependencies for content modification**: This type of dependency is recorded in the Dependencies column of the cache entries table such as Commons Cache, and it contains the ID of the existing asset. The system invalidates this cache entry whenever the associated asset changes.
- **Dependencies for content creation**: This type of dependency is system-specific. The `unknowndeps: <table name>` type of the inCache dependency keyword is used in such cases. Whenever a new entry is posted in the table, the cache is invalidated. This type of caching is useful for search and count queries that are required to be invalidated if any new content is posted.

During cache management, user operations must be constantly mapped to Community-Gadgets logic and entries stored in the cache. For example, during comment loading, the data structures shown in Figure 83–5 are associated with the "Loading..." message (the Loading operation) which is displayed initially when the Comments widget starts loading.

Figure 83–5 Data Structure Associated with the Comments Operation



The following is a step-by-step explanation of Figure 83–5. These points describe how each operation for comments is performed and cached:

1. The CommentFeed object that aggregates the whole discussion is discovered.

2. The total number of comments is discovered. This number is used to calculate and render the pagination interface and other comment-related features.
3. The site settings configured in Community-Gadgets are loaded. These settings are applied to the CommentFeed object that is rendered on the page accessed by visitors. These settings determine where the Post Comment form will be shown (top or bottom) on a page, the level of commenting permissions, and so on. After the setting IDs are discovered, Community-Gadgets searches the cache and gets cache hits before contacting the database.
4. If site settings are not cached, they are fetched from the WebCenter Sites repository by their IDs. Lookup by ID is used whenever possible because of the effectiveness of the process.
5. Loading of actual comments to be displayed on the web site begins. The Comment Feed objects and CommentRecord objects have a one-to-many relationship. First, all the IDs of comments associated with a particular feed are discovered. After the IDs of the comments associated with a particular feed are discovered, the IDs are analyzed and validated against the local cache entries. The IDs that are missing in the cache are loaded with the next request.
6. CommentRecord objects are loaded from the WebCenter Sites repository by the IDs discovered on the previous step.
7. The comments are examined, and the author IDs are extracted from comments that were posted by authenticated users. Then, the visitor profile IDs for which there are cache hits are loaded from the WebCenter Sites repository.

83.3 Configuring Cache in Community-Gadgets

The cache configuration in Community-Gadgets is similar to the cache configuration in WebCenter Sites because both products use the same inCache infrastructure.

The cache configuration file, `cos-cache.xml` and `cas-cache.xml`, is located in the `cos-standalone-config` directory for each Community node directory.

For more information about configuration parameters in this file and configuring inCache for page caching, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

83.4 Optimizing User-Generated Content (UGC) in the Community Application

There are certain peak hours when the visitor traffic and the amount of feedback are highest in a day. In other words, there is a time during a day when the maximum number of UGC entries are submitted to database. During these periods of peak loads, database cannot handle the content insertion rate at the speed at which visitors submit content entries. To address such situations, WebCenter Sites provides an option called *delayed writes*. This is a simple but effective tool that uses the producer-consumer pattern. Visitors submit content to the system in the form of comments, reviews, and so on. The content entries are then queued up for the database. A consumer thread picks up these entries from the queue and inserts them into the database one-by-one, so results of these incoming requests (UGC entries) display on the web site later. Since changes are buffered and applied in the background, there is data inconsistency, so users cannot see their changes immediately.

The delayed writes option is available only after the administrator has configured it in WebCenter Sites. For information about how to enable this option, see Chapter 77, "WEM Framework: Buffering."After the delayed writes option has been configured on the WebCenter Sites instance used by Community-Gadgets, you must also set this option in the application's `setup_cs.properties` configuration file located in the standalone_node folders (default locations are: *<cg_install_dir>*`/deploy/management/management_node1` and *<cg_install_dir>*`/deploy/production/production_node1`). In this file, set the `widgets.cs.production.attrs.delayed_writes` property to `true`:

```
#
# Enabling "delayed writes" mechanism. CS will persist the assets asynchronously.
# Default is "false"
#
widgets.cs.production.attrs.delayed_writes=true
```

Once the delayed writes option in enabled in Community-Gadgets, restart the Community production and management application servers. You will notice that the behavior of the production system will change when visitors submit content in the form of reviews, comments, and so on. For example, when a visitor posts a comment, the "Thank you for your submission" message will be displayed and comment will not appear on the site unless it is processed on the server asynchronously.

84

Community-Gadgets: Guidelines for Maintaining the Application

This chapter contains the following sections:

84.1 Widget Deployment Guidelines

The deployment process can be started for the Community-Gadgets widgets as soon as the widget functionality, permissions, and appearance are configured in the Community-Gadgets web application. During widget deployment, when the **site settings** option is selected, configurations made in Community-Gadgets are instantly reflected on the deployed widgets, after the page containing the widget is reloaded. This option enables the deployed widget to constantly monitor configuration changes in the database. However, in the **custom settings** option, some of the configuration settings use custom values that are embedded into the widget deployment code snippet itself. Note that custom configuration settings are stored in the HTML/JavaScript code of the page on which the widget is deployed, and not in the database. As a result, custom configuration settings take priority over site settings.

When you deploy a widget with the **custom settings** option selected, the **Resource ID** field becomes available for customization. The resource ID enables Community-Gadgets to associate the content posted by visitors with the page on which it is deployed. Because of the lightweight integration approach, which uses JavaScript to deploy widget functionality to the web site, this ID is used to determine the type of content to be shown on a page. For example, if no resource ID is specified during deployment, `md5` (a cryptographic hash function that produces a 128-bit (16-byte) hash value) is used as the resource ID automatically. In this case, when visitors visit a page repeatedly, the same content is shown to them by Community-Gadgets. Therefore, it is recommended that the resource ID be equal to the page ID (if it is managed by a content management system), so the association is straightforward and intuitive.

When the same comment feed needs to be shown on two separate pages, it is sufficient to specify the same resource ID. For example, `xyz` can be used as the resource ID on both pages, so that the same content is rendered on these pages.

84.2 Adjusting Logging Levels

This section contains the following topics:

- Section 84.2.1, "Configuring log4j Loggers"
- Section 84.2.2, "Enabling Logging for SEO Widget JAR Files"

84.2.1 Configuring log4j Loggers

Configuration and monitoring of log files is an important aspect of product maintenance. This section describes log4j loggers that are available for troubleshooting/monitoring.

- The `log4j-cos.properties` file is the main configuration file for the logging in Community-Gadgets web application. The `log4j-shindig.properties` file is the main configuration file for the logging in Shindig web application. These files are located in the `standalone_node` folders (default locations are: `<cg_install_dir>/deploy/management/management_node1` and `<cg_install_dir>/deploy/production/production_node1`
- To enable all logging specific to Community-Gadgets, set the following loggers to debug level:

  ```
  log4j.logger.com.fatwire=DEBUG
  log4j.logger.oracle.fatwire.widgets=DEBUG
  ```

- To monitor all the requests that Community-Gadgets sends to the WebCenter Sites data repository, enable the following logger:
  `log4j.logger.com.fatwire.cos.core.jpa.cmd.wem.WemCommandManager = TRACE`
- To monitor all the requests that Community-Gadgets sends to network and which network proxy is used, enable the following logger:
  `log4j.logger.com.fatwire.cos.registry.client.local=TRACE`
- To monitor lifecycle management of CAS tickets that are used to connect to WebCenter Sites over REST, enable the following loggers:
  `log4j.logger.com.fatwire.cos.core.jpa.session.wem.WemSessionManager = TRACE`

  `log4j.logger.com.fatwire.cos.core.sites.wem.WemManager = TRACE`
- To monitor inCache operations performed by Community-Gadgets, enable the following logger:
  `log4j.logger.com.fatwire.cos.core.cache.appcache.wem.WemAppCacheFacility = TRACE`
- To monitor inCache invalidations coming from WebCenter Sites, enable the following logger:
  `log4j.logger.com.fatwire.cos.core.cache.appcache.wem.IncacheServlet = TRACE`
- When troubleshooting data consistency issues and the synchronization functionality (locking) in the system, enable the following loggers:
  `log4j.logger.com.fatwire.cos.cluster.CacheLockClientFacility = TRACE`

  `log4j.logger.com.fatwire.cos.cluster.ClusterLockImpl = TRACE`

84.2.2 Enabling Logging for SEO Widget JAR Files

When deploying widgets that support search engine optimization (SEO), you need to download the `cos-widget-tag.jar` file containing the widget deployment and rendering logic, and place it into the classpath of your web applications. Navigate to **Comments**, then **Deploy**, then **Comments**, and then select **Custom settings** and scroll down to the **Widget Tag** field. To download the JAR file, click the link in the Note section.

This is a lightweight JAR file, and it is created to function independently of any external logging libraries. However, to enable log messages provided by this library (`cos-widget-tag.jar`), it is necessary to add the `-Dcos.widget.tag.debug=true` Java parameter to JVM that runs the web application. This ensures that the log messages display in the standard output stream (console).

To optimize this library for performance, it is recommended that you add the following JVM parameters:

```
-Dhttp.keepAlive=true, -Dhttp.maxConnections=<Number>
```

The *<Number>* value must be aligned with the concurrency rates on the application server on which the pages with the SEO deployment tag were deployed, as well as with the number of processing threads available on the application server.

84.3 Reporting Issues

Before contacting the support team, compile the information required for investigation. This will help you avoid unnecessary communication round-trips and speed up the troubleshooting process.

The following is the recommended list of items to be provided to support for quick and efficient troubleshooting:

Environment Details

- Operating system name and version
- System language, locale, and time zone
- JVM name and version
- Application server name and version

Configuration Details

Archive the contents of the product configuration directory and attach the archive to the support ticket. For example, `Production_node1` or `Management_Node1`, configurations are saved in the `<install folder>/deploy/production` and `<install folder>/deploy/management` directories. Other required files are: the `setup_*.properties` files from the `<cg_install_dir>/deploy/management/management_node1` and `<cg_install_dir>/deploy/production/production_node1` directories.

Logging Details

- Log files of Community-Gadgets, its CAS and Shindig
- Log files of the WebCenter Sites application and its CAS
- Log files of Community-Gadgets and WebCenter Sites application servers (including the standard output and standard error consoles)

85

Community-Gadgets: Analyzing Community Widget Tags

Each community widget has its own deployment screen which displays the widget's tag. This chapter provides information about the parameters defined in each widget tag.

This chapter contains the following sections:

- Section 85.1, "Comments Widget Tags"
- Section 85.2, "Reviews Widget Tags"
- Section 85.3, "Ratings Widget Tags"
- Section 85.4, "Login Bar Widget Tag"
- Section 85.5, "Poll Widget Tags"

85.1 Comments Widget Tags

The widget tags related to adding commenting functionality to your website are the following:

- Section 85.1.1, "Comments Widget Tag"
- Section 85.1.2, "Comments Summary Widget Tag"
- Section 85.1.3, "Links to Topics Widget Tag"
- Section 85.1.4, "Recently Commented Widget Tag"
- Section 85.1.5, "Most Commented Widget Tag"

85.1.1 Comments Widget Tag

The Comments Deployment screen (which you can access from the menu bar by selecting **Comments**, **Deploy**, and then **Comments**) provides designers with the Comments widget tag. This section analyzes the parameters defined in the Comments widget tag:

Comments widget tag

```
<div id="comments_container"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
```

```
5.  cos.pageWidgets.push({name: "wsdk.comments",
6.  version: "1.5",
7.  elementID: "comments_container",
8.  attributes: {"site_id":"FirstSiteII"}});
9.  setTimeout(
10. function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.comments) ==
    'undefined')) {document.getElementById('comments_container').innerHTML = "<div
    style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size: 12px;color:
    #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top: 4px;height:
    22px;'>Comments is unavailable right now. Please contact the site
    administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.comments');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Comments widget tag

- Line 1 defines the container that holds the Comments widget on the page. If you assign a resource ID to this instance of the Comments widget, the widget's resource ID is also specified in this line. For example:

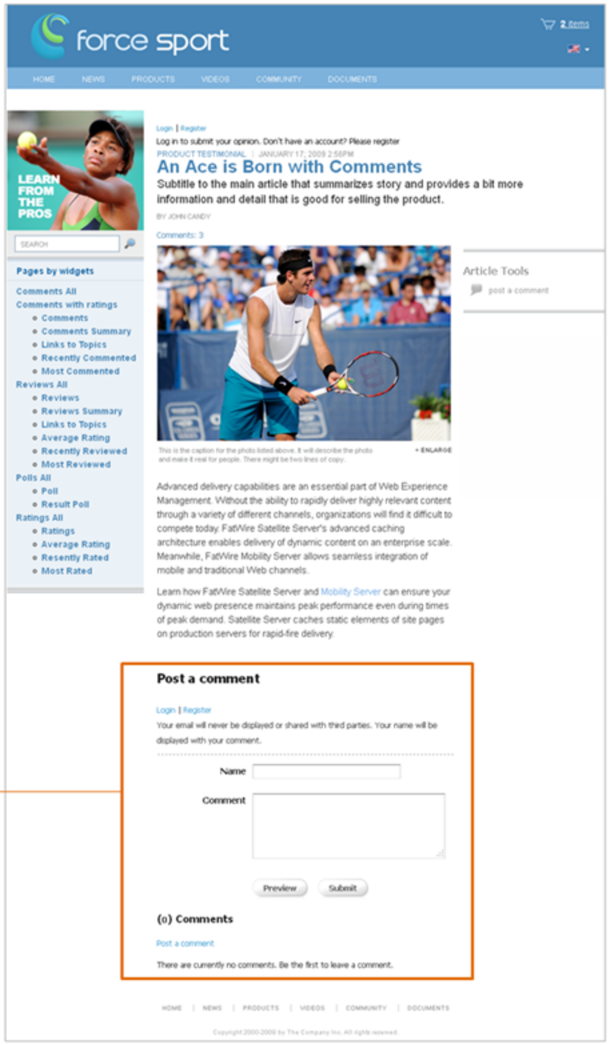
  ```
  <div id= "comments_container[resource_id]"></div>
  ```

  Where `[resource_id]` is the resource ID assigned to this instance of the Comments widget.

Note: If you do not specify a resource ID for this widget tag, the encrypted URL of the page on which this widget tag is deployed is assigned as the widget's resource ID.

- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Comments widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Comments widget inside the container defined in line 1.
- Lines 9 through 11 check if the Comments widget is rendered on the page. If the Comments widget is not rendered after 30 seconds, users are informed that there is an error and should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Comments widget tag (`wsdk.comments`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page that starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–1 Comments widget displayed on a web page



Login | Register

Log in to submit your opinion. Don't have an account? Please register

PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

Comments: 3

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content through a variety of different channels, organizations will find it difficult to compete today. FatWire Satellite Server's advanced caching architecture enables delivery of dynamic content on an enterprise scale. Meanwhile, FatWire Mobility Server allows seamless integration of mobile and traditional Web channels.

Learn how FatWire Satellite Server and Mobility Server can ensure your dynamic web presence maintains peak performance even during times of peak demand. Satellite Server caches static elements of site pages on production servers for rapid-fire delivery.

Post a comment

Login | Register

Your email will never be displayed or shared with third parties. Your name will be displayed with your comment.

(0) Comments

There are currently no comments. Be the first to leave a comment.

Copyright 2000-2009 by The Company Inc. All rights reserved.

Comments widget displaying a "Name" and "Comment" field

85.1.2 Comments Summary Widget Tag

The Comments Summary Deployment screen (which you can access from the menu bar by selecting **Comments**, **Deploy**, and then **Comments Summary**) provides designers with the Comments Summary widget tag. This section analyzes the parameters defined in the Comments Summary widget tag:

Comments Summary widget tag

```
<div id="comments_summary_container[resource_id]"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "comments-summary",
version: "0.1",
elementID: "comments_summary_container[resource_id]",
attributes: {"site_id":"FirstSiteII","resource_id":"[resource_id]","show_last_
comment_date":"false"}});
```

```
9.  setTimeout(
10. function(){if ((typeof(cos) == 'undefined')||(typeof(comments_summary) ==
    'undefined')) {document.getElementById('comments_summary_container').innerHTML
    = "<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size:
    12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top:
    4px;height: 22px;'>Comments Summary is unavailable right now. Please contact
    the site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('comments-summary');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Comments Summary widget tag

- Line 1 defines the container that holds the Comments Summary widget on the page. When you specify the resource ID of a Comments widget tag in the **Resource ID** field, that resource ID is also specified in this line. For example:

  ```
  <div id= "comments_summary_container[resource_id]"></div>
  ```

  Where `[resource_id]` is the resource ID of the desired Comments widget.

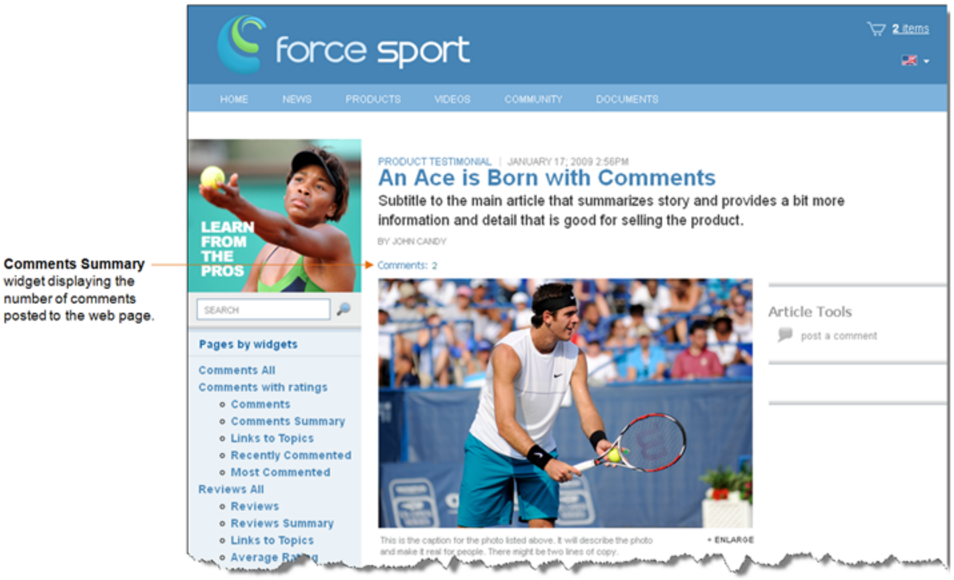
- Line 2 opens the bootstrapping JavaScript code needed for the widget.

- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.

- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.

- Lines 5 through 8 define the Comments Summary widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Comments Summary widget inside the container defined in line 1.
- Lines 9 through 11 check if the Comments Summary widget is rendered on the page. If the Comments Summary widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Comments Summary widget tag (`comments-summary`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–2 Comments Summary widget displayed on a web page



Comments Summary widget displaying the number of comments posted to the web page.

PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

Comments: 2

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

85.1.3 Links to Topics Widget Tag

The Links to Topics Deployment screen (which you can access from the menu bar by selecting **Comments**, **Deploy**, and then **Links to Topics**) provides designers with the

Links to Topics widget tag. This section analyzes the parameters defined in the Links to Topics widget tag:

Links to Topics widget tag

```
1.  <a href="[resource_url]#comments_link&resource_id=[resource_id]" ></a>
2.  <div id="comments_link_div[random]"></div>
3.  <script type="text/javascript">
4.  cos = window.cos || {};
5.  cos.pageWidgets = cos.pageWidgets || [];
6.  cos.pageWidgets.push({name: "comments-link",
7.  version: "0.1",
8.  elementID: "comments_link_div[random]",
9.  attributes: {"site_id":"FirstSiteII"}});
10. cos.pageScripts = cos.pageScripts || [];
11. cos.pageScripts.push('comments-link');
12. (function()
13. {
14. var oldOnloadHandler = window.onload || function()
15. {
16. };
17. if (!oldOnloadHandler.alreadyProcessed)
18. {
19. window.onload = function()
20. {
21. var script = document.createElement('script');
22. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
23. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
24. script.type = 'text/javascript';
25. script.charset = 'utf-8';
26. document.getElementsByTagName("head").item(0) .appendChild(script);
27. oldOnloadHandler.apply(this, arguments);
28. };
29. window.onload.alreadyProcessed = true;
30. }
31. })();
32. </script>
```

Analyzing the Links to Topics widget tag

- Line 1 is a link to the web page on which the desired Comments widget tag is deployed, and is added to the tag automatically when you enter values into the **Resource ID** and **Resource URL** fields in the Links to Topics Deployment screen.
- Line 2 defines the container that holds the Links to Topics widget on the page.
- Line 3 opens the bootstrapping JavaScript code needed for the widget.

- Line 4 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 5 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 6 through 9 define the Links to Topics widget tag and add the new element to the list of widget tags (defined in line 5). Line 8 contains a link that is used to render the Links to Topics widget inside the container defined in line 2.
- Line 10 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is left empty.
- Line 11 adds the Links to Topics widget tag (`comments-link`) to the list of page scripts (defined in line 10).
- Line 14 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 17 checks if the `onload` function is already processed.
- Line 19 overrides the code that runs when the page is loaded.
- Lines 21 through 26 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 27 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 29 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 32 closes the bootstrapping JavaScript code needed for the widget.

85.1.3.1 Deploying the Links to Topics Widget Tag (for Comments)

To configure the Links to Topics widget to display summarized information about and links to multiple web pages on which Comments (and Reviews) widgets are deployed, you will have to insert the resource ID and resource URL of each Comments (and/or Reviews) widget deployed on the desired web pages into the Links to Topics widget tag.

> **Note:** You can specify Comments widgets and Reviews widgets in the same Links to Topics widget tag. For information about specifying a Reviews widget in the Links to Topics tag, see Section 85.2.3.1, "Deploying the Links to Topics Widget Tag (For Reviews)."

For example:

1. In the menu bar, select **Comments**, **Deploy**, and then select **Links to Topics**.
2. In the Links to Topics Deployment screen, copy (**Ctrl+C**) the Links to Topics widget tag.
3. Access the WebCenter Sites Admin interface and insert the Links to Topics widget tag into the desired template. For instructions, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.

4. In line 1 of the Links to Topics widget tag, specify the resource ID of each deployed Comments (and/or Reviews) widget that the Links to Topics widget should display summarized information about and link to. For example:

   Add the following line for each Comments widget you wish to specify in the Links to Topics widget tag:

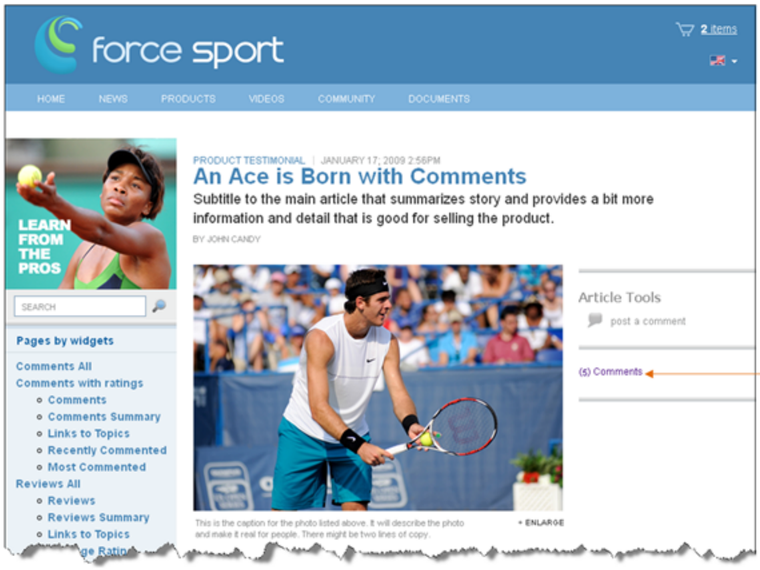
   ```
   <a href="http://URL#comments_link&resource_id=CommentsID"></a>
   ```

   Where `http://URL` is the path to the web page on which the desired Comments widget is deployed.

5. Click **Save Changes**.

6. Preview the page and then publish the template to the website. For instructions, see the *Oracle Fusion Middleware WebCenter Sites User's Guide* and the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

Figure 85–3 Links to Topics widget displayed on a web page


force sport
2 items
HOME
NEWS
PRODUCTS
VIDEOS
COMMUNITY
DOCUMENTS
LEARN FROM THE PROS
SEARCH
Pages by widgets
Comments All
Comments with ratings
Comments
Comments Summary
Links to Topics
Recently Commented
Most Commented
Reviews All
Reviews
Reviews Summary
Links to Topics
PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM
An Ace is Born with Comments
Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.
BY JOHN CANDY
Article Tools
post a comment
(5) Comments
This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.
+ ENLARGE

The **Links to Topics** widget displaying the number of comments posted to a given web page. Click this link to view the comments on the web page.

Note: When the Links to Topics widget is deployed, it does not display a title. For the Links to Topics widget to display a title, you will have to insert the title into the template to which you copied the Links to Topics widget tag.

85.1.4 Recently Commented Widget Tag

The Recently Commented Deployment screen (which you can access from the menu bar by selecting **Comments**, **Deploy**, and then **Recently Commented**) provides designers with the Recently Commented widget tag. This section analyzes the parameters defined in the Recently Commented widget tag:

Recently Commented widget tag

1. `<div id="topics_container_recently_commented_other"></div>`
2. `<script type="text/javascript">`
3. `cos = window.cos || {};`

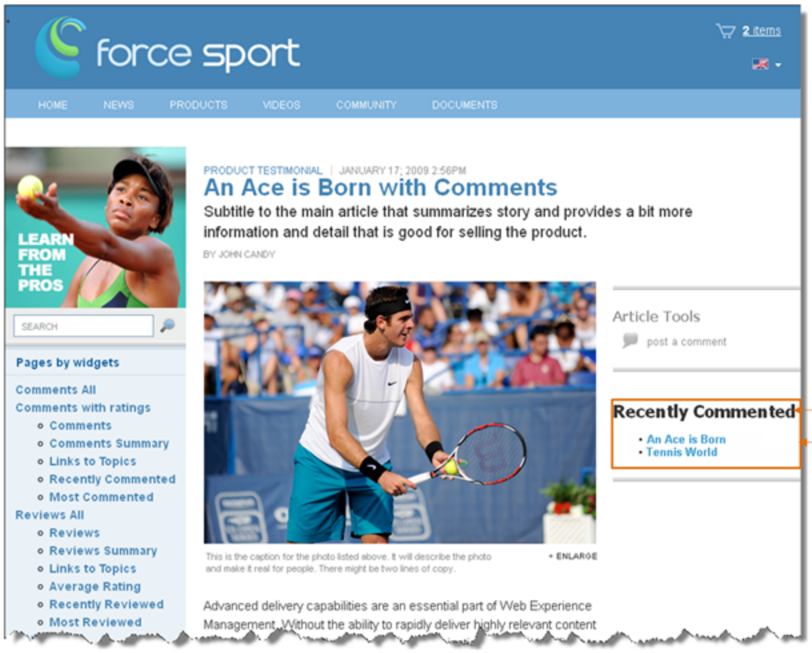
```
4.  cos.pageWidgets = cos.pageWidgets || [];
5.  cos.pageWidgets.push({name: "wsdk.topics",
6.  version: "1.5",
7.  elementID: "topics_container_recently_commented_other",
8.  attributes: {"content_type":"recently_commented","site_
    id":"FirstSiteII","resource_type":"other"}});
9.  setTimeout(
10. function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.topics) ==
    'undefined')) {document.getElementById('topics_container_recently_commented_
    other').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
    sans-serif;font-size: 12px;color: #333333;border: 1px solid
    #dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Topics is unavailable
    right now. Please contact the site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.topics');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Recently Commented widget tag

- Line 1 defines the container that holds the Recently Commented widget on the page.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the cos object on the page. The cos object contains all Community application functionality. If there are no community widget tags deployed on the page, then the cos object's value is empty.

- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Recently Commented widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Recently Commented widget inside the container defined in line 1. Line 8 specifies the Recently Commented widget tag's settings, including the type of topics the Recently Commented widget will list. If you want the Recently Commented widget to include topics of all types in its list, remove the `"resource_type":"other"` parameter.
- Lines 9 through 11 check if the Recently Commented widget is rendered on the page. If the Recently Commented widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Recently Commented widget tag (`wsdk.topics`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through28 add the JavaScript tag on the page, which includes all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–4 Recently Commented widget displayed on a web page



An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content

Title for this widget

Recently Commented widget displaying a list of web pages to which visitors have recently posted comments.

Note: When the Recently Commented widget is deployed, it does not display a title (as shown in Figure 85–4). For the Recently Commented widget to display a title, you will have to insert the title into the template to which you copied the Recently Commented widget tag.

85.1.5 Most Commented Widget Tag

The Most Commented Deployment screen (which you can access from the menu bar by selecting **Comments**, **Deploy**, and then **Most Commented**) provides designers with the Most Commented widget tag. This section analyzes the parameters defined in the Most Commented widget tag:

Most Commented widget tag

```
<div id="topics_container_most_commented_other"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "wsdk.topics",
version: "1.5",
elementID: "topics_container_most_commented_other",
attributes: {"content_type":"most_commented","site_id":"FirstSiteII","resource_
type":"other"}});
setTimeout(
function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.topics) ==
'undefined')) {document.getElementById('topics_container_most_commented_
other').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
sans-serif;font-size: 12px;color: #333333;border: 1px solid
```

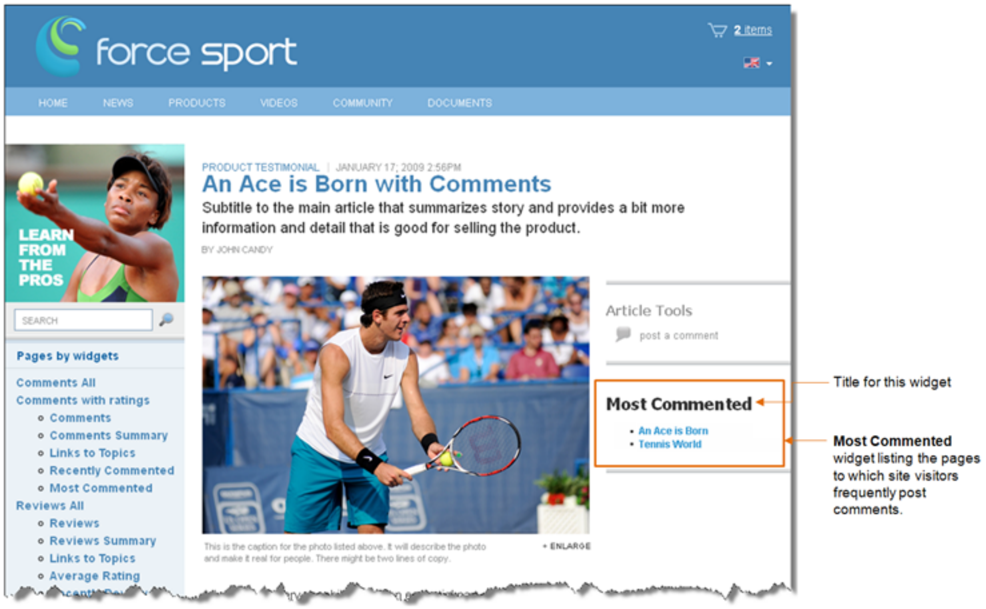
```
    #dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Topics is unavailable
    right now. Please contact the site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.topics');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Most Commented widget tag

- Line 1 defines the container that holds the Most Commented widget on the page.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Most Commented widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Most Commented widget inside the container defined in line 1. Line 8 specifies the Most Commented widget tag's settings, including the type of topics the Most Commented widget will list. If you want the Most Commented widget to include topics of all types in its list, remove the `"resource_type":"other"` parameter.
- Lines 9 through 11 check if the Most Commented widget is rendered on the page. If the Most Commented widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.

