Oracle® Fusion Middleware

Mobile Browser Developer's Guide for Oracle Application Development Framework

11*g* Release 2 (11.1.2.3.0)

E16178-04

August 2012

Documentation for Oracle Application Development Framework (Oracle ADF) developers that describes how to use Oracle JDeveloper to create mobile browser-based applications comprised of Apache MyFaces Trinidad web-client components.

ORACLE

Oracle Fusion Middleware Mobile Browser Developer's Guide for Oracle Application Development Framework 11*g* Release 2 (11.1.2.3.0)

E16178-04

Primary Author: John Bassett

Contributing Author: Mamallan Uthaman

Contents

6 Design Guidelines for BlackBerry 4.2 to 4.5

7 Narrow Screen Support and User Agent Details Support

8 Extending ADF Mobile Browser Applications

Preface

Welcome to *Mobile Browser Developer's Guide for Oracle Application Development Framework*.

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.

Audience

This document is intended for developers of browser applications for mobile devices.

Related Documents

For more information, see the following:

- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Skin Editor User's Guide for Oracle Application Development Framework*

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.

Convention	Meaning
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.

1

Overview of Oracle ADF Mobile Browser

This chapter provides an overview of Oracle Application Development Framework Mobile (ADF Mobile) browser.

This chapter includes the following sections:

1.1 About ADF Mobile Browser

Oracle Application Development Framework Mobile (ADF Mobile) browser is a standards-based framework that enables the rapid development of enterprise mobile applications. Oracle Fusion Middleware 11*g* release 2 of ADF Mobile browser extends Oracle ADF to browsers running on mobile devices. Because ADF Mobile browser is built upon the component model of Java Server Faces (JSF), you can quickly develop applications for mobile browsers. ADF Mobile browser's mobile-specific extensions to JSF enable you to develop mobile applications using the same methodologies for developing JSF applications for the desktop.

When developing an ADF Mobile browser application, you need not focus on the limitations or capabilities of different browsers, as ADF Mobile browser enables you to develop applications that function properly on different browser types. The ADF Mobile browser renderer ensures that contents can be consumed correctly by the target browser. It handles the variations in browser implementations of HTML, JavaScript, CSS, DOM, `XMLHttpRequest` and in system performance as well. For example, if a browser does not support `XMLHttpRequest` and is incapable of posting a partial page request to a server, ADF Mobile browser's support for AJAX (Asynchronous JavaScript and XML) enables the application to revert automatically to a full-page submit, thus allowing the page to function.

Note: For Oracle Fusion Middleware 11*g* release 2, ADF Mobile browser requires HTML and JavaScript support.

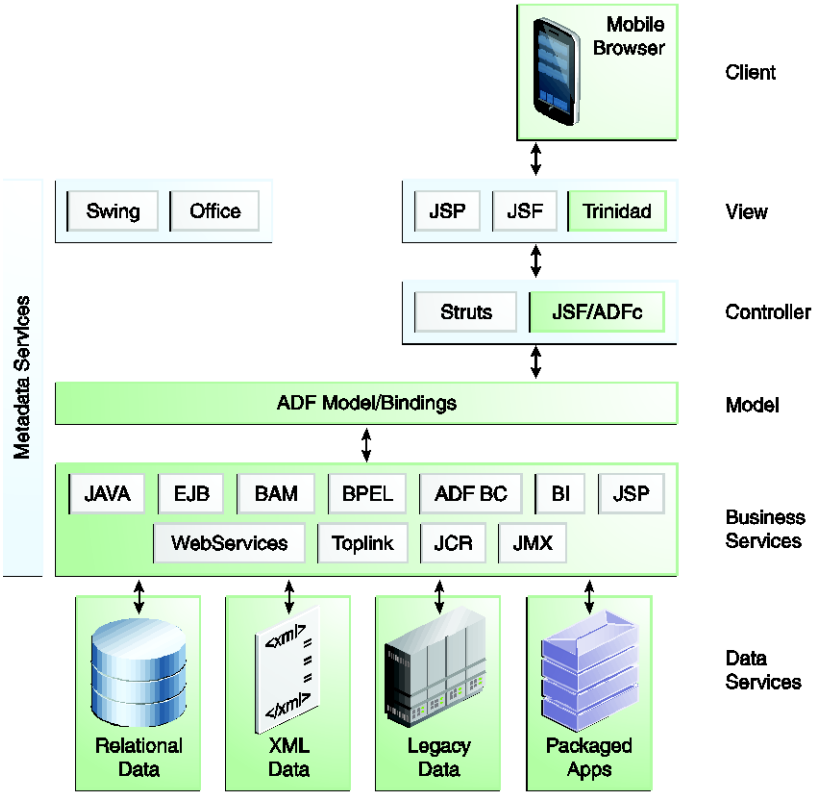
1.1.1 About Java Server Faces and the Application Development Framework

Java Server Faces (JSF) is a standard specified by JSR-127 that enables developers to create applications using pre-built components that define functionality and behavior. As illustrated in Figure 1–1, JSF provides a Model-View-Controller (MVC) mechanism that simplifies the development of web applications. Its renderkit converts components both to and from a specific markup language. Renderers abstract the

production of markup and responses to browser requests by generating the markup representations of the components and how they interpret browser requests.

JSF development focuses on components, not markup. Using JSF, you create a JSP page containing JSF component tags. When a user visits this page (through the FacesServlet), JSF uses the renderkit specified by the user's device to encode the markup used for the appropriate output. For example, if the user's device specifies HTML for a desktop browser, then the renderkit's markup encoding results in an HTML page. In addition to rendering appropriate content, JSF supports user interaction.

Figure 1–1 ADF Mobile Browser Architecture



Application Development Framework (ADF) is built on the standard JSF technology and provides the following:

- A large component set (because JSF provides only basic components).
- Renderers that support these components in HTML browsers, including a rich renderkit for applications using AJAX technologies.
- Converters, validators, and events.

1.1.2 Developing Mobile Applications Using ADF Mobile Browser

You can use the same programming model and component set for developing desktop browser applications to develop mobile browser applications for mobile devices. ADF Mobile browser application development is almost identical to ADF Web application development, except that ADF Mobile browser application development uses only mobile JSF pages that consist of Apache MyFaces Trinidad components. For more

information on developing ADF Web applications, see *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*

> **Note:** You cannot use ADF Faces components to develop an ADF Mobile browser application. You must use Apache MyFaces Trinidad components.

Developing mobile browser applications for mobile devices with ADF Mobile browser leverages the same methodologies used in developing JSF applications for the desktop but with a few specific mobile extensions. With support for over 60 Apache MyFaces Trinidad components, you can build applications with the rich component set, each of which renders appropriately for small-screen mobile devices. In this way, you can reuse the desktop browser application's model and controller layers to assemble a new view layer for mobile devices by using similar Apache MyFaces Trinidad components.

How ADF Mobile Browser Improves Performance

The PDA component renderers have been optimized to improve performance by minimizing the payload of the web page sent over the wireless network to the mobile device. In mobile environments with high-latency and low-bandwidth wireless networks, Partial Page Rendering (PPR) is essential in providing end-users with an efficient application. For mobile browsers supporting AJAX, ADF Mobile browser supports PPR for certain components to minimize the amount of data requested from the server and improve the responsiveness of the applications. See also Section 3.1.2, "Partial Page Rendering."

1.2 Supported Mobile Browsers

ADF Mobile browser supports Apache MyFaces Trinidad components on the browsers listed in Table 1–1. Later versions of Trinidad can be integrated into Oracle JDeveloper and used with Oracle Fusion Middleware 11*g* release 2 of ADF Mobile browser.

Table 1–1 Supported Browsers and Supported Mobile Features

Browser	JavaScript Support	CSS Support	PPR Support
BlackBerry version 4.6 and later	Yes	Yes	Yes
Blackberry versions 4.2 through 4.5	No	Yes	No
Microsoft Windows Mobile 5	Yes	Yes	Yes (with nuances)
Microsoft Windows Mobile 6	Yes	Yes	Yes
Apple iPhone Safari	Yes	Yes	Yes
Nokia s60 series	Yes	Yes	No
Plain HTML (such as Opera Mini, Opera Mobile and Skyfire)	No	Yes	No

2

Configuring the ADF Mobile Browser Environment

This chapter describes how to configure the development environment for ADF Mobile browser applications and how to build and test mobile browser applications.

This chapter includes the following sections:

- Section 2.1, "About the ADF Mobile Browser Development Environment"
- Section 2.2, "Configuring the ADF Mobile Browser Development Environment"
- Section 2.3, "Developing an ADF Mobile Browser Application"
- Section 2.4, "Testing an ADF Mobile Browser Application"

2.1 About the ADF Mobile Browser Development Environment

ADF Mobile browser application development is almost identical to ADF web application development, except that ADF Mobile browser application development uses only mobile JSF pages that consist of Apache MyFaces Trinidad components.

> **Note:** Except for page fragments, pop-up in dialogs, and region support, you can use the ADF task flow to develop ADF Mobile browser applications. ADF Mobile browser application that use the ADF task flow only support the `trinidad-simple` skin family.

To create an ADF Mobile browser application:

- Configure the environment by creating an application and project.
- Add a web project.
- Add the JSF pages using Apache MyFaces Trinidad components.

2.2 Configuring the ADF Mobile Browser Development Environment

ADF Mobile browser application development differs only from ADF Web application development for desktop browsers in the creation of the Web project. For more information, see Section 2.3, "Developing an ADF Mobile Browser Application."

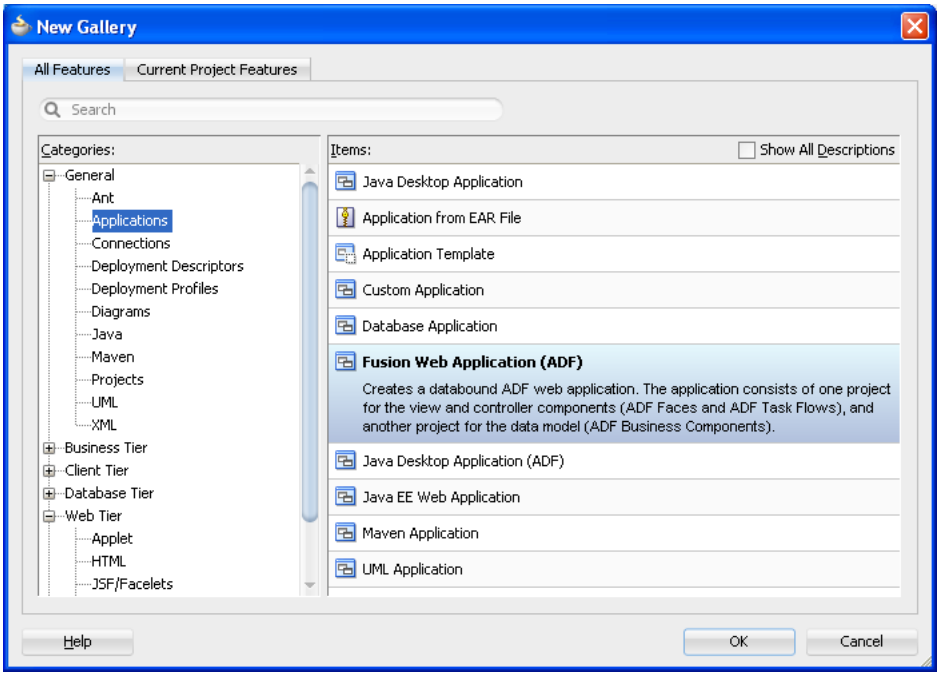
2.2.1 How to Create a Mobile Application and Project

To configure the environment, first create an ADF Mobile browser application that includes a project with the ADF Mobile browser technology.

To create the ADF Mobile browser application and the ADF Mobile browser project:

1. Choose **File** and then **New**.

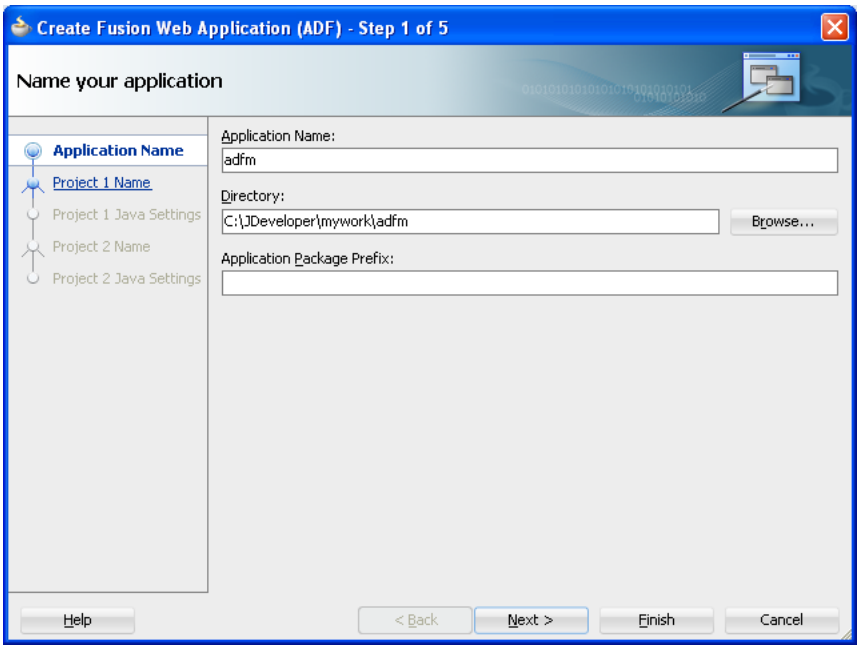
Figure 2–1 The New Gallery



2. In the New Gallery, expand **General**, select **Applications** and then **Fusion Web Application (ADF)** and then click **OK**.

