BEA WebLogic Portal®

Integrating Search

Version: 10.2
Revised: February 2008

Contents

Part II. Development

3. Using Search when Developing Your Portal

4. Multi-language Searching and Indexing

5. Metadata Searching

Part III. Staging

6. Staging Search Capabilities

Part IV. Production

7. Using Search in Production

CHAPTER 1

Introduction

BEA WebLogic Portal® provides a number of advanced search capabilities. You can implement WebLogic Portal's search engine to integrate with disparate content management systems, relational databases such as CRM systems, and external web sites. These sources of information can be exposed to your portal users via pre-packaged portlets, and developers can also author new portlets and implement business logic to search content sources.

This chapter includes the following sections:

Introducing Search

Using the search components included with WebLogic Portal, you can enable your portal to incorporate data and information from multiple sources such as databases, other web sites, and file systems.

WebLogic Portal search components work together to aggregate, categorize, and personalize content from different resources in your enterprise and across the internet. For example, when

you incorporate search within a knowledge base portal, portal users can search across multiple support databases and view the results.

Once configured, WebLogic Portal's search tools continuously index content within the sources you indicate and maintain a query-able source for your portal users. You can surface this content through out-of-the-box portlets or write your own portlets to customize search capabilities to suit your needs.

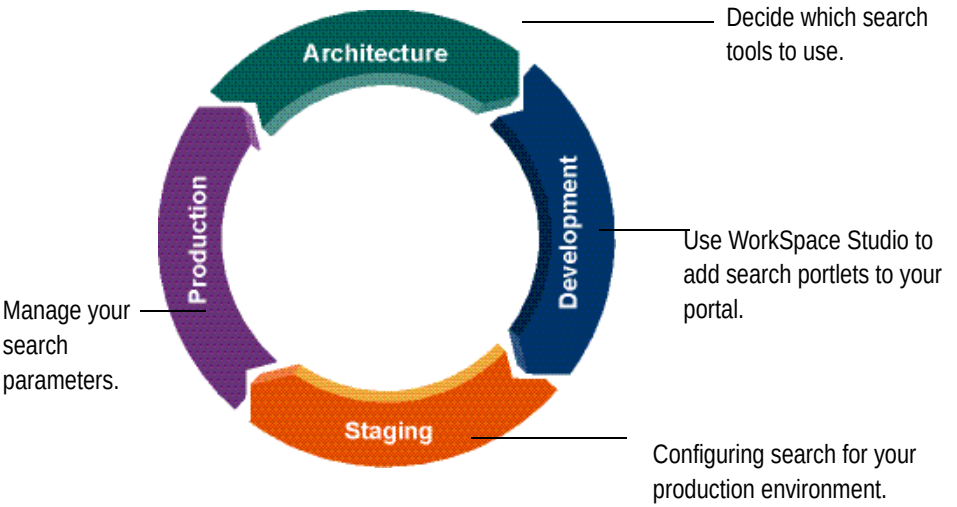
WebLogic Portal utilizes Autonomy® search for its search functionality. See "Getting Started" on page 1-3 for an introduction to the Autonomy components.

Search in the Portal Life Cycle

The tasks in this guide are organized according to the portal life cycle. For more information about the portal life cycle, see the *WebLogic Portal Overview Guide*. The portal life cycle contains four phases: architecture, development, staging, and production.

Figure 1-1 shows how search fits into the portal life cycle.

Figure 1-1 How Search Fits into the Four Phases of the Life Cycle



Architecture Phase

During the architecture phase, you determine what enterprise content you want to make available for your portal and who within your portal environment will be able to search this information.

The following chapter describes tasks within the architecture phase:

- Chapter 2, "Architectural Considerations for Search"

Development Phase

During the development phase, you add search portlets to your portal, use the APIs to retrieve content, and optionally, write portlets to surface search features for your portal users.

The following chapter describes tasks within the development phase:

- Chapter 3, "Using Search when Developing Your Portal"

Staging Phase

The staging phase is when you prepare your production environment. During this phase, you reconfigure your search configuration to match your deployment configuration and enable tools to configure search when running in a production environment.

The following chapter describes tasks within the staging phase:

- Chapter 6, "Staging Search Capabilities"

Production Phase

After you deploy your application and are running in a production environment, you can adjust how your portal searches for content, including caches and search frequencies.

The following chapter describes tasks within the production phase:

- Chapter 7, "Using Search in Production"

Getting Started

WebLogic Portal utilizes Autonomy® search for its search functionality. Search features provided by Autonomy include the following:

- Natural language queries can be input to the IDOL Server and will be processed based on the words contained in the queries.