- Line 12 defines the list of page scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Most Commented widget tag (`wsdk.topics`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–5 Most Commented widget displayed on a web page



PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Most Commented

- An Ace is Born
- Tennis World

Title for this widget

Most Commented widget listing the pages to which site visitors frequently post comments.

Note: When the Most Commented widget is deployed, it does not display a title (as shown in Figure 85–5). For the Most Commented widget to display a title, you will have to insert the title into the template to which you copied the Most Commented widget tag.

85.2 Reviews Widget Tags

The widget tags related to adding reviewing functionality to your website are the following:

- Section 85.2.1, "Reviews Widget Tag"
- Section 85.2.2, "Reviews Summary Widget Tag"
- Section 85.2.3, "Links to Topics Widget Tag"
- Section 85.2.4, "Reviews Average Rating Widget Tag"
- Section 85.2.5, "Top Ranked Reviews Widget Tag"
- Section 85.2.6, "Recently Reviewed Widget Tag"
- Section 85.2.7, "Most Reviewed Widget Tag"

85.2.1 Reviews Widget Tag

The Reviews Deployment screen (which you can access from the menu bar by selecting **Reviews**, **Deploy**, and then **Reviews**) provides designers with the Reviews widget tag. This section analyzes the parameters defined in the Reviews widget tag:

Reviews widget tag

```
<div id="reviews_container"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "wsdk.reviews",
version: "1.5",
elementID: "reviews_container",
attributes: {"site_id":"FirstSiteII"}});
setTimeout(
function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.reviews) ==
'undefined')) {document.getElementById('reviews_container').innerHTML = "<div
style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size: 12px;color:
#333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top: 4px;height:
22px;'>Reviews is unavailable right now. Please contact the site
administrator.</div>";}}
,30000);
cos.pageScripts = cos.pageScripts || [];
cos.pageScripts.push('wsdk.reviews');
(function()
{
var oldOnloadHandler = window.onload || function()
{
};
if (!oldOnloadHandler.alreadyProcessed)
{
window.onload = function()
{
var script = document.createElement('script');
script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
+ cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Reviews widget tag

- Line 1 defines the container that holds the Reviews widget on the page. If you assign a resource ID to this instance of the Reviews widget tag, the widget's resource ID is also specified in this line. For example:

  ```
  <div id= "reviews_container[resource_id]"></div>
  ```

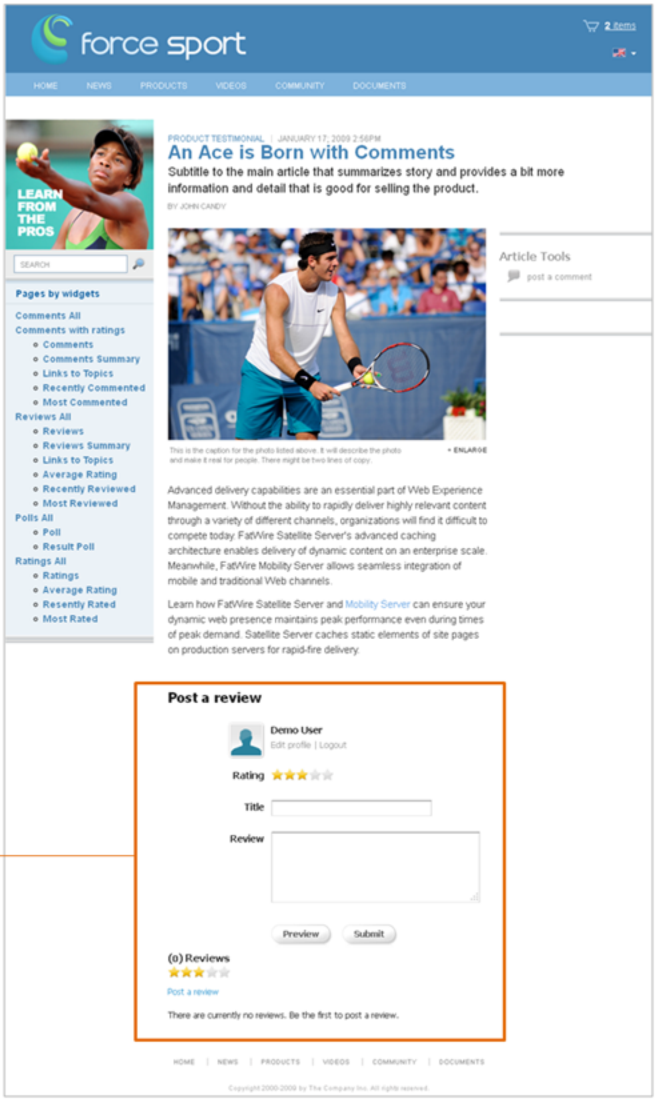
  Where `[resource_id]` is the resource ID assigned to this instance of the Reviews widget.

  > **Note:** If you do not specify a resource ID for this widget, the encrypted URL of the page on which this widget tag is deployed is assigned as the widget's resource ID.

- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Reviews widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Reviews widget inside the container defined in line 1.
- Lines 9 through 11 check if the Reviews widget is rendered on the page. If the Reviews widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrators.
- Line 12 defines the list of page scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Reviews widget tag (`wsdk.reviews`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.

- Lines 23 through 28 add the JavaScript tag to the page, which includes all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–6 Reviews widget displayed on a web page



PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content through a variety of different channels, organizations will find it difficult to compete today. FatWire Satellite Server's advanced caching architecture enables delivery of dynamic content on an enterprise scale. Meanwhile, FatWire Mobility Server allows seamless integration of mobile and traditional Web channels.

Learn how FatWire Satellite Server and Mobility Server can ensure your dynamic web presence maintains peak performance even during times of peak demand. Satellite Server caches static elements of site pages on production servers for rapid-fire delivery.

Post a review

(0) Reviews

There are currently no reviews. Be the first to post a review.

Copyright 2000-2009 by The Company Inc. All rights reserved.

Reviews widget displaying a "Rating," "Title," and "Review" field.

85.2.2 Reviews Summary Widget Tag

The Reviews Summary Deployment screen (which you can access from the menu bar by selecting **Reviews**, **Deploy**, and then **Reviews Summary**) provides designers with the Reviews Summary widget tag. This section analyzes the parameters defined in the Reviews Summary widget tag:

Reviews Summary widget tag

```
1.  <div id="reviews_summary_container[resource_id]"></div>
2.  <script type="text/javascript">
3.  cos = window.cos || {};
4.  cos.pageWidgets = cos.pageWidgets || [];
5.  cos.pageWidgets.push({name: "reviews-summary",
6.  version: "0.1",
7.  elementID: "reviews_summary_container[resource_id]",
8.  attributes: {"site_id":"FirstSiteII","resource_id":"[resource_id]","show_last_
    comment_date
9.  setTimeout(
10. function(){if ((typeof(cos) == 'undefined')||(typeof(cos.reviews-summary) ==
    'undefined')) {document.getElementById('reviews_summary_container').innerHTML =
    "<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size:
    12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top:
    4px;height: 22px;'>Reviews Summary is unavailable right now. Please contact the
    site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('reviews-summary');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Reviews Summary widget tag

- Line 1 defines the container that holds the Reviews Summary widget on the page. If you specify the resource ID of a Reviews widget in this instance of the Reviews

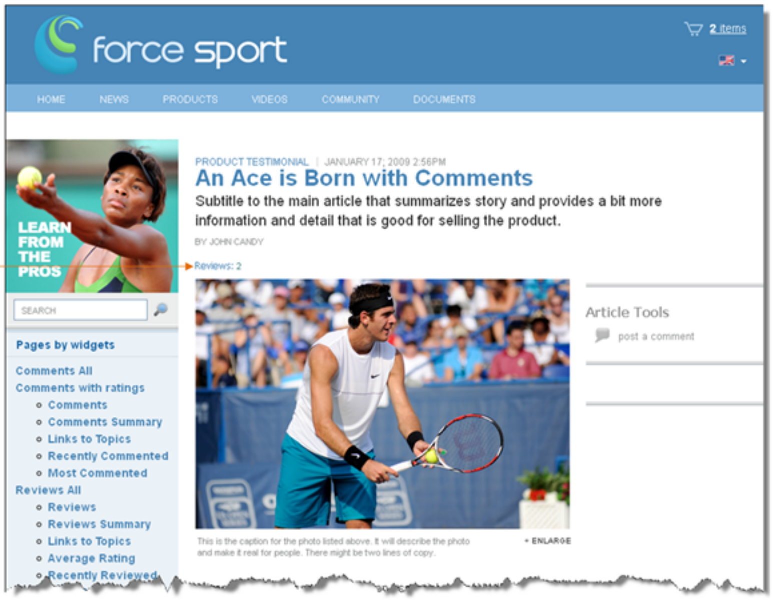
Summary widget tag, the resource ID for that Reviews widget is also specified in this line. For example:

```
<div id= "reviews_summary_container[resource_id]></div>
```

Where `[resource_id]` is the resource ID of the desired Reviews widget.

- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Reviews Summary widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Reviews Summary widget inside the container defined in line 1.
- Lines 9 through 11 check if the Reviews Summary widget is rendered on the page. If the Reviews Summary widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Reviews Summary widget tag (`reviews-summary`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag to the page, which includes all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–7 Reviews Summary widget displayed on a web page



An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Reviews Summary widget displaying the number of reviews posted to the web page.

85.2.3 Links to Topics Widget Tag

The Links to Topics Deployment screen (which you can access from the menu bar by selecting **Reviews**, **Deploy**, **Links to Topics**) provides designers with the Links to Topics widget tag. This section analyzes the parameters defined in the Links to Topics widget tag:

Links to Topics widget tag

```
<a href="[resource_url]#reviews_link&resource_id=[resource_id]" ></a>
<div id="comments_link_div[random]"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "comments-link",
version: "0.1",
elementID: "comments_link_div[random]",
attributes: {"site_id":'FirstSiteII'}});
cos.pageScripts = cos.pageScripts || [];
cos.pageScripts.push('comments-link');
(function()
{
var oldOnloadHandler = window.onload || function()
{
};
if (!oldOnloadHandler.alreadyProcessed)
{
window.onload = function()
20. {
21. var script = document.createElement('script');
22. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
23. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
24. script.type = 'text/javascript';
25. script.charset = 'utf-8';
26. document.getElementsByTagName("head").item(0) .appendChild(script);
27. oldOnloadHandler.apply(this, arguments);
28. };
29. window.onload.alreadyProcessed = true;
30. }
31. })();
32. </script>
```

Analyzing the Links to Topics widget tag

- Line 1 is a link to the web page on which the desired Reviews widget tag is deployed, and is added to the tag automatically when you enter values into the **Resource ID** and **Resource URL** fields in the Links to Topics Deployment screen.
- Line 2 defines the container that holds the Links to Topics widget on the page.
- Line 3 opens the bootstrapping JavaScript code needed for the widget.
- Line 4 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 5 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 6 through 9 define the Links to Topics widget tag and add the new element to the list of widget tags (defined in line 5). Line 8 contains a link that is used to render the Links to Topics widget inside the container defined in line 2.
- Line 10 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 11 adds the Links to Topics widget tag (`comments-link`) to the list of page scripts (defined in line 10).
- Line 14 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic, then the function is left empty.
- Line 17 checks if the `onload` function is already processed.
- Line 19 overrides the code that runs when the page is loaded.
- Lines 21 through 26 add the JavaScript tag to the page, which includes all the scripts that need to be deployed.
- Line 27 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 29 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.

- Line 32 closes the bootstrapping JavaScript code needed for the widget.

85.2.3.1 Deploying the Links to Topics Widget Tag (For Reviews)

To configure the Links to Topics widget to display summarized information about and links to multiple web pages on which Reviews (and/or Comments) widgets are deployed, you will have to insert the resource ID and resource URL of each Reviews (and/or Comments) widget deployed on the desired web pages into the Links to Topics widget tag.

> **Note:** You can specify Reviews widgets and Comments widgets in the same Links to Topics widget tag. For information about specifying a Comments widget, see Section 85.1.3.1, "Deploying the Links to Topics Widget Tag (for Comments)."

For example:

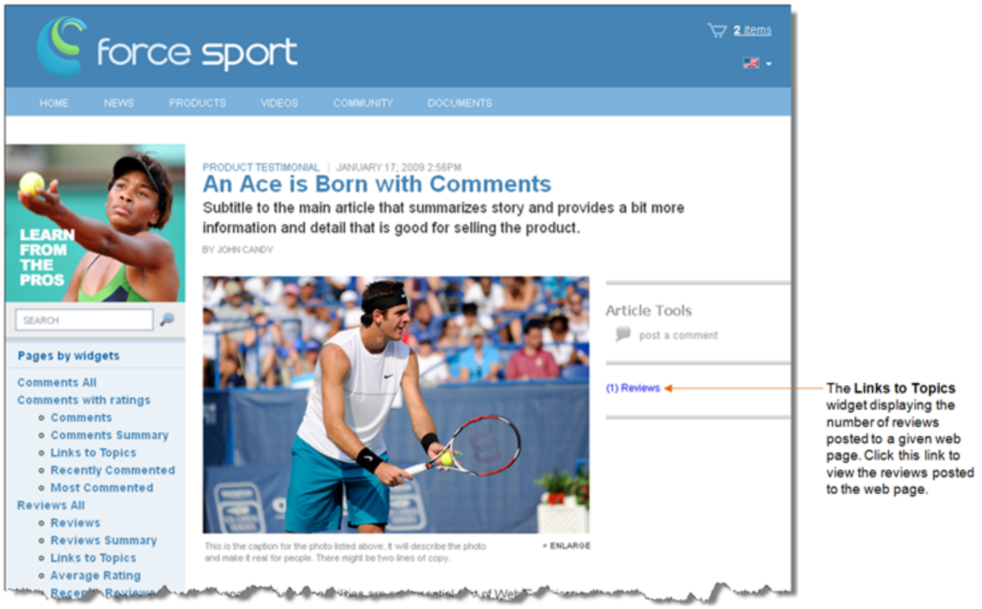
1. In the menu bar, select **Reviews**, **Deploy**, and then select **Links to Topics**.
2. In the Links to Topics Deployment screen, copy (**Ctrl+C**) the Links to Topics widget tag.
3. Access the WebCenter Sites Admin interface and insert the Links to Topics widget tag into the desired template. For instructions, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.
4. In line 1 of the Links to Topics widget tag, specify the resource ID and resource URL of each deployed Reviews (and/or Comments) widget that the Links to Topics widget should display summarized information about and link to. For example:

   Add the following line for each Reviews widget you wish to specify in the Links to Topics widget tag:

   ```
   <a href="http://URL#reviews_link&resource_id+ReviewsID"></a>
   ```

   Where `http://URL` is the path to the web page on which the desired Reviews widget is deployed.
5. Click **Save Changes**.
6. Preview the page and then publish the template to the website. For instructions, see the *Oracle Fusion Middleware WebCenter Sites User's Guide* and the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

Figure 85–8 Links to Topics widget displayed on a web page



PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

The **Links to Topics** widget displaying the number of reviews posted to a given web page. Click this link to view the reviews posted to the web page.

Note: When the Links to Topics widget is deployed, it does not display a title. For the Links to Topics widget to display a title, you will have to insert the title into the template to which you copied the Links to Topics widget tag.

85.2.4 Reviews Average Rating Widget Tag

The Reviews Average Rating Deployment screen (which you can access from the menu bar by selecting **Reviews**, **Deploy**, and then **Average Rating**) provides designers with the Reviews Average Rating widget tag. This section analyzes the parameters defined in the Reviews Average Rating widget tag:

Reviews Average Rating widget tag

```
<divid="reviews_average_container[resource_id]"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "wsdk.reviews_average",
version: "1.5",
elementID: "reviews_average_container[resource_id]",
attributes: {"site_id":"FirstSiteII","resource_id":"[resource_id]"}});
setTimeout(
function(){if ((typeof(wsdk) == 'undefined')||(typeof(wsdk.reviews_average) ==
'undefined')) {document.getElementById('reviews_average_container[resource_
id]').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
sans-serif;font-size: 12px;color: #333333;border: 1px solid
#dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Reviews Average
Rating is unavailable right now. Please contact the site
administrator.</div>";}}
               ,30000);
```

```
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.reviews_average');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Reviews Average Rating widget tag

- Line 1 defines the container that holds the Reviews Average Rating widget on the page. You must specify the resource ID of the Reviews widget that is deployed on the web page whose average rating the Reviews Average Rating widget should display. For example:

  ```
  <div id="reviews_average_container[resource_id]"></div>
  ```

  Where `[resource_id]` is the resource ID of the desired Reviews widget.

- Line 2 opens the bootstrapping JavaScript code that is needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Reviews Average Rating widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Reviews Average Rating widget inside the container defined in line 1.
- Lines 9 through 11 check if the Reviews Average Rating widget is rendered on the page. If the Reviews Average Rating widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.

- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Reviews Average Rating widget tag (`wsdk.reviews_average`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add a JavaScript tag to the page, which includes a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

85.2.5 Top Ranked Reviews Widget Tag

The Top Ranked Reviews Deployment screen (which you can access from the menu bar by selecting **Reviews**, **Deploy**, and then **Top Ranked Topics**) provides designers with the Top Ranked Reviews widget tag. This section analyzes the parameters defined in the Top Ranked Topics widget tag:

Top Ranked Reviews widget tag

```
1.  <div id="topics_container_top_ranked_other"></div>
2.  <script type="text/javascript">
3.  cos = window.cos || {};
4.  cos.pageWidgets = cos.pageWidgets || [];
5.  cos.pageWidgets.push({name: "wsdk.topics",
6.  version: "1.5",
7.  elementID: "topics_container_top_ranked_other",
8.  attributes: {"content_type":"top_ranked","site_id":"FirstSiteII","resource_
    type":"other","rating_type":"stars"}});
9.  setTimeout(
10. function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.topics) ==
    'undefined')) {document.getElementById('topics_container_top_ranked_
    other').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
    sans-serif;font-size: 12px;color: #333333;border: 1px solid
    #dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Topics is unavailable
    right now. Please contact the site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.topics');
14. (function()
15. {
```

```
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

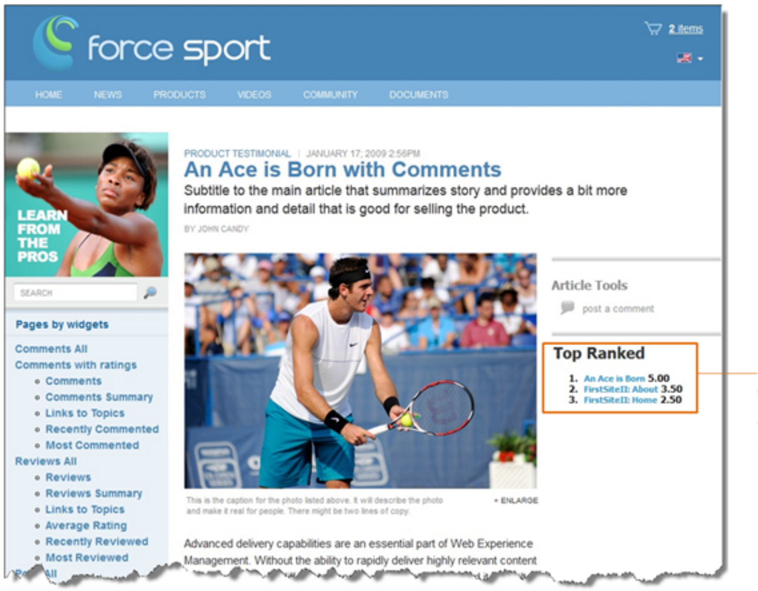
Analyzing the Top Ranked Reviews widget tag

- Line 1 defines the container that holds the Top Ranked Reviews widget on the page.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Top Ranked Reviews widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Top Ranked Reviews widget inside the container defined in line 1. Line 8 specifies the Top Ranked Reviews widget tag's settings, including the type of topics the Top Ranked Reviews widget will list. If you want the Top Ranked Reviews widget to include topics of all types in its list, remove the `"resource_type":"other"` parameter.
- Lines 9 through 11 check if the Top Ranked Reviews widget is rendered on the page. If the Top Ranked Reviews widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Top Ranked Reviews widget tag (`wsdk.topics`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the

function that implements it. If there is no such logic deployed, then the function is left empty.

- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–9 Top Ranked Reviews widget displayed on a web page



PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content

Top Ranked

1. An Ace is Born 5.00
2. FirstSiteII: About 3.50
3. FirstSiteII: Home 2.50

Top Ranked Reviews widget displaying the names of the highest rated web pages and the average rating of each page.

85.2.6 Recently Reviewed Widget Tag

The Recently Reviewed Deployment screen (which you can access from the menu bar by selecting **Reviews**, **Deploy**, and then **Recently Reviewed**) provides designers with the Recently Reviewed widget tag. This section analyzes the parameters defined in the Recently Reviewed widget tag:

Recently Reviewed widget tag

```
1. <div id="topics_container_recently_reviewed_other"></div>
2. <script type="text/javascript">
3. cos = window.cos || {};
4. cos.pageWidgets = cos.pageWidgets || [];
5. cos.pageWidgets.push({name: "wsdk.topics",
6. version: "1.5",
7. elementID: "topics_container_recently_reviewed_other",
8.  attributes: {"content_type":"recently_reviewed","site_
    id":"FirstSiteII","resource_type":"other"}});
9.  setTimeout(
10. function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.topics) ==
    'undefined')) {document.getElementById('topics_container_recently_reviewed_
    other').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
    sans-serif;font-size: 12px;color: #333333;border: 1px solid
    #dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Topics is unavailable
    right now. Please contact the site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.topics');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

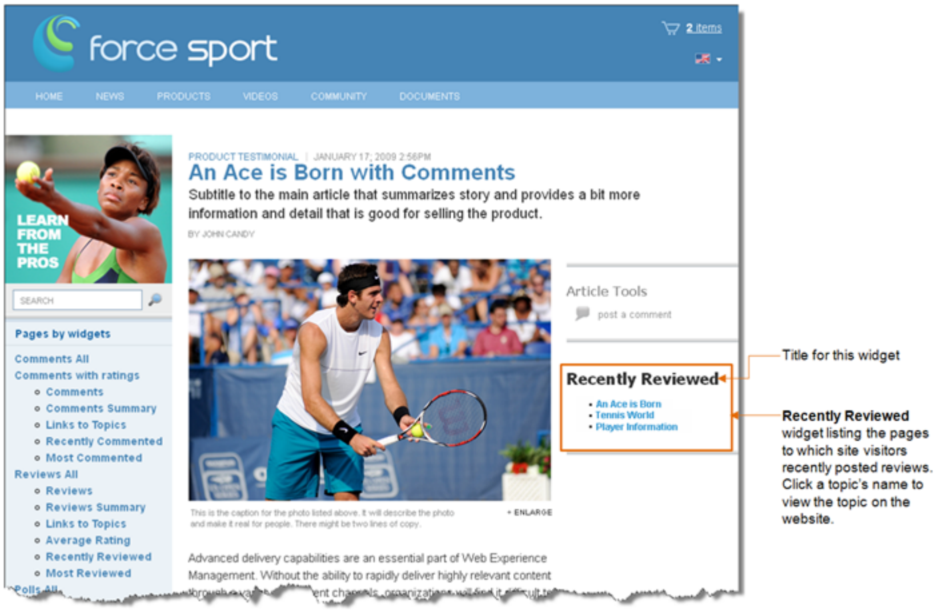
Analyzing the Recently Reviewed widget tag

- Line 1 defines the container that holds the Recently Reviewed widget tag on the page.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widget tags deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Recently Reviewed widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is

used to render the Recently Reviewed widget inside the container defined in line 1. Line 8 specifies the Recently Reviewed widget tag's settings, including the type of topics the Recently Reviewed widget will list. If you want the Recently Reviewed widget to include topics of all types in its list, remove the `"resource_type":"other"` parameter.

- Lines 9 through 11 check if the Recently Reviewed widget is rendered on the page. If the Recently Reviewed widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Recently Reviewed widget tag (`wsdk.topics`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag to the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–10 Recently Reviewed widget displayed on a web page



PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content through a variety of different channels, organizations will find it difficult to

Recently Reviewed

- An Ace is Born
- Tennis World
- Player Information

Title for this widget

Recently Reviewed widget listing the pages to which site visitors recently posted reviews. Click a topic's name to view the topic on the website.

Note: When the Recently Reviewed widget is deployed, it does not display a title (as shown in Figure 85–10). For the Recently Reviewed widget to display a title, you will have to insert the title into the template to which you copied the Recently Reviewed widget tag.

85.2.7 Most Reviewed Widget Tag

The Most Reviewed Deployment screen (which you can access from the menu bar by selecting **Reviews**, **Deploy**, and then **Most Reviewed**) provides designers with the Most Reviewed widget tag. This section analyzes the parameters defined in the Most Reviewed widget tag:

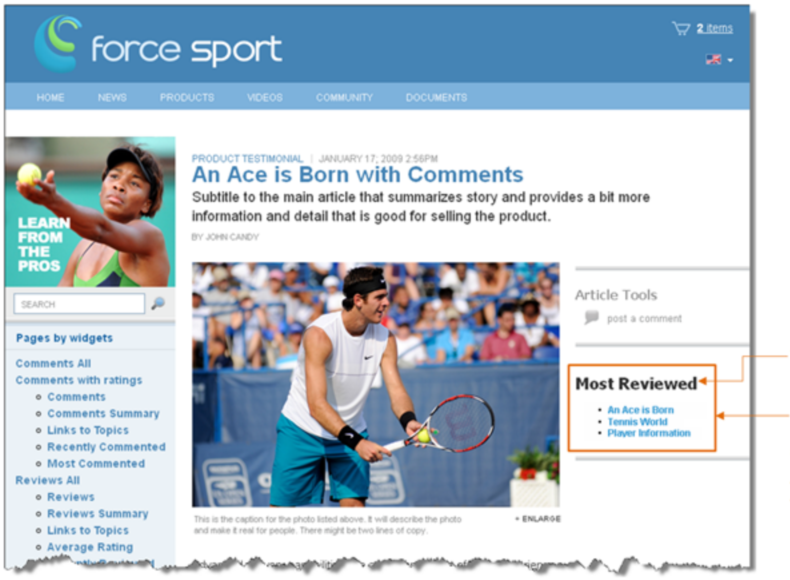
Most Reviewed widget tag

```
<div id="topics_container_most_reviewed_other"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "wsdk.topics",
version: "1.5",
elementID: "topics_container_most_reviewed_other",
attributes: {"content_type":"most_reviewed","site_id":"FirstSiteII","resource_
type":"other"}});
setTimeout(
function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.topics) ==
'undefined')) {document.getElementById('topics_container_most_reviewed_
other').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
sans-serif;font-size: 12px;color: #333333;border: 1px solid
#dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Topics is unavailable
right now. Please contact the site administrator.</div>";}}
,30000);
cos.pageScripts = cos.pageScripts || [];
cos.pageScripts.push('wsdk.topics');
(function()
{
var oldOnloadHandler = window.onload || function()
{
};
if (!oldOnloadHandler.alreadyProcessed)
{
window.onload = function()
{
var script = document.createElement('script');
script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
+ cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
script.type = 'text/javascript';
script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Most Reviewed widget tag

- Line 1 defines the container that holds the Most Reviewed widget on the page.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Most Reviewed widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Most Reviewed widget inside the container defined in line 1. Line 8 specifies the Most Reviewed widget tag's settings, including the type of topics the Most Reviewed widget will list. If you want the Most Reviewed widget to include topics of all types in its list, remove the `"resource_type":"other"` parameter.
- Lines 9 through 11 check if the Most Reviewed widget is rendered on the page. If the Most Reviewed widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no such scripts deployed, then the list's value is empty.
- Line 13 adds the Most Reviewed widget tag (`wsdk.topics`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag to the page, including a list of scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–11 Most Reviewed widget displayed on a web page



Title for this widget

Most Reviewed widget listing the topics to which site visitors frequently post reviews. Click a topic's name to view the topic on the website.

> **Note:** When the Most Reviewed widget is deployed, it does not display a title (as shown in Figure 85–11). For the Most Reviewed widget to display a title, you will have to insert the title into the template to which you copied the Most Reviewed widget tag.