3. In the Name your application page of the Create Fusion Web Application (ADF) wizard, enter a name and, if needed, a location for the application in the Directory field, as shown in Figure 2–2.

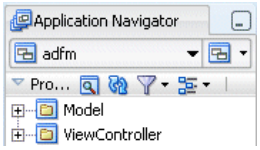
Figure 2–2 The Name your application Page



4. Click **Finish**.

   Selecting **Fusion Web Application (ADF)** creates the model project used by the mobile view project. Figure 2–3 shows the application's Model and its generated View Controller projects that appear in the Application Navigator.

Figure 2–3 The Fusion Web Application and its Projects in the Application Navigator



> **Note:** Do not use the generated View Controller project to design the business logic in the Model project. Instead, create a mobile Model View Controller project as described in the following steps.

5. Create the mobile Model View Controller as follows:

   a. Create the Choose **File** and then **New**. The New Gallery appears.

   b. In the New Gallery, expand **Categories**, select **Projects** and then select **Custom Project** and click **OK**.

   c. In the Create Custom Project wizard, complete the wizard by first entering a name for the project. For example, enter *mvc* (a short name for Model View Controller).

      Tip: To make entering a URL on a mobile device easier, enter short, lower-case names for both the application and the project.

d. Select the ADF Mobile browser feature for the project by moving **ADF Mobile Browser** from the **Available** list to the **Selected** list.

Figure 2–4 Selecting the ADF Mobile Browser Technology for an ADF Mobile Browser Project

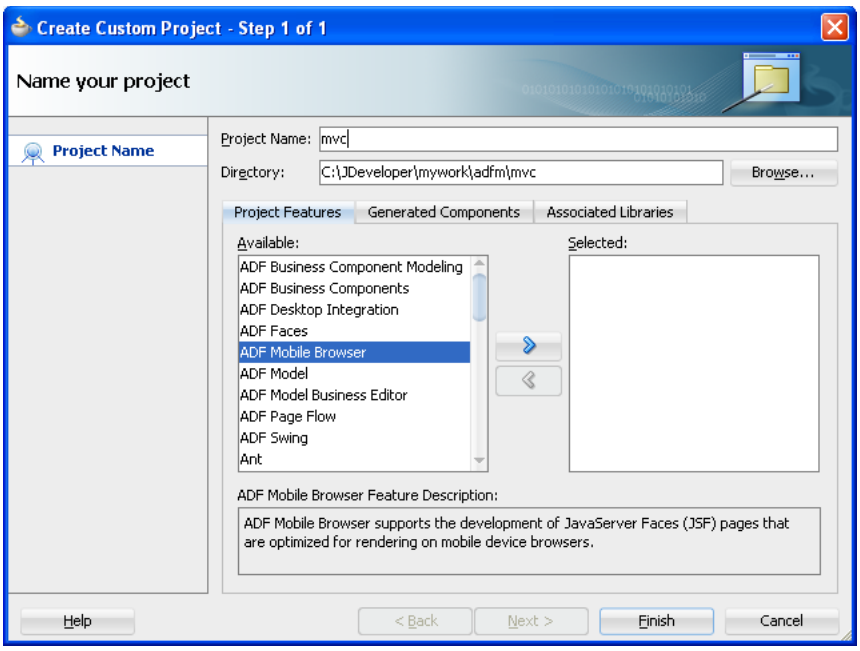
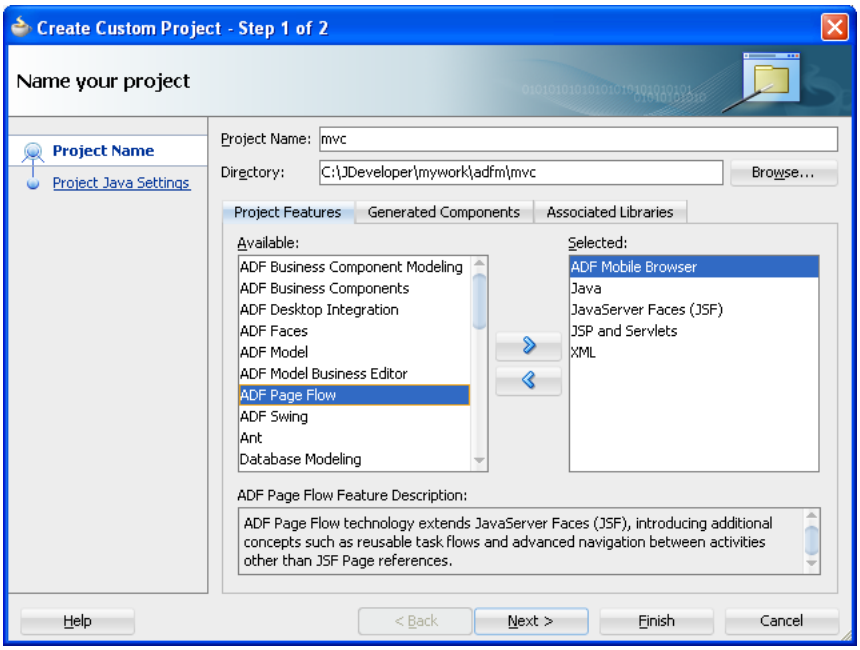


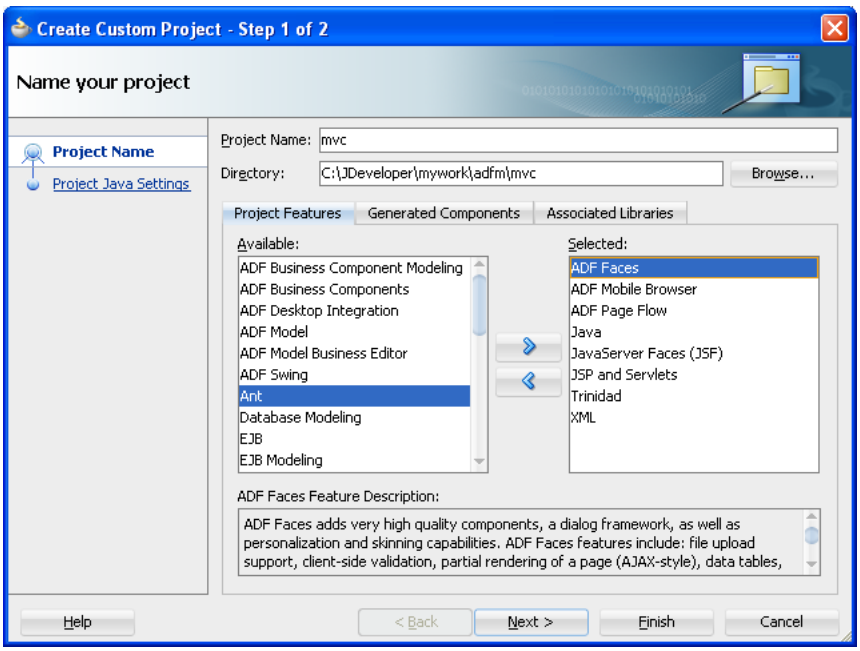
Figure 2–4 shows the ADF Mobile Browser feature in the **Available** window. After you move the **ADF Mobile Browser** feature to the **Selected** window, the Java, JavaServer Faces (JSF), JSP, and Servlets technologies are made available to the project and also appear in the **Selected** window, as shown in Figure 2–5.

Figure 2–5 ADF Mobile Browser and Supporting Features Selected for an ADF Mobile Browser Project



e. If the application uses an ADF task flow, add the **ADF Page Flow** feature to the **Selected** window. As shown in Figure 2–6, the ADF Page Flow feature adds the ADF Faces, ADF Page Flow, Trinidad, and XML technologies.

Figure 2–6 Adding ADF Page Flow Feature to the ADF Mobile Browser Project

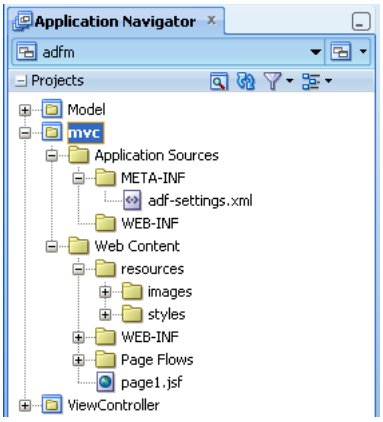


6. Click **Next** to navigate through the Configure Java settings page and then click **Finish**.

2.2.2 What Happens When You Create a Mobile Application and Project

As shown in Figure 2–7, the mobile Model View Controller project (*mvc*) appears in the Application Navigator within the Fusion web application (*adfm*).

Figure 2–7 The Mobile ADF Model View Controller Project in the Application Navigator



The creation of the Model View Controller project also results in the creation of the mobile application's page flow definition file, `faces-config.xml`. For information on creating a page flow, see "Defining Page Flows" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

Because you added the ADF Mobile Browser feature, the Apache MyFaces Trinidad library is automatically loaded to the workspace and the Trinidad component palette is loaded when you create mobile JSF pages, shown in Figure 2–10.

2.3 Developing an ADF Mobile Browser Application

For mobile browser applications, you develop an application by creating web pages within the web project. Otherwise, you develop a mobile browser application in the same way that you develop an ADF web application for a desktop browser. Typically, you create a web project within the application to implement a user interface and ADF Business Components or an Oracle EclipseLink project to implement a business layer.

2.3.1 How to Develop a Mobile JSF Page

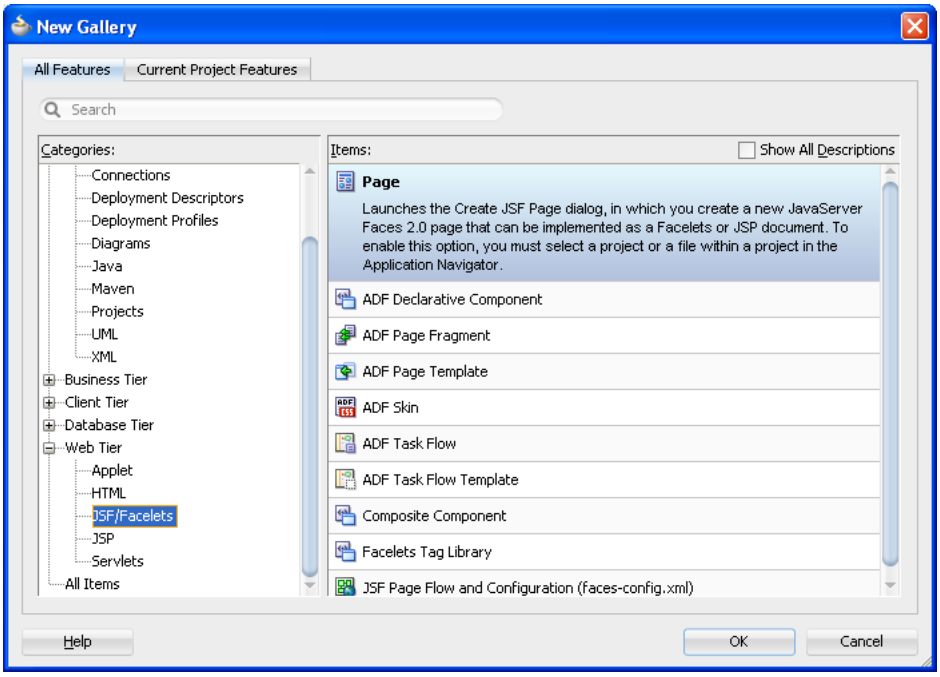
You develop an ADF Mobile browser application by first creating a JSP page and then populating it with the Apache My Faces Trinidad components.

To create a mobile JSF page:

1. Choose **File** and then **New**.
2. In the New Gallery, expand **Categories**, select **Web Tier**, select **JSF/Facelets** and then **Page**. Click **OK**.

> **Note:** **Project Technologies** (the default) must be selected from the Filter By list.

Figure 2–8 The New Gallery for JSF Pages



3. Enter a name for the JSF page in the JSF Page dialog, as shown in Figure 2–9. If needed, enter a directory location for the page.