- Basic XML Search - basic features to index and search XML documents Natural Language Support.
- Relevance Ranking. Each retrieval operation produces a relevancy score which can be used in the search results interface
- Document Similarity Search – “More Like This” feature using keyword similarity between documents.
- Proximity Controls. Basic Boolean, proximity, and field searches are provided.

Table 1-1 lists the components of Autonomy search tools and what each provides.

Table 1-1 Autonomy Search Components Used with WebLogic Portal

Autonomy Component	What It Does:
Autonomy IDOL Server	The Intelligent Data Operating Layer (IDOL) server is responsible for indexing content as well as processing content queries made from your portal. For more information about the Autonomy IDOL Server, see the Autonomy IDOL Server documentation.
Autonomy DiSH	The Distributed Service Handler – DiSH, provides the crucial maintenance, administration, control and monitoring functionality of the Intelligent Data Operating Layer (IDOL). DiSH delivers a unified way to communicate with all Autonomy services from a centralized location. DiSH can be managed with the Autonomy Service Dashboard. For more information about the Autonomy DiSH, see the Autonomy DiSH documentation.
Autonomy Service Dashboard	The Autonomy Service Dashboard is an stand-alone front-end web application that communicates with one or more Autonomy Distributed Service Handler (DiSH) modules that provide the back-end process for monitoring and controlling all the Autonomy child services, such as fetches. For more information about the Autonomy Service Dashboard, see the Autonomy DiSH documentation.

Table 1-1 Autonomy Search Components Used with WebLogic Portal

Autonomy Component	What It Does:
Autonomy HTTP Fetch	HTTP Fetch allows documents from internet or intranet sites to be aggregated from remote servers and indexed into Autonomy IDOL server. For more information about the HTTPFetch, see the Autonomy HTTP Fetch documentation.
Autonomy ODBC Fetch	ODBC Fetch is an Autonomy connector that automatically retrieves data that is stored in ODBC data sources, imports it into IDX file format and indexes it into Autonomy IDOL server. For more information about the ODBC Fetch, see the Autonomy ODBC Fetch documentation.
Autonomy File System Fetch	File System Fetch analyzes file systems on local or network machines (including Novell, NT, UNIX file systems and Samba-mounted servers) for new documents to aggregate into the Autonomy IDOL server. It keeps the IDOL server's view of the file system in sync so that files deleted are automatically removed from IDOL server, and modifications to files are reflected automatically. **Note:** If a file name contains any Japanese characters using Shift JIS encoding, Autonomy will not index them. This means that if a file with Shift JIS characters in the file name are placed in a directory to be indexed by the File System Fetch utility, it will not be indexed by Autonomy and not be returned within the search results provided by the Enterprise Search portlet. Therefore, you must rename any files that contain Shift JIS characters to a name without any Shift JIS characters. For more information about the File System Fetch, see the Autonomy File System Fetch documentation.
Autonomy Portlets	Autonomy portlets are designed to integrate search functionality with your portal. For more information about the Autonomy portlets, see "Installing Autonomy Portlets" on page 3-3. For additional documentation on the Autonomy portlets, see *Autonomy Portlets for WebLogic Guide* or the *Autonomy Portlets User Guide*.

Locating the Autonomy Product

Autonomy is bundled with the WebLogic Portal and WebLogic Platform installers. The files are located in the `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10` directory.

Licensing Autonomy Modules

The license model for Autonomy is one portal to one Autonomy IDOL Server. You can install one instance each on production, development, and a failover instance within your portal deployment. Licensing is paper-based, and all development instances ship with a full production version of Autonomy. A production instance of Autonomy is included with the WebLogic Portal installation.

Note: The evaluation license included with Autonomy allows a document limit of 10,000. On purchase of WebLogic Portal, you will receive a full Autonomy production license that provides a 500,000-document limit. For information about updating your Autonomy license, see "Updating the Autonomy License" on page 6-4.

Determining the Number of CPUs for Your Search Needs

The number of CPUs that you need for a production instance varies with the number and type of documents you are exposing, as well as the way they are exposed (for example, automated searching, user driven, and so on).

A single instance of one CPU can potentially support tens of thousands of users and millions of documents. Contact your BEA or Autonomy sales representative for additional licenses, if needed.

Choosing an Operating System

During development mode, Autonomy services are automatically started for the operating system of the host computer, which allows developers to use Autonomy during portal development.

However, when you deploy your portal and install Autonomy within your portal environment, you will need to install the operating system-specific version of Autonomy on your server on which you run the Autonomy services. For more information about installing and deploying the Autonomy services, see Chapter 6, "Staging Search Capabilities."

Note: Autonomy binary executable files are named with a `.exe` extension (Windows style) for all operating systems.

System Requirements

When configuring Autonomy search for your portal application, please note Autonomy's system requirements, see the Autonomy documentation.