85.3 Ratings Widget Tags

The widget tags related to adding rating functionality to your website are the following:

- Section 85.3.1, "Stars Ratings Widget Tag"
- Section 85.3.2, "Thumbs Up/Down Ratings Widget Tag"
- Section 85.3.3, "Like It Ratings Widget Tag"
- Section 85.3.4, "Recommend Ratings Widget Tag"
- Section 85.3.5, "Ratings Average Rating Widget Tag"
- Section 85.3.6, "Recently Rated Widget Tag"
- Section 85.3.7, "Most Rated Widget Tag"

85.3.1 Stars Ratings Widget Tag

The Stars Ratings Deployment screen (which you can access from the menu bar by selecting **Ratings**, **Deploy**, and then **Stars Ratings**) provides designers with the Stars Ratings widget tag. This section analyzes the parameters defined in the Stars Ratings widget tag:

Stars Ratings widget tag

```
1.  <div id="ratingsstars_container"></div>
2.  <script type="text/javascript">
3.  cos = window.cos || {};
```

```
4.  cos.pageWidgets = cos.pageWidgets || [];
5.  cos.pageWidgets.push({name: "wsdk.ratings",
6.  version: "1.5",
7.  elementID: "ratingsstars_container",
8.  attributes: {"rating_type":"stars","site_id":"FirstSiteII"}});
9.  setTimeout(
10. function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.ratings) ==
    'undefined')) {document.getElementById('ratings_stars_container').innerHTML =
    "<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size:
    12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top:
    4px;height: 22px;'>Ratings is unavailable right now. Please contact the site
    administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.ratings');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Stars Ratings widget tag

- Line 1 defines the container that holds the Stars Ratings widget on the page. If you assign a resource ID to this instance of the Stars Ratings widget tag, the widget's resource ID is also specified in this line. For example:

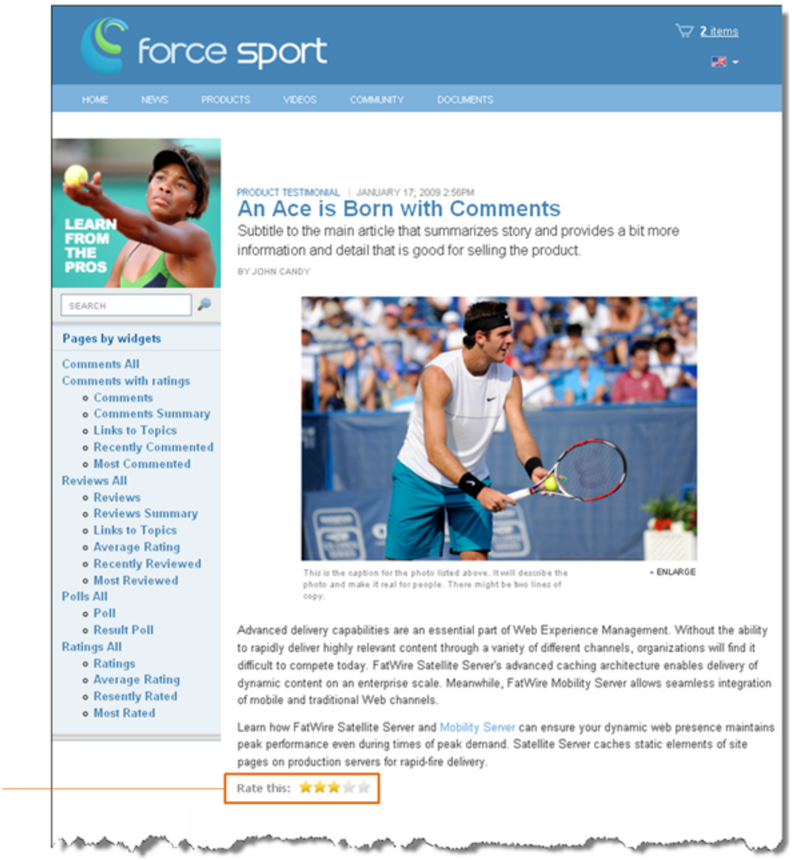
  ```
  <divid="ratingsstars_container[resource_id]"></div>
  ```

  Where `[resource_id]` is the resource ID assigned to this instance of the Stars Ratings widget.

Note: If you do not specify a resource ID for this widget tag, the encrypted URL of the page on which this widget tag is deployed is assigned as the widget's resource ID.

- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Stars Ratings widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Stars Ratings widget inside the container defined in line 1.
- Lines 9 through 11 check if the Stars Ratings widget is rendered on the page. If the Stars Ratings widget is not rendered after 30 seconds, users are informed that there is an error and that they should contact the site administrator.
- Line 12 defines the list of scripts that are deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Stars Ratings widget tag (`wsdk.ratings`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–12 Stars Ratings widget deployed on a web page



PRODUCT TESTMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content through a variety of different channels, organizations will find it difficult to compete today. FatWire Satellite Server's advanced caching architecture enables delivery of dynamic content on an enterprise scale. Meanwhile, FatWire Mobility Server allows seamless integration of mobile and traditional Web channels.

Learn how FatWire Satellite Server and Mobility Server can ensure your dynamic web presence maintains peak performance even during times of peak demand. Satellite Server caches static elements of site pages on production servers for rapid-fire delivery.

Stars Ratings widget. Rate the topic by clicking the desired number of stars.

85.3.2 Thumbs Up/Down Ratings Widget Tag

The Thumbs Up/Down Ratings Deployment screen (which you can access from the menu bar by selecting **Ratings**, **Deploy**, and then **Thumbs Up/Down Ratings**) provides designers with the Thumbs Up/Down Ratings widget tag. This section analyzes the parameters defined in the Thumbs Up/Down Ratings widget tag:

Thumbs Up/Down Ratings widget tag

1. `<div id="ratingsthumbs_container"></div>`
2. `<script type="text/javascript">`
3. `cos = window.cos || {};`
4. `cos.pageWidgets = cos.pageWidgets || [];`
5. `cos.pageWidgets.push({name: "wsdk.ratings",`
6. `version: "1.5",`
7. `elementID: "ratingsthumbs_container",`
8. `attributes: {"rating_type":"thumbs","site_id":"FirstSiteII"}});`
9. `setTimeout(`
10. `function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.ratings) == 'undefined')) {document.getElementById('ratings_thumbs_container').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size: 12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Ratings is unavailable right now. Please contact the site administrator.</div>";}}`

```
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.ratings');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Thumbs Up/Down Ratings widget tag

- Line 1 defines the container that holds the Thumbs Up/Down Ratings widget on the page. If you assign a resource ID to this instance of the Thumbs Up/Down Ratings widget, the resource ID is also specified in this line. For example:

  ```
  <div id="ratingsthumbs_container[resource_id]"></div>
  ```

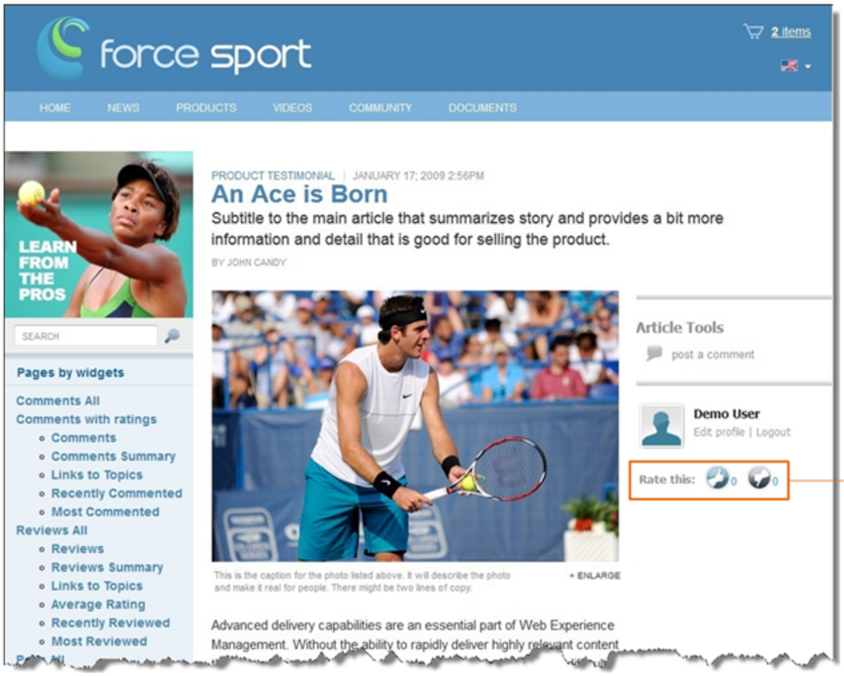
  Where `[resource_id]` is the resource ID assigned to this instance of the Thumbs Up/Down Ratings widget.

  > **Note:** If you do not specify a resource ID for this widget tag, the encrypted URL of the page on which this widget tag is deployed is assigned as the widget's resource ID.

- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.

- Lines 5 through 8 define the Thumbs Up/Down Ratings widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Thumbs Up/Down Ratings widget inside the container defined in line 1.
- Lines 9 through 11 check if the Thumbs Up/Down Ratings widget is rendered on the page. If the Thumbs Up/Down Ratings widget is not rendered after 30 seconds, users are informed that there is an error and that they should contact the site administrator.
- Line 12 defines the list of scripts that are deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Thumbs Up/Down Ratings widget tag (`wsdk.ratings`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–13 Thumbs Up/Down Ratings widget displayed on a web page



PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content

Thumbs Up/Down Ratings widget displaying a field which visitors can use to indicate whether the topic is informative.

85.3.3 Like It Ratings Widget Tag

The Like It Ratings Deployment screen (which you can access from the menu bar by selecting **Ratings**, **Deploy**, and then **Like It Ratings**) provides designers with the Like It Ratings widget tag. This section analyzes the parameters defined in the Like It Ratings widget tag:

Like It Ratings widget tag

```
<div id="ratingslike_it_container"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "wsdk.ratings",
version: "1.5",
elementID: "ratingslike_it_container",
attributes: {"rating_type":"like_it","site_id":"FirstSiteII"}});
setTimeout(
function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.ratings) ==
'undefined')) {document.getElementById('ratingslike_it_container').innerHTML =
"<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size:
12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top:
4px;height: 22px;'>Ratings is unavailable right now. Please contact the site
administrator.</div>";}}
,30000);
cos.pageScripts = cos.pageScripts || [];
cos.pageScripts.push('wsdk.ratings');
(function()
{
var oldOnloadHandler = window.onload || function()
{
};
if (!oldOnloadHandler.alreadyProcessed)
{
window.onload = function()
{
var script = document.createElement('script');
script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
+ cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
script.type = 'text/javascript';
script.charset = 'utf-8';
document.getElementsByTagName("head").item(0) .appendChild(script);
oldOnloadHandler.apply(this, arguments);
};
window.onload.alreadyProcessed = true;
}
})();
```

```
34. </script>
```

Analyzing the Like It Ratings widget tag

- Line 1 defines the container that holds the Like It Ratings widget on the page. If you assign a resource ID to this instance of the Like It Ratings widget, the widget's resource ID is also specified in this line. For example:

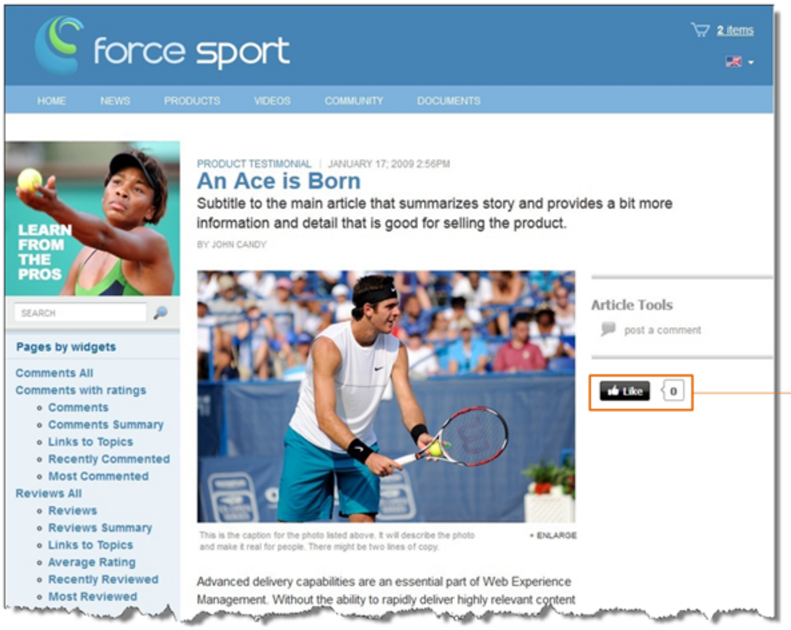
  ```
  <div id="ratingslike_it_container[resource_id]"></div>
  ```

  Where `[resource_id]` is the resource ID assigned to this instance of the Like It Ratings widget.

  > **Note:** If you do not specify a resource ID for this widget tag, the encrypted URL of the page on which this widget tag is deployed is assigned as the widget's resource ID.

- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Like It Ratings widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Like It Ratings widget tag inside the container defined in line 1.
- Lines 9 through 11 check if the Like It Ratings widget is rendered on the page. If the Like It Ratings widget is not rendered after 30 seconds, users are informed that there is an error and that they should contact the site administrator.
- Line 12 defines the list of scripts that are deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Like It Ratings widget tag (`wsdk.ratings`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–14 Like It Ratings widget displayed on a web page



An Ace is Born

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content

Like it Ratings widget displaying a **Like** link which visitors can click to indicate whether they liked a given topic.

85.3.4 Recommend Ratings Widget Tag

The Recommend Ratings Deployment screen (which you can access from the menu bar by selecting **Ratings**, **Deploy**, and then **Recommend Ratings**) provides designers with the Recommend Ratings widget tag. This section analyzes the parameters defined in the Recommend Ratings widget tag:

Recommend Ratings widget tag

```
<div id="ratingsrecommend_container"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "wsdk.ratings",
version: "1.5",
elementID: "ratingsrecommend_container",
attributes: {"rating_type":"recommend","site_id":"FirstSiteII"}});
setTimeout(
function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.ratings) ==
'undefined')) {document.getElementById('ratingsrecommend_container').innerHTML
= "<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size:
12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top:
4px;height: 22px;'>Ratings is unavailable right now. Please contact the site
administrator.</div>";}}
,30000);
cos.pageScripts = cos.pageScripts || [];
cos.pageScripts.push('wsdk.ratings');
(function()
{
```

```
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Recommend Ratings widget tag

- Line 1 defines the container that holds the Recommend Ratings widget on the page. If you assign a resource ID to this instance of the Recommend Ratings widget, the widget's resource ID is also specified in this line. For example:

  ```
  <div id="ratingsrecommend_container[resource_id]"></div>
  ```

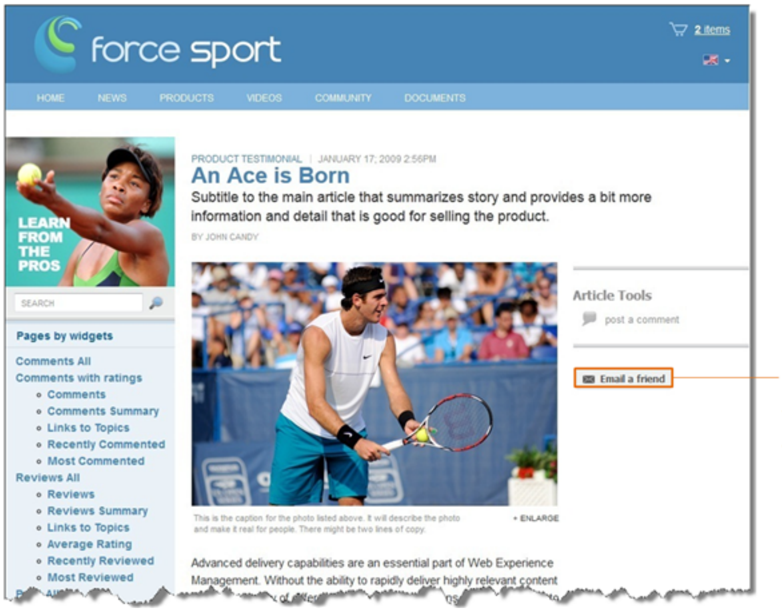
  Where `[resource_id]` is the resource ID assigned to this instance of the Recommend Ratings widget.

  > **Note:** If you do not specify a resource ID for this widget tag, the encrypted URL of the page on which this widget tag is deployed is assigned as the widget's resource ID.

- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Recommend Ratings widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Recommend Ratings widget inside the container defined in line 1.

- Lines 9 through 11 check if the Recommend Ratings widget is rendered on the page. If the Recommend Ratings widget is not rendered after 30 seconds, users are informed that there is an error and that they should contact the site administrator.
- Line 12 defines the list of scripts that are deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Recommend Ratings widget tag (`wsdk.ratings`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–15 Recommend Ratings widget displayed on a web page



PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content

Recommend Ratings widget displaying an Email a Friend link which visitors can click to email the URL of the web page to a contact who may find the content informative.

85.3.5 Ratings Average Rating Widget Tag

The Ratings Average Rating Deployment screen (which you can access from the menu bar by selecting **Ratings**, **Deploy**, and then **Average Rating**) provides designers with the Ratings Average Rating widget tag. This section analyzes the parameters defined in the Ratings Average Rating widget tag:

Ratings Average Rating widget tag

```
1.  <div id="ratings_average_container[resource_id]"></div>
2.  <script type="text/javascript">
3.  cos = window.cos || {};
4.  cos.pageWidgets = cos.pageWidgets || [];
5.  cos.pageWidgets.push({name: "wsdk.ratings_average",
6.  version: "1.5",
7.  elementID: "ratings_average_container[resource_id]",
8.  attributes: {"site_id":"FirstSiteII","resource_id":"[resource_id]"}});
9.  setTimeout(
10. function(){if ((typeof(wsdk) == 'undefined')||(typeof(wsdk.ratings_average) ==
    'undefined')) {document.getElementById('ratings_average_container').innerHTML =
    "<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size:
    12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top:
    4px;height: 22px;'>Average Rating is unavailable right now. Please contact the
    site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.ratings_average');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Ratings Average Rating widget tag

- Line 1 defines the container that holds the Ratings Average Rating widget on the page. You must specify the resource ID of the Stars Ratings widget tag deployed on the web page whose average rating the Ratings Average Ratings widget should display. For example:

```
<div id="ratings_average_container[resource_id]"></div>
```

Where `[resource_id]` is the resource ID of the Stars Ratings widget deployed on the web page whose average rating the Ratings Average Rating widget should display.

- Line 2 opens the bootstrapping JavaScript code that is needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Ratings Average Rating widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Ratings Average Rating widget tag inside the container defined in line 1.
- Lines 9 through 11 check if the Ratings Average Ratings widget is rendered on the page. If the Ratings Average Rating widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Ratings Average Rating widget tag (`wsdk.ratings_average`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag to the page, which includes a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

85.3.6 Recently Rated Widget Tag

The Recently Rated Deployment screen (which you can access from the menu bar by selecting **Ratings**, **Deploy**, and then **Recently Rated**) provides designers with the Recently Rated widget tag. This section analyzes the parameters defined in the Recently Rated widget tag:

Recently Rated widget tag

```
1. <div id="topics_container_recently_rated_other"></div>
```

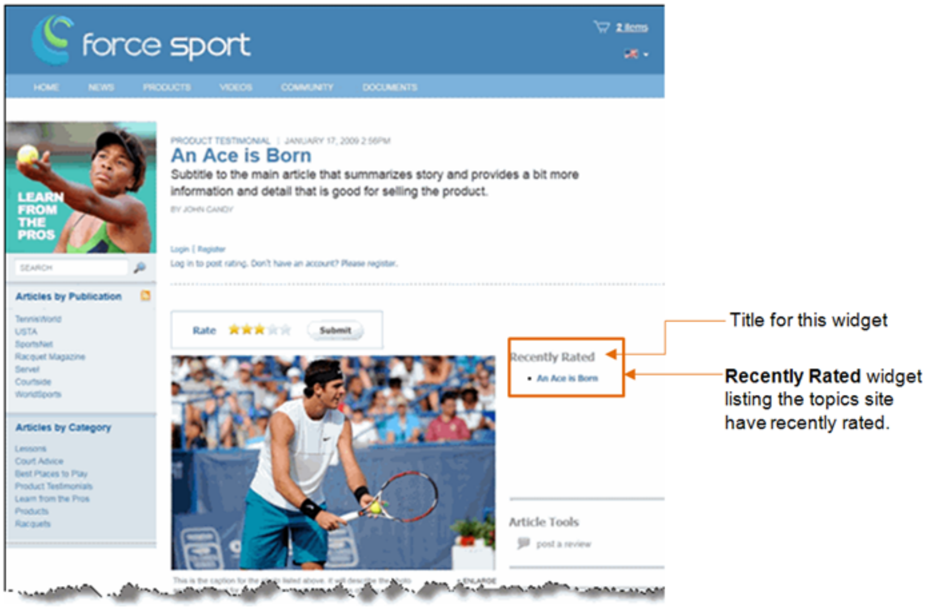
```
2.  <script type="text/javascript">
3.  cos = window.cos || {};
4.  cos.pageWidgets = cos.pageWidgets || [];
5.  cos.pageWidgets.push({name: "wsdk.topics",
6.  version: "1.5",
7.  elementID: "topics_container_recently_rated_other",
8.  attributes: {"content_type":"recently_rated","site_id":"FirstSiteII","resource_
    type":"other"}});
9.  setTimeout(
10. function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.topics) ==
    'undefined')) {document.getElementById('topics_container_recently_rated_
    other').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
    sans-serif;font-size: 12px;color: #333333;border: 1px solid
    #dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Topics is unavailable
    right now. Please contact the site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.topics');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Recently Rated widget tag

- Line 1 defines the container that holds the Recently Rated widget on the page.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.

- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Recently Rated widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Recently Rated widget inside the container defined in line 1. Line 8 specifies the Recently Rated widget tag's settings, including the type of topics the Recently Rated widget will list. If you want the Recently Rated widget to include topics of all types in its list, remove the `"resource_type":"other`" parameter.
- Lines 9 through 11 check if the Recently Rated widget is rendered on the page. If the Recently Rated widget is not rendered after 30 seconds, then users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Recently Rated widget tag (`wsdk.topics`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag to the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–16 Recently Rated widget displayed on a web page



Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

Login | Register

Log in to post rating. Don't have an account? Please register.

Title for this widget

Recently Rated widget listing the topics site have recently rated.

Note: When the Recently Rated widget is deployed, it does not display a title (as shown in Figure 85–16). For the Recently Rated widget to display a title, you will have to insert the title into the template to which you copied the Recently Rated widget tag.

85.3.7 Most Rated Widget Tag

The Most Rated Deployment screen (which you can access from the menu bar by selecting **Ratings**, **Deploy**, and then **Most Rated**) provides designers with the Most Rated widget tag. This section analyzes the parameters defined in the Most Rated widget tag:

Most Rated widget tag

```
<div id="topics_container_most_rated_other"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "wsdk.topics",
version: "1.5",
elementID: "topics_container_most_rated_other",
attributes: {"content_type":"most_rated","site_id":"FirstSiteII"'"resource_
type":"other"}});
setTimeout(
function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.topics) ==
'undefined')) {document.getElementById('topics_container_most_rated_
other').innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
sans-serif;font-size: 12px;color: #333333;border: 1px solid
#dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Topics is unavailable
right now. Please contact the site administrator.</div>";}}
,30000);
```

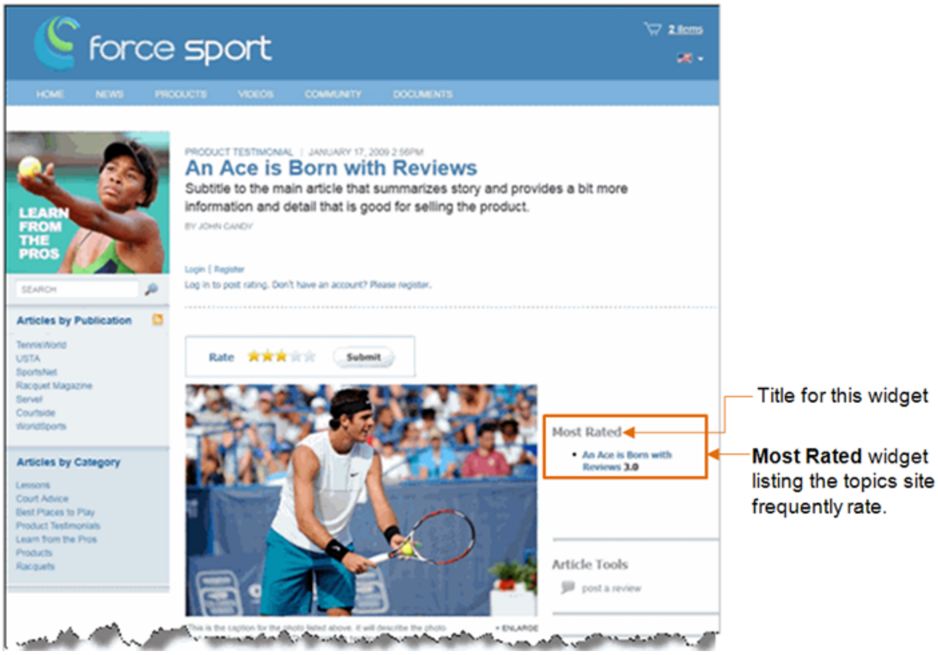
```
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.topics');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Most Rated widget tag

- Line 1 defines the container that holds the Most Rated widget on the page.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Most Rated widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Most Rated widget inside the container defined in line 1. Line 8 specifies the Most Rated widget tag's settings, including the type of topics the Most Rated widget will list. If you want the Most Rated widget to include topics of all types in its list, remove the `"resource_type":"other"` parameter.
- Lines 9 through 11 check if the Most Rated widget is rendered on the page. If the Most Rated widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.

- Line 13 adds the Most Rated widget tag (`wsdk.topics`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–17 Most Rated widget displayed on a web page



Title for this widget

Most Rated widget listing the topics site frequently rate.

Note: When the Most Rated widget is deployed, it does not display a title (as shown in Figure 85–17). For the Most Rated widget to display a title, you will have to insert the title into the template to which you copied the Most Rated widget tag.

85.4 Login Bar Widget Tag

The "Login Bar Deployment" screen (which you can access from the menu bar by selecting **Login Bar** and then **Deploy**) provides designers with the Login Bar widget tag. This section analyzes the parameters defined in the Login Bar widget tag:

Login Bar widget tag

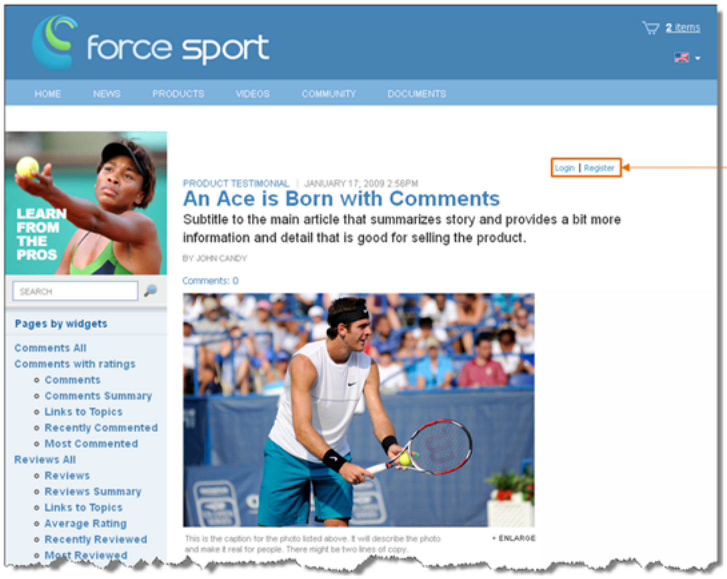
```
1.  <div id="session_box_container"></div>
2.  <script type="text/javascript">
3.  cos = window.cos || {};
4.  cos.pageWidgets = cos.pageWidgets || [];
5.  cos.pageWidgets.push({name: "wsdk.session",
6.  version: "0.1",
7.  elementID: "session_box_container",
8.  attributes:{"view_type":"short","site_id":"FirstSiteII"}});
9.  setTimeout(
10. function(){if ((typeof(session_box) == 'undefined') || (typeof(session_box.v0_
    1) == 'undefined')) {document.getElementById('session_box_container').innerHTML
    = "<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size:
    12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top:
    4px;height: 22px;'>Login Bar is unavailable right now. Please contact the site
    administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('wsdk.session');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0).appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Login Bar widget tag

- Line 1 defines the container that holds the Login Bar widget on the page.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.

- Line 3 defines the cos object on the page. The cos object contains all Community application functionality. If there are no community widget tags deployed on the page, then the cos object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Login Bar widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Login Bar widget inside the container defined in line 1.
- Lines 9 through 11 check if the Login Bar widget is rendered on the page. If the Login Bar widget is not rendered after 30 seconds, users are informed that there is an error and they should contact their site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Login Bar widget tag (session-box) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the onload function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new onload function call to the old onload function, with all necessary arguments.
- Line 31 defines the alreadyProcessed variable in order to avoid processing the onload function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–18 Standalone login bar widget displayed on a web page



PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

An Ace is Born with Comments

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

Comments: 0

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Standalone **Login Bar** widget. Visitors can use this widget to log in to the web site, register to become an authenticated visitor, and request a new password.

85.5 Poll Widget Tags

The Deploy Poll Name Poll screen (which you can access from the menu bar by selecting **Polls**, **All Polls**, and then *navigating to the desired poll* and selecting **Deploy**) provides designers with a given poll's Poll and Results widget tags. This section analyzes the parameters defined in the following widget tags:

- Section 85.5.1, "Main Poll Widget Tag"
- Section 85.5.2, "Poll Results Widget Tag"

85.5.1 Main Poll Widget Tag

The Poll Tag is the main Poll widget tag which contains all of the configurations for the poll you are deploying.

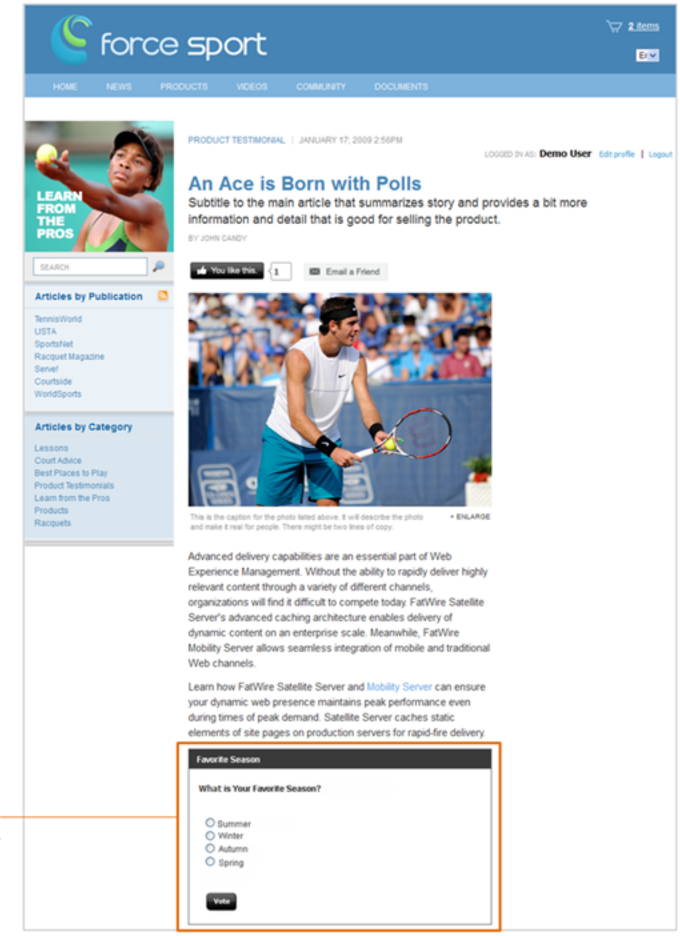
Poll Tag widget tag

```
<div id="poll_container1322111446303"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "poll",
version: "1.0",
elementID: "poll_container1322111446303",
attributes: {"poll_
id":"1322111446303","uid":"35b48e18-f1c7-4e97-ab1a-98cea2564f66","site_
id":"FirstSiteII"}});
setTimeout(
function(){if ((typeof(poll) == 'undefined') || (typeof(poll.v1_0) ==
'undefined')) {document.getElementById('poll_container1322111446303').innerHTML
= "<div style='font-family: Tahoma, Verdana, Geneva, sans-serif;font-size:
12px;color: #333333;border: 1px solid #dbdfe1;padding-left: 5px;padding-top:
4px;height: 22px;'>Poll is unavailable right now. Please contact the site
administrator.</div>";}}
,30000);
cos.pageScripts = cos.pageScripts || [];
cos.pageScripts.push('poll');
(function()
{
var oldOnloadHandler = window.onload || function()
{
};
if (!oldOnloadHandler.alreadyProcessed)
{
window.onload = function()
{
var script = document.createElement('script');
script.src = 'http://prod-cg.example.com:8080/cg/wsdk/widget/'
+ cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Poll Tag widget tag

- Line 1 defines the container that holds the Poll widget on the page. This line also specifies the unique identifier of the Poll widget you are deploying. Each poll is automatically assigned a unique identifier upon creation.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Poll widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Poll widget inside the container defined in line 1.
- Lines 9 through 11 check if the Poll widget is rendered on the page. If the Polls widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrator.
- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's value is empty.
- Line 13 adds the Poll widget tag (`poll`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of all the scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–19 Poll widget displayed on a web page



Articles by Publication

TennisWorld
USTA
SportsNet
Racquet Magazine
Serve!
Courtside
WorldSports

Articles by Category

Lessons
Court Advice
Best Places to Play
Product Testimonials
Learn from the Pros
Products
Racquets

PRODUCT TESTIMONIAL | JANUARY 17, 2009 2:56PM

LOGGED IN AS: **Demo User** Edit profile | Logout

An Ace is Born with Polls

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content through a variety of different channels, organizations will find it difficult to compete today. FatWire Satellite Server's advanced caching architecture enables delivery of dynamic content on an enterprise scale. Meanwhile, FatWire Mobility Server allows seamless integration of mobile and traditional Web channels.

Learn how FatWire Satellite Server and Mobility Server can ensure your dynamic web presence maintains peak performance even during times of peak demand. Satellite Server caches static elements of site pages on production servers for rapid-fire delivery.

Favorite Season

What is Your Favorite Season?

○ Summer
○ Winter
○ Autumn
○ Spring

Poll widget displayed on a web page. This tag displays the poll's question and available answer options. Site visitors can select an answer and click **Vote** to participate in the poll.

85.5.2 Poll Results Widget Tag

The Results Tag contains a summary of the main poll's results (Section 85.5.1, "Main Poll Widget Tag"), which dynamically change with each poll vote submitted by a site visitor.