Note: Because you added the **ADF Mobile Browser** feature for the application, the **Render in Mobile Device** option is selected by default, as shown in Figure 2–9.

Figure 2–9 The Create JSF Page Dialog Box

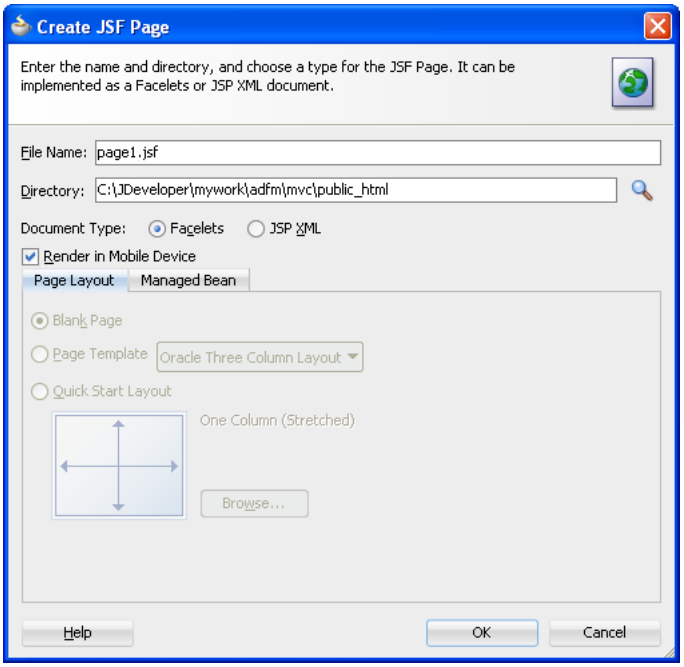
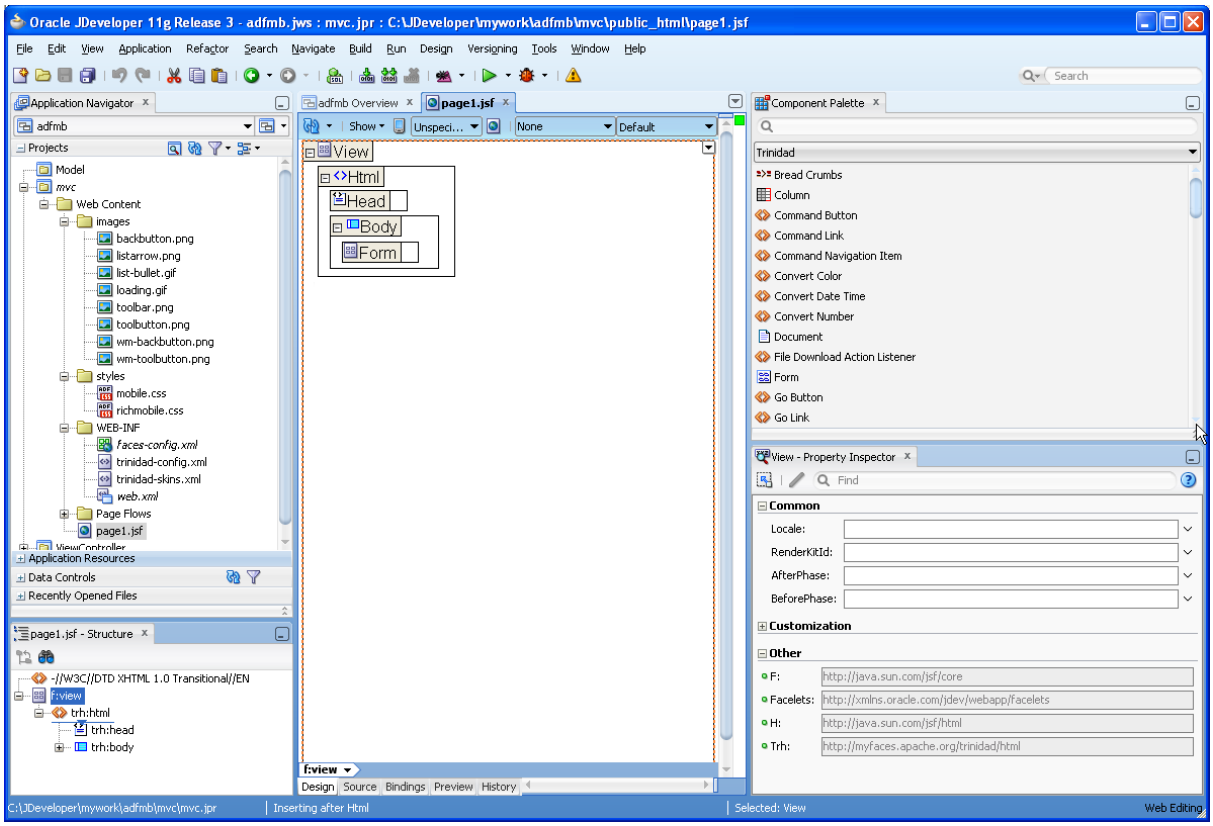


Figure 2–10 shows the designer for a mobile JSP page called *page1.jsp*.

4. From the Components Palette, select **Trinidad** as shown in Figure 2–10 and then create the page using the Apache MyFaces Trinidad components. You can create the page in the same manner as a desktop ADF web page.

Figure 2–10 Using the Trinidad Component Palette



2.3.2 What Happens When You Create a Mobile JSF Page

Because the **Render in Mobile Device** option is selected by default, the page designer in the visual editor reflects the size of a mobile device, as illustrated in Figure 2–10. This option also results in the creation of the following ADF skin-related configuration files and style sheets which are configured for ADF Mobile browser. Table 2–1 describes these files, which are located in the ADF Mobile browser View Controller project's Web Content folder.

Table 2–1 Skinning-Related Artifacts

Skinning File	Location	Description
`trinidad-config.xml`	WEB-INF node of the mobile View Controller project	Defines the skins used in the ADF Mobile browser application. For more information on using this file in a mobile context, see Section 4.2, "Implementing ADF Mobile Browser Skinning." See also "Configuration in trinidad-config.xml" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

Table 2–1 (Cont.) Skinning-Related Artifacts

Skinning File	Location	Description
`trinidad-skins.xml`	WEB-INF node of the mobile View Controller project	Lists the skins files that are automatically generated when you create the mobile JSF page. By default, these are `mobile.css` and `richmobile.css`. See also Section 4.2, "Implementing ADF Mobile Browser Skinning" and the "Configuration in trinidad-skins.xml" section in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.
`mobile.css`	Styles node of the mobile View Controller project	Used for basic HTML browsers and browsers used in older versions of smartphones.
`richmobile.css`	Styles node of the mobile View Controller project	Used for webkit-based smartphone browsers, such as those powered by iOS or Android.

2.4 Testing an ADF Mobile Browser Application

You can test an ADF Mobile browser application on a mobile device, a mobile device emulator, or a desktop browser. Testing on an actual mobile device or mobile device emulator provides more accurate results than does testing on a desktop browser.

Testing an ADF Mobile browser application with a desktop browser produces only approximate results because it provides a fairly uniform testing environment; web pages appear and behave similarly and business logic executes identically in any type of desktop browser. Testing an application on an actual mobile device, however, produces more accurate results, because the capabilities of mobile browsers may cause controls to behave differently than they do on a desktop browser. Mobile browsers, which are usually smaller than desktop browsers, render pages differently because web servers optimize the look and feel by generating mobile browser-specific pages.

Testing ADF Mobile browser applications directly on mobile devices has limitations as well, in that you may not have access to all of the devices that you must test. Furthermore, firewalls can complicate testing, as many mobile devices can access only the internet and cannot reach development environments behind firewalls. In such cases, mobile device emulators provide an alternative testing method. For example, to test applications on BlackBerry smartphone simulators or Windows Mobile device emulators (shown in Figure 2–11 and Figure 2–13, respectively), download smartphone simulators from the RIM developer site (`http://us.blackberry.com/`) and device emulators from the Microsoft developer site (`http://www.microsoft.com`). You must then configure them and connect them to the web server. For information on downloading and configuring simulators and emulators for ADF Mobile browser, refer to *Running Mobile Device Simulators with ADF Mobile and JDeveloper*, available through the *Mobile Application Development with Oracle ADF Mobile* page of the Oracle Technology Network. You can access this page by selecting **Oracle ADF Mobile** from the Oracle Technology Network's Oracle Application Development Framework overview page (`http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html`).

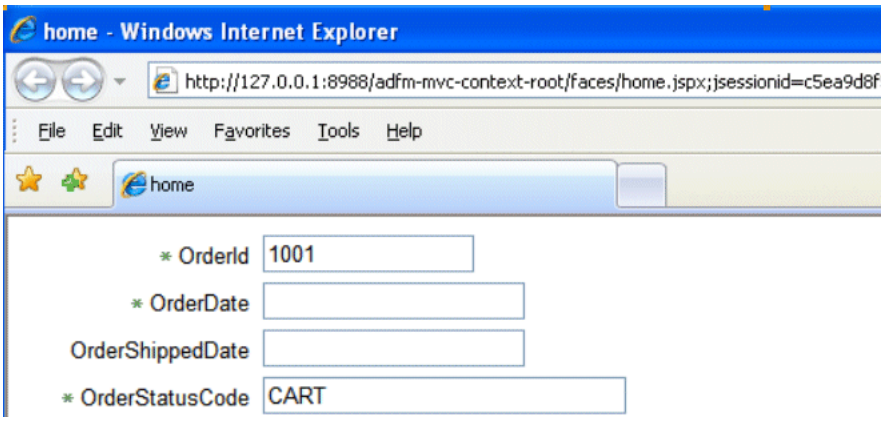
Figure 2–11 Testing an ADF Mobile Browser Application on a BlackBerry Smartphone Simulator

2.4.1 How to Test ADF Mobile Browser Applications on Emulators

After you test an application on a desktop browser, you can then test it on an emulator. You can use the URL displayed in the desktop browser, but if it uses the localhost IP address (127.0.0.1), you must change it to the network IP address of your computer.

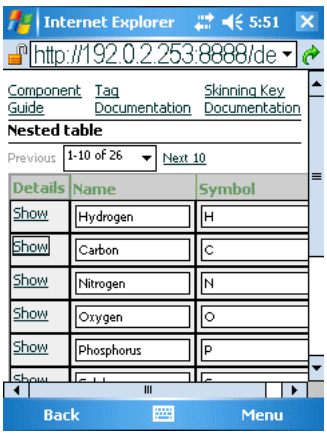
> **Tip:** To obtain the network IP address, use the `ipconfig` command interface on Windows systems and the `ifconfig` command on Linux/UNIX systems.

Figure 2–12 Testing an ADF Mobile Browser Application on a Desktop Browser



For example, to test an application using a Windows Mobile 6 emulator, change the address from the desktop's localhost IP address (*127.0.0.1*, shown in Figure 2–12) to that of the computer's network IP address (*192.0.2.253*, shown in Figure 2–13).

Figure 2–13 Testing an ADF Mobile Browser Application on a Windows Mobile Device Emulator



In addition, you must remove the session specification that follows the page name. The page name is typically appended with either `.jspx` or `.jsp`. In Figure 2–12, the page name, *home*, is appended with `.jspx`.

In general, you debug an application by repeating cycles of code and then by testing the application. When you test an application that has been modified, you must do one or both of the following:

- Refresh the page.
- Clear the browser's cache.

> **Tip:** Because the URL does not change if you develop the same application, you are not required to enter it again.

2.4.2 What You May Need to Know About Browser Settings

To view ADF Mobile browser applications properly requires adjustments to the browser settings for Windows Mobile and BlackBerry browsers.

Microsoft Windows Mobile 5 and 6, Microsoft Pocket Internet Explorer

For optimal viewing, select the *Fit to Screen* view (accessed by selecting **Menu**, **View** and then **Fit to Screen**).

> **Note:** Do not select the *One Column* view because it causes layout problems.

BlackBerry Browser 4.x

ADF Mobile browser only works if JavaScript support is enabled. To ensure that JavaScript support is enabled:

1. Select **Options** and then **Browser Configuration**.
2. Ensure that the following tables are selected:
   - Support JavaScript
   - Allow JavaScript Popup
   - Support HTML Tables

3

Component Support

This chapter describes the Apache MyFaces Trinidad components that are supported by ADF Mobile browser.

This chapter includes the following sections:

3.1 About Apache My Faces Trinidad Components

ADF Mobile browser supports more than 60 of the Apache MyFaces Trinidad components, enabling you to build applications with a rich component set that renders appropriately to the screens of mobile devices. For more information, refer to the Apache MyFaces Trinidad site (`http://myfaces.apache.org/trinidad/`).