Upgrading and Getting Support

BEA provides front-line support for Autonomy components—contact the BEA Support Department. BEA Support will contact Autonomy for additional back-line support as needed. Additional connectors (fetches) and tools are available from Autonomy as well.

Disabling Search/ Search Indexing

To disable search indexing:

1. Start your portal domain.
2. Start the WebLogic Portal Administration Console.
3. Select **Content > Content Management** from the navigation menu at the top of the console.
4. Select **Manage | Repositories**.
5. In the resource tree, click the repository for which you want to disable search indexing.
6. In the Summary tab, click **Advanced** to view the Edit Advanced Properties for Repository dialog.
7. In the Edit Advanced Properties for Repository dialog, clear the **Search Enabled**/**Search Indexing Enabled** check boxes.
8. When finished making changes, click **Save**.

Autonomy Documentation

Review this guide to become familiar with how WebLogic Portal uses Autonomy search. For additional information, see the Autonomy documentation.

The Autonomy documentation is included in your WebLogic Portal installation directory at `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

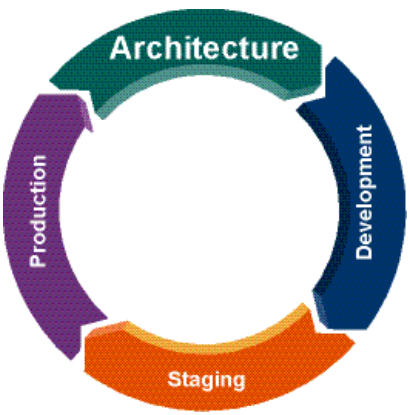
Part I Architecture

Part I contains instructions for tasks you should accomplish in the architecture phase. When you finish the architecture phase, you can proceed to the development phase, and then on to the other phases.

Part I includes the following chapter:

- Chapter 2, "Architectural Considerations for Search"

For a description of the architecture phase of the portal life cycle, see the *WebLogic Portal Overview*. The portal life cycle is shown in the following graphic:



CHAPTER 2

Architectural Considerations for Search

During the architecture phase, you determine what enterprise content you want to make available for your portal and who within your portal environment will be able to search this information.

This chapter includes the following sections:

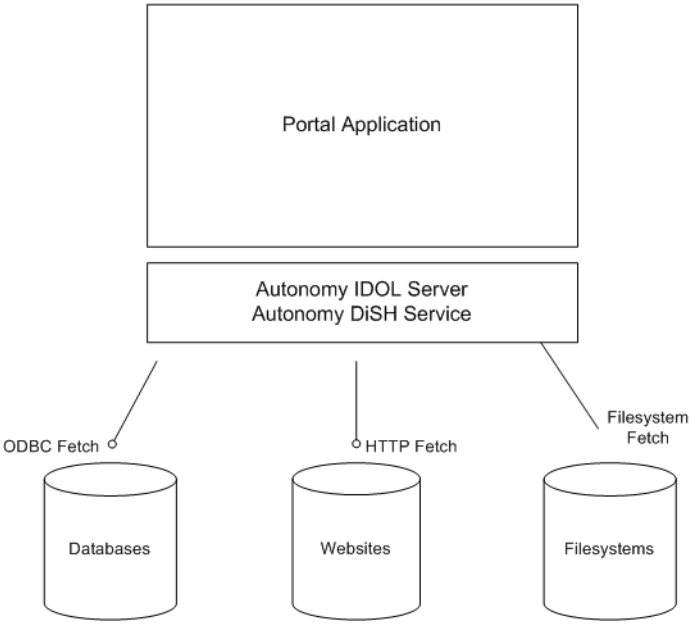
Understanding How Search is Implemented

WebLogic Portal uses Autonomy search components to implement search functions such as allowing portal users to search external web sites, integrated databases, and available file systems. To do this, you can incorporate search portlets that ship with WebLogic Portal or write your own. However you decide to implement search, the same tools will be used. You will need to install and configure these tools before integrating search within your portal.

The Autonomy IDOL server is used to manage the indexes created by the Autonomy fetches you use. You can use this indexed data in your portal application by using the portlets that come with WebLogic Portal or you can write your own portlets using the Autonomy API. For more information about using and developing portlets, see Chapter 3, "Using Search when Developing Your Portal."

Figure 2-1 shows a diagram of how Autonomy search tools integrate with WebLogic Portal.

Figure 2-1 Diagram of a WebLogic Portal Integration with Autonomy Search Tools



Deciding What Information to Index

You can allow portal users to search a variety of information sources from your portal. Before developing and deploying your portal, you should decide what information sources you want to index. Consult the respective Autonomy documentation for more details. For example, if you want to include web sites, see the *Autonomy HTTP Fetch Administrator's Guide*.