Results widget tag

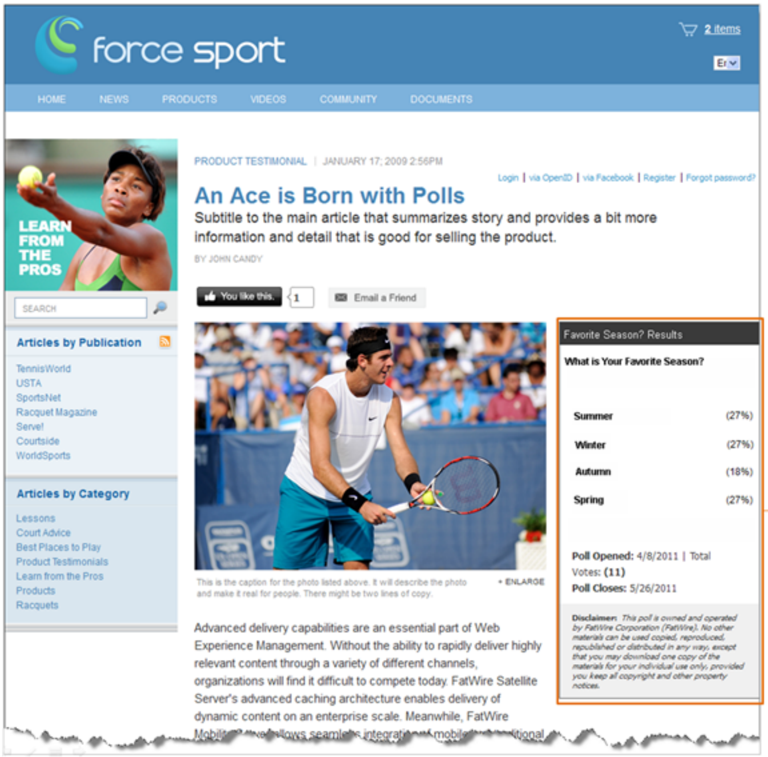
```
<div id="poll_summary_container1322111446303"></div>
<script type="text/javascript">
cos = window.cos || {};
cos.pageWidgets = cos.pageWidgets || [];
cos.pageWidgets.push({name: "poll-summary",
version: "1.0",
elementID: "poll_summary_container1322111446303",
attributes: {"poll_
id":"1322111446303","uid":"35b48e18-f1c7-4e97-ab1a-98cea2564f66","site_
id":"FirstSiteII"}});
setTimeout(
function(){if ((typeof(poll) == 'undefined') || (typeof(poll.v1_0) ==
'undefined')) {document.getElementById('poll_summary_
container1322111446303').innerHTML = "<div style='font-family: Tahoma, Verdana,
    Geneva, sans-serif;font-size: 12px;color: #333333;border: 1px solid
    #dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Poll is unavailable
    right now. Please contact the site administrator.</div>";}}
11. ,30000);
12. cos.pageScripts = cos.pageScripts || [];
13. cos.pageScripts.push('poll');
14. (function()
15. {
16. var oldOnloadHandler = window.onload || function()
17. {
18. };
19. if (!oldOnloadHandler.alreadyProcessed)
20. {
21. window.onload = function()
22. {
23. var script = document.createElement('script');
24. script.src = 'http://prod-cg.example.com:8080/cg//wsdk/widget/'
25. + cos.pageScripts.join(':') + '.js?site_id=FirstSiteII';
26. script.type = 'text/javascript';
27. script.charset = 'utf-8';
28. document.getElementsByTagName("head").item(0) .appendChild(script);
29. oldOnloadHandler.apply(this, arguments);
30. };
31. window.onload.alreadyProcessed = true;
32. }
33. })();
34. </script>
```

Analyzing the Results widget tag

- Line 1 defines the container that holds the Results widget on the page. This line also specifies the unique identifier of the poll with which the Results widget is associated. The Results widget displays the results of the poll whose unique identifier is specified in this line.
- Line 2 opens the bootstrapping JavaScript code needed for the widget.
- Line 3 defines the `cos` object on the page. The `cos` object contains all Community application functionality. If there are no community widget tags deployed on the page, then the `cos` object's value is empty.
- Line 4 defines the list of widget tags that are deployed on the page. If there are no widgets deployed on the page, then the list's value is empty.
- Lines 5 through 8 define the Results widget tag and add the new element to the list of widget tags (defined in line 4). Line 7 contains a link that is used to render the Results widget inside the container defined in line 1.
- Lines 9 through 11 check if the poll results widget is rendered on the page. If the poll results widget is not rendered after 30 seconds, users are informed that there is an error and they should contact the site administrators.

- Line 12 defines the list of scripts deployed on the page. If there are no scripts deployed, then the list's values is empty.
- Line 13 adds the Results widget tag (`poll`) to the list of page scripts (defined in line 12).
- Line 16 checks if there is any logic deployed on the page, which starts after the page has been loaded. If there is logic deployed on the page, this line defines the function that implements it. If there is no such logic deployed, then the function is left empty.
- Line 19 checks if the `onload` function is already processed.
- Line 21 overrides the code that runs when the page is loaded.
- Lines 23 through 28 add the JavaScript tag on the page, including a list of scripts that need to be deployed.
- Line 29 adds a new `onload` function call to the old `onload` function, with all the necessary arguments.
- Line 31 defines the `alreadyProcessed` variable in order to avoid processing the `onload` function again.
- Line 34 closes the bootstrapping JavaScript code needed for the widget.

Figure 85–20 Results widget displayed on a web page



An Ace is Born with Polls

Subtitle to the main article that summarizes story and provides a bit more information and detail that is good for selling the product.

BY JOHN CANDY

This is the caption for the photo listed above. It will describe the photo and make it real for people. There might be two lines of copy.

Advanced delivery capabilities are an essential part of Web Experience Management. Without the ability to rapidly deliver highly relevant content through a variety of different channels, organizations will find it difficult to compete today. FatWire Satellite Server's advanced caching architecture enables delivery of dynamic content on an enterprise scale. Meanwhile, FatWire Mobility Server allows seamless integration of mobile and traditional

Favorite Season? Results

What is Your Favorite Season?

Summer	(27%)
Winter	(27%)
Autumn	(18%)
Spring	(27%)

Poll Opened: 4/8/2011 | Total Votes: **(11)**
Poll Closes: 5/26/2011

Disclaimer: *This poll is owned and operated by FatWire Corporation (FatWire). No other materials can be used copied, reproduced, republished or distributed in any way, except that you may download one copy of the materials for your individual use only, provided you keep all copyright and other property notices.*

Results widget displayed on a web page. This widget displays the results of its associated poll.

86

Community-Gadgets: Enabling SEO Support for Community Widgets

This chapter provides information about SEO support and instructions on how to enable SEO support for your website.

This chapter contains the following sections:

- Section 86.1, "Overview"
- Section 86.2, "Enabling SEO Support"

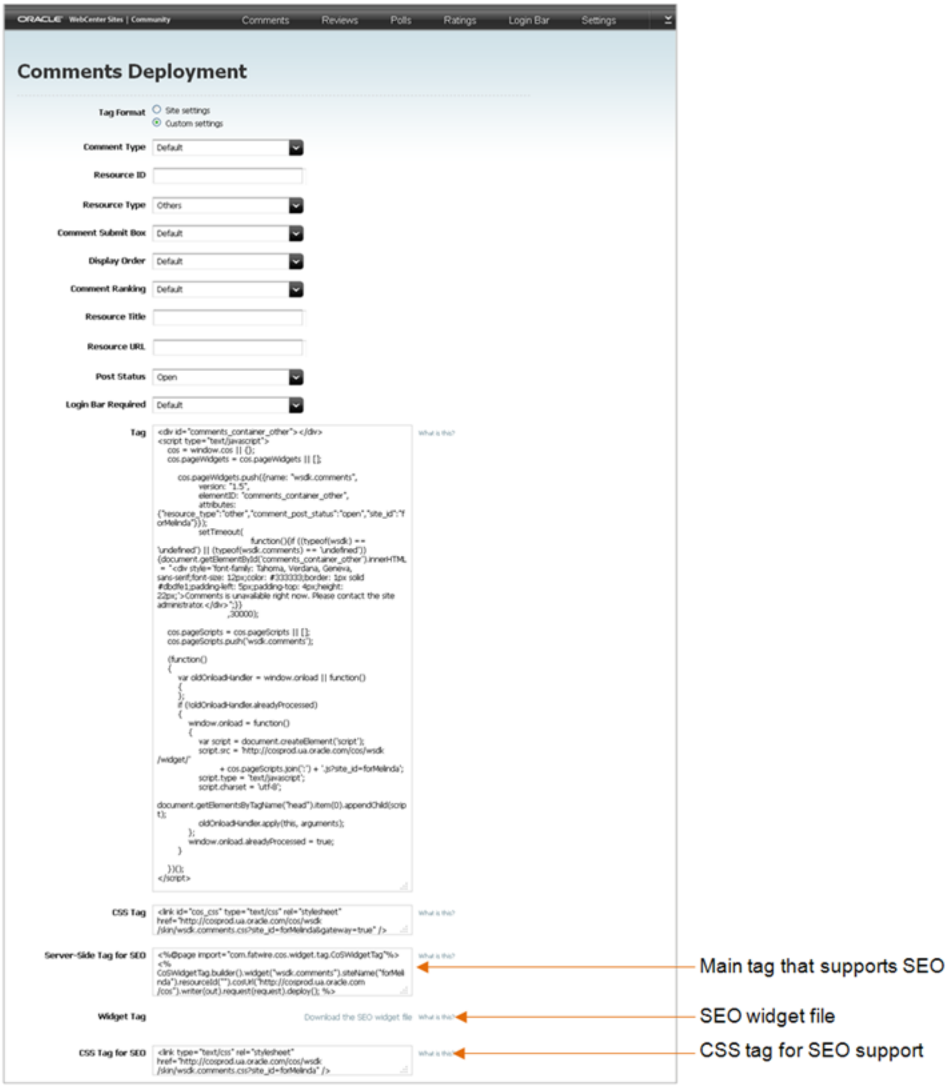
86.1 Overview

Community widget tags are based on JavaScript code. A browser that is set to disable JavaScript does not display any community widgets. Enabling SEO support ensures that visitors' comments and reviews are visible in browsers set to disable JavaScript. However, visitors are limited to having read-only access to comments and reviews. All other widgets and widget functionality remain hidden.

Additionally, SEO support enables search engines such as *Google* to look for search terms within the content of your website's comments and reviews. If a comment or review contains a search term, the search engine will display a link to the web page, on which that comment or review was posted, in its search results list.

86.2 Enabling SEO Support

The fields used to enable SEO support are located at the bottom of the Comments and Reviews widget tag custom deployment screens. Below shows the SEO fields in the Comments widget tag's deployment screen (the Reviews widget tag's custom deployment screen contains the same SEO field):



Comments Deployment

Tag Format ○ Site settings ● Custom settings

Comment Type: Default

Resource ID:

Resource Type: Others

Comment Submit Box: Default

Display Order: Default

Comment Ranking: Default

Resource Title:

Resource URL:

Post Status: Open

Login Bar Required: Default

Tag:

```
<div id="comments_container_other"></div>
<script type="text/javascript">
   cos = window.cos || {};
   cos.pageWidgets = cos.pageWidgets || [];

      cos.pageWidgets.push({name: "wsdk.comments",
            version: "1.5",
            elementID: "comments_container_other",
            attributes:
{"resource_type":"other","comment_post_status":"open","site_id":"f
orMelinda"}});
            setTimeout(
                        function(){if ((typeof(wsdk) ==
'undefined') || (typeof(wsdk.comments) == 'undefined'))
{document.getElementById('comments_container_other').innerHTML
 = "<div style='font-family: Tahoma, Verdana, Geneva,
sans-serif;font-size: 12px;color: #333333;border: 1px solid
#dbdfe1;padding-left: 5px;padding-top: 4px;height:
22px;'>Comments is unavailable right now. Please contact the site
administrator.</div>";}}
                  ,30000);

   cos.pageScripts = cos.pageScripts || [];
   cos.pageScripts.push('wsdk.comments');

   (function()
   {
      var oldOnloadHandler = window.onload || function()
      {
      };
      if (!oldOnloadHandler.alreadyProcessed)
      {
         window.onload = function()
         {
            var script = document.createElement('script');
            script.src = 'http://cosprod.ua.oracle.com/cos/wsdk
/widget/'
                  + cos.pageScripts.join(':') + '.js?site_id=forMelinda';
            script.type = 'text/javascript';
            script.charset = 'utf-8';

document.getElementsByTagName("head").item(0).appendChild(scrip
t);
            oldOnloadHandler.apply(this, arguments);
         };
         window.onload.alreadyProcessed = true;
      }

   })();
</script>
```

CSS Tag:

```
<link id="cos_css" type="text/css" rel="stylesheet"
href="http://cosprod.ua.oracle.com/cos/wsdk
/skin/wsdk.comments.css?site_id=forMelinda&gateway=true" />
```

Server-Side Tag for SEO:

```
<%@page import="com.fatwire.cos.widget.tag.CoSWidgetTag"%>
<%
CoSWidgetTag.builder().widget("wsdk.comments").siteName("forMeli
nda").resourceId("").cosUrl("http://cosprod.ua.oracle.com
/cos").writer(out).request(request).deploy(); %>
```

Widget Tag: Download the SEO widget file

CSS Tag for SEO:

```
<link type="text/css" rel="stylesheet"
href="http://cosprod.ua.oracle.com/cos/wsdk
/skin/wsdk.comments.css?site_id=forMelinda" />
```

To enable SEO support

1. Access the custom deployment screen for either the Comments or Reviews widget tag. In the menu bar, do one of the following:

   - To access the custom Comments Deployment screen, select **Comments**, **Deploy**, **Comments**, and then select **Custom Settings**.

   - To access the custom Reviews Deployment screen, select **Reviews**, **Deploy**, **Reviews**, and then select **Custom Settings**.

2. In the Resource ID field, specify a resource ID for the Comments or Reviews widget. The Server-Side Tag for SEO inherits the specified resource ID.

3. Place the `cos-widget-tag.jar` file in the application server's classpath (for example, `WEB-INF/lib`):

   a. In the Widget Tag field, click **Download the SEO Widget file** to download the SEO widget file (`cos-widget-tag.jar`).

   b. Place the SEO widget file (`cos-widget-tag.jar`) into the application server's classpath.

This `jar` file defines the functionality of the Server-Side Tag for SEO. Without this `jar` file, SEO support cannot be enabled.

4. Deploy the Server-Side Tag for SEO on the page on which the Comments (or Reviews) widget is deployed:

   a. In the Server-Side Tag for SEO field, copy (**Ctrl+C**) the tag.

   b. Access the template into which you inserted the Comments (or Reviews) widget tag and insert the Server-Side Tag for SEO field directly below the Comments (or Reviews) widget tag. For instructions, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.

      The Server-Side Tag for SEO is the main tag that enables SEO support for your web pages.

5. Deploy the CSS Tag for SEO. This tag is required to display the web page's comments and reviews in the proper layout when a visitor's browser is set to disable JavaScript. Do the following:

   a. In the CSS Tag for SEO field, copy (**Ctrl+C**) the tag.

   b. Access the template into which you inserted the Comments (or Reviews) widget tag and insert the CSS Tag for SEO between the `<head></head>` tags of the template's source code. For instructions, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.

      The CSS Tag for SEO inherits the appearance settings that are configured for the Comments and Reviews widgets. When deployed, this tag applies JavaScript formatting to the comments and reviews listed on the web page.

6. Publish the template. For instructions, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

7. Verify that you have enabled SEO support properly. Using a browser set to disable JavaScript, access the website and view a web page on which a Comments (or Reviews) widget is deployed.

87

Community-Gadgets: Deploying the CSS Tag

This chapter provides information about the CSS Tag and instructions for deploying this tag on web pages.

This chapter contains the following sections:

- Section 87.1, "Overview"
- Section 87.2, "Deploying the CSS Tag"

87.1 Overview

Each widget and gadget contains its own CSS which applies the desired look and feel to the widget and/or gadget deployed on a web page. A widget's or gadget's CSS is loaded when a request is made for the web page on which that widget or gadget is deployed. If multiple widgets and gadgets are deployed on the same page, the system loads each widget and gadget's CSS separately, which increases the load time of the web page.

To decrease the load time of a web page on which multiple widgets and gadgets are deployed, the Community and Gadgets applications provide the CSS tag. When the CSS tag is deployed, it combines all CSS files of the widgets and gadgets deployed on that page into one CSS. This means that each widget's and gadget's CSS is loaded at the same time, which reduces the amount of requests the system makes to load the page's content. Therefore, decreasing the time it would take the Community and Gadgets applications to load all CSS files for the widgets and gadgets deployed on the page.

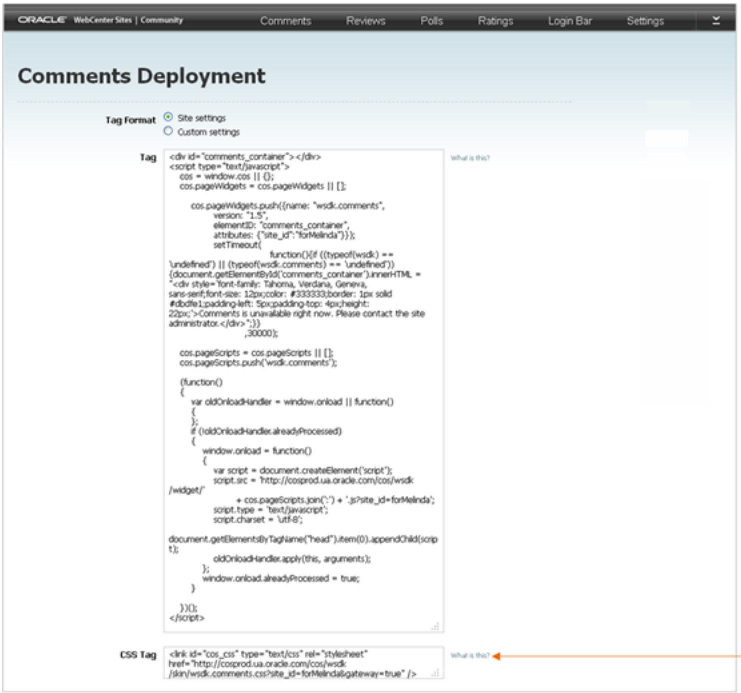
87.2 Deploying the CSS Tag

The CSS Tag field is available in the deployment screen of any widget and gadget. This scenario uses the CSS tag in the Comments Deployment screen (**Comments, Deploy,** and then **Comments**).

For the CSS tag to load all CSS files for the widgets and gadgets deployed on a given web page, you will have to configure the CSS tag by specifying the widgets, gadgets, and the associated CSS files that will be loaded with the page. When the page is requested, the Community-Gadgets application scans the code of the CSS tag and makes a single request for all widget and gadget CSS files that are specified in the CSS tag.

> **Note:** If a widget or gadget that is deployed on the page is not specified inside the CSS tag, the page will not load that widget's or gadget's CSS.

Below shows the CSS Tag field in the Comments Deployment screen (all other widget and gadget deployment screens contain a similar CSS Tag field).



Comments Deployment

Tag Format ● Site settings ○ Custom settings

Tag

```
<div id="comments_container"></div>
<script type="text/javascript">
   cos = window.cos || {};
   cos.pageWidgets = cos.pageWidgets || [];

      cos.pageWidgets.push({name: "wsdk.comments",
            version: "1.5",
            elementID: "comments_container",
            attributes: {"site_id":"forMelinda"}});
            setTimeout(
                        function(){if ((typeof(wsdk) ==
'undefined') || (typeof(wsdk.comments) == 'undefined'))
{document.getElementById('comments_container').innerHTML =
"<div style='font-family: Tahoma, Verdana, Geneva,
sans-serif;font-size: 12px;color: #333333;border: 1px solid
#dbdfe1;padding-left: 5px;padding-top: 4px;height:
22px;'>Comments is unavailable right now. Please contact the site
administrator.</div>";}}
                        ,30000);

   cos.pageScripts = cos.pageScripts || [];
   cos.pageScripts.push('wsdk.comments');

   (function()
   {
      var oldOnloadHandler = window.onload || function()
      {
      };
      if (!oldOnloadHandler.alreadyProcessed)
      {
         window.onload = function()
         {
            var script = document.createElement('script');
            script.src = 'http://cosprod.ua.oracle.com/cos/wsdk
/widget/'
                  + cos.pageScripts.join(':') + '.js?site_id=forMelinda';
            script.type = 'text/javascript';
            script.charset = 'utf-8';

document.getElementsByTagName("head").item(0).appendChild(scrip
t);
            oldOnloadHandler.apply(this, arguments);
         };
         window.onload.alreadyProcessed = true;
      }

   })();
</script>
```

CSS Tag

```
<link id="cos_css" type="text/css" rel="stylesheet"
href="http://cosprod.ua.oracle.com/cos/wsdk
/skin/wsdk.comments.css?site_id=forMelinda&gateway=true" />
```

"CSS Tag" field containing the CSS tag

To configure and deploy the CSS tag

1. Access the deployment screen of any widget tag or gadget. In this example, we access the Comments Deployment screen.
2. In the CSS Tag field, copy (**Ctrl+C**) the CSS tag.
3. Access the WebCenter Sites Admin interface, and insert the CSS tag into the template that renders the desired web page:
   a. In the menu bar, point to the down-arrow icon, located at the extreme right of the screen, to render the applications bar.
   b. In the applications bar, click the **Admin** icon to render the WebCenter Sites Admin interface.
   c. Locate the template into which you wish to insert the CSS tag:
      a. From the start menu options, click **Search**.
      b. In the Search results list, select **Find Template**.
      c. In the Search for Templates form, click **Search**.
      d. In the List of Templates form, select the template into which you wish to insert the widget tag.
      e. In the template's Inspect form, click **Edit**.

f. In the template's Element screen, insert (**Ctrl+V**) the widget tag into the Element Logic field between the `<head></head>` tags. For example:

```
<head>
<link id="cos_css" type="text/css"rel="stylesheet"
href="http://prod-cg.example.com:8080/cg/wsdk/skin/wsdk.comments.css?si
te_id=FirstSiteII&gateway=true" />
...
</head>
```

d. In the CSS tag, specify each widget, gadget, and CSS file that will be loaded with the page. For example:

If the CSS tag will load the CSS files for the Comments, Reviews, Polls, and Login Bar widgets, as well as a single gadget's CSS file, the CSS tag should be modified as follows:

```
<link id="cos_css" type="text/css" rel="stylesheet"
href="http://prod-cg.example.com:8080/cg/wsdk/skin/wsdk.comments:wsdk.revie
ws:poll:wsdk.session:gadgets_dashboard.css?site_
id=FirstSiteII&gateway=true" />
```

Where:

– `wsdk.comments`: Name of the Comments widget.

– `wsdk.reviews`: Name of the Review Summary widget.

– `poll`: Name of both the Poll and Poll Results widgets.

– `wsdk.session`: Name of the Login Bar widget.

– `gadgets_dashboard`: Name of the single gadget.

The CSS `href` tag references the location of the CSS files that will be loaded for the widgets and gadgets deployed on the page. Notice that each widget's and gadget's name is separated by a colon.

Each widget's name is located in its CSS tag in the widget's deployment screen. The widget names are as follows:

– `wsdk.comments`: Name of the Comments widget.

– `comments-summary`: Name of the Comments Summary widget.

– `comments-link`: Name of the Links to Topics widget for comments and reviews.

– `wsdk.topics`: Name of the Most Commented, Recently Commented, Most Reviewed, Recently Reviewed, Top Ranked, Most Rated, and Recently Rated widgets.

– `wsdk.reviews`: Name of the Reviews widget.

– `reviews-summary`: Name of the Reviews Summary widget.

– `wsdk.reviews_average`: Name of the Reviews Average Rating widget.

– `poll`: Name of both the Poll and Poll Results widgets.

– `wsdk.ratings`: Name of the Stars, Thumbs Up/Down, Like It, and Recommend ratings widgets.

– `wsdk.ratings_average`: Name of the Ratings Average Rating widget.

– `wsdk.session`: Name of the Login Bar widget.

   - `gadgets_dashboard`: Name of a gadget.

   e. Click **Save Changes**.

4. Publish the template to the website. For instructions about publishing, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

Part VII

Developing Community Blogs

This part contains procedures for customizing the blog data model. It also covers template code and provides guidelines for developing blog functionality on different CM sites.

This part contains the following chapters:

88

Community Blogs: Data Model

This chapter provides information about the Community Blogs flex family hierarchy and the assets that are created for it during the process of installing the Community Blogs module.

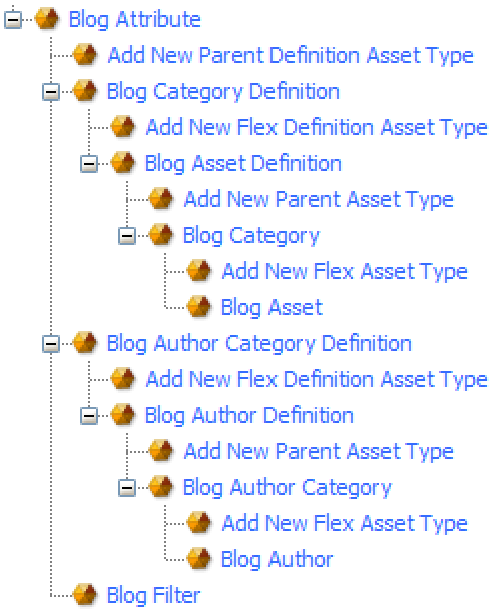
This chapter contains the following sections:

- Section 88.1, "Overview of the Blog Flex Family"
- Section 88.2, "Flex Family Members"

88.1 Overview of the Blog Flex Family

The `Blog` flex family provides the data structure for storing blog assets and authors. The `Blog` flex family member organization is shown in Figure 88–1. Blog assets are stored in a four-level hierarchy.

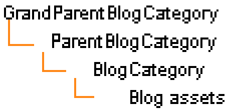
Figure 88–1 Blog Asset Hierarchy



- Blog Attribute
  - Add New Parent Definition Asset Type
  - Blog Category Definition
    - Add New Flex Definition Asset Type
    - Blog Asset Definition
      - Add New Parent Asset Type
      - Blog Category
        - Add New Flex Asset Type
        - Blog Asset
  - Blog Author Category Definition
    - Add New Flex Definition Asset Type
    - Blog Author Definition
      - Add New Parent Asset Type
      - Blog Author Category
        - Add New Flex Asset Type
        - Blog Author
  - Blog Filter

- `Blog Asset` is used to create the unit of content that will be displayed on the website.
- `Blog Category` is used to name the blog categories.

- Blog Category Definition is used to place blog categories in hierarchical order. By default the blog category definitions are: GrandParentBlogCategory (first level), ParentBlogCategory (second level), and BlogCategory (third level):



- Blog Asset Definition is used to determine a blog asset's parent. By default, a blog asset's parent is a category at the BlogCategory level (third level). The Blog Asset Definition also specifies which blog attributes make up the Blog Asset form.
- Blog Attribute is used to create attributes, which you specify as fields for the Blog Asset and Blog Author forms.

Blog authors are stored in a flat structure, under a single category named BlogAuthors:

- Blog Author is used to create author assets. Authors' names (and optionally images) will be displayed with the blogs they create.
- Blog Author Category is used to name the categories of blog authors. By default, there is only one author category, named BlogAuthors, which is used to contain all the author assets.
- Blog Author Category Definition is used to place blog author categories in hierarchical order. By default, there is only one blog author category definition (BlogAuthorCategory), which creates a flat structure (all blog author categories at the same level).
- Blog Author Definition is used to place author assets under the parent named BlogAuthors. The Blog Author Definition also specifies which blog attributes make up the Blog Author form.

Blog filters are used for data transformation:

- Blog Filter is used to take the data from one blog attribute, transform or evaluate it in some way, and then store the results in another blog attribute when you save the blog asset. The resulting value from a blog filter action is called a derived attribute value.

88.2 Flex Family Members

The rest of this chapter provides detailed descriptions of the default Blog flex family members:

- Section 88.2.1, "Default Blog Attributes"
- Section 88.2.2, "Default Blog Category Definitions and Blog Categories"
- Section 88.2.3, "Default Blog Asset Definition and Blog Assets"
- Section 88.2.4, "Default Blog Author Category Definition and Blog Author Category"
- Section 88.2.5, "Default Blog Author Definition and Blog Authors"
- Section 88.2.6, "Default Blog Filters"

88.2.1 Default Blog Attributes

The `Blog Attribute` asset type contains the following default attributes:

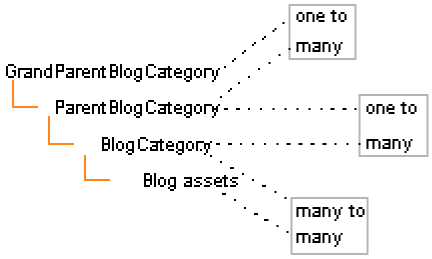
- `Abstract`: Defines the **Abstract** field for the blog asset form.
- `AssetId`: Is used to get the parent asset ID of the blog asset. Since the blog asset can be assigned to multiple blog categories under the same parent blog category, this attribute is used to find all blog categories to which the current blog asset belongs.
- `Author`: Defines the **Author** field of the blog asset form.
- `BlogAuthorFullName`: Defines the **Full Name** field of the blog author form.
- `BlogAuthorImage`: Defines the **Author Profile Image** field of the blog author form.
- `BlogAuthorImageHeight`: Is used to identify the default height of an uploaded author image.
- `BlogAuthorImageMimeType`: Is used by the Blog Author filters to identify the media type (for example, `jpeg` or `.gif`) of an uploaded author image.
- `BlogAuthorImageWidth`: Is used to identify the default width of an uploaded author image.
- `BlogAuthorLargeThumbImage`: Is used to render the large thumbnail of the author image in the blog author form.
- `BlogAuthorLThumbHeight`: Is used to specify the appropriate height for the large thumbnail of the uploaded author image.
- `BlogAuthorLThumbWidth`: Is used to specify the appropriate width for the large thumbnail of the uploaded author image.
- `BlogAuthorSmallThumbImage`: Is used to render the small thumbnail of the uploaded author image.
- `BlogAuthorSThumbHeight`: Is used to specify the appropriate height for the small thumbnail of the uploaded author image.
- `BlogAuhtorSThumbWidth`: Is used to specify the appropriate width for the small thumbnail of the uploaded author image.
- `Body`: Defines the **Blog Body** field for the blog asset form.
- `Category`: Defines the **Blog Category** field for the blog asset form.
- `Date`: Defines the **Blog Date and Time** field for the blog asset form.
- `GrandParentCategory`: Is used to retrieve the top most category to which the blog asset belongs. The `GrandParentCategory` encapsulates the parent and asset categories.
- `GrandParentDescription`: Contains the description for the grandparent category to which the blog asset belongs.
- `ParentCategory`: Is similar in concept to the `GrandParentCategory`, only it contains the category information about the parent.
- `ParentDescription`: Contains the description of the parent category to which the blog asset belongs.

The default attributes can be used as a quick start for your own site's blog functionality. You can also create new attributes, and modify or delete the default attributes to fit your requirements. For more information, see Section 90.1, "Customizing the Blog Asset Form" in Chapter 90, "Community Blogs: Customizing Blog Components."

88.2.2 Default Blog Category Definitions and Blog Categories

The `Blog Category Definition` asset type contains the following definitions: `GrandParentBlogCategory`, `ParentBlogCategory`, and `BlogCategory`. Each definition specifies the level at which a category can be placed (Figure 88–2).