3.1.1 Supported Features

ADF Mobile browser supports the following renderer-specific features for the supported browsers:

- Partial Page Rendering
- Dialogs

3.1.2 Partial Page Rendering

The high latency and low bandwidth of networks in mobile environments decrease application responsiveness for mobile users. Screens refresh slowly, diminishing the mobile user experience. ADF Mobile browser's support of Partial Page Rendering (PPR) compensates for the negative impact that slow connections have on screen updates by minimizing the amount of data requested from the server; using PPR, mobile device screen updates do not require a full refresh. Browsers that do not

support AJAX (Asynchronous JavaScript and XML) use full page rendering instead of PPR. For example, a page submission on basic HTML browsers (which do not support JavaScript) results in the refresh of a full page.

Note: Browsers for BlackBerry 4.5 and earlier versions do not support PPR. Specifying the `autosubmit` attribute on certain form components results in the submission of the page after the user exits the field. A full, not partial, refresh of the page then follows.

3.1.3 Dialogs

ADF Mobile browser supports dialogs, the pages used by applications to obtain user input. Because mobile browsers cannot open a new window that contains a dialog (a pop-up window), dialogs appear as new pages within the main browser window after automatically preserving the state of the current page.

3.1.4 Rendering Specific to the BlackBerry Browser 4.5 and Earlier Versions

On browsers for BlackBerry 4.5 and earlier versions, the bullets in a list sublevel (such as those in a `tr:panelList` component) appear large and are not indented. BlackBerry's table handling may affect complex layouts, as its browser does not allow horizontal scrolling. Instead, it wraps a table row onto multiple display lines which may disturb the layout. For more information, see Chapter 6, "Design Guidelines for BlackBerry 4.2 to 4.5."

3.2 Input Components

ADF Mobile browser supports input text fields and lists, core components that support user input.

3.2.1 Creating Input Text Fields

You can create input fields using the following components:

- `tr:inputColor`

  Note: Mobile browsers do not support an inline `chooseColor` or a `color` dialog for the `tr:inputColor` component.

- `tr:inputDate`
- `tr:inputHidden`
- `tr:inputText`

  Note: Basic HTML browsers do not support the `autosubmit` attribute of the `tr:inputText` component.

  Note: Trinidad optimizes the `tr:inputText` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Chapter 7, "Narrow Screen Support and User Agent Details Support."

3.2.2 Creating Lists

You can create lists using the following components:

- `tr:panelChoice`
- `tr:panelList`
- `tr:selectBooleanCheckBox`
- `tr:selectBooleanRadio`
- `tr:selectItem`

> **Note:** Mobile browsers do not support the `disabled` attribute for the `tr:selectItem` component.

- `tr:selectManyCheckBox`
- `tr:selectManyListBox`
- `tr:selectOneChoice`
- `tr:selectOneListBox`
- `tr:selectOneRadio`
- `tr:resetButton`

> **Note:** Basic HTML browsers do not support the `autosubmit` attribute for the `tr:resetButton` component.

3.3 Output Components

ADF Mobile browser uses the Apache MyFacesTrinidad core components that support output on mobile device applications. These components include those for displaying text and images and also components for displaying or hiding text.

3.3.1 Displaying Text

The following components enable you to display text:

- `tr:iterator`
- `tr:message`
- `tr:messages`

> **Note:** When using the `tr:message` and `tr:messages` components, the component-specific messages do not display as they do in a desktop browser. Instead, they display in the region where the message component is placed on the Web page.

- `tr:outputDocument`
- `tr:outputForwarded`
- `tr:outputLabel`
- `tr:outputText`

3.3.2 Displaying Images

The following components enable you to display images:

- `tr:icon`
- `tr:image`
- `tr:panelTip`

3.3.3 Showing (or Hiding) Components

The following components enable showing or hiding items:

- `tr:panelAccordion`

  Note: Mobile browsers only support a full-page update; they do not support the `partialTriggers` attribute of the `tr:panelAccordion` component.

- `tr:panelTabbed`

  Note: To save space on mobile devices, the renderer intentionally prevents the display of tab bars on both the top and bottom of the `tr:panelTabbed` component. Valid values for the attribute positions are `top` and `bottom`. If `both` is specified, then the renderer displays the tabs on top.

- `tr:showDetail`

  Note: For the `tr:showDetail` component, the disclosure arrow does not display; instead [+] and [-] display.

- `tr:showDetailHeader`

  Note: For the `tr:showDetailHeader` component, the disclosure arrow does not appear on mobile browsers.

- `tr:showDetailItem`

  Note: For the `tr:showDetailItem` component, the disclosure arrow does not appear on mobile browsers and `flex` is not supported.

3.4 Layout Components

The layout components supported by ADF Mobile browser include those for managing the page itself (such as `tr:document` and `tr:form`) as well as such components for laying out the sections of a page as `tr:group`, `tr:panelFormLayout`, and `tr:panelGroupLayout`.

3.4.1 Managing the Page

The following components enable you to manage the page:

- `tr:document`
- `tr:form`

  Note: Mobile browsers do not support the `defaultCommand` attribute of the `tr:form` component.

- `tr:page`

  Note: Mobile browsers do not support the `tr:page` facet of the `tr:page` component.

3.4.2 Laying Out Sections of the Page

The following ADF Faces core tags support page layout for mobile device applications:

- `tr:group`
- `tr:panelBorderLayout`

  Note: Only the `top` and `bottom` facets are supported for the `tr:panelBorderLayout` component. Mobile browsers do not support the following facets:

  - `left`
  - `right`
  - `start`
  - `end`
  - `innerLeft`
  - `innerRight`
  - `innerStart`
  - `innerEnd`

  The `tr:panelBorderLayout` component does not render if you use any of these unsupported facets.

- `tr:panelBox`
- `tr:panelFormLayout`
- `tr:panelGroupLayout`
- `tr:panelHeader`
- `tr:panelHorizontalLayout`

  Note: Mobile devices do not support the `halign=end` in the `tr:panelHorizontalLayout` component.

- `tr:panelLabelAndMessage`

  Note: Trinidad optimizes the `tr:panelLabelAndMessage` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Section 7.1, "Determining Narrow Screen Support."

- `tr:panelPage`
- `tr:panelPageHeader`

  Note: Mobile devices only support the following facets of the `tr:panelPageHeader` component:

  - `branding`
  - `brandingApp`
  - `navigation1`
  - `navigation2`

- `tr:panelRadio`

  Note: Trinidad optimizes the `tr:panelRadio` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Section 7.1, "Determining Narrow Screen Support."

- `tr:panelCaptionGroup`

3.4.3 Inserting Spaces

The following components control the space allocation on pages:

- `tr:separator`
- `tr:spacer`
- `tr:subform`

3.5 Navigation Components

ADF Mobile browser supports components as buttons, links, and breadcrumbs that enable users to navigate to other pages of the application or to external locations.

3.5.1 Creating Buttons

ADF Mobile browser supports the following button types:

- `tr:commandButton`

Note: Because the `text` attribute cannot display if the `icon` attribute has been set, buttons on mobile devices can have either text or an image, but not both. If you set the `disabled` attribute to `true`, then the `tr:commandButton` component with an `icon` attribute renders as a static image with no links.

- `tr:goButton`

See Chapter 8, "Extending ADF Mobile Browser Applications" for information on how to use the `tr:goButton` component to integrate e-mail, telephony, and Google maps into an application.

3.5.2 Creating Links

ADF Mobile browser supports the following components for creating hyper-links:

- `tr:commandLink`

Note: Because the `tr:commandLink` component renders as an input element in basic mobile HTML browsers, its child components cannot render. For more information on input elements in basic mobile HTML browsers, see Section 5.2, "Developing Applications for Basic HTML Mobile Browsers."

- `tr:goLink`

See Chapter 8, "Extending ADF Mobile Browser Applications" for information on how to use the `tr:goLink` component to integrate e-mail, telephony, and Google maps into an application.

3.5.3 Navigation Components

ADF Mobile browser supports the following navigation components:

- `tr:breadcrumbs`

Note: Trinidad optimizes the `tr:breadcrumbs` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Chapter 7.1, "Determining Narrow Screen Support."

- `tr:commandNavigationItem`

Note: `tr:commandNavigationItem` does not render when you set the `disabled` attribute to `true` for the following:

- `tr:selectOneListBox`
- `tr:selectOneChoice`
- `tr:processChoiceBar`
- `tr:navigationPane` with hint, `"choice"`
- `tr:selectRangeChoiceBar`

- `tr:navigationPane`

  Note: `tr:navigationPane hint = "choice"` with a destination value is not supported for basic HTML browsers.

  Note: Trinidad optimizes the `tr:navigationPane` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Chapter 7.1, "Determining Narrow Screen Support."

- `tr:train`

  Note: The `tr:train` component appears as x of y instead of listing each item. This is a display-only component in ADF Mobile browser; users cannot navigate through the application by clicking the x of y component. To enable navigation, you must add a separate link or button.

- `tr:processChoiceBar`

  Note: Trinidad optimizes the `tr:processChoiceBar` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Section 7.1, "Determining Narrow Screen Support."

- `tr:selectRangeChoiceBar`

  Note: Trinidad optimizes the `tr:selectRangeChoiceBar` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Chapter 7.1, "Determining Narrow Screen Support."

3.6 Data Visualization (Graphs and Gauges)

ADF Mobile browser supports data visualization tools (DVTs), described in the "Creating Databound ADF Data Visualization Components" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

ADF Mobile browser supports the following types of graphs:

- area
- bar
- bar (horizontal)
- bubble
- combination (horizontal bar and line)
- funnel
- line

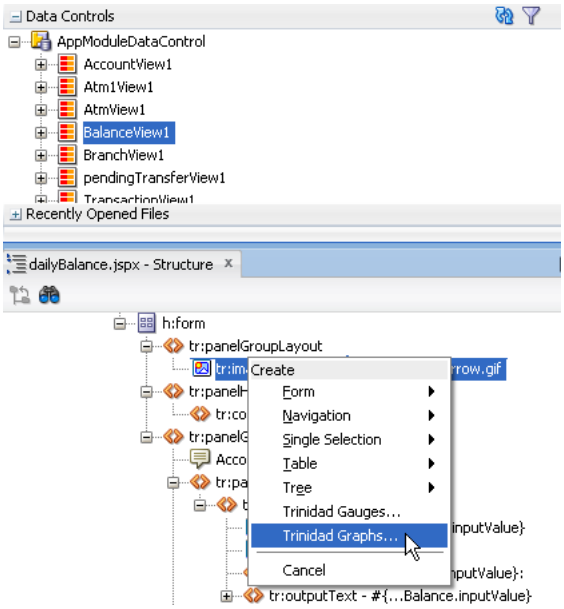
- pareto
- pie
- radar
- scattar/polar
- spark
- stock

ADF Mobile browser supports the following types of gauges:

- dial
- status meter
- status meter (vertical)
- LED

To add these components to an application, first move a data control into the editor or Structure window using a drag-and-drop operation and then select **Trinidad Gauges** or **Trinidad Graphs** from the context menu. For example, Figure 3–1 illustrates the context menu that appears when an iterator called *BalanceView1* is dragged and dropped into the editor window.

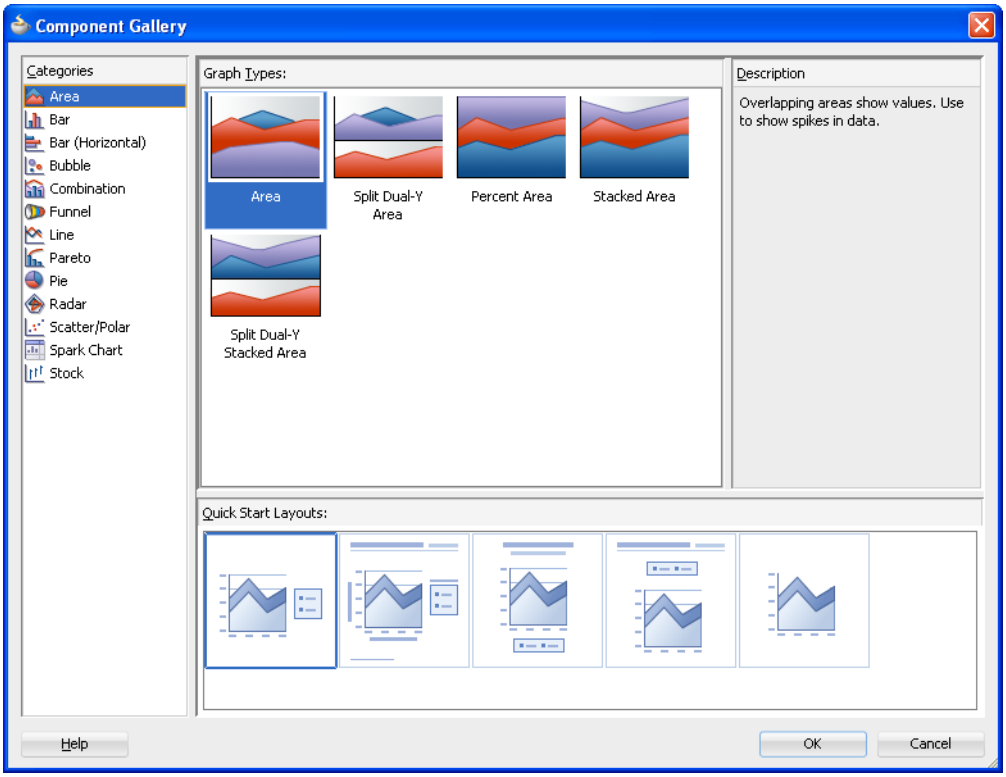
Figure 3–1 Creating a DVT Component



After you select either the **Trinidad Graphs** or **Trinidad Gauges** options, the DVT wizard appears and opens to the Component Gallery page, shown in Figure 3–2. You select the DVT type from this page.