Maximum Amount of Indexed Content

The Autonomy license that comes with WebLogic Portal allows you to index 500,000 pieces of content. If you need to index more content than 500,000 items, you will need to obtain a different license from Autonomy.

Architectural Recommendations

BEA recommends using a separate machine for your Autonomy IDOL Server to ensure the most processing power to service indexing and query requests from your portal clients. You can also

install each Autonomy engine (such as HTTP Fetch, File System Fetch and IDOL Server) on a separate server if you find you need additional resources.

Autonomy recommends a dual-processor server for hosting the IDOL Server and the DiSH Handler. For complete system requirements, see the Autonomy documentation.

Choosing an Operating System

When you install WebLogic Portal, the Autonomy engine for the target operating system is included. If you need a version of Autonomy for a different operating system than the operating system on which you installed WebLogic Portal, you will need to download and install WebLogic Portal onto the operating system for which you need Autonomy. You can then retrieve the respective operating system files for Autonomy.

Note: Autonomy binary executable files are named with a `.exe` extension (Windows style) for all operating systems.

For example, if you downloaded and installed WebLogic Portal on a Windows server, the Windows version of Autonomy was included in the download. If you want to install Autonomy on a Linux server, you need to download and install the Linux version of WebLogic Portal in order to have the correct version of Autonomy for a Linux machine.

Note: You can also choose to install Autonomy on an operating system different from what is supported for WebLogic Portal, see "Configuring the BEA Content Management Fetch When Using a Non-WebLogic Portal Supported Operating System" on page 6-14

For more information about installing and deploying Autonomy services, see Chapter 6, "Staging Search Capabilities."

Part II Development

During the development phase, portal developers can add search features to your portal by using provided portlets or creating their own.

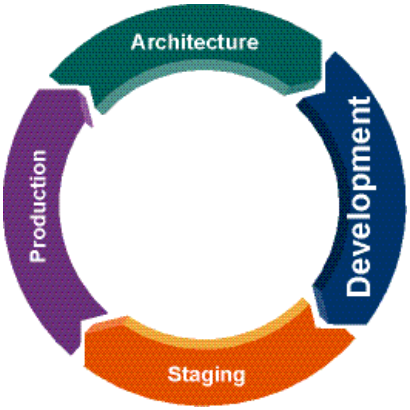
Part II includes the following chapters:

- Chapter 3, “Using Search when Developing Your Portal”
- Chapter 4, “Multi-language Searching and Indexing”

When you finish the development phase you can proceed to the staging phase. Consider setting up a common development environment for the development phase and the staging phase. You might move iteratively between these two phases, developing and then testing what you created.

If you moved on to the production phase and then go back to make changes that affect the development phase, you must redeploy your portal application in order to view your changes. The BEA Propagation Utility performs the redeployment; see the *WebLogic Portal Production Operations User Guide* for more information.

For a detailed description of the development phase of the portal life cycle, see the *WebLogic Portal Overview*. The portal life cycle is shown in the following graphic:


Architecture
Development
Staging
Production

CHAPTER 3

Using Search when Developing Your Portal

During the development phase, you can add Autonomy portlets to allow your portal users access to search functions. You can also build your own portlets or customize existing ones.

Note: The Autonomy documentation is included in your WebLogic Portal installation directory at `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

This chapter discusses the following topics:

- Preparing to Develop a Portal with Search
- Writing Autonomy-based Applications and Portlets
- Creating Search Portlets for BEA Content Repositories

Preparing to Develop a Portal with Search

Before you add and create search capabilities within your portal, you need to ensure your development environment has the necessary tools and the proper settings. Specifically, you need to add the Autonomy portlets to your web application library, and ensure that search features are optimized for your environment.

This section includes the following topics:

- Configuring Search Capabilities in Your Development Environment
- Installing Autonomy Portlets

Configuring Search Capabilities in Your Development Environment

Running Autonomy's search services and BEA's full-text search capabilities while developing your portal could use unnecessary memory resources. If you do not need to run search capabilities continuously in your development environment, it is recommended that configure a staging environment for your Autonomy search features, see "Staging Search Capabilities" on page 6-1.

Optionally, you could also disable search within your development environment. Disabling search involves disabling BEA's full-text search as well as turning off Autonomy's search services.

Disabling BEA's Full-Text Search

To disable BEA's full-text search,

1. Start your portal domain.
2. Start the WebLogic Portal Administration Console.
3. Select **Content > Content Management** from the navigation menu at the top of the console.
4. Select **Manage | Repositories**.
5. In the resource tree, click the repository for which you want to disable full-text search.
6. In the Advanced section, click **Advanced** to view the Edit Advanced Properties for Repository dialog.
7. In the Edit Advanced Properties for Repository dialog, edit the properties listed in Table 3-1.