Figure 88–2 Category Levels

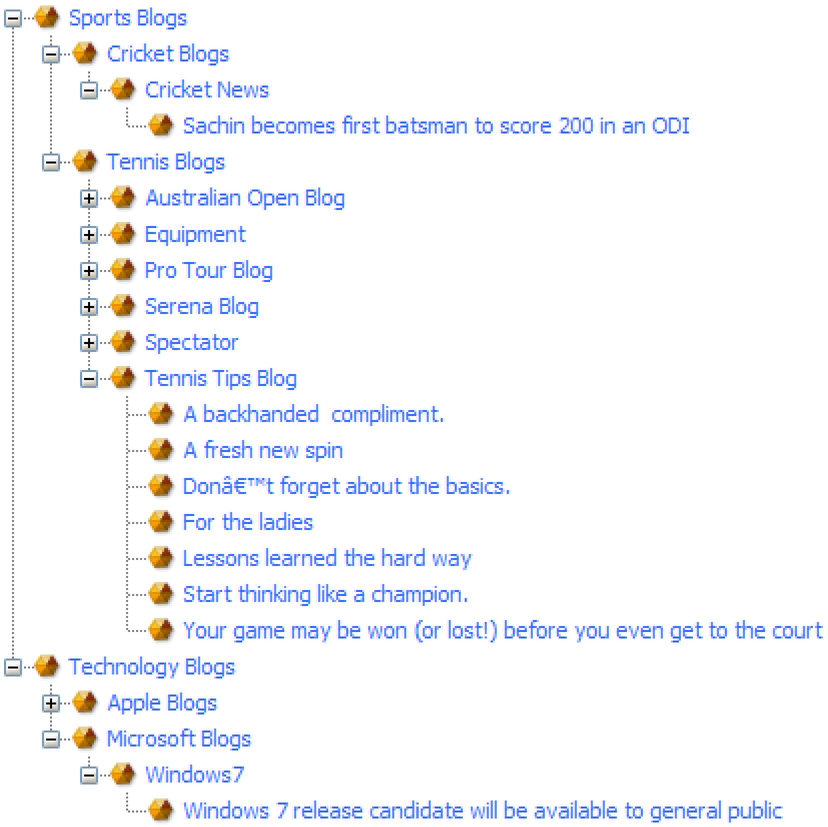


`BlogCategory` defines the third level of categories. All blog assets belong to categories at the `BlogCategory` level. A given blog asset can belong to many blog categories under the same `ParentBlogCategory`.

The `Blog Category` asset type is used to name categories and place them in hierarchical order (by means of the blog category definitions described above). The `Blog Category` asset type contains the following grandparent categories: Sports Blogs and Technology Blogs (as shown in Figure 88–3). Each grandparent category can contain multiple parent categories. Each parent category can contain multiple blog categories, and each blog category can contain multiple blog assets. Figure 88–3 displays a sample tree with the default categories and blog assets.

Note: Blog assets cannot have more than one distinct inherited value for a single-valued attribute. Therefore, if you are going to assign a blog asset to more than one blog category, make sure the blog categories are under the same parent blog category.

Figure 88–3 Sample Blog Category Hierarchy



- Sports Blogs
  - Cricket Blogs
    - Cricket News
      - Sachin becomes first batsman to score 200 in an ODI
  - Tennis Blogs
    - Australian Open Blog
    - Equipment
    - Pro Tour Blog
    - Serena Blog
    - Spectator
    - Tennis Tips Blog
      - A backhanded compliment.
      - A fresh new spin
      - Donâ€™t forget about the basics.
      - For the ladies
      - Lessons learned the hard way
      - Start thinking like a champion.
      - Your game may be won (or lost!) before you even get to the court
- Technology Blogs
  - Apple Blogs
  - Microsoft Blogs
    - Windows7
      - Windows 7 release candidate will be available to general public

When creating blog categories for the blog assets of your website, you can reuse the hierarchy that is defined by the default blog category definitions. To display the hierarchy of blog categories and blog assets in the WebCenter Sites tree (as shown in Figure 88–3), you will need to create a blog tree tab. For instructions on enabling blog components see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

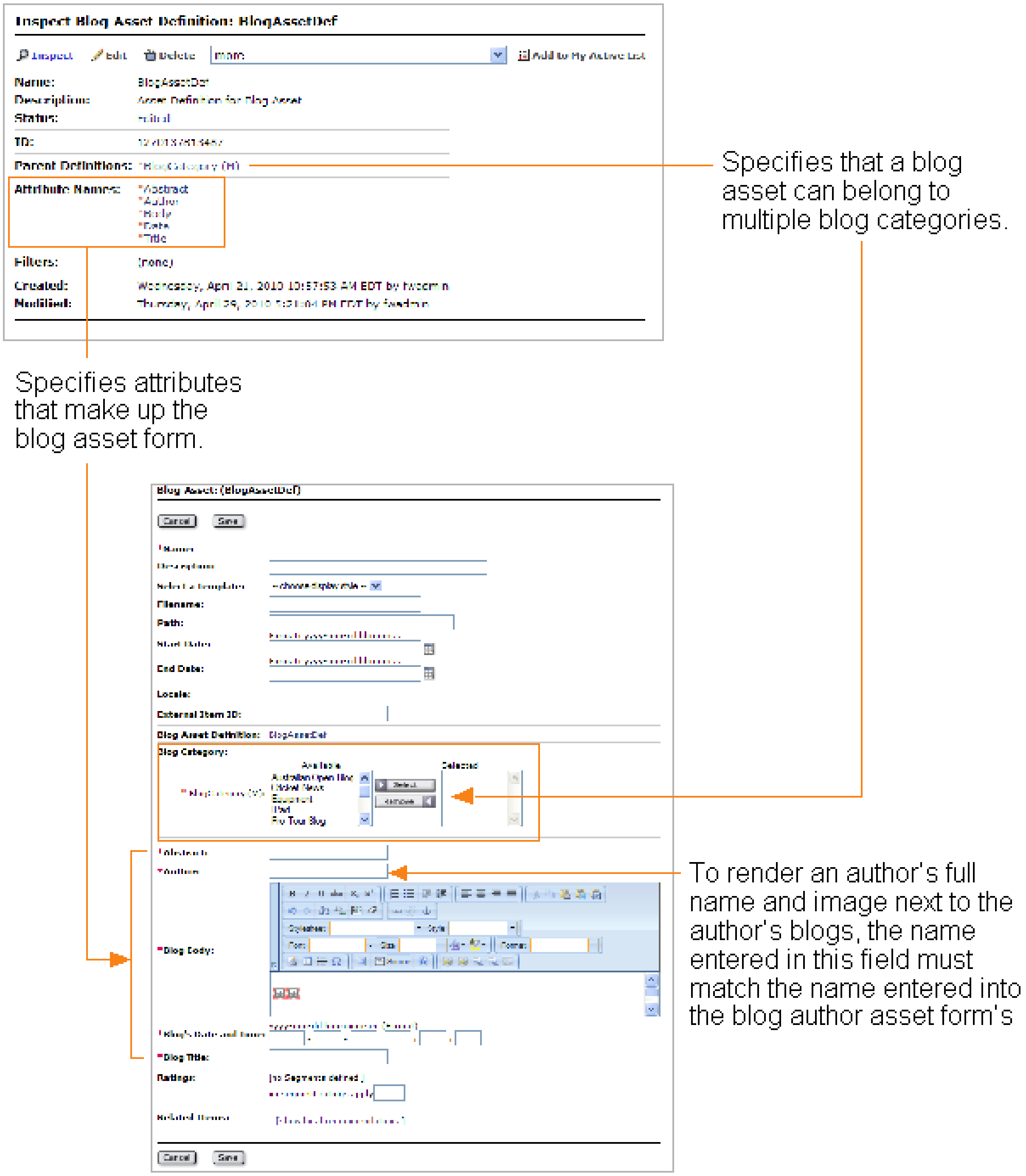
88.2.3 Default Blog Asset Definition and Blog Assets

The `Blog` flex family contains one blog asset definition, named `BlogAssetDef`, which specifies that a blog asset can belong to multiple blog categories (third-level categories) under the same parent blog category (second-level category). `BlogAssetDef` also specifies attributes that make up the blog asset's form:

- Abstract
- Author
- Body
- Date
- Title

The `Blog Asset` asset type supplies the form that content providers fill out to create blog assets. Figure 88–4 displays `BlogAssetDef` and shows how its parameters are used in the Blog Asset form.

Figure 88–4 Relationship Between Blog Asset Definition and Blog Asset Form



Inspect Blog Asset Definition: BlogAssetDef

Name: BlogAssetDef
Description: Asset Definition for Blog Asset
Status: Edited
ID: [illegible]
Parent Definitions: *BlogCategory (M)
Attribute Names: *Abstract
*Author
*Body
*Date
*Title
Filters: (none)
Created: Wednesday, April 21, 2010 10:57:53 AM EDT by fwadmin
Modified: Thursday, April 29, 2010 [illegible] PM EDT by fwadmin

Specifies that a blog asset can belong to multiple blog categories.

Specifies attributes that make up the blog asset form.

Blog Asset: (BlogAssetDef)

*Name:
Description:
Select a template: -- choose display style --
Filename:
Path:
Start Date: [illegible]
End Date: [illegible]
Locale:
External Item ID:
Blog Asset Definition: BlogAssetDef
Blog Category:
*BlogCategory (M): Available: Australian Open Blog, Cricket News, Equipment, iPad, For Tour Blog; Selected:

*Abstract:
*Author:
*Blog Body:

*Blog's Date and Time:
*Blog Title:
Ratings: (no Segments defined.)
Related Items: [illegible]

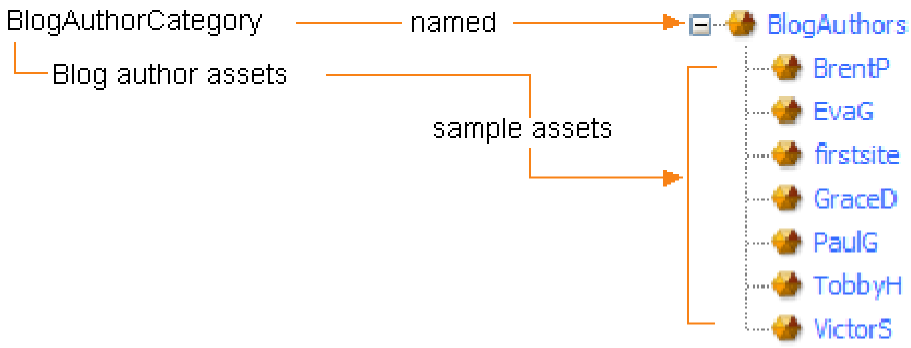
To render an author's full name and image next to the author's blogs, the name entered in this field must match the name entered into the blog author asset form's

If you create your own attributes for the blog asset form, you will need to add those attributes to the default blog asset definition. For instructions on creating new attributes and adding them to the blog asset definition, see Section 90.1, "Customizing the Blog Asset Form" in Chapter 90, "Community Blogs: Customizing Blog Components."

88.2.4 Default Blog Author Category Definition and Blog Author Category

By default, the `Blog` flex family contains one blog author category definition, named `BlogAuthorCategory`, which defines a flat structure for storing blog authors. The `Blog Author Category` asset type is used to name the categories. Only one blog author category exists, named `BlogAuthors`, which is used to contain all the author assets created (or shared) on the site.

Figure 88–5 Sample Blog Author Category Hierarchy



88.2.5 Default Blog Author Definition and Blog Authors

The `Blog` flex family contains one blog author definition, `BlogAuthorDef`, which specifies that blog author assets can belong to only the BlogAuthors category. `BlogAuthorDef` also specifies attributes that make up the blog author form:

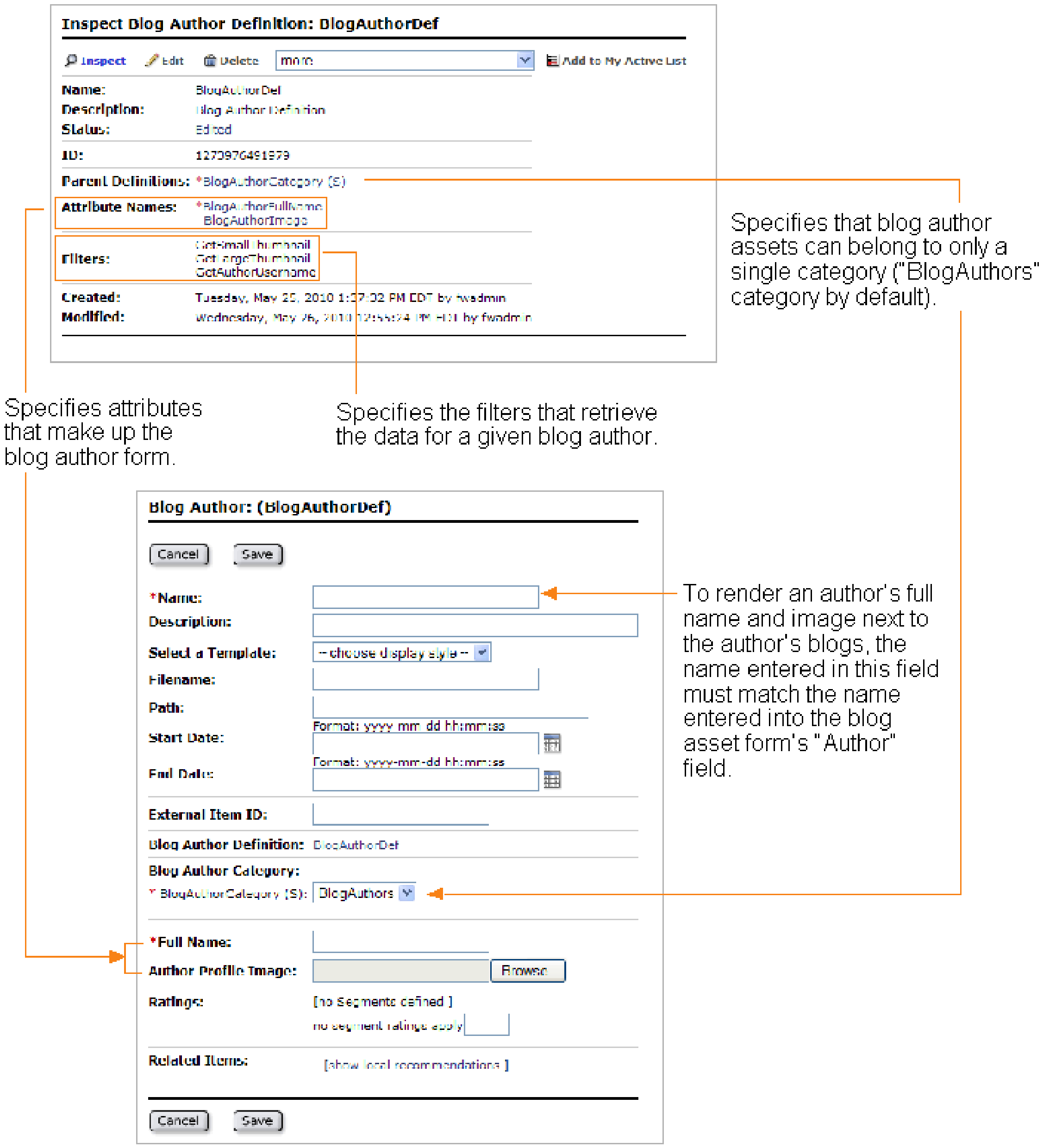
- Full Name
- Author Profile Image

In addition, `BlogAuthorDef` specifies filters that retrieve the author's full name, and large and small thumbnails of the image (optional).

The `Blog Author` asset type supplies the form that administrators fill out to create blog author assets for display next to published blogs. Blog authors can modify their names and images.

Figure 88–6 displays `BlogAuthorDef` and shows how its parameters are used in the blog author form.

Figure 88–6 Relationship Between Blog Author Definition and Blog Author Form



Inspect Blog Author Definition: BlogAuthorDef

Name: BlogAuthorDef
Description: Blog Author Definition
Status: Edited
ID: 1273976491979
Parent Definitions: *BlogAuthorCategory (S)
Attribute Names: *BlogAuthorFullName
BlogAuthorImage
Filters: GetSmallThumbnail
GetLargeThumbnail
GetAuthorUsername
Created: Tuesday, May 25, 2010 1:37:02 PM EDT by fwadmin
Modified: Wednesday, May 26, 2010 12:55:24 PM EDT by fwadmin

Blog Author: (BlogAuthorDef)

*Name:
Description:
Select a Template: -- choose display style --
Filename:
Path:
Start Date: Format: yyyy-mm-dd hh:mm:ss
End Date: Format: yyyy-mm-dd hh:mm:ss
External Item ID:
Blog Author Definition: BlogAuthorDef
Blog Author Category:
* BlogAuthorCategory (S): BlogAuthors
*Full Name:
Author Profile Image:
Ratings: [no Segments defined]
no segment ratings apply
Related Items: [show local recommendations]

Specifies that blog author assets can belong to only a single category ("BlogAuthors" category by default).

Specifies attributes that make up the blog author form.

Specifies the filters that retrieve the data for a given blog author.

To render an author's full name and image next to the author's blogs, the name entered in this field must match the name entered into the blog asset form's "Author" field.

88.2.6 Default Blog Filters

The Blog flex family contains default filters, which use the field copier filter to copy the content of the system-defined attributes into user-defined flex attributes.

The default blog filters are:

- GetAssetId: Applied to the Blog Category Definition to retrieve the ID of the blog asset.
- GetAuthorUsername: Defined in the Blog Author Definition and applied to the Blog Author to dynamically generate the full name of a given blog's author.
- GetCategory: Applied to the Blog Category Definition to retrieve the appropriate category associated with the blog asset.

- `GetGrandParentCategory`: Applied to the `GrandParentBlogCategory` to retrieve the grandparent category associated with a given blog asset.
- `GetGrandParentDesc`: Applied to the `GrandParentBlogCategory` to retrieve the grandparent category's attributes.
- `GetLargeThumbnail`: Defined in the `Blog Author Definition` and applied to the Blog Author to dynamically generate the large thumbnail image of a given blog's author.
- `GetParentCategory`: Applied to the `ParentBlogCategory` to retrieve the parent category associated with a given blog asset.
- `GetParentDesc`: Applied to the `ParentBlogCategory` to retrieve the parent category's attributes.
- `GetSmallThumbnail`: Defined in the `Blog Author Definition` and applied to the `Blog Author` to dynamically generate the small thumbnail image of a given blog's author.

For instructions on creating additional filters for blog assets, see Section 17.2, "Defining a Flex Filter Class and Creating a Flex Filter Asset."

89

Community Blogs: Sample Blog Pages

This chapter provides information about the default blog components that are supplied by the Community Blogs module. These ready-to-use components render the sample blog pages.

This chapter contains the following sections:

- Section 89.1, "Overview of the Sample Blog Pages"
- Section 89.2, "Components of the Sample Blog Pages"
- Section 89.3, "Next Steps"

89.1 Overview of the Sample Blog Pages

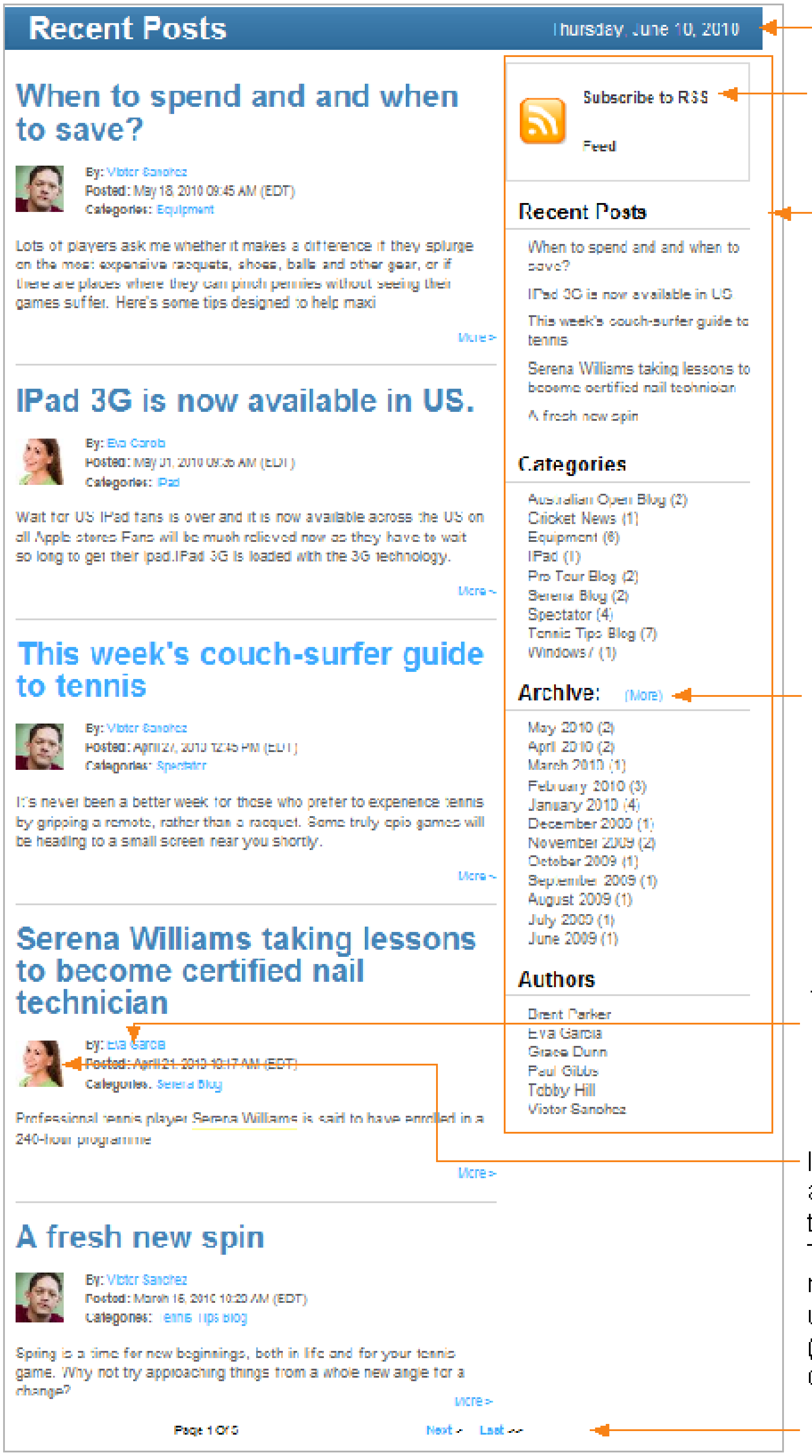
Blog pages are composed of blog assets, blog author assets, a right navigation panel, a header, and a footer. Blog pages are rendered by default blog templates, CSElements, and a SiteEntry asset (as shown in Figure 89–1). By default, all blog pages are cached (except the wrapper page).

The `FW_RecentBlogs` page is the Community Blogs module's main page. It is called by the `FW_RecentBlogsDetail` template.

The wrapper element calls the sample layout template (`FW_BlogLayout`). The layout defines the overall appearance of the sample blog pages and calls the `FW_BlogContainer` template, which then calls the corresponding detail templates for the page, and for the asset types `Blog Category` and `Blog Asset`.

The wrapper's parameters (including the number of blogs that can appear on a page at one time) are defined by the default SiteEntry asset. The Community Blogs module's SiteEntry asset is rendered by the `FW_Blogwrapper CSElement`.

Figure 89–1 Sample Home Page



Recent Posts Thursday, June 10, 2010

When to spend and and when to save?

By: Victor Sanchez
Posted: May 18, 2010 09:45 AM (EDT)
Categories: Equipment

Lots of players ask me whether it makes a difference if they splurge on the most expensive racquets, shoes, balls and other gear, or if there are places where they can pinch pennies without seeing their games suffer. Here's some tips designed to help maxi

More >

IPad 3G is now available in US.

By: Eva Garcia
Posted: May 31, 2010 09:36 AM (EDT)
Categories: IPad

Wait for US IPad fans is over and it is now available across the US on all Apple stores Fans will be much relieved now as they have to wait so long to get their ipad.IPad 3G is loaded with the 3G technology.

More >

This week's couch-surfer guide to tennis

By: Victor Sanchez
Posted: April 27, 2010 12:45 PM (EDT)
Categories: Spectator

It's never been a better week for those who prefer to experience tennis by gripping a remote, rather than a racquet. Some truly epic games will be heading to a small screen near you shortly.

More >

Serena Williams taking lessons to become certified nail technician

By: Eva Garcia
Posted: April 21, 2010 10:17 AM (EDT)
Categories: Serena Blog

Professional tennis player Serena Williams is said to have enrolled in a 240-hour programme

More >

A fresh new spin

By: Victor Sanchez
Posted: March 16, 2010 10:20 AM (EDT)
Categories: Tennis Tips Blog

Spring is a time for new beginnings, both in life and for your tennis game. Why not try approaching things from a whole new angle for a change?

More >

Page 1 Of 5 Next > Last >>

Subscribe to RSS Feed

Recent Posts

When to spend and and when to save?
IPad 3G is now available in US
This week's couch-surfer guide to tennis
Serena Williams taking lessons to become certified nail technician
A fresh new spin

Categories

Australian Open Blog (2)
Cricket News (1)
Equipment (6)
IPad (1)
Pro Tour Blog (2)
Serena Blog (2)
Spectator (4)
Tennis Tips Blog (7)
Windows 7 (1)

Archive: (More)

May 2010 (2)
April 2010 (2)
March 2010 (1)
February 2010 (3)
January 2010 (4)
December 2009 (1)
November 2009 (2)
October 2009 (1)
September 2009 (1)
August 2009 (1)
July 2009 (1)
June 2009 (1)

Authors

Brent Parker
Eva Garcia
Grace Dunn
Paul Gibbs
Tobby Hill
Victor Sanchez

Blog header

RSS Feed rendered by the `FW_BlogRSS` template.

Right navigation panel renders the names of the blog pages and links to the blog assets on those pages.

The default blog pages are:

- Recent Posts
- Archive
- Authors

The **More** link is used to fetch another 12 months of content on top of what is displayed by default.

The full name of the blog's author is displayed next to her blog post(s).

Image of the blog's author displayed next to her blog post(s). This image is retrieved from the user's author asset. (The author asset is optional.)

Blog footer

89.2 Components of the Sample Blog Pages

This section provides information about the default blog templates, CSElements, and SiteEntry asset. These components are pre-configured with default parameters and code which render the sample blog pages.

WebCenter Sites renders the sample blog pages by executing the code associated with the sample page names. You can use the sample blog pages as a reference to create your own blog pages that will conform to the look and feel of your own website.

> **Note:** Page assets cannot be shared between sites. If you wish to use the Community Blogs module on a different CM site, you must create new blog pages for those sites. All other components can be shared. If you wish to completely customize blog functionality on your sites, you can add the necessary blog code to your own site's components, and reconfigure them to fit your requirements. For instructions on creating pages, see Section 90.2.1, "Creating Blog Pages" in Chapter 90, "Community Blogs: Customizing Blog Components."

The sample blog pages can be found in the WebCenter Sites tree (select **Site Plan** tab and then **Unplaced Pages**). The sample pages are described below:

- `FW_RecentBlogs`: This page is the sample home page for the Community Blogs module and displays all recently published blogs.
- `FW_AuthorBlogs`: This page displays the blogs posted by a selected author. The blogs are rendered when you click the author's name in the **Authors** section of the right navigation panel.
- `FW_ArchiveBlogs`: This page displays all the blogs that have been posted in a given month. The blogs are rendered on this page when you click the month's name in the Archive section of the right navigation panel.

These pages are rendered by the `FW_BlogLayout` template. However, to navigate through the pages, start from the sample home page (`FW_RecentBlogs`), because this page initializes the parameters that are required for the other sample blog pages.

For information about the components of the sample blog pages, see the following topics:

- Section 89.2.1, "Default Blog CSElements"
- Section 89.2.2, "Default Blog SiteEntry Asset"
- Section 89.2.3, "Default Blog Templates"
- Section 89.2.4, "Blog Archive More Link Functionality"

89.2.1 Default Blog CSElements

The Community Blogs module provides you with default CSElements, which store reusable code (in the form of XML or JSP and Java) that build the sample blog pages. For example, since the Share link is displayed on all three sample blog pages, a CSElement was created for it (`FW_Blogs/Utils/AddThisWidget`) which contains the full code for the link's functionality.

Table 89–1 lists and describes the Community Blogs module's default CSElements.

Table 89–1 Default Blog CSElements

Name and ElementCatalog Entry	Description
FW_BlogWrapper	The wrapper for the Community Blogs module's sample pages.
FW_Blogs/Utils/AddThisWidget	Renders the Share link, which enables visitors to share blogs between websites.
FW_Blogs/Utils/CreateArchiveLink	Renders a link that calls the FW_ArchiveBlogs template.
FW_Blogs/Utils/CreateAuthorLink	Renders a link that calls the FW_AuthorBlogsDetail template.
FW_Blogs/Utils/GetAssetData	Retrieves the data for the blog asset. For example, this element can retrieve the attributes the blog asset inherits from its parent.
FW_Blogs/Utils/GetAuthor	Retrieves the full name and profile image of a given blog's author from that author's author asset (if any).
FW_Blogs/Utils/GetBlogAuthors	Retrieves the list of authors, and provides links to each author.
FW_Blogs/Utils/GetBlogCategories	Retrieves the list of categories, and provides links to each category.
FW_Blogs/Utils/GetBlogsByMonth	Retrieves the list of months, and provides links to each month.
FW_Blogs/Utils/GetChildren	Retrieves the children for a particular blog parent.
FW_Blogs/Utils/GetExternalURL	Creates the external URLs of the blog assets whose titles and summaries are listed in the RSS Feed. By default, this element creates the local WebCenter Sites URL. To modify the URL you must customize this element, see Section 90.3, "Customizing URLs for the RSS Feed."
FW_Blogs/Utils/GetRecentBlogs	Retrieves the list of recently published blogs, and provides links to their detail pages.

89.2.2 Default Blog SiteEntry Asset

The Community Blogs module's SiteEntry asset represents a pagelet and is associated with the FW_Wrapper CSElement, which is the root element that generates the sample pages' wrapper. The Community Blogs module's SiteEntry asset specifies the blogsperpage pagelet parameter, which defines the number of blogs that can be displayed on a page at one time.

Table 89–2 lists the default pagelet parameters for the FW_BlogWrapper SiteEntry asset.

Table 89–2 FW_BlogWrapper SiteEntry Asset Pagelet Parameters

Pagelet Parameter	Value
blogsperpage	5
rendermode	live
seid	This value is auto-generated.

Table 89–2 (Cont.) FW_BlogWrapper SiteEntry Asset Pagelet Parameters

Pagelet Parameter	Value
`site`	The name of the site on which you installed the Community Blogs module.

The Number of Blogs Per Page

The number of blogs per page is passed as the parameter `blogsperpage` from the `FW_BlogWrapper` SiteEntry asset. If nothing is passed, the default value of `10` is used to specify the number of blogs per page. For instructions on specifying the number of blogs that can be displayed at one time on your site's blog pages, see Section 90.2.4, "Adding Blog Parameters to Your Site's SiteEntry Asset."

89.2.3 Default Blog Templates

The Community Blogs module supplies you with default templates that render the sample blog pages. When a link is clicked, the corresponding template is called to render the content. For example, if you click the **More** link for a given blog post, the associated detail template renders the post's full content.

The page layout of your site is rendered by your own headers, footers, navigation, and content containers. To add blog pages to your website, you can add the necessary code specified in the Community Blogs module's default templates to your site's templates. Table 89–3 lists and provides information about the Community Blogs module's default templates.