Note: For Oracle Fusion Middleware 11*g* release 2, ADF Mobile browser supports only static graphs and gauges, which are rendered as PNG images. Any mobile device that supports this image format can display these graphs and gauges.

Figure 3–2 The Component Gallery for Trinidad Graphs



3.7 Tables and Trees

ADF Mobile browser applications can display structured data in the rows and columns of a table or hierarchically in and trees.

3.7.1 Creating Tables

ADF Mobile browser supports tables comprised of the following components:

- `tr:table`

  Note: ADF Mobile browser does not support the `allDetailsEnabled` attribute for the `tr:table` component; this attribute is always set to `false`.

- `tr:column`

  Note: When you nest `tr:column` tags to create column groups, the header facets do not render for the column groups.

3.7.2 Creating Trees

ADF Mobile browser supports the `tr:tree` component.

Note: `tr:tree` may not render on basic HTML browsers.

3.8 Generating HTML <meta> Tags

The <meta> tag designates how pages display within a browser. Example 3–1 illustrates examples of <meta> tag usage that include setting an application to display in full-screen mode and setting the viewport width on such devices as Apple's iPhones and iPads. This example also includes a <meta> tag used to designate how a page reloads after a given number of sections.

Example 3–1 Using the <meta> Tag to Set Page Behavior

```
<meta name="viewport" content="width=device-width, user-scalable=no">
<meta name="apple-mobile-web-app-capable" content="yes">
<meta http-equiv="refresh" content="2;url=./test/index.jspx">
```

For more information on the use of the <meta> tag in applications running on Apple devices, see *Apple-Specific Meta Tag Keys* in the "Supported Meta Tags" section of *Safari HTML Reference*, available at: http://developer.apple.com.

3.8.1 Using <trh:meta> to Generate HTML <meta> Tags

The <trh:meta> component tag generates an HTML <meta> tag. Example 3–2 shows using this component to:

- Configure the viewport dimensions on mobile devices.
- Disable user resizing of the viewport on mobile devices.
- Enable the home screen bookmark of the web page to launch it in its own process rather than from within Safari on iOS devices.
- Configure the page to navigate to another page after two seconds.

Example 3–2 Using <trh:meta> to Generate Several HTML <meta> Tags

```
<af:document ...>
  <f:facet name="metaContainer">
    <af:group id="metaContainer">
      <trh:meta name="viewport" content="width=device-width, user-scalable=no"/>
      <trh:meta name="apple-mobile-web-app-capable" content="yes"/>
      <trh:meta type="httpEquiv" name="refresh"
content="2;url=./test/index.jspx"/>
    </af:group>
   </f:facet>
</af:document>
```

3.8.1.1 About Default Viewport Size on Mobile Devices

Web pages commonly have a hard-coded width that leaves empty white space on both sides of the page. While this does not interfere with pages displaying on desktop browsers, it can make the same pages appear too small on mobile devices. In addition, the display of the page controls is too small for a users' finger tips as well, forcing users to zoom in if they need to interact with the page. This scaling occurs when web pages display on mobile devices because operating systems for Apple (iOS), Google (Android), and Micosoft (Windows Phone 7) assume a standard viewport width and render the page to fit the smaller screens.

To enable a page to display properly at the native resolution of a mobile device, you can specify the viewport dimension using the viewport meta key within the <trh:meta> tag, as illustrated in Example 3–3.

Example 3–3 Setting the Viewport Dimensions

```
<af:document ...>
   <f:facet name="metaContainer">
     <trh:meta name="viewport" content="width=device-width"/>
   </f:facet>
   ...
 </af:document>
```

You can set a specific numeric value for pixels for the `width` property of the `viewport` meta key or you can use the `device-width` constant as shown in Example 3–3. On Apple (iOS) systems, the value set for the page's width in both the portrait and landscape orientations is the same (that is, for the landscape width, the Safari browser uses the value set for the portrait width).

> **Note:** The pixels on high-resolution displays, such as Retina on iPhone 4, do not have a one-to-one match to pixels on a lower-resolution devices such as iPhone 3GS. Despite this, the number of pixels defined for the `width` property for an iPhone 3GS equates to the same physical length on the iPhone 4, even though the iPhone 4 has more display pixels.

3.9 Unsupported Components and Attributes

Release 11*g* of ADF Mobile browser does not support some components or attributes.

3.9.1 Unsupported Components

Release 11*g* of ADF Mobile browser does not support the following components:

- `tr:chart`
- `tr:chooseColor`
- `tr:chooseDate`
- `tr:inputFile`
- `tr:inputListOFVariables`
- `tr:inputNumberSpinbox`
- `tr:legend`
- `tr:media`
- `tr:navigationTree`
- `tr:panelButtonBar`
- `tr:panelPopup`
- `tr:panelSideBar`
- `tr:poll`
- `tr:progressIndicator`
- `tr:selectManyShuttle`
- `tr:selectOrderShuttle`
- `tr:singleStepButtonBar`

- `tr:statusIndicator`
- `tr:switcher`
- `tr:treeTable`

3.9.2 Unsupported Attributes

Release 11*g* of ADF Mobile browser does not support the following component attributes on any component.

- `accessKey`
- `shortDesc (tooltip)`

4 Skinning

This chapter describes skinning for ADF Mobile browser applications.

This chapter includes the following sections:

- Section 4.1, "About ADF Mobile Browser Skinning"
- Section 4.2, "Implementing ADF Mobile Browser Skinning"
- Section 4.3, "Applying ADF Mobile Browser Skinning"

4.1 About ADF Mobile Browser Skinning

Skinning enables a page to display consistently on a variety of devices through the automatic delivery of device-dependent style sheets. These style sheets enable the optimal display of pages that share the same page definitions on various mobile browsers. Within these style sheets, which enable you to set the look and feel of an application, you not only tailor a component to a specific browser by setting its size, location, and appearance, but you also specify the types of browsers in which components can be displayed or hidden.

Note: Browsers must support the Cascading Style Sheet (CSS) syntax.

4.2 Implementing ADF Mobile Browser Skinning

As noted in Section 2.3.2, "What Happens When You Create a Mobile JSF Page," JDeveloper creates two mobile-specific stylesheets, `mobile.css` and `richmobile.css` within the ADF Mobile browser-specific View Controller project, as shown in Figure 4–1. Creating a mobile JSF page also populates the `trinidad-config.xml` with an EL expression for selecting the skin families defined by these skinning files and also populates `trinidad-skins.xml` with definitions for the ADF Mobile browser skins.

Figure 4–1 The Mobile Browser Skinning Files

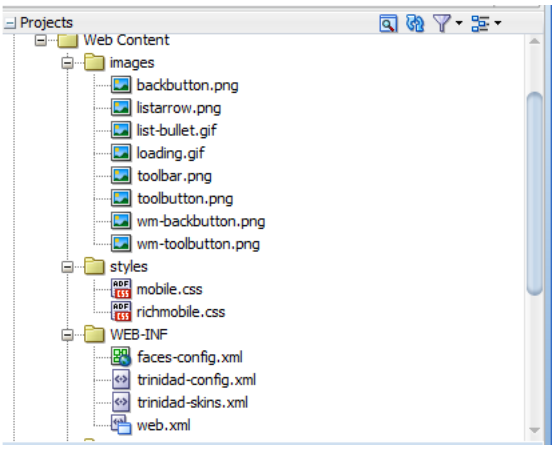


Table 4–1 lists the skinning files provided to ADF Mobile pages.

Table 4–1 The ADF Mobile Browser Skins

CSS File	Skin Family	Use
`mobile.css`	mobile	Used for basic HTML browsers. This family is used for rendering on Windows Mobile and BlackBerry, Version 4.6 and higher. See Chapter 5, "Supporting Basic HTML Mobile Browsers."
`richmobile.css`	richmobile	Used for smartphone browsers that use the Webkit rendering engine. Such browsers are used on the Nokia S60 and also iOS- and Android-powered devices.

Example 4–1 illustrates the `trinidad-config.xml` with the EL expression embedded within its `<skin-family>` element that evaluates to the string that returns the skin family type requested by the browser.

Example 4–1 The Skin Family Selection Logic within Trinidad-config.xml

```
<?xml version="1.0" encoding="windows-1252"?>
<trinidad-config xmlns="http://myfaces.apache.org/trinidad/config">
  <skin-family>#{requestContext.agent.type == 'desktop'? 'richmobile': 'mobile'}</skin-family>
</trinidad-config>
```

Example 4–2 illustrates `trinidad-skins.xml`, whose `<skin>` elements are defined for the default ADF Mobile browser skins when you create a mobile JSF page.

Example 4–2 trinidad-skins.xml Populated with ADF Mobile browser <skin> Definitions

```
<?xml version="1.0" encoding="windows-1252"?>
<!-- To use mobile skin families in your app, please update trinidad-config.xml with below tags -->
<!-- <skin-family>#{requestContext.agent.type == 'desktop'?'richmobile': 'mobile'}</skin-family>
-->
  <skins xmlns="http://myfaces.apache.org/trinidad/skin">
    <skin>
      <id>richmobile</id>
      <family>richmobile</family>
      <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
```

```
    <style-sheet-name>styles/richmobile.css</style-sheet-name>
  </skin>
  <skin>
    <id>mobile</id>
    <family>mobile</family>
    <render-kit-id>org.apache.myfaces.trinidad.pda</render-kit-id>
    <style-sheet-name>styles/mobile.css</style-sheet-name>
  </skin>
</skins>
```

4.2.1 Extending the ADF Mobile Skins

You can configure `trinidad-config.xml` and `trinidad-skins.xml` to call other CSS files. You can also modify the `richmobile.css` and `mobile.css` files to render to a specific mobile device or platform using `@rule`.

To add mobile features to a non-mobile project:

1. Create a skin (`trinidad-skins.xml`, located in the either the **WEB-INF** or **META-INF** nodes).
2. Create a style sheet.
3. Set the skin family in `trinidad-config.xml` (located in the **WEB-INF** node).

   For more information, see "Create a Skin—An Overview" in *Development Guidelines for Apache MyFaces Trinidad* (`http://myfaces.apache.org/trinidad/devguide/skinning.html`).

Skin families in Apache MyFaces Trinidad are associated with a renderkit and a unique CSS file. Because Trinidad uses the desktop renderkit for Webkit-based mobile browsers and the PDA renderkit for all other mobile browsers, you can support all mobile browsers by creating two skin families, both of which reference one of these renderkits and a CSS file. Use the `@agent` and `@platform` selector rules to enable rendering based on the browser's name, version, or platform. For more information, see "Skinning CSS Features" in *Development Guidelines for Apache MyFaces Trinidad* (`http://myfaces.apache.org/trinidad/devguide/skinning.html`).

4.2.2 What Happens at Runtime

The EL expression defined within the `trinidad-config.xml` `<skin-family>` element selects the renderkit appropriate to the browser's user agent. The ADF Mobile browser framework then applies the style defined for the renderkit.

4.3 Applying ADF Mobile Browser Skinning

Although CSS styles are applied automatically for many components, some components require you to manually set the style classes to its `styleClass` attribute.

4.3.1 Headers

Augmenting the `<tr:panelHeader>` component with the `styleClass` attribute enables you to display title-only headers and headers with a title and links on various browsers.

4.3.1.1 Creating a Title-Only Header

To create a title-only header, add `styleClass="af_m_toolbar"` to the `<tr:panelHeader>` component as illustrated in Example 4–3.

Example 4–3 Adding Attributes to Create a Title-Only Header

```
<tr:panelHeader styleClass="af_m_toolbar" text="Welcome"/>
```

Figure 4–2 shows how this ADF Mobile browser attribute creates a title-only header on an Apple iPhone.

Figure 4–2 A Title-Only Header on the Apple iPhone

Table 4–2 lists examples of how title-only headers display on Windows Mobile devices, BlackBerry smartphones, and the Nokia Webkit.

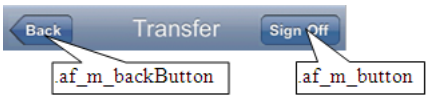
Table 4–2 Title-Only Header Displays on Various Platforms

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

4.3.1.2 Creating Headers with Titles and Links

As illustrated in Figure 4–3, you can add links and a title within a header. Figure 4–3 shows such a header as it displays in on the Apple iPhone.