Table 3-1 Advanced Repository Properties

Advanced Property	What it does:
Search Indexing Enabled	Allows content to be indexed for portal search. This enables portal developers to use full-text content search in any portlets that they develop.
Full-Text Search Enabled	Enables users to search the repository using the full-text of the content.

8. When finished making changes, click **Save**.

Your modifications display in the Advanced section of the Summary page.

Note: After you make any changes to repository properties, Portal Administration Console users must log out and log back in to view the changes.

Disabling Autonomy's Services

You can disable Autonomy's search services. When you disable Autonomy's services, the following features will not be available:

- Full Text Search on a BEA repository
- Full Text Search within the WebLogic Portal GroupSpace application
- Enterprise Search within the WebLogic Portal GroupSpace application
- Autonomy portlets
- Any application/portlet which uses the Autonomy API

To disable Autonomy's services:

1. Stop your portal domain.
2. Set the following environment variable on your portal domain server:
   `CONTENT_SEARCH_OPTION=none`
3. Restart your portal domain.

Installing Autonomy Portlets

Several Autonomy portlets ship with WebLogic Portal. In order to use these portlets within your portal application, you must first install and configure them.

For additional information about configuring these portlets, see the *Autonomy Portlets for WebLogic Administration Guide* or the *Autonomy Portlets User Guide*. These documents are located in `<BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

The portlets are listed in Table 3-2.

Table 3-2 Autonomy Portlets

Portlet	What It Provides:
Autonomy2DMap	The 2D Cluster Map is used to identify conceptual similarities and differences between clusters. Also based on JSP, the landscape is generated from the inter-relationships between clusters and the documents contained within those clusters. Designed to provide a single overview of the clusters contained within the data, clusters that are close together correlate to higher degrees of similarity, whilst dissimilar clusters are situated further apart. By scrolling over the ClusterMap automatic titles are generated and assigned to every cluster. By clicking on the cluster, the results and respective information can be viewed.
Agent	The Agent portlet allows individual users to create their own personalized information channels, either from Natural Language, legacy Keywords, Boolean expressions, Parametric Searches or even simply by example. These agents then monitor all incoming information and can target and alert useful content on a continual basis, automatically. Users begin by customizing the Agent Portlet by setting up 'interest agents'. This is done by the user describing in plain natural language what it is they are interested in. The Interest Agent persistently identifies all relevant content and presents it in a concise personalized page, complete with URL links. As new information becomes available the agent will monitor new data submissions ensuring that the Portlet user is always provided with up-to-date information. Moreover, as user interests change users are also given the option to refine their interest by retraining the agents.
Breaking News	The Breaking News portlet identifies what's new in the information space. Taking the cluster analysis from a previous time period and comparing it to a current one allows automatic identification of new clusters that weren't previously present, allowing automation of 'breaking news' pages, alerting to new areas of information or new interest trends in subscriber groups.
Community	The Community portlet notifies individual users of any agents that people in the work community may have set up using the Agent portlet, which resemble their own personalized agents. This Portlet brings together the benefits of collaboration, reducing duplicated effort as well as identifying experts within the organization.

Table 3-2 Autonomy Portlets

Portlet	What It Provides:
Cluster	Cluster portlets provide a range of classification Portlets and visualization tools that can be added to the BEA environment, further enriching the portal experience. Autonomy's automatic clustering features identify areas of intense research, breaking news or emerging trends and market opportunities based on information found within the knowledge base. Autonomy's Cluster Portlets can take large sets of document data or user-profile information and automatically identify the main set of concepts/ themes inherent within the knowledge base. Furthermore, clustering can be used in identifying the 'gap' between the users interests and the data being provided to the users thereby allowing 'knowledge/ content gaps' to be eliminated through provision or aggregation of further content relevant to the community.
Expertise Locator	The Expertise Locator portlet allows users to find people who have been dealing with a specific subject by entering a brief natural language description of the subject. It returns all agents and profiles that match this description together with the names of the users who own the agents or profiles.
Hot News	The Hot News Cluster portlet can identify what is most popular or the main topics/clusters of information or interests found within the information assets or an organization. This allows the business to instantly receive a high-level view of the entire knowledge base providing a catalyst that enables informed decisions to be made faster.
Administration	The Administration portlet that enables you to administer and maintain all Autonomy Portlet settings from a central location.
Profile	The Profile portlet brings new documents to users attention based on each users individual interest and according to their profile. This Portlet creates a profile on each user based on the concepts of the documents that the user has been reading within the Autonomy Portlet suite (Agents, Retrieval, Clustering, Community and so on). Every time a user opens a new document within the Autonomy Portlet suite, the user's profile will be update based on the information read.