Table 89–3 Default Blog Templates

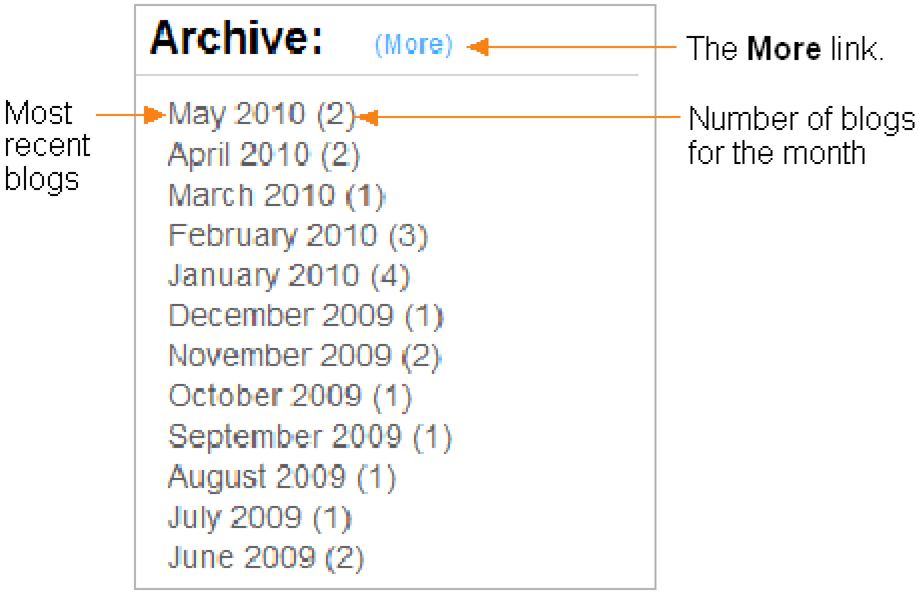
Template	Default Name	Asset Type	Description
Detail	`FW_ArchiveBlogsDetail`	Page	Used to render the blogs listed in the Blogs Archive page.
Detail	`FW_AuthorBlogsDetail`	Page	Used to render the blogs listed under the Blogs By Author page.
Detail	`FW_BlogAssetDetail`	`FW_BlogAsset`	Used to render the full content of a selected blog.
Detail	`FW_BlogCategoryDetail`	`FW_BlogParent`	Used to render the blog categories listed under the Categories section of the right navigation panel.
Detail	`FW_RecentBlogsDetail`	Page	Used to render the blog home page which lists all recently published blogs.
Container	`FW_BlogContainer`	Page	Used to call the detail templates.
Container	`FW_BlogContainer`	`FW_BlogParent`	Used to call the detail templates.
Container	`FW_BlogContainer`	`FW_BlogAsset`	Used to call the detail templates.

Table 89–3 (Cont.) Default Blog Templates

Template	Default Name	Asset Type	Description
Blog Layout	`FW_BlogLayout`	N/A	Used to render the main layout of the sample blog pages.
Blog RSS	`FW_BlogRSS`	N/A	(Typeless) Used to render the `FW_BlogSummaryRSS`.
Blog RSS	`FW_BlogSummaryRSS`	`FW_BlogAsset`	(Typed) Renders the RSS feed on a Web page. Called by `FW_BlogRSS`.
Summary	`FW_BlogSummary`	`FW_BlogAsset`	Used to list the summaries of each blog asset. The summary of a blog asset includes the title, abstract, author's full name and profile image, date, and time.
Link	`FW_Link`	`FW_BlogParent`	Used to render the blog category links.
Link	`FW_Link`	`FW_BlogAsset`	Used to render the blog links.
Navigation	`FW_Nav`	`FW_BlogParent`	For each detail template there is a navigation link which internally calls all the elements (Recent Posts, Categories, Authors, and Archive).
Navigation	`FW_Nav`	`FW_BlogAsset`	For each detail template there is a navigation link which internally calls all the elements (Recent Posts, Categories, Authors, and Archive).
Navigation	`FW_Nav`	N/A	For each detail template there is a navigation link which internally calls all the elements (Recent Posts, Categories, Authors, and Archive).
Navigation	`FW_Nav`	Page	For each detail template there is a navigation link which internally calls all the elements (Recent Posts, Categories, Authors, and Archive).

89.2.4 Blog Archive More Link Functionality

The **More** link, located next to the Archive section in the right navigation panel, is used to fetch additional data associated with the Archive page. By default, the Archive page lists the last 12 months of blogs. Clicking the **More** link multiple times retrieves another 12 months of blog data for every click.



The **More** link is coded in the Community Blogs module's default layout template. For more information about the layout template's code, see Section 90.2.2, "Adding Blog Code" in Chapter 90, "Community Blogs: Customizing Blog Components."

89.3 Next Steps

So far this section of the guide has provided information about the `Blog` flex family and the default components that make up the sample blog pages. The rest of the chapters are procedural. For developers, Chapter 90, "Community Blogs: Customizing Blog Components" contains procedures and code for adding blog functionality to custom pages. For administrators, the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide* contains instructions on enabling blog asset types for the content providers. The *Oracle Fusion Middleware WebCenter Sites User's Guide* provides information on testing the data model and creating blog assets for publication.

90

Community Blogs: Customizing Blog Components

This section provides general information and instructions for reconfiguring the default blog components to render custom pages.

This chapter contains the following sections:

- Section 90.1, "Customizing the Blog Asset Form"
- Section 90.2, "Adding Blog Functionality to CM Sites"
- Section 90.3, "Customizing URLs for the RSS Feed"

90.1 Customizing the Blog Asset Form

The Blog flex family hierarchy can be modified to fit your requirements. This section provides instructions for creating a new blog attribute and adding that attribute to the blog asset definition to display the attribute as a field in the blog asset form.

> **Note:** For your own reference, if you wish to see the hierarchical relationships between blog categories and blog assets, you can create a tree tab. For instructions on creating a tree tab, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

This section contains the following topics:

- Section 90.1.1, "Creating a Blog Attribute"
- Section 90.1.2, "Adding a Blog Attribute to the Blog Asset Definition"

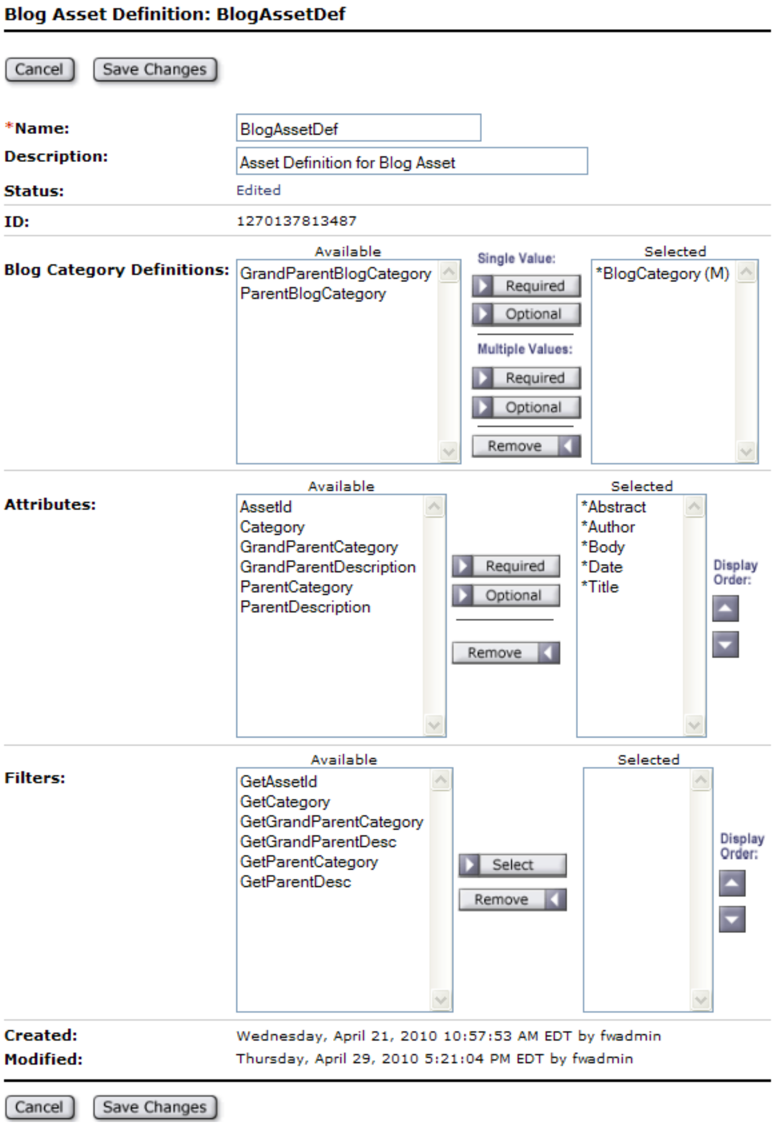
90.1.1 Creating a Blog Attribute

The blog attribute you create will be displayed as a field in the blog asset form once you add the attribute to the blog asset definition.

To create a blog attribute

1. Log in to the WebCenter Sites Admin interface as a general administrator.
2. Select the site on which the Community Blogs module is enabled.
3. In the button bar, click **New**.
4. Click **New Blog Attribute**.
5. In the Blog Attribute form, fill in the fields:

Figure 90–1 Blog Attribute Form



Blog Asset Definition: BlogAssetDef

***Name:**	BlogAssetDef
Description:	Asset Definition for Blog Asset
Status:	Edited
ID:	1270137813487

Blog Category Definitions:

Available	Selected
GrandParentBlogCategory ParentBlogCategory	*BlogCategory (M)

Attributes:

Available	Selected
AssetId Category GrandParentCategory GrandParentDescription ParentCategory ParentDescription	*Abstract *Author *Body *Date *Title

Filters:

Available	Selected
GetAssetId GetCategory GetGrandParentCategory GetGrandParentDesc GetParentCategory GetParentDesc	

Created:	Wednesday, April 21, 2010 10:57:53 AM EDT by fwadmin
Modified:	Thursday, April 29, 2010 5:21:04 PM EDT by fwadmin

Note: The fields you need to fill in can differ significantly based on the data type that you select for your attribute.

- **Name**: Enter a name of up to 64 characters (the name cannot contain spaces).
- **Description**: Enter a short summary that describes the use or function of the attribute.
- **Value Type**: Select a data type for this attribute.
- **Asset Type**: If the attribute is of type `asset`, select an asset from the drop-down list.
- **Mirror Dependency Type**: If the attribute is of type `asset`, select a dependency type.
- **Folder**: (Optional) If the attribute is of type `blob`, enter a path to the directory that you want to store the attribute values in.

- **Allow Embedded Links**: If the attribute is of type `text`, `blob`, or `URL`, select whether links to other pages or websites can be embedded in the attribute's content field.
- **Number of Values**: Select either **single** or **multiple** from the drop-down list, depending on the data type selected for the **Value Type** field.
- **Attribute Editor**: (Optional) If you do not wish to use the default input type, click in the **Attribute Editor** field to select the appropriate attribute editor for the field.
- (Optional) **Character Set**: If you need to override the default ISO character set (ISO-8859-1), enter the character set you want to use for this attribute.

If you are creating a foreign attribute (keeping data in an external system) fill in the following fields:

- **Editing Style**: If you want this attribute to be available to users in its native table on the external system, select **external**.
- **Storage Style**: Select **external**. For more information, see Section 16.3.6.4, "Creating Foreign Flex Attributes."
- **External ID**: Specify the name of the column that serves as the primary key for the table that holds this foreign attribute (the column that uniquely identifies the attribute).
- **External Table**: Enter the name of the table that stores this attribute.
- **External Column**: Enter the name of the column in the table specified in the **External Table** field that holds the value of the attribute.

6. Click **Save**.

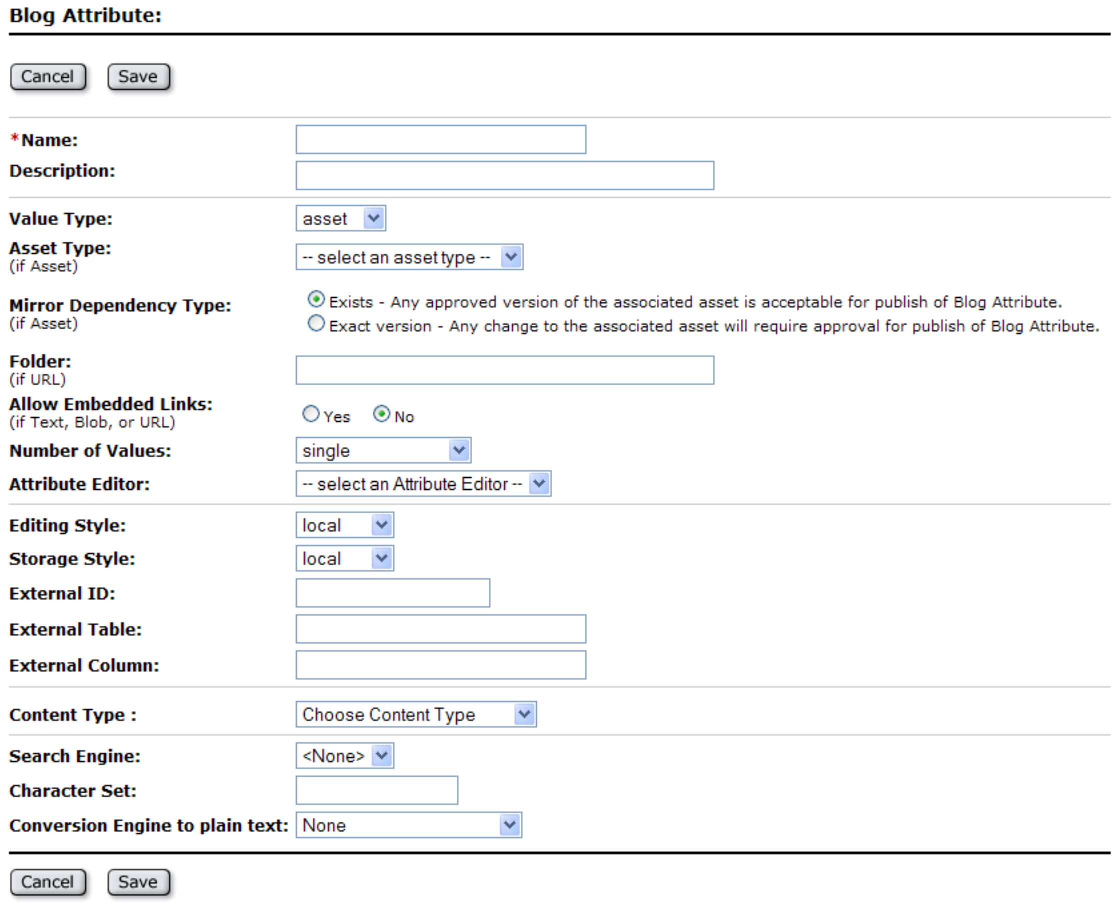
   Now that you have created the attribute, add the attribute to the blog asset definition. For instructions, see the next section.

90.1.2 Adding a Blog Attribute to the Blog Asset Definition

To add an attribute to the blog asset form you must add the attribute to the blog asset definition.

To add an attribute to the blog asset definition

1. Log in to the WebCenter Sites Admin interface as a general administrator, and select the site on which the Community Blogs module is enabled.
2. Access the blog asset definition's Inspect form:
   a. In the button bar, click **Search**.
   b. In the Search form, click **Find Blog Asset Definition** , and click **Search**.
   c. Select **BlogAssetDef**.
3. In the Inspect form, click **Edit**.



Blog Attribute:

***Name:**	
Description:	
Value Type:	asset
Asset Type: (if Asset)	-- select an asset type --
Mirror Dependency Type: (if Asset)	● Exists - Any approved version of the associated asset is acceptable for publish of Blog Attribute. ○ Exact version - Any change to the associated asset will require approval for publish of Blog Attribute.
Folder: (if URL)	
Allow Embedded Links: (if Text, Blob, or URL)	○ Yes ● No
Number of Values:	single
Attribute Editor:	-- select an Attribute Editor --
Editing Style:	local
Storage Style:	local
External ID:	
External Table:	
External Column:	
Content Type :	Choose Content Type
Search Engine:	<None>
Character Set:	
Conversion Engine to plain text:	None

4. In the **Attributes** field, select the attribute(s) from the **Available** list and use the **Required** or **Optional** button to move the attribute(s) to the **Selected** list. Which button you choose determines whether the attribute(s) will be required or optional in the blog asset form.

5. Click **Save**.

   The attributes you selected are now included as fields in the blog asset form. When a user creates a blog asset, the new attributes will be displayed as either required or optional fields.

90.2 Adding Blog Functionality to CM Sites

If you wish to use the Community Blogs module on a different CM site, you must create new blog pages for those sites. Adding blog functionality to your website entails the following steps:

1. Create pages on the content management site that will be used to render blog assets on the website.

2. Copy the blog code from the default blog templates and CSElements to your site's templates and CSElements. How your site is set up determines the modifications you must make to the default code once you insert it into your templates and CSElements.

3. Add the `blogsperpage` parameter that is specified in the Community Blogs module's SiteEntry asset to your own site's SiteEntry asset.

This section contains the following topics:

- Section 90.2.1, "Creating Blog Pages"
- Section 90.2.2, "Adding Blog Code"
- Section 90.2.3, "Adding Community-Gadgets Functionality"
- Section 90.2.4, "Adding Blog Parameters to Your Site's SiteEntry Asset"

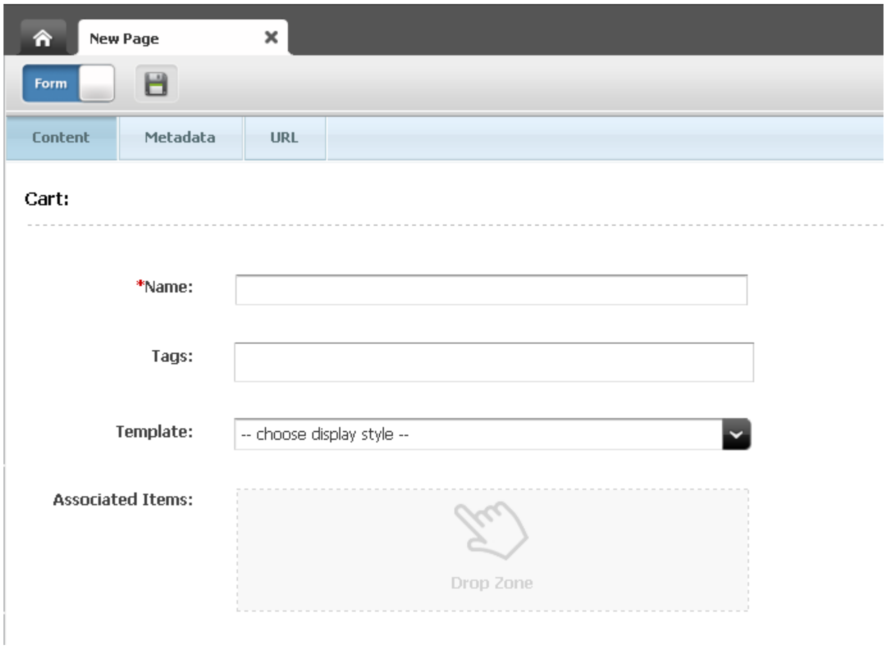
90.2.1 Creating Blog Pages

Before creating blog pages, map out their types: the main blog page (which is the `FW_RecentBlogs` page in the Community Blogs module), category pages, and so on. Also determine your site's graphical, navigational, and functional features in order to create blog pages that will conform to the layout of your website.

To create blog pages

1. Log in to WebCenter Sites as a general administrator, select the site on which you wish to create the pages for displaying blogs to website visitors and then select the icon for the WebCenter **Contributor** interface.
2. In the menu bar, select **Content**, then select **New**, and then select **New Page**.

   A tab opens displaying the New Page form.



3. In the New Page form, fill in the fields:
   - **Name**: Enter a name of up to 64 characters.
   - **Tags**: Enter a single word or phrase to attach to the page.
   - **Template**: Select the template that will render the page.
   - **Associated Items**: Add content (for example, related articles) to this field's Drop Zone.
4. Click the **Save** icon.

Now that you have created a page to display blog assets, code your site's templates and CSElements to call the new page and render blog functionality on your website. For more information, see Section 90.2.2, "Adding Blog Code."

90.2.2 Adding Blog Code

The Community Blogs module is configured to render sample blog pages. Your own site's layout is likely to differ from the layout of the sample blog pages. For example, your site may call a left navigation, while the sample blog pages call a right navigation. Instead of coding your templates from scratch to incorporate blog functionality, you can reuse the sample code by inserting it into your own templates and CSElements, then reconfiguring the code as necessary.

> **Note:** Your site's wrapper element renders the layout of your site. To ensure that your wrapper element renders blog pages in addition to existing pages, copy the relevant blog code from the default `FW_Wrapper` element to your site's wrapper element

To add blog code to your site's layout template

1. Log in to the WebCenter Sites Admin interface as a general administrator.
2. Select the site on which the Community Blogs module was installed.
3. Access the `FW_BlogLayout` template:
   a. In the button bar, click **Search**.
   b. In the Search list, select **Find Template**.
   c. In the Search field, enter **FW_BlogLayout**, and then click **Search**.
   d. Click **FW_BlogLayout**.
4. In the layout template's Inspect form, click **Edit**.
5. Copy the necessary code from the Community Blogs module's layout template and insert it into your own layout template:

   The following lines retrieve the site description from the wrapper and load the site:

```
<ics:if condition='<%=ics.GetVar("tid")!=null%>'>
  <ics:then>
    <render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
  </ics:then>
</ics:if>

<publication:load name='Publication' field="name"
    value='<%=ics.GetVar("site")%>'/>
<publication:get name='Publication' field="id" output="spubid"/>
<publication:get name='Publication' field="description" output="pubdesc"/>
```

   The following line retrieves the body of the page:

```
String sContainerTName = "FW_BlogContainer";
```

   The following lines load the site, page, and asset descriptions:

```
   String sTitle = "";
  if (!"Page".equals(ics.GetVar("c")))
  {
     %><asset:load name='t2' type='<%=ics.GetVar("c")%>'
     objectid='<%=ics.GetVar("cid")%>' /><%
     %><asset:get name='t2' field='name' output='t2Name' /><%
     %><asset:get name='t2' field='description' output='t2Desc' /><%
     sTitle += ": "+(Utilities.goodString(ics.GetVar("t2Desc")) ?
     ics.GetVar("t2Desc") : ics.GetVar("t2Name"));
     } else if (Utilities.goodString( ics.GetVar("p") )) {
     %><asset:load name='t1' type='Page' objectid='<%=ics.GetVar("p")%>' /><%
     %><asset:get name='t1' field='name' output='t1Name' /><%
     %><asset:get name='t1' field='description' output='t1Desc' /><%
     sTitle += ": "+(Utilities.goodString(ics.GetVar("t1Desc")) ?
     ics.GetVar("t1Desc") : ics.GetVar("t1Name"));
}
```

The following line calls the style sheet for the blog layout template. The style sheet defines the look and feel of the sample blog pages. Because your site already has its own style sheet, copy the parameters you need from the Community Blogs module's style sheet and insert them into your own site's style sheet. Make sure you resolve any conflicts between the Community Blogs module's style sheet and your own style sheet:

```
<render:callelement elementname="FW_Blogs/CSS/blogsCSS/">
```

The following lines are the JavaScript method that retrieves the **More** link for the Archive blogs page:

```
<script type="text/javascript">
function getMoreBlogs(url) {
  var xhtReq = getXMLHttpRequest();
  xhtReq.open(GET, url, true);
  xhtReq.onreadystatechange = function() {
    if(xhtReq.readyState==3) {
      document.getElementById('moreLink').innerHTML =
        ' <img src="<%=ics.GetSSVar("baseurl")%>images/wait_ax_tiny.gif/>';
    }
    else if(xhtReq.readyState==4) {
      document.getElementById('archiveDiv').innerHTML = xhtReq.responseText;
    }
  };
  xhtReq.send(null);
}

function getXMLHttpRequest() {
  try { return new XMLHttpRequest(); } catch(e) {}
  try { return new ActiveXObject("Msxml2.XMLHTTP"); } catch (e) {}
  alert("XMLHttpRequest not supported");
  return null;
}
</script>
```

The following lines call the main blog page or the body of a given asset. These lines also call the header and footer for the blog sample pages. Since your site has its own headers and footers, copy only the code you require for your site's blog functionality, and retain your own site's headers and footers:

```
<!--main start-->
<div id="main">

  <!--header start-->
  <%-- <div id="header">
  <!--Call header from here-->
```

```
      </div> --%>

      <!--header end container start-->
      <div id="container">
      <!--Body -->
      <%-- Call the container template for the current page subtype --%>
        <render:calltemplate
         tname='<%=sContainerTName%>'
         site='<%=sSite%>'
         tid='<%=ics.GetVar("tid")%>'
         slotname="BlogBodyContainer"
         c='<%=ics.GetVar("c")%>'
         cid='<%=ics.GetVar("cid")%>'
         ttype="Template">
            <render:argument name=p
             value='<%=ics.GetVar("p")%>' />
            <render:argument name=locale
             value='<%=ics.GetVar("locale")%>'/>
            <render:argument name=packedargs
             value='<%=ics.GetVar("packedargs")%>'/>
            <render:argument name=site
             value='<%=ics.GetVar("site")%>'/>
            <render:argument name=spubid
             value='<%=ics.GetVar("spubid")%>'/>
            <render:argument name=blogsperpage
             value='<%=ics.GetVar("blogsperpage")%>'/>
        </render:calltemplate>
      </div><!-- End of container -->
    </div><!-- End of main -->

    <!-- Footer -->
    <%-- <div id="footer">
    <!--Call footer from here-->
    </div> --%>
    </body>
```

6. Reconfigure the blog code you inserted into your site's layout template to fit your requirements.
7. Inspect the code of the other default blog templates and CSElements and copy the relevant sections into your own templates and CSElements.

90.2.3 Adding Community-Gadgets Functionality

If the Community Blogs module was integrated with Community-Gadgets, the tags for the comments widget and comments link were added to the `FW_BlogSummary` and `FW_BlogAssetDetail` templates. To add Community-Gadgets functionality to custom pages, copy the widget tags from those templates to your custom templates.

To add a comments widget to a page, copy the following lines from the default blog templates to the custom templates:

```
<%
boolean cosEnabled = Boolean.parseBoolean(ics.GetProperty("fwblogs.cos.enabled",
"futuretense_xcel.ini", true));
if(cosEnabled)
{
String cosProtocol = ics.GetProperty("cos.protocol", "futuretense_xcel.ini",
true);
String cosHost = ics.GetProperty("cos.hostname", "futuretense_xcel.ini", true);
String cosPort = ics.GetProperty("cos.portnumber", "futuretense_xcel.ini", true);
String cosContext = ics.GetProperty("cos.contextroot", "futuretense_xcel.ini",
true);
String cosUrl = cosProtocol + "://" + cosHost + ":" + cosPort + "/" + cosContext;
%>

<div id="comments_container"></div>
<script type="text/javascript">
    cos = window.cos || {};
    cos.pageWidgets = cos.pageWidgets || [];

cos.pageWidgets.push({name: "wsdk.comments",
version: "1.5",
elementID: "comments_container",
attributes: {"site_id":"<%=ics.GetVar("site")%>","resource_
id":"<%=ics.GetVar("cid")%>"}});
setTimeout(
    function(){if ((typeof(wsdk) == 'undefined') || (typeof(wsdk.comments) ==
    'undefined')) {element:document.getElementById('comments_container')
    .innerHTML = "<div style='font-family: Tahoma, Verdana, Geneva,
    sans-serif;font-size: 12px;color: #333333;border: 1px solid
    #dbdfe1;padding-left: 5px;padding-top: 4px;height: 22px;'>Comments is
    unavailable right now. Please contact the site administrator.</div>";}}
    ,30000);

    cos.pageScripts = cos.pageScripts || [];
    cos.pageScripts.push('wsdk.comments');

    (function()
    {
        var oldOnloadHandler = window.onload || function()
        {
        };
        if (!oldOnloadHandler.alreadyProcessed)
        {
            window.onload = function()
            {
                var script = document.createElement('script');
                script.src = '<%=cosUrl%>/wsdk/widget/'
                        + cos.pageScripts.join(':') + '.js?site_
id=<%=ics.GetVar("site")%>';
                script.type = 'text/javascript';
                script.charset = 'utf-8';
                document.getElementsByTagName("head").item(0).appendChild(script);
                oldOnloadHandler.apply(this, arguments);
            };
            window.onload.alreadyProcessed = true;
        }

    })();
</script>

<%}%>
```

To add the comments link widget to a page, copy the following lines from the default blog templates to the custom templates:

```
boolean cosEnabled = Boolean.parseBoolean(ics.GetProperty("fwblogs.cos.enabled",
"futuretense_xcel.ini", true));
if(cosEnabled)
{
```

```
String cosProtocol = ics.GetProperty("cos.protocol", "futuretense_xcel.ini",
true);
String cosHost = ics.GetProperty("cos.hostname", "futuretense_xcel.ini", true);
String cosPort = ics.GetProperty("cos.portnumber", "futuretense_xcel.ini", true);
String cosContext = ics.GetProperty("cos.contextroot", "futuretense_xcel.ini",
true);
String cosUrl = cosProtocol + "://" + cosHost + ":" + cosPort + "/" + cosContext;
%>

<script type="text/javascript">
    cos = window.cos || {};
    cos.pageWidgets = cos.pageWidgets || [];

    cos.pageWidgets.push({name: "comments-link",
        version: "0.1",
        elementID: "comments_link_div",
        attributes: {"site_id":"<%=ics.GetVar("site")%>"}});

    cos.pageScripts = cos.pageScripts || [];
    cos.pageScripts.push('comments-link');

    (function()
    {
        var oldOnloadHandler = window.onload || function()
        {
        };
        if (!oldOnloadHandler.alreadyProcessed)
        {
            window.onload = function()
            {
                var script = document.createElement('script');
                script.src = '<%=cosUrl%>/wsdk/widget/'
                        + cos.pageScripts.join(':') + '.js?site_
id=<%=ics.GetVar("site")%>';
                script.type = 'text/javascript';
                script.charset = 'utf-8';
                document.getElementsByTagName("head").item(0).appendChild(script);
                oldOnloadHandler.apply(this, arguments);
            };
            window.onload.alreadyProcessed = true;
        }

    })();
</script>

<%}%>
```

90.2.4 Adding Blog Parameters to Your Site's SiteEntry Asset

The Community Blogs module's SiteEntry asset specifies the `blogsperpage` parameter, which enables you to specify the number of blog assets that can be displayed on a Web page at one time. You can specify this parameter in your own site's SiteEntry asset.

Note: If you do not specify the `blogsperpage` parameter in your site's SiteEntry asset, then the default number of blogs per page, which is 10, will be used. For information about the `blogsperpage` parameter, see Section 89.2.2, "Default Blog SiteEntry Asset."

To add the `blogsperpage` parameter to your custom SiteEntry asset

1. Log in to the WebCenter Sites Admin interface as a general administrator.
2. Select the site to which you are adding blog functionality.
3. In the button bar, click **Search** to find your site's SiteEntry asset.
   a. In the Search form, click **Find SiteEntry**.
   b. Click **Search**.
   c. Select your site's SiteEntry asset.
4. In the SiteEntry asset's Inspect form, click **Edit**.
5. In the **Pagelet Parameters** field, add the following:
   - **Name**: Enter `blogsperpage`.
   - **Value**: Enter the number of blogs that can be displayed on a page at one time.
6. Click **Save**.

90.3 Customizing URLs for the RSS Feed

The right navigation panel of the sample blog pages contain an RSS Feed link. When a visitor clicks the RSS Feed link, it renders an up-to-date listing of the titles and summaries of the blog assets that are published to the sample blog pages. When a visitor clicks the title of a blog, the entire content of the selected blog is rendered. External URLs for the blog assets included in the RSS Feed are created by the `GetExternalURL` CSElement. By default this element creates the local WebCenter Sites URL:

```
http://<host name>:<port number>/<application context>/<path to file>
```

where `<host name>` is the host name of the WebCenter Sites installation from which the assets of the RSS Feed are accessible, `<port number>` is the port number of the WebCenter Sites application, and `<application context>` is the context of the WebCenter Sites application on which the Community Blogs module is running.

If you wish to use the RSS Feed with your own site's external URLs, you must modify the `GetExternalURL` element's URL string, and add the parameters listed in Table 90–1 to the `futuretense_xcel.ini` file with the values for the host information and context of your site. The `GetExternalURL` element reads these parameters from the `futuretense_xcel.ini` file to create the external URLs for the blog assets listed in your site's RSS Feed.