Figure 4–3 Title and Links Within a Header on Apple iPhone



As described in Section 4.3.1.1, "Creating a Title-Only Header," you define the title for the header (in Figure 4–3, a title called *Transfer*) by adding `styleClass="af_m_toolbar"` within the `<tr:panelHeader>` element. The links are defined as buttons (`styleClass="af_m_backButton"` and `styleClass="af_m_button"`, respectively) within the child `<tr:commandLink>` element as illustrated in Example 4–4. In Example 4–4, the `<tr:panelHeader>` element includes these attributes (in bold).

Example 4–4 Adding Titles and Links to Headers

```
<tr:panelHeader styleClass="af_m_toolbar"
```

```
          text="Transfer">
    <tr:commandLink styleClass="af_m_backButton"
                    text="Back"
                    action="back"/>
    <tr:spacer rendered="#{requestContext.agent.skinFamilyType eq 'blackberryminimal'}"
               height="5" width="105"/>
    <tr:spacer rendered="#{requestContext.agent.skinFamilyType eq 'windowsmobile'}"
               height=""
               width="28"/>
     <tr:commandLink styleClass="af_m_button"
                     text="Sign Off"
                     action="signoff"/>
</tr:panelHeader>
```

Table 4–3 lists examples of how the `<tr:panelHeader>` that includes a title and links display on Windows Mobile devices, BlackBerry smartphones, and the Nokia Webkit.

Table 4–3 Title and Link Headers on Various Platforms

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

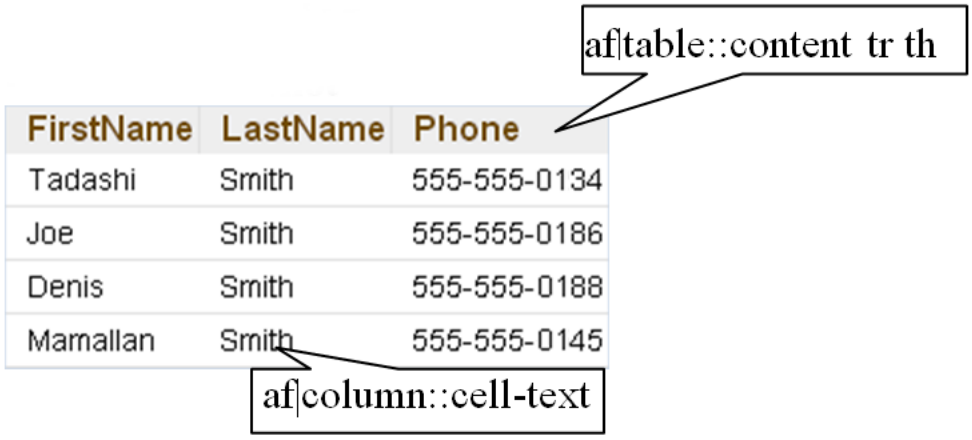
4.3.2 Table Components

Using the `styleClass` attribute enables table components within ADF Mobile browser application to render appropriately on various browsers.

4.3.2.1 Multi-Column Tables

Unlike panel headers, which require that you include the `styleClass` attribute to apply the style appropriately on the target platform, the table column headers do not require any attributes. Instead, you use the `<tr:columns>` component described in Section 3.7.1, "Creating Tables." Figure 4–4 illustrates how column headers render on the Apple iPhone.

Figure 4–4 Column Headers and Cells on Apple iPhone



Example 4–5 illustrates how to define the `<tr:columns>` element (in bold).

Example 4–5 Creating Column Headers

```
<tr:table  var="row" …./>
            <tr:column  headerText="LastName">
                <tr:outputText value="#{row.bindings.LastName.inputValue}"/>
            </tr:column>
            <tr:column  headerText="FirstName">
                <tr:outputText value="#{row.bindings.FirstName.inputValue} "/>
             </tr:column>
             <tr:column headerText="Phone">
                <tr:outputText value="#{row.bindings.Phone.inputValue}"/>
            </tr:column>
</tr:table>
```

Table 4–4 shows examples of how column headers display on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

Table 4–4 Column Headers on Various Platforms

Platform	Example
BlackBerry 4.6	
Windows Mobile	

Table 4–4 (Cont.) Column Headers on Various Platforms

Platform	Example
Nokia Webkit	
BlackBerry 4.2	

4.3.2.2 Adding Images and Primary Details with Links

Figure 4–5 demonstrates creating the links and details within a table using the `styleClass` values `af_m_listingLink` and `af_m_listingDetails`.

Figure 4–5 Images, Links and Details as Rendered on the Apple iPhone

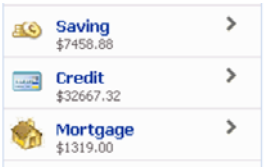
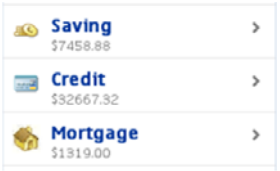
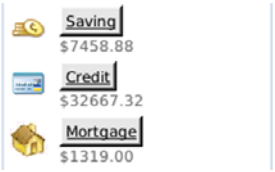
As illustrated in Example 4–6, you create these features by adding a `<tr:panelGroupLayout>` component as a child of a `<tr:column>` component. You then add the `styleClass="af_m_listingLink"` and `styleClass="af_m_listingDetails"` attributes to the `panelGroupLayout`'s `<tr:commandLink>` and `<tr:outputText>` subcomponents. See Chapter 3, "Component Support" for information on the `tr:panelGroupLayout`, `tr:commandLink`, and `tr:outputText`.

Example 4–6 Adding Links with Details

```
<tr:table  horizontalGridVisible="false"
           var="row"
           width="100%">
            <tr:column>
                <tr:image source="#{row.bindings.TypeIconUrl.inputValue}"/>1
            </tr:column>
            <tr:column inlineStyle="width:100%;">
                <tr:panelGroupLayout layout="vertical">
                    <tr:commandLink text="#{row.bindings.DescShort.inputValue}"
                                    action="detail"
                                    styleClass="af_m_listingLink">
                    </tr:commandLink>
                    <tr:outputText value="#{row.bindings.Balance.inputValue}"
                                   styleClass="af_m_listingDetails">
                    </tr:outputText>
                </tr:panelGroupLayout>
            </tr:column>
 </tr:table>
```

Table 4–5 shows examples of how images, links, and details display on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

Table 4–5 Images, Links, and Details on Various Platforms

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

4.3.2.3 Creating Primary Details with Links

Figure 4–6 illustrates how to create primary details and links within a table.

Figure 4–6 Primary Details with Links as Rendered on Apple iPhone

Similar to adding the primary links and details with images described in Section 4.3.2.2, "Adding Images and Primary Details with Links," you create these features by adding a `<tr:panelGroupLayout>` component as a child of a `<tr:column>` component. As illustrated in Example 4–7, you then add the `styleClass="af_m_listingLink"` and `styleClass="af_m_listingDetails"` attributes to the `panelGroupLayout`'s `<tr:commandLink>` and `<tr:outputText>` subcomponents. See Chapter 3, "Component Support" for information on the `tr:panelGroupLayout`, `tr:commandLink`, and `tr:outputText`.

Example 4–7 Primary Details and Links

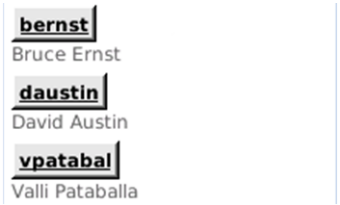
```
<tr:table  horizontalGridVisible="false"
           var="row"
           width="100%">
            <tr:column>
                <tr:panelGroupLayout layout="vertical">
                    <tr:commandLink text="#{row.bindings.Email.inputValue}"
                          styleClass=" af_m_listingLink">
                    </tr:commandLink>
                    <tr:outputText value="#{row.bindings.FirstName.inputValue}"
                          styleClass="af_m_listingDetails"/>
                </tr:panelGroupLayout>
            </tr:column>
</tr:table>
```

Table 4–6 shows examples of how links and details display on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

Table 4–6 Images and Links on Various Platforms

Platform	Example
BlackBerry 4.6	

Table 4–6 (Cont.) Images and Links on Various Platforms

Platform	Example
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

4.3.2.4 Creating Primary Details Without Links

As illustrated in Figure 4–7, `af_m_listingPrimaryDetails` and `af_m_listingDetails` style classes enable you to create details that do not function as links; their behavior is different from the primary details described in Section 4.3.2.2, "Adding Images and Primary Details with Links."

Figure 4–7 Primary Details without Links on Apple iPhone

As illustrated in Example 4–8, you create non-linking primary details by adding `styleClass=af_m_listingPrimaryDetails` and `styleClass="af_m_listingDetails"` to the `<tr:outputText>` element. This element is a child of the

<tr:panelGroupLayout> element (which is itself a child of the <tr:column> element).

Example 4–8 Adding Non-Linking Primary Details

```
tr:table  horizontalGridVisible="false"
          var="row"
          width="100%">
             <tr:column>
                <tr:panelGroupLayout layout="vertical">
                    <tr:outputText value="#{row.bindings.Amount.inputValue}
                        styleClass="af_m_listingPrimaryDetails">
                    </tr:outputText>
                    <tr:outputText value="
#{row.bindings.FromAccountName.inputValue} "
                        styleClass="af_m_listingDetails"/>
                </tr:panelGroupLayout>
             </tr:column>
</tr:table>
```

Table 4–7 shows examples of how non-linking details display on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

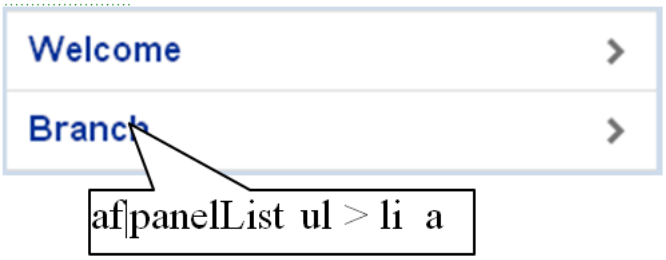
Table 4–7 Non-Linking Details on Various Platforms

Platform	Example
BlackBerry 4.6	**$586.6 on 2008-12-09** From Credit To Saving **$53.9 on 2008-12-09** From Saving To Checking **$263.31 on 2008-12-09** From Checking To Loc
Windows Mobile	**$836.94 on 2008-12-09** From Mortgage To Credit **$663.39 on 2008-12-09** From Saving To Checking **$588.92 on 2008-12-09** From Credit To Saving
Nokia Webkit	**$765.55 on 2008-12-09** From Credit To Saving **$817.08 on 2008-12-09** From Credit To Saving **$703.09 on 2008-12-09** From Credit To Saving
BlackBerry 4.2	$817.08 on 2008-12-09 From Credit To Saving $703.09 on 2008-12-09 From Credit To Saving $20.52 on 2008-12-09 From Checking To Loc

4.3.3 Panel List Components

Defining the value of the styleClass as af_m_panelBase within the <tr:panelGroupLayout> component applies padding to the <tr:panelList> components, as shown in Figure 4–8.

Figure 4–8 Rendering Padding on an Apple iPhone



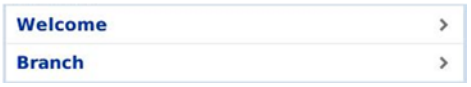
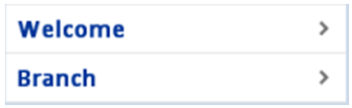
As illustrated in Example 4–9, you do not have to include a `styleClass` attribute in the child `<tr:panelList>` component. For more information on using `<tr:panelList>` and `<tr:panelGroupLayout>`, see Section 3.2.2, "Creating Lists" and Section 3.4, "Layout Components," respectively.

Example 4–9 Adding Padding to panelList Components

```
<tr:panelGroupLayout styleClass="af_m_panelBase">
        <tr:panelList>
            <tr:commandLink text="Welcome" action="welcome"/>
            <tr:commandLink text="Branch" action="branch"/>
         </tr:panelList>
</tr:panelGroupLayout>
```

Table 4–8 shows examples of padding in the `<tr:panelList>` component on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

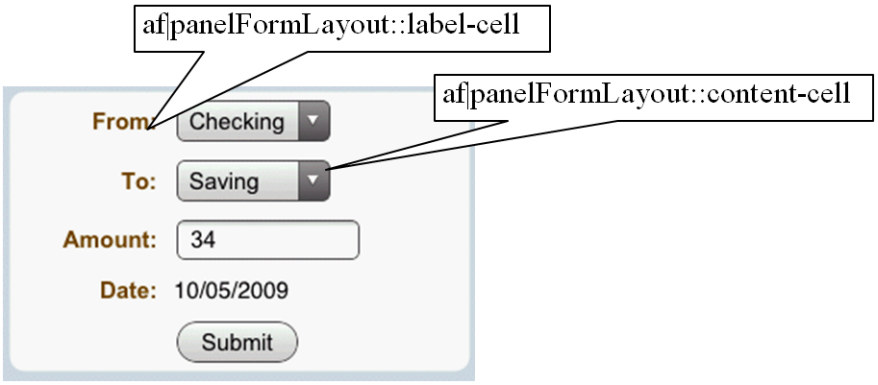
Table 4–8 Padding Applied to <tr:panelList> on Various Platforms

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

4.3.4 PanelFormLayout

Defining the value of the `styleClass` attribute as `af_m_panelBase` within the `<tr:panelGroupLayout>` component applies padding to the child `<tr:panelFormLayout>` components, as shown in Figure 4–9.