Table 3-2 Autonomy Portlets

Portlet	What It Provides:
Retrieval	Autonomy's Retrieval portlet provides a fully-automated and precise means of retrieving information. It allows content to be searched in any language and any format, wherever it is stored, and presented with hyperlinks to similar information, automatically and in real-time. Unlike ordinary searches that look for keywords the Autonomy Retrieval Portlet allows you to enter a natural language query. The Retrieval Portlet submits the natural language query to one or more databases that have been set up, in order to find documents that are related to your query.
Similar People	The Similar People portlet notifies a user or other people in the same organization of other users who have been using the same type of documents that you have been looking at. This feature helps users avoid spending time on searching for information that may already be available.
Spectrograph	The Spectrograph portlet displays the relationship between clusters on successive periods and sets of data. Clusters are presented as a JSP-based spectrograph, whereby the x-axis represents information over time (enabling users to visualize how clusters develop over a given time period), whilst the y-axis represents the range of concepts defined within the knowledge base. Moreover, the spectrograph is able to display hot and breaking news in the same instance. The importance of clusters over time can be seen through the change of color and width. The color/ intensity of the lines is an indication of the size of cluster. The brighter colors indicate what is popular, and the width of the lines is an indication of the quality of the cluster. Navigation features are identical to the 2D ClusterMap enabling users to browse clusters with a click of the mouse.

After installing the Autonomy portlets, you can use customize them or add them to your portal without customizing them. If installing the Autonomy portlets in a non-GroupSpace portal application, you should also configure an additional servlet, see "Enabling Autonomy Portlets in Non-GroupSpace Portal Applications" on page 3-8.

For more information about working with portlets, see the *WebLogic Portal Portlet Development Guide*.

To add these portlets:

Note: These instructions assume you have already created and deployed your portal application.

1. Locate the `AutonomyPortlets.zip` file in your WebLogic Portal installation directory. For example, `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/common/AutonomyPortlets.zip`
2. Extract the contents of the `AutonomyPortlets.zip` file to a temporary directory of your choosing. For example, `c:/temp/`.
3. Add the following to WorkSpace Studio by right-clicking the appropriate folder and then selecting **Paste**:
   a. Copy the `autonomyPiB.jar` file to *yourEAR*`/EARContent/APP-INF/lib` directory.
   b. Copy the portlet folder to your `webApp/webContent` directory. For example, `<your_projects>/w4WP_workspaces/<project_name>/<yourWebApp>/WebContent/`
   c. Copy the `portalInabox.css` file to the `<yourWebApp>/webContent/portlets` directory that you added in step b.
   d. Copy the `AutonomyPortletSettings.usr` file to the `userprofiles` subdirectory of your Datasync project directory. For example, `<yourDataSyncProject>/src/userprofiles/`.
   e. Copy the `*.properties` files in the temporary `WEB-INF/classes` directory to your web application's `WEB-INF/classes/` directory. If you don't have this directory, create it. For example, `<yourWebapp>/WebContent/WEB-INF/classes`
4. Move the `portlets.cfg` file to your domain directory, or if running in a cluster, to a shared directory outside the application.

   This insures that the directory location will be constant regardless of how the application is deployed.
5. Using a text editor, modify the `default-value` setting in `AutonomyPortletSettings.usr` to point to the `portlets.cfg` file.

   Note: You must use an absolute path to point to the `portlets.cfg` file.
6. When finished editing, save your changes.

7. Using a text editor, edit the `portlets.cfg` file and update the following values to point to your IDOL Server port, if changed: `UAPort`, `ClassPort`, and `DREPort`. Edit other settings if needed. The `AutonomyIDOLServer.cfg` is located in the `<BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/<os>/IDOLserver/IDOL` directory.

   For more information about configuring this file, see the *Autonomy Portlets for WebLogic Administration Guide* at `<BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

8. Disable JSP fragment validation:

   a. Right-click your web application project and select **Properties**.

   b. In the tree, select **Validation - AppXRay**.

   c. In Validation - AppXRay, deselect **Report Java compilation errors in JSP pages**, and then click **OK**.

   d. Click **Yes** when asked to rebuild the AppXRay database.

9. When finished editing, save changes. If using Autonomy portlets within a non-GroupSpace portal application, continue to the next section, "Enabling Autonomy Portlets in Non-GroupSpace Portal Applications".

Note: To view the portlets, you must add them to your portal application.

Enabling Autonomy Portlets in Non-GroupSpace Portal Applications

Note: If you are using Autonomy portlets within a GroupSpace portal application, disregard this section.

When operating in a non-GroupSpace portal application, Autonomy portlets require the DownloadServerFileServlet. You need to register this servlet in your web project by editing the the `web.xml` file for your respective web project.

To register this servlet, do the following in WorkSpace Studio:

1. From the Package Explorer, navigate to the respective web.xml file for your web project. For example, `<myWebProject>\WebContent\WEB-INF\web.xml`.