Table 90–1 Parameters read by GetExternalURL element to create external URLs

Parameter	Description
`fwblogs.hostscheme`	Specifies the top level of the URL naming structure. For example, `http`.
`fwblogs.hostname`	Specifies the host name of the WebCenter Sites installation from which the assets of the RSS Feed are accessible.
`fwblogs.portnumber`	Specifies the port number of the WebCenter Sites application.
`fwblogs.contextinfo`	Specifies the context of the WebCenter Sites installation on which the Community Blogs module is running.

Part VIII

Developing Gadgets

This part introduces template developers to the process of creating gadgets for the Oracle WebCenter Sites: Gadgets application. Sample gadgets, which are included with the Gadgets application, are used throughout this section of the guide to illustrate the development task. This set of fully operational gadgets runs on the FirstSiteII sample site, available in Oracle WebCenter Sites. The sample gadgets' underlying asset model and template framework provide the tools developers need to get started with creating their own gadgets.

This part contains the following chapters:

- Chapter 91, "About Developing Gadgets"
- Chapter 92, "Gadgets: Template Flow"
- Chapter 93, "Gadgets: Creating Your Own Gadgets"

91

About Developing Gadgets

This chapter provides guidelines for creating gadgets based on WebCenter Sites template code.

This chapter contains the following sections:

91.1 Before You Begin

Users of this section of the guide must have:

- A developer's knowledge of WebCenter Sites' basic and flex asset models and templating.
- The Google Gadget API. Related resources are available at the following URLs:
  - The API Reference is available at:

    `http://code.google.com/apis/gadgets/docs/reference/`
  - The API Developer's Guide is available at:

    `http://code.google.com/apis/gadgets/docs/dev_guide.html`
- Familiarity with OpenSocial Standards, used to create the environment that enables gadget users to set preferences. OpenSocial documentation is available at:

  `http://wiki.opensocial.org/`
- The Gadgets application's sample gadgets installed on the WebCenter Sites FirstSiteII sample site. For Gadgets application installation and configuration details, see the *Oracle Fusion Middleware WebCenter Sites Installation Guide* For more information on the Gadgets application, see the *Oracle Fusion Middleware WebCenter Sites User's Guide*.

91.2 Gadget Specifications

Four sample gadgets are included with the Gadgets application. They are enabled on the FirstSiteII sample site. You can develop your own gadgets, using the processes outlined in this guide. The sample gadgets are:

- **List Gadget**, which presents a listing of headlines and article summaries, linking to their respective full articles.
- **ThumbList Gadget**, which presents a list of products, with a thumbnail accompanying each product's description.
- **Slideshow Gadget**, which renders a series of product images into a slideshow, where the user can click on a thumbnail to view a larger image, then click the larger image to open the page containing full details on the product.
- **RSS Feed**, which presents a list of headlines retrieved from an RSS feed. Each headline links to a full article.

The rest of this chapter provides information about the gadgets' major components and describes the sample gadgets in detail.

This section contains the following topics:

- Section 91.2.1, "Asset Model and Templates"
- Section 91.2.2, "Sample Assets"
- Section 91.2.3, "Auxiliary Files"

91.2.1 Asset Model and Templates

Installing the sample gadgets on FirstSiteII installs the basic components for creating and rendering gadgets. The components are the data model, templates, and sample assets that provide the gadgets' content:

- `FW_CSGadget` asset type (its description is `CS-Based Gadget`). All sample gadgets are of type `FW_CSGadget` .
- `FW_RSS` asset type (its description is `RSS Feed`). This asset type is used to specify a URL as the source of content for the RSS Feed gadget.
- `FW_CSGadget/GenerateGadgetXML` template, which is accessed by the Gadgets application. This template is used to render the gadget descriptor XML (also referred to as gadget specification XML).
- `FW_CSGadget/ListSiteGadgets` template, which provides a gadget descriptor URL for each gadget on the current content management site.

91.2.2 Sample Assets

The sample assets either provide content for the gadgets or they render the gadgets. The sample assets are referenced by the sample gadgets as described below:

- Gadget content is provided by:
  - Content assets of type `Content_C` (with parent of type `Content_P`), representing sports articles. These assets are used by the List Gadget.
  - Product assets of type `Product_C` (with parent of type `Product_P`), representing sports products. These assets are used by the ThumbList and Slideshow gadgets.
  - Media assets of type `Media_C` (with parent of type `Media_P`), representing images used by the Product assets. These assets are used by the ThumbList and Slideshow gadgets.
  - Recommendation (AdvCols) assets, encapsulating the Content and Product assets. Recommendation assets are used by the List, ThumbList, and Slideshow sample gadgets.

- Content of type `FW_RSS`, which specifies a URL as the source of content for the RSS Feed gadget.

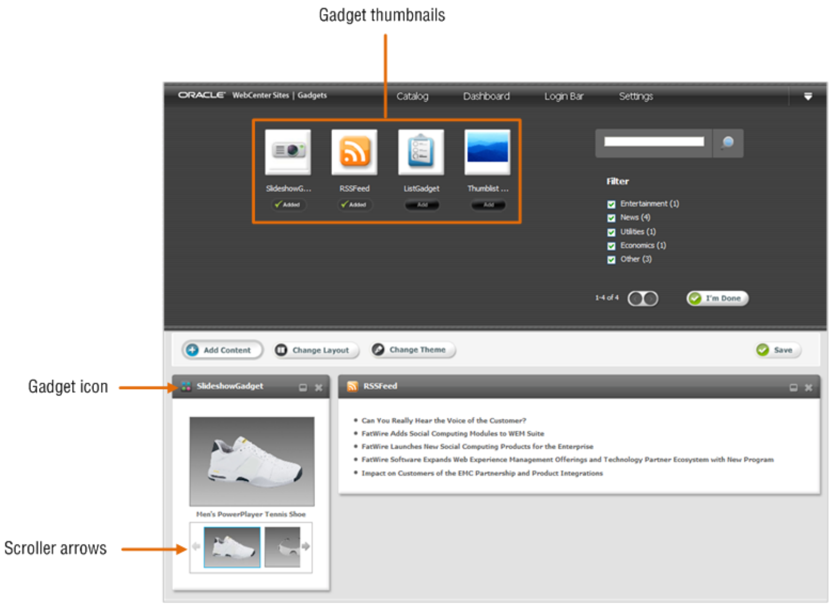
- Templates render the gadgets:
  - The `GenerateGadgetXML` template is accessed by the Gadgets application and calls the templates listed below.
  - An `FW_CSGadget` -typed template exists for each of the sample gadgets. Each of these templates outputs the body of a gadget descriptor XML understandable by the Gadgets application. The templates are `G_List`, `G_RSS`, `G_Slideshow`, and `G_ThumbList`.
  - A typed template named G_JSON exists for each of the asset types referenced by the sample gadgets: `Content_C`, `Media_C`, `Product_C`, and `Recommendation` (`AdvCols`). These templates provide JSON-formatted output containing data necessary to render the HTML for each asset that is displayed in the gadgets. The templates are invoked via remote requests made in the gadget code.

91.2.3 Auxiliary Files

The following image files are used by the sample gadgets:

- Scroller arrow images used by the Slideshow gadget (Figure 91–1 and Figure 91–2). These static images are located in the `FirstSiteII/gadgets` subdirectory under the WebCenter Sites web application.
- Images used as icons, thumbnails, and previews to represent a gadget's various sections. The default images are located in the `sample/GadgetImages` directory in the `gadgetserver.zip` installation package.

Figure 91–1 Images used by sample gadgets



Gadget thumbnails

Gadget icon

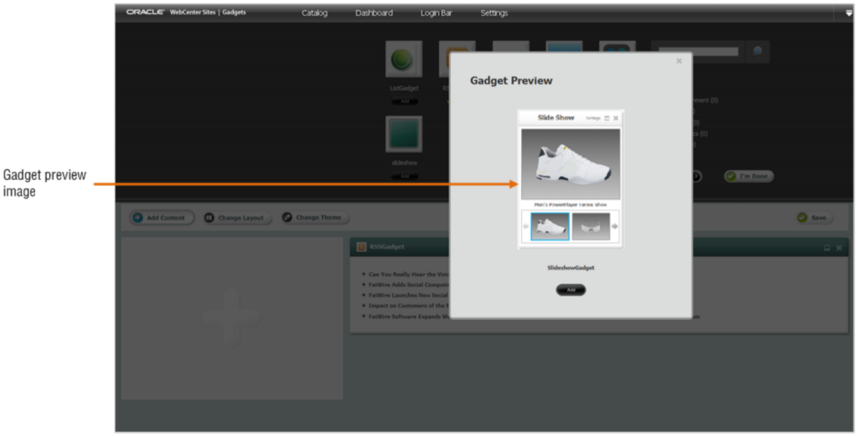
Scroller arrows

☑ Entertainment (1)
☑ News (4)
☑ Utilities (1)
☑ Economics (1)
☑ Other (3)

- Can You Really Hear the Voice of the Customer?
- FatWire Adds Social Computing Modules to WEM Suite
- FatWire Launches New Social Computing Products for the Enterprise
- FatWire Software Expands Web Experience Management Offerings and Technology Partner Ecosystem with New Program
- Impact on Customers of the EMC Partnership and Product Integrations

Men's PowerPlayer Tennis Shoe

Figure 91–2 Images used by sample gadgets



Gadget preview image

91.3 Sample Gadgets

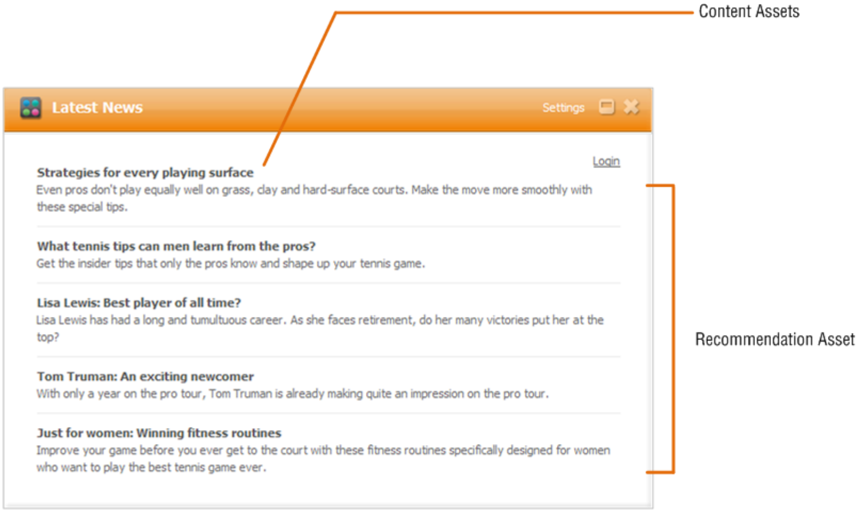
This section outlines the WebCenter Sites sample gadgets and the assets they use. Each asset is listed in the same order in which it is called by the Gadgets application when the Gadgets application initiates the request. This section of the guide uses the List Gadget in particular to illustrate various concepts.

This section contains the following topics:

- Section 91.3.1, "List Gadget"
- Section 91.3.2, "ThumbList Gadget"
- Section 91.3.3, "Slideshow Gadget"
- Section 91.3.4, "RSS Feed Gadget"

91.3.1 List Gadget

The List Gadget renders a Recommendation asset containing a list view of Content assets. Each headline links to its full article.



Content Assets

Recommendation Asset

Strategies for every playing surface
Even pros don't play equally well on grass, clay and hard-surface courts. Make the move more smoothly with these special tips.

What tennis tips can men learn from the pros?
Get the insider tips that only the pros know and shape up your tennis game.

Lisa Lewis: Best player of all time?
Lisa Lewis has had a long and tumultuous career. As she faces retirement, do her many victories put her at the top?

Tom Truman: An exciting newcomer
With only a year on the pro tour, Tom Truman is already making quite an impression on the pro tour.

Just for women: Winning fitness routines
Improve your game before you ever get to the court with these fitness routines specifically designed for women who want to play the best tennis game ever.

The List Gadget asset references the Recommendation asset to be rendered in the gadget and the template (`G_List`) that will render the gadget. The relevant assets are:

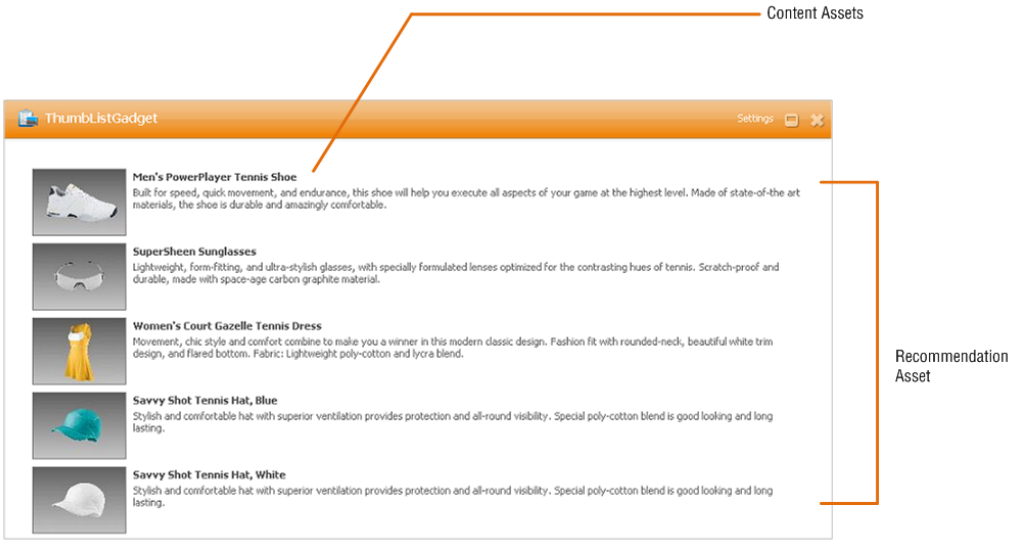
- The `ListGadget` asset of type `FW_CSGadget`, referencing the Recommendation asset to be rendered.
- The G_List template, which produces the body of the gadget descriptor XML used by the Gadgets application to render the gadget. This process is outlined in Chapter 92, "Gadgets: Template Flow."
- `G_JSON` templates, which render the Recommendation asset and its content:

  `AdvCols/G_JSON and Content_C/G_JSON`

- The Recommendation (AdvCols) asset to be rendered in the gadget.
- Content assets included in the Recommendation.

91.3.2 ThumbList Gadget

The ThumbList Gadget is similar to the List Gadget in that it also renders a Recommendation asset. The gadget contains a list view of Product assets. Each headline links to its product page. However, in this gadget, each product is accompanied by a thumbnail image from a Media asset, which is associated with the Product assets via their "Image" attribute.



Men's PowerPlayer Tennis Shoe
Built for speed, quick movement, and endurance, this shoe will help you execute all aspects of your game at the highest level. Made of state-of-the art materials, the shoe is durable and amazingly comfortable.

SuperSheen Sunglasses
Lightweight, form-fitting, and ultra-stylish glasses, with specially formulated lenses optimized for the contrasting hues of tennis. Scratch-proof and durable, made with space-age carbon graphite material.

Women's Court Gazelle Tennis Dress
Movement, chic style and comfort combine to make you a winner in this modern classic design. Fashion fit with rounded-neck, beautiful white trim design, and flared bottom. Fabric: Lightweight poly-cotton and lycra blend.

Savvy Shot Tennis Hat, Blue
Stylish and comfortable hat with superior ventilation provides protection and all-round visibility. Special poly-cotton blend is good looking and long lasting.

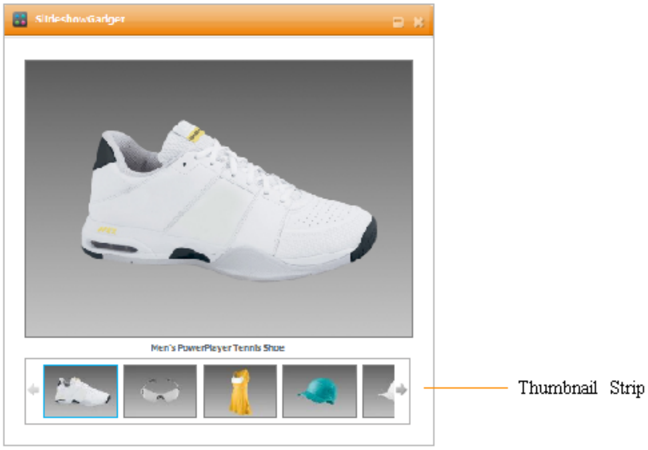
Savvy Shot Tennis Hat, White
Stylish and comfortable hat with superior ventilation provides protection and all-round visibility. Special poly-cotton blend is good looking and long lasting.

The relevant assets are:

- The `ThumbListGadget` asset of type `FW_CSGadget` , referencing the Recommendation to be rendered and the template (`G_ThumbList`) that will render the gadget.
- The G_ThumbList template which produces the body of the gadget descriptor XML used by the Gadgets application to render the gadget.
- `G_JSON` templates, which render the Recommendation asset and its content:`AdvCols/G_JSON` , `Product_C/G_JSON` , and `Media_C/G_JSON`
- The `Recommendation (AdvCols)` asset to be rendered in the gadget.
- Product assets included in the Recommendation asset.
- Media assets referenced by the "Image" attribute of each of the Product assets.

91.3.3 Slideshow Gadget

The Slideshow gadget uses the same content as the ThumbList gadget, but presents the content in an entirely different manner. This gadget displays a thumbnail strip filled with product images. Clicking on an image in the strip displays the image at the maximum size allowed by the gadget area. Clicking the full-size image opens the product's detail page.

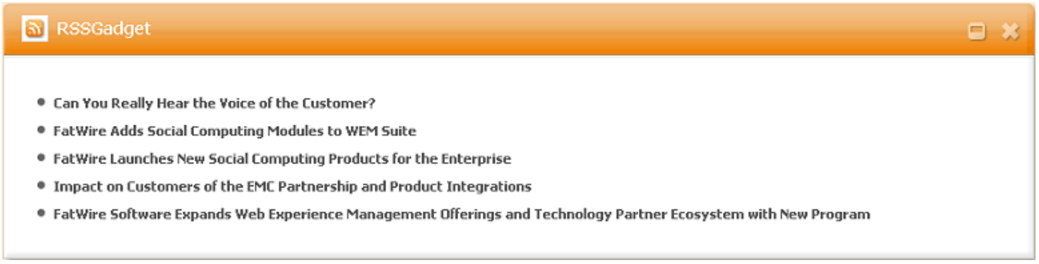


The relevant assets are:

- The `SlideshowGadget` asset referencing the Recommendation asset to be rendered and the template (`G_Slideshow`) that will render the gadget.
- The G_Slideshow template which produces the body of the gadget descriptor XML used by the Gadgets application to render the gadget.
- `G_JSON` templates which render the Recommendation asset and its content:`AdvCols/G_JSON` , `Product_C/G_JSON` , and `Media_C/G_JSON`
- The `Recommendation (AdvCols)` asset to be rendered in the gadget.
- Product assets included in the Recommendation asset.
- Media assets referenced by the "Image" attribute of each Product asset.

91.3.4 RSS Feed Gadget

The RSS Feed gadget displays the most recent items from an RSS feed, utilizing the Google Gadget API for its built-in feed-reading functionality.



- Can You Really Hear the Voice of the Customer?
- FatWire Adds Social Computing Modules to WEM Suite
- FatWire Launches New Social Computing Products for the Enterprise
- Impact on Customers of the EMC Partnership and Product Integrations
- FatWire Software Expands Web Experience Management Offerings and Technology Partner Ecosystem with New Program

The relevant assets are:

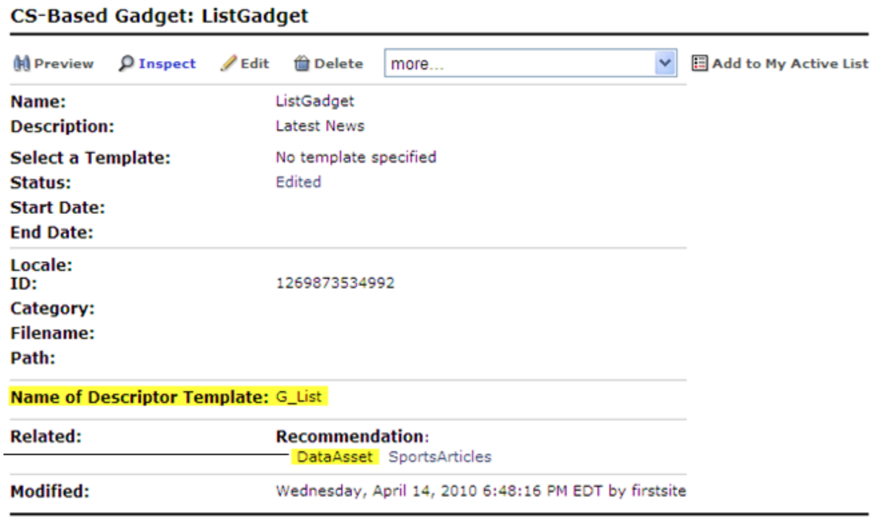
- The `RSSGadget` asset of type `FW_CSGadget` referencing the feed (`FW_RSS`) to be requested and the template (`G_RSS`) that will render the gadget.

- The G_RSS template which produces the body of the gadget descriptor XML used by the Gadgets application to render the gadget.
- An `FW_RSS` asset containing the feed URL to be requested in the gadget.

91.4 Asset Structure

The sample gadgets are based on a new asset type, `FW_CSGadget` , which is used to specify information required for rendering the gadgets. It is a basic asset type, containing the following attributes:

- `Name of descriptor template` is used to specify the name of the `FW_CSGadget` template that will be invoked to generate the gadget's descriptor XML. For the List Gadget, the template is FW_CSGadget/G_List (hence, G_List is specified in this field and later resolved as a typed template).
- The `DataAsset` association is a single, any-type asset association. This association references the asset that provides the main content of the gadget. For example, the `DataAsset` association for the List Gadget references a Recommendation asset containing a static list of Content assets to be rendered by the gadget.



CS-Based Gadget: ListGadget

Name:	ListGadget
Description:	Latest News
Select a Template:	No template specified
Status:	Edited
Start Date:	
End Date:	
Locale:	
ID:	1269873534992
Category:	
Filename:	
Path:	
Name of Descriptor Template:	G_List
Related:	**Recommendation:** DataAsset SportsArticles
Modified:	Wednesday, April 14, 2010 6:48:16 PM EDT by firstsite

The descriptor template attribute is read by the `FW_CSGadget/GenerateGadgetXML` template. The DataAsset association is read by the descriptor template itself (for example, `G_List`).

92

Gadgets: Template Flow

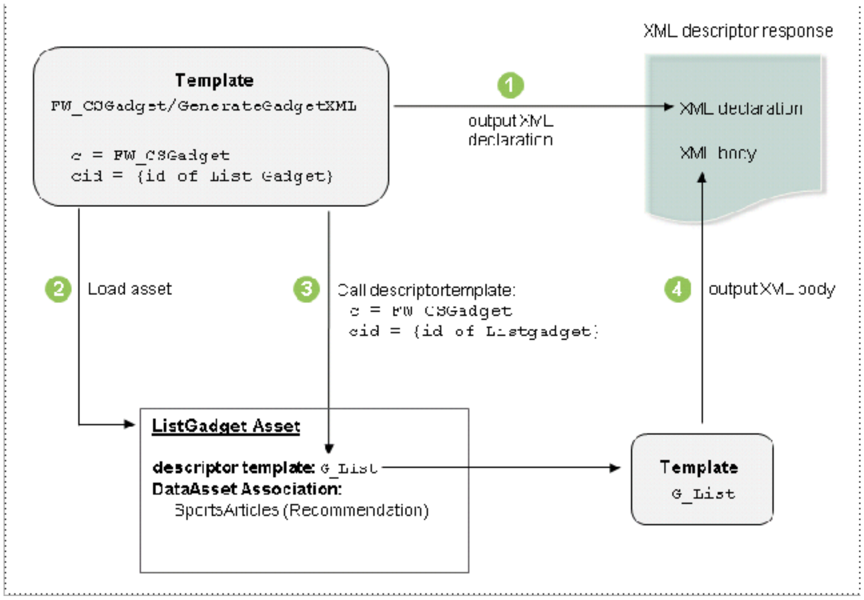
This chapter describes the flow of template execution, using the List Gadget as an example.

- Section 92.1, "Template Flow for the List Gadget"
- Section 92.2, "Differences in Template Flow"
- Section 92.3, "Why Server Calls are Done Separately"

92.1 Template Flow for the List Gadget

By design, the only way to request a gadget's descriptor XML is by invoking the `GenerateGadgetXML` template on an `FW_CSGadget` asset. When the template is invoked, the rendering process begins, as shown in Figure 92–1 and outlined below:

Figure 92–1 Gadget XML Output



1. `GenerateGadgetXML` first outputs the XML declaration, a recommended feature for all XML documents. Outputting the XML declaration at the beginning of the process saves gadget developers from having to consider it in every gadget they write.
2. GenerateGadgetXML then loads the asset that corresponds to the passed *c* (`FW_CSGadget`) and cid (id of `ListGadget`) and looks up its descriptortemplate value.

Note: Together, the *c* and *cid* point to a specific asset in the WebCenter Sites system. Parameter c is the type of asset and *cid* is the identifier of the asset. This information is required whenever a template is invoked.

---

3. `GenerateGadgetXML` invokes the template in the descriptortemplate field (`FW_CSTemplate/G_List` in the case of the List Gadget), passing it the same `c` and `cid` as in step 2.

   Notice that the `G_List` template is in itself not externally callable (its SiteCatalog entry's `pageletonly` field is set to `T`). This setting prevents these templates from being called directly (for example, in cases where they are not designed to be called).

4. The called template, `G_List` , outputs the entire body of the gadget descriptor XML. At this point, the gadget descriptor XML is complete.

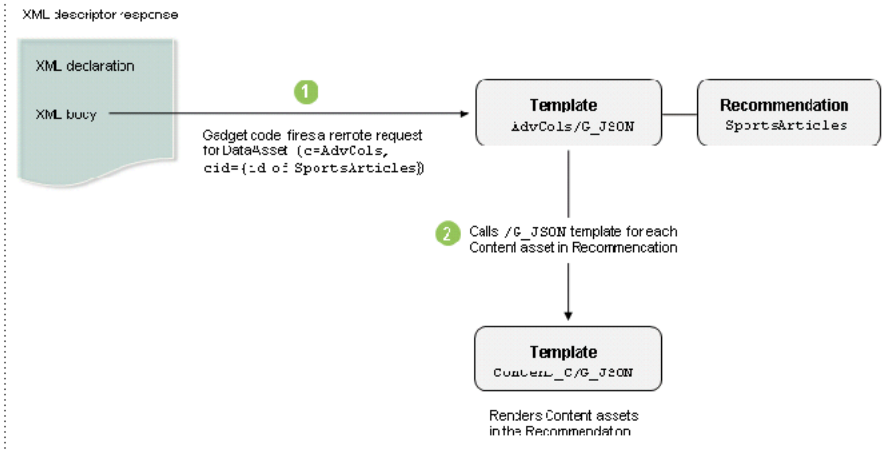
Note: See Section 91.1, "Before You Begin" for links to additional resources regarding the authoring of gadget descriptor XMLs.

Once the gadget is rendered based on its descriptor XML, the gadget retrieves its content. Figure 92–2 illustrates how the List Gadget retrieves content. This process varies, depending on the gadget. The List Gadget retrieves articles from WebCenter Sites via the following JavaScript output by the *G_List* template (this line of code calls a Google Gadgets API function):

```
gadgets.io.makeRequest(url, handleJson, params);
```

The `G_List` template crafts an additional WebCenter Sites URL, pointing to this gadget's `DataAsset` (in this case a Recommendation), via the `G_JSON` template for that asset type. That URL is eventually fed to the `gadgets.io.makeRequest` function, which can be used to make additional remote server calls expecting various forms of data such as XML, JSON, or even Atom/RSS. This request initiates the process of retrieving content.

Figure 92–2 Retrieving Content


XML descriptor response
XML declaration
XML body
1
Gadget code fires a remote request for DataAsset (c=AdvCols, cid={id of SportsArticles})
Template
AdvCols/G_JSON
Recommendation
SportsArticles
2
Calls /G_JSON template for each Content asset in Recommendation
Template
Content_C/G_JSON
Renders Content assets in the Recommendation

1. Gadget code in the XML body invokes the `AdvCols/G_JSON` template and prepares the beginning of a JSON response. Since Recommendations contain lists of other

assets, the template simply begins the output of a JSON array, then loops through the list of assets, expecting to call a respective G_JSON template on each.

2. In the case of the List Gadget, all of the children are Content assets, so Content_C/G_JSON is invoked for each asset in the Recommendation. For each invocation, the Content_C/G_JSON template outputs a complete JSON object representation containing relevant fields of the asset.
3. In between individual Content_C/G_JSON calls, AdvCols/G_JSON outputs a comma to properly delimit objects in the array it is constructing. After the loop is completed, the AdvCols/G_JSON template closes the array.
4. This array is received by the gadget code: (gadgets.io.makeRequest(url, handleJson, params);), which then executes the handleJson callback function as prescribed by the original makeRequest call. The handleJson function wraps the content in HTML, which renders that gadget's view (a list view in this example).

92.2 Differences in Template Flow

The previous section illustrated template flow for the List Gadget. While template flows for the other sample gadgets are very similar, differences in their WebCenter Sites template logic are worth noting.

92.2.1 ThumbList and Slideshow

Gadget rendering follows a path in WebCenter Sites that is very similar to the path of the List Gadget. The main difference is what is invoked by the AdvCols/G_JSON template. While the List Gadget references a Recommendation filled with Content assets, the ThumbList and Slideshow gadgets reference a Recommendation containing Product assets. These gadgets also inspect attributes specific to the response of the Product_C/G_JSON template (which includes data produced by the Media_C/G_JSON template).

92.2.2 RSS

The RSS Gadget's second request is fired straight to the URL of the RSS feed, which may or may not be on WebCenter Sites. The URL is read from the associated FW_RSS asset, which is loaded within the G_RSS template.

92.3 Why Server Calls are Done Separately

You may be asking why an additional server call is always involved. For instance, why not simply retrieve all of the articles for the List Gadget from the G_List template itself? The answer to this is twofold:

1. Gadget descriptor XML may be expected to be cacheable by the gadget container. This means that developers should avoid embedding volatile data in the XML.
2. An additional server call helps to separate presentation logic (for example, in G_List) from the underlying model (in the asset types' G_JSON templates).

93

Gadgets: Creating Your Own Gadgets

This chapter contains the following sections:

- Section 93.1, "Creating Gadgets on Different CM Sites"
- Section 93.2, "Custom Gadgets"
- Section 93.3, "Prerequisites for Registering Gadgets"

93.1 Creating Gadgets on Different CM Sites

The sample gadgets must be enabled on FirstSiteII. Developers will be interested in creating gadgets on different CM sites. The following steps outline the steps for enabling gadget support on additional CM sites:

1. Enable the `FW_CSGadget` asset type and its start menu items on the CM site. (By default, the start menu items are accessible to users with the Designer, Site Admin, or General Admin role.)
2. Share the following Template assets to the site:

   `FW_CSGadget/GenerateGadgetXML` and `FW_CSGadget/ListSiteGadgets`

(If you are reusing the sample gadgets, share their templates from FirstSiteII, enable the relevant asset types, and if necessary share the assets. For example, if you are reusing the RSS Feed Gadget, share the `FW_CSGadget/G_RSS` Template asset from FirstSiteII and enable the `FW_RSS` asset type.)

93.2 Custom Gadgets

In addition to having a knowledge of sample gadget architecture, it is helpful to see what other gadgets of type `FW_CSGadget` can be created. This section outlines the requirements for creating different types of gadgets. This section also provides information about the parameters defined in the descriptor XML of the RSS feed sample gadget.