Figure 4–9 Padding Rendered in panelFormLayout on Apple iPhone



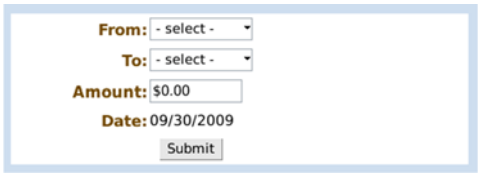
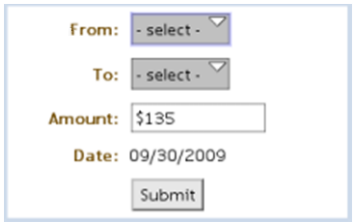
As illustrated in Example 4–10, you do not need to add `styleClass` to the `<tr:panelFormLayout>` component.

Example 4–10 Applying Padding to the PanelFormLayout Component

```
<tr:panelGroupLayout styleClass="af_m_panelBase">
     <tr:panelFormLayout labelWidth="35%" fieldWidth="65%">
         <tr:selectOneChoice value="#{transferBean.transferFromAccount}"
              label="From:" showRequired="false">
                   <f:selectItems value="#{bindings.AccountView1.items}"/>
         </tr:selectOneChoice>
         <tr:selectOneChoice  value="#{transferBean.transferToAccount}"
                              showRequired="false"
                              unselectedLabel="- select -"
                              label="To:">
                   <f:selectItems value="#{bindings.AccountView1.items}"/>
         </tr:selectOneChoice>
         <tr:inputText id="amount"
                       columns="#{requestContext.agent.capabilities.narrowScreen ? '8' : '12'}"
                       required="false" showRequired="false"
                       value="#{transferBean.transferAmount}"
                       label="Amount:">
                   <f:converter converterId="Bank10.amountConverter"/>
         </tr:inputText>
         <tr:panelLabelAndMessage label="Date: ">
              <tr:outputText value="#{transferBean.transferDate}"/>
         </tr:panelLabelAndMessage>
        <f:facet name="footer">
            <tr:panelGroupLayout>
                 <tr:spacer rendered="#{requestContext.agent.skinFamilyType eq
                                        'blackberryminimal'}"
                            height="5"
                            width="75"/>
                  <tr:commandButton text="Submit"
                                    action="#{transferBean.validateTransferRequest}"/>
            </tr:panelGroupLayout>
        </f:facet>
     </tr:panelFormLayout>
</tr:panelGroupLayout>
```

Table 4–9 shows examples of padding in the `<tr:panelList>` component on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

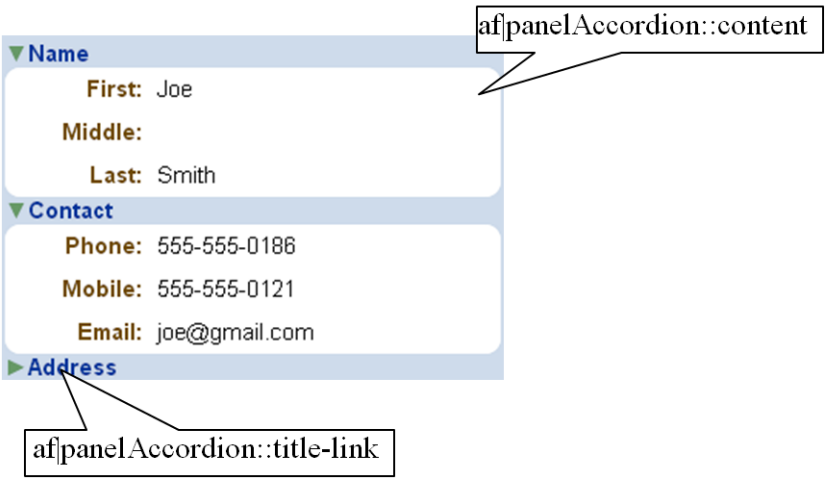
Table 4–9 Padding Applied to <tr:panelFormLayout> Component on Various Platforms

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

4.3.5 Panel Accordion

Defining the value of the `styleClass` component as `af_m_panelBase` within the `<tr:panelGroupLayout>` component applies padding to its `<tr:panelAccordion>` component, as shown in Figure 4–10.

Figure 4–10 Padding Applied to the Panel Accordion on Apple iPhone



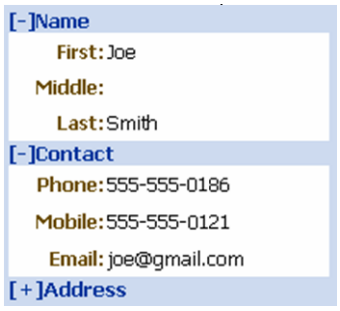
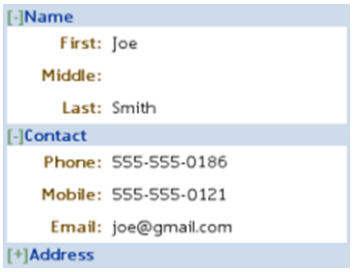
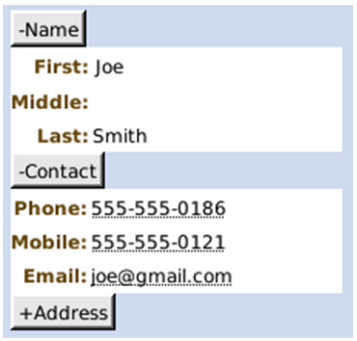
As illustrated in Example 4–11, you do not need to add the `styleClass` attribute to the `<tr:panelAccordion>` component.

Example 4–11 Applying Padding to the <tr:panelAccordion> Component

```
<tr:panelGroupLayout styleClass="af_m_panelBase">
        <tr:panelAccordion discloseMany="true">
            <tr:showDetailItem text="Name" disclosed="true">
                <tr:panelFormLayout fieldWidth="70%" labelWidth="30%">
                    ...
                </tr:panelFormLayout>
            </tr:showDetailItem>
            <tr:showDetailItem text="Contact" disclosed="true">
                <tr:panelFormLayout fieldWidth="70%" labelWidth="30%">
                    ...
                </tr:panelFormLayout>
            </tr:showDetailItem>
            <tr:showDetailItem text="Address">
                <tr:panelFormLayout fieldWidth="70%" labelWidth="30%">
                   ...
                </tr:panelFormLayout>
            </tr:showDetailItem>
        </tr:panelAccordion>
</tr:panelGroupLayout>
```

Example 4–10 shows examples of padding in the `<tr:panelAccordion>` component on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

Table 4–10 <tr:panelAccordion> on Various Platforms

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

5 Supporting Basic HTML Mobile Browsers

This chapter describes ADF Mobile browser's support for basic HTML mobile browsers.

This chapter includes the following sections:

- Section 5.1, "About Basic HTML Mobile Browser Support"
- Section 5.2, "Developing Applications for Basic HTML Mobile Browsers"
- Section 5.3, "Styling Basic HTML Mobile Browsers"

5.1 About Basic HTML Mobile Browser Support

Because basic HTML browsers for mobile devices do not support JavaScript, they are less robust than the other browsers supported by ADF Mobile browser, such as those for BlackBerry smartphones or the Apple iPhone. Aside from the browsers listed in Section 1.2, "Supported Mobile Browsers," ADF Mobile browser considers most common browsers as basic HTML mobile browsers. (ADF Mobile browser may not recognize certain mobile browsers, however.)

5.1.1 Requirements for Basic HTML Mobile Browser Support

The minimum requirement for ADF Mobile browser's support is XHTML Basic or the XHTML Mobile Profile that includes WAP2.x browsers.

> **Note:** ADF Mobile browser does not support the WAP1.x browsers that cannot support either the XHTML Basic or the XHTML Mobile Profile.

5.2 Developing Applications for Basic HTML Mobile Browsers

Because the ADF Mobile browser serves pages to mobile browsers that are appropriate to a browser's capabilities or limitations, you do not have to create user interfaces that are specific to basic HTML mobile browsers. However, the absence of JavaScript support by these browsers limits the functionality of certain HTML elements.

- Basic HTML mobile browsers do not support the `autosubmit` attribute. Add a submit button to the form only if the form submission responds to a component's autosubmit feature. For composite components with built-in autosubmit features, ADF Mobile browser adds a submit button to enable users to submit the form.
- Basic HTML mobile browsers do not support form-submitting links. All submitting elements are rendered as buttons. Basic HTML mobile browsers do not

support the child components of such input elements. As a consequence, the child components of the `tr:commandLink` component cannot render in a basic HTML mobile browser. For more information on `tr:commandLink`, see Section 3.5.2, "Creating Links."

5.3 Styling Basic HTML Mobile Browsers

ADF Mobile browser provides basic CSS support for basic HTML mobile browsers. Although most of these browsers support CSS, ADF Mobile browser applications can even run on the browsers that do not support CSS. In these cases, however, the user interface may be difficult to use. As a precaution, you should test the ADF Mobile browser application on as many browsers as possible.

6

Design Guidelines for BlackBerry 4.2 to 4.5

This chapter describes how to accommodate the behavior of BlackBerry browsers 4.2 to 4.5.

This chapter includes the following sections:

6.1 About BlackBerry Browser Display Behavior

The BlackBerry browser behaves differently than many other browsers in that it does not display pages using horizontal scrolling. Instead, it fits a page to the width of the screen. This chapter presents guidelines to help you format pages to display properly on BlackBerry smartphones.

6.2 Formatting Tables to Prevent Wrapping

Because browsers wrap long words between fields, avoid long words on lines that contain multiple fields when formatting tables.

Note: Within this chapter, a word refers to a series of characters. In this context, a word does not include white space.

Because the default mode of the BlackBerry browser limits the browser width to that of the physical screen, any field that does not fit within a line is displayed on the next line. If the intent of an application is to display multiple elements in one line, then you must ensure that the total width of the fields are within the width of the browser. Like other browsers, the BlackBerry browser wraps multiple lines when needed. The column width cannot be reduced beyond the size of the longest word in the field.

6.2.1 How to Prevent Fields from Wrapping in Tables

To prevent fields from wrapping, ensure that the total of the size attribute values in a table's row satisfies the following formula when all of the fields in a row are input fields.

```
3*Number of columns + the Sum of the size attributes in all columns <=X, when X=48
```

In general, field sizes in table columns should satisfy the following formula:

```
3 * Number of Columns +
Sum of size attributes in all input field columns +
Sum of number of characters in longest words in all output field columns <=X, when X=48
```

If the fields still wraps, decrease the value of `X` until it fits.

6.3 Formatting Label and Message Panels

To preserve the intended programming flexibility, `nowrap` attributes are supported and inserted when they are explicitly programmed for the Trinidad component. You may encounter problems if you add `nowrap` to a component definition when you program pages.

6.4 Formatting Column Width

When formatting columns, set the percentage width specification for both the label and the field in the `tr:panelFormLayout` component so that the total width is at 100%.

6.5 What You May Need to Know About Display Variations on BlackBerry Smartphones

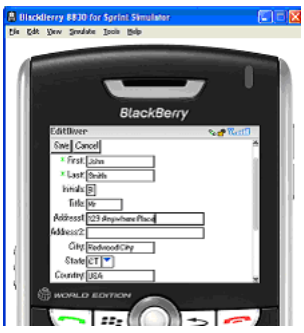
This section describes how the same application can display differently on different devices. This section includes the following topics:

- Changing the Minimum Font Size
- Form Factor Variations

6.5.1 Changing the Minimum Font Size

Changing the minimum font size through user preferences affects the formatting ability of the ADF Mobile browser renderer. For example, input fields and their corresponding labels align properly when the font is set to its default size of 6 pt., as shown in Figure 6–1.

Figure 6–1 Application Display Using the Default Font Size of 6 pt.



However, increasing the font size to 10 pt. disrupts the display by shifting the input fields beneath their corresponding labels. As a result, the page is difficult to read.

Figure 6–2 shows a page that is too large for the display screen.

Figure 6–2 Increasing the Font Size

6.5.2 Form Factor Variations

Differing screen sizes can affect display. Even if the font size is at the default of 6 pt. (illustrated in Figure 6–1), the same application appears differently on different devices. In Figure 6–3, the input fields barely fit the device's screen, even though they are easily accommodated on other devices running the same application as shown in Figure 6–1.