2. Right-click your `web.xml` file and select **Open With > XML Editor**.

3. Add the following sections to your `web.xml` file.

   a. Register the servlet:

```
<servlet>
<servlet-name>DownloadServerFileServlet</servlet-name>
<servlet-class>com.bea.apps.groupspace.util.DownloadServerFileServlet</servle
t-class>
</servlet>
```

b. Register the servlet mapping:

```
<servlet-mapping>
   <servlet-name>DownloadServerFileServlet</servlet-name>
   <url-pattern>/DownloadServerFileServlet/*</url-pattern>
</servlet-mapping>
```

c. Register the context parameters:

```
<context-param>

<param-name>com.bea.apps.groupspace.search.enterprise.IDOLServerHost</p
aram-name>
   <param-value>localhost</param-value>
</context-param>
<context-param>
<param-name>com.bea.apps.groupspace.search.enterprise.IDOL_aciPort</par
am-name>
   <param-value>9014</param-value>
</context-param>
```

4. To enable basic authentication for administrators (in lieu of a login portlet):

```
<security-constraint>
   <web-resource-collection>
      <web-resource-name>Site</web-resource-name>
         <url-pattern>/*</url-pattern>
         <http-method>GET</http-method>
         <http-method>POST</http-method>
      </web-resource-collection>
      <auth-constraint>
         <role-name>Admin</role-name>
      </auth-constraint>
</security-constraint>
<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>myrealm</realm-name>
```

```
</login-config>
<security-role>
   <role-name>Admin</role-name>
</security-role>
```

5. Save your `web.xml` file.
6. Re-deploy your application.

Writing Autonomy-based Applications and Portlets

You can create Autonomy-based applications and portlets using the Autonomy APIs. The Autonomy API is included in the `content-management-app-lib` shared library within WorkSpace Studio.

For Autonomy API documentation, review the Autonomy JavaDoc.

Note: Do not execute queries against any IDOL database which is prefixed with `WLP_CM_REPO` as these indexes contain information on the BEA content repositories in use for your portal. If you want to execute queries against BEA's content management repositories, you need to use the WebLogic Portal API, see the *WebLogic Portal JavaDoc*.

Creating Search Portlets for BEA Content Repositories

You can also create portlets that can be used to search WebLogic Portal's content management system. To do this, you use the WebLogic Portal Content Search API. You must also install and configure the BEACMRepoFetch in enable this search capability. WebLogic Portal ships with this custom fetch, see "Setting up BEA Content Management Search" on page 6-9.

Although you use Autonomy's APIs to create most search portlets, you must use WebLogic Portal Content Search API to create portlets that will search the WebLogic Portal content management system. See the *WebLogic Portal JavaDoc* for more details.

Table 3-3 Helpful Content Search APIs

Package/Class	What it does:
com.bea.content.expression.search	This package enables you to build search queries.
FullTextSearchFactory	This class enables you to incorporate full-text search on content within the Virtual Content Repository.

CHAPTER 4

Multi-language Searching and Indexing

WebLogic Portal provides several methods for configuring full-text search and indexing in multiple languages. Each method provides different capabilities. You need to decide on a per-repository basis which method is desirable. If you decided to change methods later, you also need to re-index your repository. Note that each document indexed can be associated with only one language.

The following sections describe each full-text search method and how to configure them:

- One Language per Autonomy Server
- One Language per Repository
- Mixing Languages Within a Repository
- Enterprise Search for Microsoft Word, Excel, and PowerPoint Files in Multibyte Languages

You need to decide on a per-repository basis which approach is desirable. You should also consult the Autonomy documentation, which is included in your WebLogic Portal installation directory at `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

One Language per Autonomy Server

The default configuration for an Autonomy server is one language and one encoding across all repositories using that server. When you use this configuration for multiple languages, you need separate Autonomy servers for each language. In this case you need to configure all indexed content and all full-text queries against that content to use the same `LanguageType` (language and encoding).

For example, you could have three repositories accessing a single Autonomy server. All three repositories must use the same `LanguageType`, such as FrenchUTF8, and all documents indexed in each repository would need to be in French. Additionally, all queries on all repositories would need to be in French language with UTF8 encoding. If you needed two languages, you would have to set up two Autonomy servers, two repositories, and manually configure the default language type in each server.

To set a default language type for a server, you edit the `DefaultLanguageType` in the `[LanguageTypes]` section in the server's configuration file (`AutonomyIDOLServer.cfg`). For more information about defining a global default language type, see the *IDOL Server Administration Guide* at `<BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

One Language per Repository

To mix multiple repositories, possibly with different languages, in the same Autonomy server, you need to specify the language and encoding for each repository. This means that all nodes in a repository and all queries must use the same language type and encoding. Both the language type and encoding are defined by the `LanguageType`. Some examples of language types are `frenchUTF8` (French language, UTF8 encoding), `frenchASCII`, and `russianCYRILLIC`. When you use a language type, such as `frenchUTF8`, all documents in the French-UTF8 repository must be in French and all queries in that repository must be in the French language with UTF8 encoding.