This section contains the following topics:

- Section 93.2.1, "New Gadget, WebCenter Sites Generates Only XML"
- Section 93.2.2, "New Gadget, WebCenter Sites Generates XML and Fields Additional Requests"
- Section 93.2.3, "Same Gadget Logic, Different Content"
- Section 93.2.4, "Analyzing the Sample RSSFeed Gadget"

93.2.1 New Gadget, WebCenter Sites Generates Only XML

You may want to create a new gadget where the only additional content is from another website. The RSS Feed Gadget is an example of a gadget that functions in this way. See Section 91.3.4, "RSS Feed Gadget."

Requirements in this scenario are:

- A gadget of type `FW_CSGadget` referencing the template and asset listed below.
- `FW_CSGadget` template that generates the body of the gadget descriptor XML (for example, `FW_CSGadget /G_RSS`).
- An asset that specifies the URL to be requested in the gadget.

93.2.2 New Gadget, WebCenter Sites Generates XML and Fields Additional Requests

In this scenario, you will populate a gadget with content directly from one of WebCenter Sites' CM sites. The List, ThumbList, and Slideshow gadgets are examples of gadgets that function in this way. See Figure 92–1 and Figure 92–2.

Requirements in this scenario are:

- A gadget of type `FW_CSGadget` referencing the template and asset listed below.
- `FW_CSGadget` template (such as `G_List`) for rendering the body of the gadget descriptor XML.
- Template(s) for rendering data to be returned in response to the additional requests made by the gadget code (for example, the `G_JSON` templates included with the sample gadgets).
- Assets to be referenced by the gadget (or its additional requests).

93.2.3 Same Gadget Logic, Different Content

You can create gadgets that have the same functionality but pull different sets of content. Requirements in this scenario are:

- An `FW_CSGadget` asset that references a pre-existing `FW_CSGadget` template.
- Assets that provide the gadget with content.

93.2.4 Analyzing the Sample RSSFeed Gadget

A gadget's descriptor XML contains all the data defined for the gadget within the `Module` tag. This tag holds information about the gadget's dependencies, user preferences (if any), appearance settings, functionality, and so on.

This section analyzes the descriptor XML files of `FW_CSGadget` assets, using code snippets from the sample RSSFeed gadget as an example.

Analyzing the code of the RSSFeed gadget's descriptor XML file

These lines specify the properties and dependencies of the gadget:

```
<ModulePrefs title="FatWire RSS" height="350">
screenshot="http://localhost:8100/cs/FirstSiteII/gadgets/RSS/screenshot.png"
thumbnail="http://localhost:8100/cs/FirstSiteII/gadgets/RSS/thumbnail.png">
<Require feature="dynamic-height"/>
</ModulePrefs>
```

The following lines define the gadget's user preferences, which are values for the gadget that visitors can modify. In this code snippet, visitors will be able to select the number of news feeds that the RSSFeed gadget displays at one time:

```
<UserPref name="max" display_name="Number of Items"
datatype="enum" default_value="5">
   <EnumValue value="1" display_value="1"/>
   <EnumValue value="3" display_value="3"/>
   <EnumValue value="5" display_value="5"/>
   <EnumValue value="10" display_value="10"/>
</UserPref>
```

The `Content` tag contains all of the content for the RSSFeed gadget's descriptor XML file, including the gadget's CSS file, which defines the gadget's appearance, and the gadget's JavaScript (contained within the `<script>` tag), which specifies the interactive components and initialization functionality of the gadget.

For example, the CSS file defined within the RSSFeed gadget's `Content` tag looks as follows:

```
<style type="text/css">
/* === generic styles === */
body #container {
   padding: 15px;
   padding-bottom: 5px; /* 5 plus 10 from bottom-most entry */
   margin: 0;
}
body #container * {
   padding: 0;
   margin: 0;
   font-size: 11px;
   font-family: Tahoma,Arial,Helvetica,sans-serif;
   color: #666;
}
body #container a, body #container * a {
   text-decoration: none;
   color: #555;
}
body #container a:hover, body #container * a:hover {
   color: #3b9cce;
}
.error {
  background-color: #fcc;
  color: #c00;
}
/* === gadget-specific styles === */
#container .list {
   padding-left: 1em; /* provide proper indentation space for bullets */
}
#container .list li {
   padding-bottom: 10px;
}
#container .headline {
   font-weight: bold;
}
</style>
```

93.3 Prerequisites for Registering Gadgets

For gadgets to be recognized by the Gadgets application, they must be registered (typically by administrators of the Gadgets application). A gadget is accessed via the WebCenter Sites URL that generates the gadget descriptor XML. This URL can be quickly obtained for every `FW_CSGadget` asset on the content management site by querying the `FW_CSGadget/ListSiteGadgets` template. The template can be easily accessed in one of the following ways:

- In the WebCenter Sites Advanced interface, preview any `FW_CSGadget` asset on the site. In the InSite window, select `ListSiteGadgets` from the Template drop-down list.

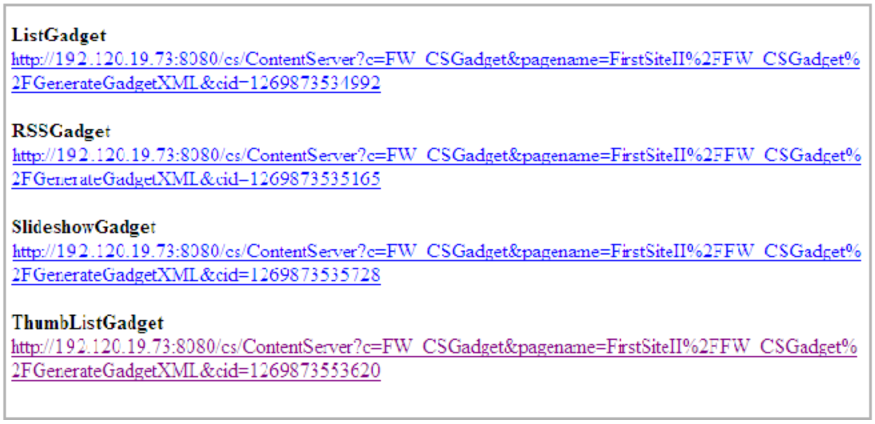
Or:

- Open a browser and navigate directly to

  ```
   http://<host>:<port>/<application context>/wem/<sitename>/FW_
  CSGadget/ListSiteGadgets
  ```

  (where `host`, `port`, and `application context` correspond to the WebCenter Sites installation, and `sitename` is the name of the content management site where the gadget exists).

The list of sample gadgets and their corresponding URLs is shown in the following figure.



ListGadget
http://192.120.19.73:8080/cs/ContentServer?c=FW_CSGadget&pagename=FirstSiteII%2FFW_CSGadget%2FGenerateGadgetXML&cid=1269873534992

RSSGadget
http://192.120.19.73:8080/cs/ContentServer?c=FW_CSGadget&pagename=FirstSiteII%2FFW_CSGadget%2FGenerateGadgetXML&cid=1269873535165

SlideshowGadget
http://192.120.19.73:8080/cs/ContentServer?c=FW_CSGadget&pagename=FirstSiteII%2FFW_CSGadget%2FGenerateGadgetXML&cid=1269873535728

ThumbListGadget
http://192.120.19.73:8080/cs/ContentServer?c=FW_CSGadget&pagename=FirstSiteII%2FFW_CSGadget%2FGenerateGadgetXML&cid=1269873553620

Part IX

Developing a Java Connector and Plugin for CIP

This part shows developers how to extend Oracle WebCenter Sites: Content Integration Platform to publish from systems of their own choice to Oracle WebCenter Sites.

This part contains the following chapters:

94

Integrating with Custom Source Systems

This chapter outlines methods for extending Oracle WebCenter Sites: Content Integration Platform to support content delivery from custom source systems to Oracle WebCenter Sites.

This chapter contains the following sections:

- Section 94.1, "Customizing WebCenter Sites: Content Integration Platform"
- Section 94.2, "Content Integration Agent"

94.1 Customizing WebCenter Sites: Content Integration Platform

Content Integration Platform (CIP), by default, delivers content from file systems, Microsoft SharePoint, and EMC Documentum to Oracle WebCenter Sites. Developers can extend Content Integration Platform to publish from a system of their own choice, such as a database or custom content management system, by writing a Java-based implementation consisting of a source adapter and plug-in(s), or just the adapter. Both the adapter and the plug-ins are supported by the Content Integration Agent component. See Section 94, "Integrating with Custom Source Systems."

A Java source adapter must be written for each source system whose content will be delivered to Oracle WebCenter Sites. The adapter queries the source system to retrieve its metadata and binary content. The adapter must be registered with Content Integration Agent by means of a statement in the `catalog.xml` file.

A plug-in is required only if items retrieved by the adapter must be processed before they are published to WebCenter Sites. Processing could include for example, extracting thumbnails from image files or performing a validation step while publishing. Typically, plug-ins are written to support different file formats or to filter selected items from the publishing process. Any number of plug-ins can be used with *any* adapter. Like the adapter, a plug-in must be registered with the Content Integration Agent (in the `types.xml` file).

94.2 Content Integration Agent

Content Integration Agent is written in C++ and provides the following components to support Java-based custom connectors and plug-ins:

- Solid runtime system.
- Pluggable interfaces, used to implement Java-based source connectors and plug-ins.
- XML files named `catalog.xml` and `types.xml`, both used to register the custom source adapter and plug-ins.

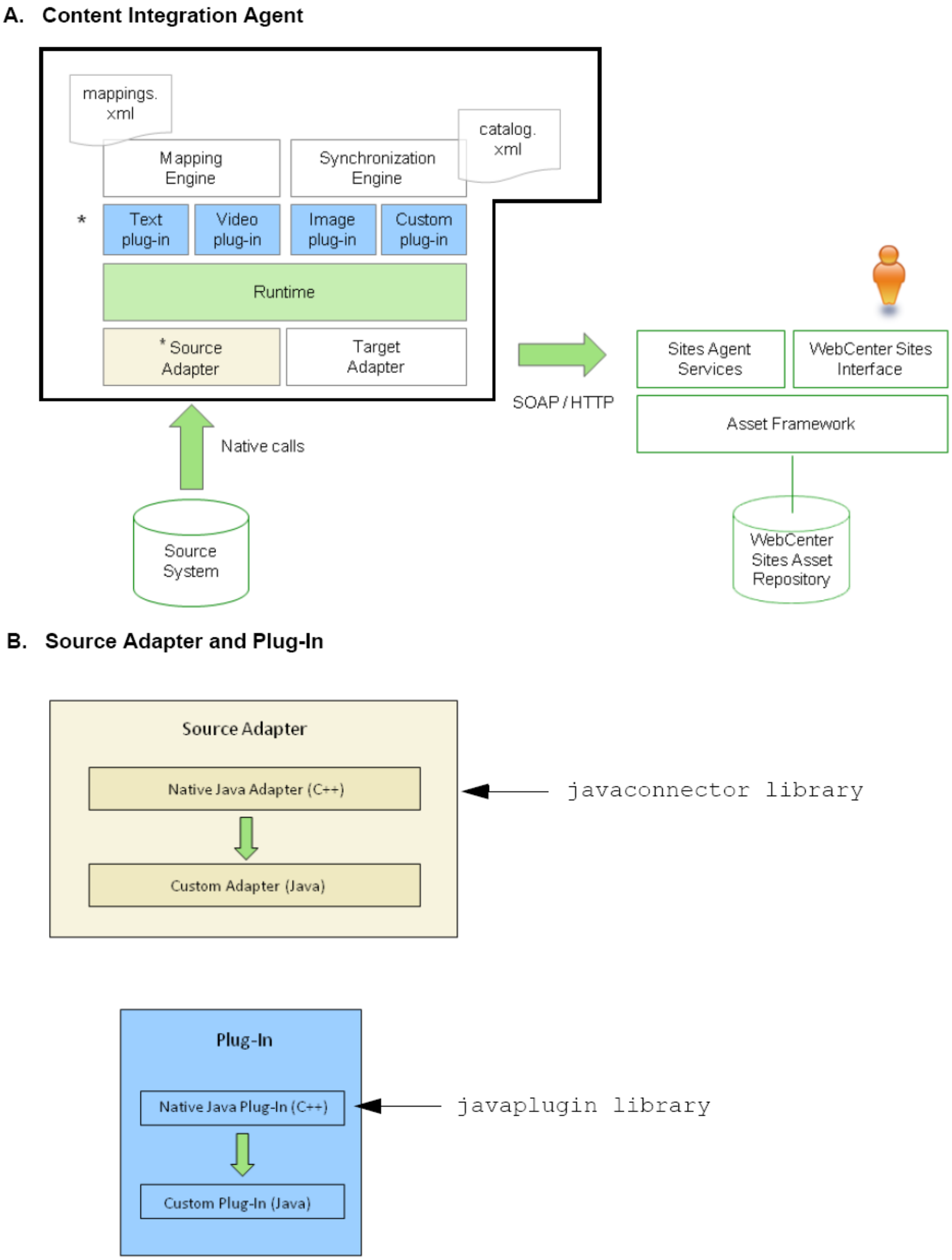
- Native source adapter (`javaconnector` library) and native plug-in (`javaplugin` library). Both are written in C++. They are used to make calls to Java code.
- Facilities, which are construction blocks providing some set of functionality to the Agent runtime. Content Integration Agent hosts the Java Virtual Machine in its process space in order to delegate calls from the C++ runtime environment to Java code. The JVM is enabled by registering `javafacility` in `facilities.xml`.

Once the Java-based adapter is created and the JVM is enabled, the C++ Agent runtime system can use the JVM to call Java code via the native adapter (similar process for plug-ins).

Procedures for creating Java-based connectors and plug-ins are given in Chapter 95, "Creating Adapters and Plug-Ins" along with instructions for completing the integration.

- For more information about Content Integration Agent, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.
- For more information on the Content Integration Platform for File Systems and Microsoft SharePoint, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.
- For more information on the Content Integration Platform for EMC Documentum, see the *Oracle Fusion Middleware WebCenter Sites Administrator's Guide*.

Figure 94–1 Content Integration Platform



A. Content Integration Agent

B. Source Adapter and Plug-In

95

Creating Adapters and Plug-Ins

This chapter provides instructions for creating a complete integration solution to support content delivery from any source system to WebCenter Sites.

This chapter contains the following sections:

95.1 Overview of Creating Adapters and Plug-Ins

Creating an adapter and plug-in involves the following steps:

1. Implementing the pluggable interfaces that are provided within Content Integration Agent.
2. Registering the implementation(s) with the Content Integration Agent runtime system.
3. Registering `javafacility` in order to enable the Java Virtual Machine to delegate calls from the C++ Agent runtime to Java code.

> **Note:** A custom plug-in can be used with *any* adapter. You can implement and deploy as many plug-ins as necessary.

Before a custom adapter (or plug-in) can be successfully used, the data model for the publishable objects must exist on the WebCenter Sites system and be mapped to the WebCenter Sites system. The following steps are required:

1. Reproduce the objects' metadata in WebCenter Sites by creating a dedicated flex family (or re-using an existing flex family) to store the object types, their attributes, and the objects themselves.
2. Map object types and attributes to their respective flex family asset instances (created in the previous step). The map can be created directly in the adapter implementation, or in the `mappings.xml` file.

95.2 Creating a Java Source Adapter

Publishing from an unsupported source system to WebCenter Sites requires you to create a Java-based source adapter. A plug-in is not required unless objects retrieved by the adapter must be processed before they are published.

> **Note:** If you are using a relational database, implement custom views or custom queries in order for the adapter to work.

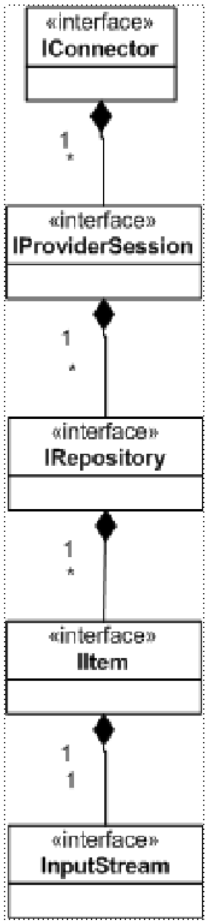
To create a Java source adapter

1. Implement the adapter:

   Implement the `IConnector`, `IProviderSession`, `IRepository`, and `IItem` interfaces. You can optionally implement the `InputStream` interface if items on your source system have primary binary content.

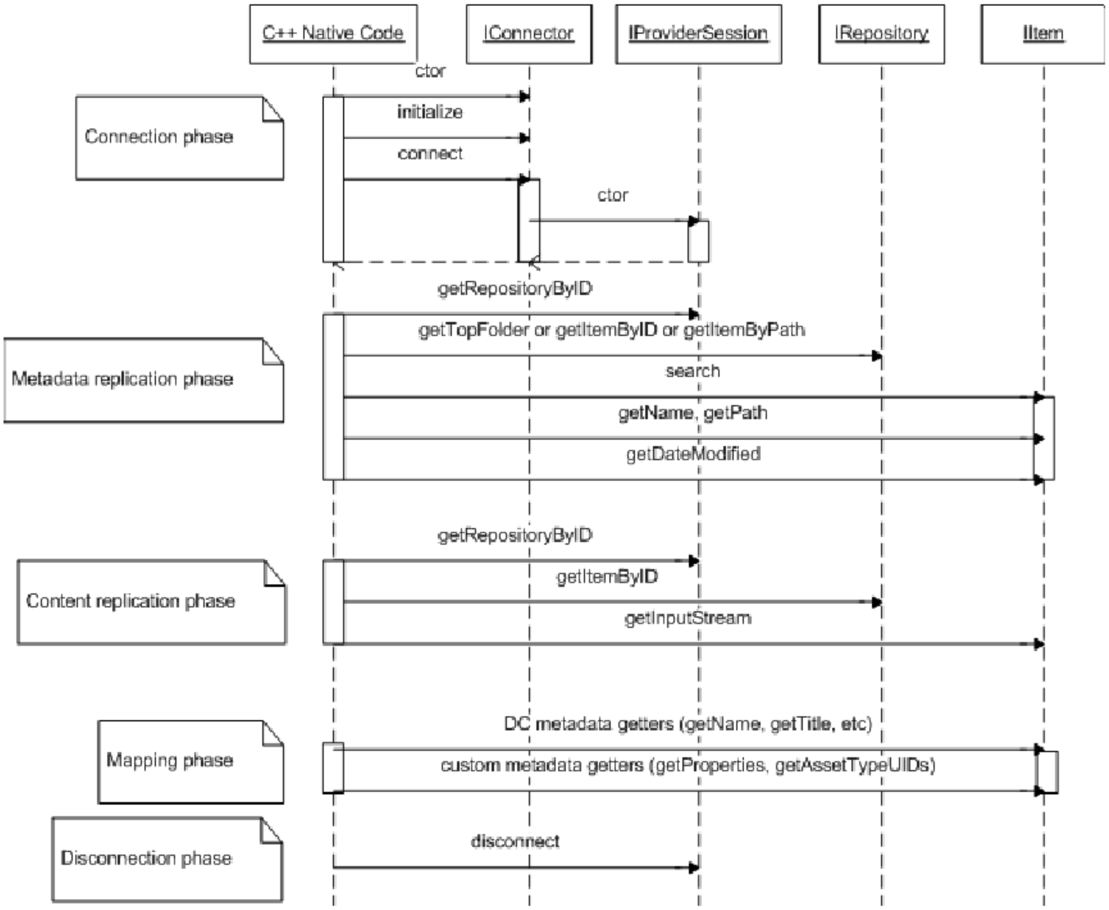
   Figure 95–1 shows the relationships among the interfaces. The entry point for the adapter's code is a factory class: the `IConnector` interface implementation.

Figure 95–1 Adapter and Plug-in Class Diagram



There are different phases in an adapter's lifetime. Depending on the phase, different methods are invoked. Figure 95–2 shows the sequence of calls during each phase.

Figure 95–2 Source Adapter Calls Sequence



Note: "DC metadata" is "Dublin Core metadata" (`http://en.wikipedia.org/wiki/Dublin Core`)

Analyzing Figure 3: Source adapter calls sequence

The ID, which is passed to the `getRepositoryByID` function, is taken from one of the corresponding workspace elements in the `catalog.xml` file.

Depending on what you pass to the `cipcommander`, one of the following functions is invoked:

- If `-source_itemid` is passed, then `getItemByID` is invoked passing the `itemid`.
- If `-source_itemid` is omitted, and `-source_path` is specified, then the `getItemByPath` function is invoked.
- If neither `-source_itemid` or `-source_path` is specified, then the `getTopFolder` function is invoked. (In this case, the entire repository is published.)

To ensure uniqueness, maintain a different `versionid`, `itemid`, and `path` for all items inside the same repository, and keep the names different for all items inside the same folder. The `path` must be in the form: `<parent path>/<this item name>`.

2. Register the adapter:

a. Register the `IConnector` interface implementation with Content Integration Agent by adding a '`connector`' element to `catalog.xml` (located in `integration_agent/conf/`):

```
<connector id="connector_id" name="connector_descriptive_name">
      <library>javaconnector</library>
        <init-params>
         <param name="className">connector_class_name</param>
           connector-specific_parameters
        </init-params>
</connector>
```

Parameter	Description
`connector_id`	Any unique identifier.
`connector_descriptive_name`	Any descriptive name (does not have to be unique).
`connector_class_name`	Name of the `IConnector` implementation created.
`connector-specific_parameters`	Set of parameters that will be passed to `IConnector.initialize` during the call.

b. Enable publishing by adding a new '`provider`' element to `catalog.xml`:

```
<provider id="provider_id" name="provider_descriptive_name">
      <connector-ref refid="connector_id"/>
        <init-params/>
          provider-specific_parameters
        </init-params>
</ provider >
```

Parameter	Description
`provider_id`	Any unique identifier.
`provider_descriptive_name`	Any descriptive name (doesn't have to be unique).
`connector_id`	Adapter's unique identifier.
`provider-specific_parameters`	Set of parameters that will be passed to `IConnector.login` during the call.

c. Deploy the adapter:

Place the adapter's `jar` files into the folder `<resource>/java/<connector_id>/lib`, and the `class` files into `<resource>/java/<connector_id>/classes`.

The `<resource>` folder is located within Content Integration Agent.

On Windows: `<resource>` is `<%INSTALLDIR%>`

On Unix: `<resource>` is `<$INSTALLDIR/shared/cipagent>`

Note: Adapter classes are loaded by different class loaders to prevent collisions with different implementations and loading/unloading features. We strongly advise placing all adapter `jar` and `class` files into the `<connector_id>` folder, instead of including them into the `CLASSPATH` environment variable, or the `java.class.path` property, or the `jre/lib/ext` folder.

3. If you require a Java plug-in (to process items retrieved by the adapter), continue to section Section 95.3, "Creating a Java Plug-In." Otherwise, enable `javafacility` (to allow the Java Virtual Machine to delegate calls to Java code from the C++ Agent runtime). For instructions, see Section 95.4, "Enabling javafacility."

95.3 Creating a Java Plug-In

Note: A custom plug-in can be used with *any* adapter. You can create and deploy as many plug-ins as necessary.

A plug-in is not required unless objects retrieved by the adapter must be processed before they are published to the WebCenter Sites system. The main purpose of a plug-in is to modify the metadata of retrieved items, add metadata to retrieved items, and reject items.

Creating a plug-in is similar to creating a adapter. The steps are as follows:

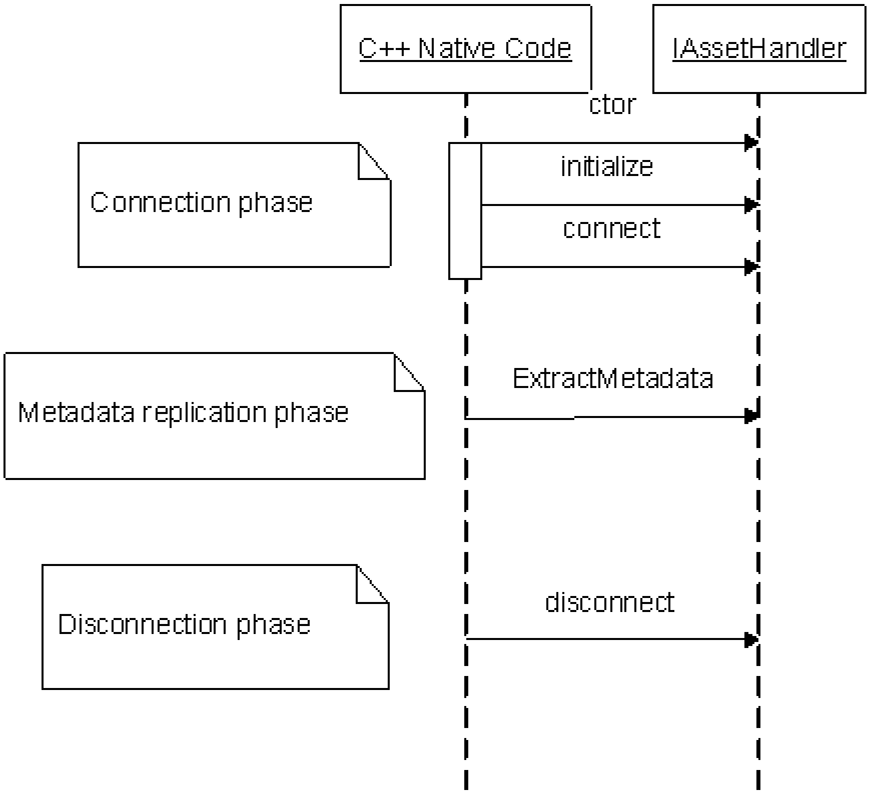
To create a Java plug-in

1. Implement the plug-in by implementing the `IAssetHandler` interface (in Content Integration Agent).

   The entry point for a plug-in is the `IAssetHandler` interface. This is the only interface which is directly used by the runtime system.

   In most cases `ExtractMetadata` is the only method you need to implement. Figure 95–3 shows the calls sequence in a plug-in's lifetime.

Figure 95–3 Plug-in Calls Sequence



2. Register the plug-in with Content Integration Agent.

   a. Add a new plug-in 'handler' element to the `types.xml` file (located in the `integration_agent/conf/` folder):

```
<handler id="handler_id">
      <library>javaplugin</library>
        <init-params/>
          plugin-specific_parameters
        </init-params>
</handler>
```

Parameter	Description
handler_id	Custom plug-in's unique identifier.
plugin-specific parameters	Plugin-specific parameters that are passed when the plug-in is initialized.

   b. If you are using Content Interation Platform for EMC Documentum, complete this step. Otherwise, skip to step c.

   Enable the handler for the selected handler sets. Which handler set to use is specified during the publication initiation process. Each handler set contains the list of handlers, which are invoked during the metadata extraction phase in the Content Integration Agent. Handlers are matched by either MIME type or asset type.

   MIME type has the following form: `<major type>/<minor type>`

(image/jpeg, for example). There is an option to use '*' for MIME types. It can be applied either to the minor part or the whole MIME type. For example, */* matches all assets, while text/* matches only text files.

When using the IConnectorContext.guessMIMEType function, it will look into mimetypes.xml to get the corresponding MIME type for the supplied file extension. For example the call with "txt" parameter will produce the "text/plain" result.

Asset types also support the '*' notation, which matches all asset types.

If more then one handler matches a specific item, then both are invoked in the same sequence in which they are registered within the handler set used. If any of the matching handlers returns the null object from the IItem.extracMetadata call, then the item is discarded from future processing and not sent to the target adapter.

c. Enable the custom plug-in for selected object types by adding "asset-type" elements to the types.xml file. Items for which this plug-in is invoked will be filtered according to MIME type.

Note: The asset-type element in the context of a plug-in is a MIME type group.

```
<asset-type type="MIME_type">
      <extensions>
        <ext>ext</ext>
          . . .
      </extensions>
   <handler-ref refid="handler_id" />
</asset-type>
```

Parameter	**Description**
MIME type	MIME type of the item for which this plug-in will be invoked. *MIMEtype* must be of the form <major_type/minor_type>, e.g., text/plain. A wild-card symbol (*) can also be used. For example: ■ To enable the plug-in for all text files, specify: text/* ■ To enable the plug-in for all items, specify: */*
ext	File extension, e.g., .txt for text files. The file extension does not directly affect the plug-in selection process. However, it is used to "guess" the MIME type for those systems where MIME type is not directly available (e.g., file system).
handler_id	Custom plug-in's unique identifier (specified in the handler element, in the previous step).

d. Deploy the plug-in:

Place the plug-in's jar files into the folder <resource>/java/<plugin_id>/lib, and the class files into <resource>/java/<plugin_id>/classes.

The <resource> folder is located within Content Integration Agent.

On Windows: `<resource>` is `<%INSTALLDIR%>`

On Unix: `<resource>` is `<$INSTALLDIR/shared/cipagent>`

Note: Plug-in classes are loaded by different class loaders to prevent collisions with different implementations and loading/unloading features. We strongly advise placing all plug-in `jar` and `class` files into the `<plugin_id>` folder, instead of including them into the `CLASSPATH` environment variable, or the `java.class.path` property, or the `jre/lib/ext` folder.

---

3. If you created a custom adapter but have not enabled `javafacility`, continue to Section 95.4, "Enabling javafacility."

95.4 Enabling javafacility

Calling Java code from C++ Agent runtime requires a special facility named `java` to be registered in `facilities.xml`.

To enable javafacility

1. Verify that `facilities.xml` is not commented (`facilities.xml` is located in the `integration_agent/conf/` folder).
2. Add the following lines:

```
<facility name="javafacility">
     <library>java</library>
       <init-params>
         <param name="VMArgparam_id">Java_VM_argument
</param>
         <param name="VMLibraryPath">VM_library_path</param>
       </init-params>
</facility>
```

Parameter	Description
`param_id`	Parameter's unique id (any unique value). In order to pass multiple arguments to the JVM, construct multiple parameters with different *param_id*'s.

`Java_VM_argument`	Java VM argument to be passed to the Java VM created within the Agent runtime process.
	Example:
	`<param name="VMArg0">-Xmx256m`
	`</param>`

`VM_library_path`	Full path to the Java VM library (DLL or shared library) within the JRE/JDK installation.
	For example, for Sun JDK on Windows, `VM_library_path` is either:
	`%JAVA_HOME%\jre\bin\server\jvm.dll`
	or:
	`%JAVA_HOME%\jre\bin\client\jvm.dll`

95.5 Troubleshooting and Debugging

When developing custom components for CIP, it is often helpful to see more than just the default logging messages displayed in the production environment. CIP Agent supports five different logging levels:

- `fatal`
- `error`
- `warning`
- `info`
- `debug`

Use the instructions below to debug custom components in CIP.

> **Note:** Do not use the settings shown below on a production system, as they can slow down the system's performance.

- **Escalating the logging level in CIP Agent**

  CIP is set to `error` by default. To increase the logging level, CIP Agent must run as a console executable:

  1. Stop the CIP Agent system service.
  2. Run the **cipagent -t debug** command.

- **Debugging Java custom components**

  To debug custom Java implementations hosted within the Agent runtime, enable remote debugging in CIP Agent. For example, to start the remote debugger on port 7007 and suspend CIP Agent to wait until a debugger attaches, add the following lines to `javafacility`:

  ```
  <param name="VMArg1">-Xdebug</param>
  <param name="VMArg2">-Xrunjdwp:transport=dt_socket,
  address=7007,server=y,suspend=y</param>
  ```

- **Escalating the logging level for Sites Agent Services**

  To get more data about an error in the Sites Agent Services application, set the `DEBUG` level in the `commons-logging.properties` file for the `com.fatwire.logging.csagentservices` category. We also recommend setting the `DEBUG` logging level in the `commons-logging.properties` file for the `com.fatwire.logging.cs.db` category.