Figure 6–3 Difficulty Displaying Input Fields and Labels with Font Size at 6 pt.

In addition, input fields may display properly on the screen of one device, but may appear crowded on the screen of another type of device.

Figure 6–4 shows an application whose table cells are not wide enough to accommodate the text, causing it to wrap.

Figure 6–4 Wrapping Text

7

Narrow Screen Support and User Agent Details Support

This chapter describes how the Trinidad infrastructure determines narrow screen support and how it uses EL expressions to expose user agent details.

This chapter includes the following sections:

- Section 7.1, "Determining Narrow Screen Support"
- Section 7.2, "Determining User Agent Capabilities Using EL Expressions"

7.1 Determining Narrow Screen Support

Mobile devices have a wide range of screen widths. As a result, the UI components of a web application may render properly on a device with a screen width measuring 240 pixels, but not align correctly when the application runs on a device that has a screen width of only 100 pixels. In such a situation, Trinidad optimizes its rendering for narrow-screen devices. Trinidad considers any device with a screen width of less than 240 pixels as a narrow screen and optimizes the rendering for the following components accordingly:

- `tr:breadcrumbs`
- `tr:inputText`
- `tr:navigationPane`
- `tr:panelFormLayout`
- `tr:panelLabelAndMessage`
- `tr:panelRadio`
- `tr:processChoiceBar`
- `tr:selectRangeChoiceBar`

7.1.1 How Trinidad Determines Narrow-Screen Optimization

Because Trinidad only considers a device with a screen width that measures less than 240 pixels as a narrow screen, it does not consider iPhones (Safari browsers) or BlackBerry smartphones (BlackBerry browsers), which usually have screens that are greater than 240 pixels, as such. For a Windows Mobile browser, Trinidad determines the screen width from the UA-pixels request header and only applies narrow screen optimization if the screen-width is less than 240 pixels. For all other user agents, however, Trinidad optimizes its rendering for a narrow screen device.

7.2 Determining User Agent Capabilities Using EL Expressions

Trinidad exposes a requesting user agent's details to developers using the EL expression, `#{requestContext.agent}`, which returns an `agent` object that describes the requesting user agent. By adding the detail name or capability name properties to this expression, you enable Trinidad to return details that include the user agent's name, its version and platform, the version of the platform, the model (which is applicable only to BlackBerry), and the browser's support for JavaScript and PPR (Partial Page Rendering). For information on exposing user agent details, see Section 7.2.1, "How To Determine User Agent Details." For information on determining browser capabilities, see Section 7.2.2, "How to Determine Browser Capabilities."

7.2.1 How To Determine User Agent Details

When Trinidad receives a request, it parses user agent strings for a variety of user agent details (listed in Table 7–1) that include type, the name and version of the agent, and the agent's platform name and platform version. Trinidad uses the EL expression `#{requestContext.agent.<detail-name>}` to expose these details to developers. For example, to enable developers to retrieve the category appropriate to the user agent type (that is, desktop for a desktop browser or PDA for mobile browsers), Trinidad uses the `type` detail in the EL expression as follows:

```
#{requestContext.agent.type}
```

> **Note:** Trinidad may return a null value for such details as `PlatformName`, `PlatformVersion` if it cannot parse them from the user agent string.

Table 7–1 Browser Details Exposed through EL Expressions

Detail Name	Description
`type`	Identifies a user agent type. For desktop and mobile browsers, the values are *desktop* and *PDA*, respectively. Because Safari provides all desktop browser features when it runs on a mobile device, the agent object exposes this detail as a desktop type.
`agentName`	The name of the agent
`agentVersion`	The version of the agent
`platformName`	The platform on which the agent runs
`platformVersion`	The version of the platform on which the agent runs
`hardwareMakeModel`	The model of the mobile device

7.2.2 How to Determine Browser Capabilities

Trinidad sends its response to a user agent's request based on capabilities it assigns to a user agent. These capabilities include a user agent's support for JavaScript and PPR. Some of these capabilities (listed in Table 7–2) are exposed to developers through the EL expression `#{requestContext.agent.capabilities}`.

Use the EL expression `#{requestContext.agent.capabilities.<capability-name>}` to determine the specific capability assigned to a user agent by Trinidad. For example, to determine whether Trinidad assigns JavaScript capability to a user agent, use the following EL expression:

```
# {requestContext.agent.capabilities.scriptingSpeed!='none'}
```

Table 7–2 Browser Capabilities Exposed through EL Expressions

Capability Name	Detail
`narrowScreen`	Indicates whether Trinidad optimizes is rendering for a narrow-screen device. It returns `true` (a boolean type) if Trinidad optimizes its rendering for a narrow-screen device.
`scriptingSpeed`	Indicates JavaScript support for a user agent. Returns `none` (a String type) if the user agent does not support JavaScript.
`partialRendering`	Indicates PPR support for a user agent. Returns `true` (a boolean type) if the browser supports PPR.

8

Extending ADF Mobile Browser Applications

This chapter describes how to add e-mail, telephony, and Google Maps functionality to ADF Mobile browser applications.

This chapter includes the following sections:

- Section 8.1, "Introduction to Extending Applications for E-Mail, Telephony, and Google Maps"
- Section 8.2, "Integrating an E-Mail Client"
- Section 8.3, "Integrating Telephony"
- Section 8.4, "Integrating Google Maps"
- Section 8.5, "What You May Need to Know About Page Display Dimensions"

8.1 Introduction to Extending Applications for E-Mail, Telephony, and Google Maps

In addition to using the style sheets described in Chapter 4, "Skinning", you can further tailor an ADF Mobile browser application by using the `tr:goButton` and `tr:goLink` components to integrate links to phone numbers, e-mail addresses, and Google Maps.

8.2 Integrating an E-Mail Client

To invoke an e-mail application from a web application:

1. Use either the `tr:goButton` or the `tr:goLink` components.
2. Prepend the `mailto:` protocol in an HTML link.
3. Set the `destination` property to the HTML link (represented as the Expression Language statement `#{sessionScope.empDetails.Email}` in Example 8–1).

Example 8–1 Integrating the iPhone E-Mail Client Using the mailto: Protocol

```
<tr:goLink styleClass="messageText"
           text="#{sessionScope.empDetails.Email}"
           destination="mailto:#{sessionScope.empDetails.Email}"/>
```

8.2.1 Adding Mail Properties

The `mailto:` protocol enables you to add the mail properties that are listed in Table 8–1.

Table 8–1 Mail Properties

Property	Syntax
Multiple Recipients	A comma (,) separating each e-mail address
Message Subject	`subject =<subject text>`
cc Recipients	`cc=<name@address.com>`
bcc Recipients	`bcc=<name@address.com>`
Message Text	`body=<Message Text>`

To specify these properties, append the e-mail address with question mark (`?`) as illustrated by `#{sessionScope.empDetails.Email}?` in Example 8–2 and then add the properties, separating each with an ampersand (`&`).

Example 8–2 Adding E-Mail Properties

```
<tr:goLink styleClass="messageText"
           text="#{sessionScope.empDetails.Email}"
           destination="mailto:#{sessionScope.empDetails.Email}?subject=hello
                                                              &cc=myboss@oracle.com
                                                              &bcc=me@oracle.com
                                                              &body=good morning!"/>
```

8.3 Integrating Telephony

To invoke a call dialog box for a phone number displayed in the application, prepend the phone number with the `tel:` protocol in an HTML link.

> **Note:** The phone number must support the portion of the RFC 2806 protocol (`http://www.ietf.org/rfc/rfc2806.txt`) which enables you to add pauses or dial extensions after a user dials the primary phone number. Because Apple does not specify which portions of RFC 2086 that it supports, you must test each portion.

In Example 8–3, the EL expression, `#{sessionScope.empDetails.PhoneNumber}` represents the phone number.

Example 8–3 Enabling the Call Dialog Box

```
<tr:goLink styleClass="messageText"
           text="#{sessionScope.empDetails.PhoneNumber}"
           destination="tel:#{ sessionScope.empDetails.PhoneNumber}"/>
```

8.4 Integrating Google Maps

To create a link that displays a map that shows the data available in the application, specifying the `destination` property of the `tr:goLink` component as follows:

1. Define `destination=` as the URL of Google Maps. (`destination=http://maps.google.com/maps`, as illustrated in Example 8–4.)
2. To search for a location, append the Google Maps URL with `?q=`.

3. Define `q=` using the address string of the target location. This value can be a full street address, a city, landmark, or any item that Google Maps can search and locate. If multiple items are found, Google Maps drops multiple pins automatically.

> **Note:** The address string must be well formatted, including commas between words. Also, replace spaces with plus sign (+) characters.

Example 8–4 illustrates how to define the `tr:goLink` component to invoke a Google Maps application and then drop a pin on 200 Oracle Parkway.

Example 8–4 Specifying Locations in Google Maps

```
<tr:goLink styleClass="messageAddrText"
           text="200 Oracle Parkway, Redwood City, CA, USA"
           destination="http://maps.google.com/maps?q=200+Oracle+Parkway,+Redwood+City,+CA,+USA"/>
```

Example 8–5 illustrates specifying a location using an address represented by EL expressions.

Example 8–5 Specifying Locations in Google Maps Using EL Expressions

```
<tr:goLink styleClass="messageAddrText"
           text="#{sessionScope.empDetails.StreetAddress},
                 #{sessionScope.empDetails.City},
                 #{sessionScope.empDetails.StateProvince},
                 #{sessionScope.empDetails.CountryName}"
destination=" http://maps.google.com/maps?q=#{sessionScope.empDetails.StreetAddress},
                                           +#{sessionScope.empDetails.City},
                                           +#{sessionScope.empDetails.StateProvince},
                                           +#{sessionScope.empDetails.CountryName}"/>
```

The address string, such as the one in Example 8–4, must be have plus sign (+) characters rather than spaces.

8.4.1 Programming Driving Directions

Google Maps also supports driving directions. Modify the string following the question mark (`?`) in the Google Maps URL with the starting and destination addresses (`saddr=<starting address>&daddr=<destination address>`). Using this format, the directions from Oracle headquarters at 200 Oracle Parkway in Redwood City to 1 Telegraph Hill in San Francisco, California are as follows:

```
http://maps.google.com/maps?
                        saddr=200+Oracle+Parkway,+Redwood+City,+CA,+USA
                        &daddr=1+Telegraph+Hill,+San+Francisco,+CA,+USA
```

> **Note:** Apple and Google have not yet published other APIs.

8.4.2 Supporting Google Maps on iPhone

iPhone Safari supports both Google Maps and YouTube applications in that it automatically intercepts certain URL calls and invokes a native application rather than opening the URL using the target web site. For example, when a user clicks an HTML link to Google Maps (`http://maps.google.com`), Safari invokes a native Google

Maps application rather than navigating to the Google Maps web site. Because the native Google maps application accepts some URL parameters supported by `maps.google.com`, users can specify a location and drop a pin.

8.5 What You May Need to Know About Page Display Dimensions

To control the correct zoom ratio, add a `viewport` meta tag in the header of a page. The `viewport` is a device-specific meta tag used to ensure that a page displays at the correct scale. Example 8–6, illustrates setting the viewports for both iPhones and BlackBerry smartphones. For more information on using the `viewport` specification, see `http://developer.apple.com/`.

Example 8–6 Setting Viewports

```
<trh:head title="Online Banking Demo">
    <meta http-equiv="Content-Type"
                content="text/html; charset=windows-1252"/>
   <f:verbatim rendered="#{requestContext.agent.skinFamilyType eq 'blackberry'}">
   <meta name="viewport"
         content="width=device-width;
                  height=device-height;
                  initial-scale=1.0;
                  maximum-scale=1.0;
                  user-scalable=0;"/>

   </f:verbatim>
   <f:verbatim rendered="#{requestContext.agent.skinFamilyType eq 'iPhonewebkit'}">
   <meta name="viewport"
         content="width=device-width;
         initial-scale=1.0;
         maximum-scale=1.0;
         user-scalable=0;"/>
   </f:verbatim>
</trh:head>
```

Note: Versions 4.6 and later of BlackBerry support the `HandheldFriendly` meta tag which is similar to `viewport`. Include the following line in the header to enable the page to scale appropriately:

```
<meta name="HandheldFriendly" content="True">
```

8.5.1 Setting the Viewports for iPhone

While some mobile browser applications may display correctly on desktop Safari browsers, they may not scale not correctly for the smaller screen of the iPhone and may appear too large. As a result, the iPhone shrinks pages until they are too small to read. The following line from Example 8–6, illustrates how to set the iPhone `viewport` specifications in the `<head>` element to ensure that applications display properly on iPhones.

```
<f:verbatim rendered="#{requestContext.agent.skinFamilyType eq 'iPhonewebkit'}">
   <meta name="viewport"
         content="width=device-width;
         initial-scale=1.0;
         maximum-scale=1.0;
         user-scalable=0;"/>
```

```
</f:verbatim>
```