The supported language types are listed in `[LanguageTypes]` section in the server's configuration file (`AutonomyIDOLServer.cfg`), which is located in the `<BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/<os>/IDOLserver/IDOL` directory.

To use this approach, you need to add two properties to the repository configuration. For example, to use the French language with UTF8 encoding add:

- `fullTextSearchIndexLanguageType=frenchUTF8`
- `fullTextSearchQueryLanguageType=frenchUTF8`

Generally you set these properties to the same value. All queries need use the same `fullTextSearchQueryLanguageType` language and encoding.

For instructions on how to add a property to a repository, see "Adding Custom Properties" in Configuring BEA Repositories in the *Content Management Guide*.

Note: After you disconnect a repository or make any changes to repository properties, Portal Administration Console users must log out and log back in to view the changes.

Mixing Languages Within a Repository

If you mix data of multiple language types within a repository, you can use Automatic Language Detection. This approach provides the greatest flexibility for both repository content and search options.

Automatic Language Detection identifies the language and encoding of a document when it is indexed and provides the ability to query data by language and/or encoding. For example you could specify that you want to find only French and Italian matches; regardless of encoding; or only Russian matches with UTF8 encoding; or all matches, regardless of language and encoding.

You configure Automatic Language Detection on a per-repository basis. This means you could have three different repositories with different indexing and querying abilities: two repositories might use Automatic Language Detection and have a mixture of documents of type `frenchUTF8`, `englishASCII`, and `russianCYRILLIC`, while the third repository contains only `italianUTF8` documents.

Caution: When you configure Automatic Language Detection, any repository using the default configuration (one language and one encoding across all repositories using that server), will be automatically configured to use Automatic Language Detection. If you do not want this behavior, you must specify the language type for each language and its encoding for those repositories, as described in "One Language per Repository" on page 4-2.

Configuring Automatic Language Detection

When Automatic Language Detection is set, the server automatically identifies the language and encoding of a document when it is indexed. For more information about Automatic Language Detection, see the *IDOL Server Administration Guide* at `<BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

Note: Enabling this feature may have an impact on the ability to search for existing content in Content Management and GroupSpace repositories other than content defined as the `DefaultLanguageType`. This is because language reclassification can occur when this feature is enabled.

To configure Automatic Language Detection on a repository:

1. Set the `AutoDetectLanguagesAtIndex` to true in the `[Server]` section of the `AutonomyIDOLServer.cfg` file, which is located in the `<BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/<os>/IDOLserver/IDOL` directory.
2. Do not set a `fullTextSearchIndexLanguageType` property on the repository; remove if already set.
3. Optionally, specify the default query language type by adding a property to the repository. For example:

   ```
   fullTextSearchQueryLanguageType=frenchUTF8
   ```

4. Optionally, specify that results are returned across all languages, not just the language of the `fullTextSearchQueryLanguageType` by adding the following property to the repository:

   ```
   fullTextSearchQueryAnyLanguage=true
   ```

5. Re-index your repository content. For information on how to do this, see "Re-Indexing BEA Content" on page 7-1.

   Note: During indexing, if the language type cannot be determined automatically, the `DefaultLanguageType` is used. This is a global server setting, not a repository setting.

Creating Queries

Queries are very flexible; they can be in any language and encoding. For example, you can construct a query that return results for Japanese documents using UTF-8, Shift_JIS, and EUC-JP encodings.

Use the following examples to specify the search results from your repositories. For additional information about these examples, see the *WebLogic Portal Javadoc*.

Query Text in Same Language and Any Encoding

If the query text is in the language and encoding defined by the `fullTextSearchQueryLanguageType` and you want results in the language of `fullTextSearchQueryLanguageType` regardless of the encoding, you do not need to create additional code.

Query Text in Same Language with Specific Encoding

If the query text is in the language and encoding defined by the `fullTextSearchQueryLanguageType` and you want the results in the same language as the `fullTextSearchQueryLanguageType` with a specific encoding:

```
params = new AutonomyLanguageParameterSet();
params.setLanguageType("englishASCII");
params.setMatchEncoding("UTF8");
context.setParameter(FullTextSearchLanguageParameterSet.
   QUERY_LANGUAGE_PARAMETER_SET_KEY, params);
```

Query Text in Another Language and Encoding

If the query text is in a language and encoding different from `fullTextSearchQueryLanguageType`, you can override the repository `fullTextSearchQueryLanguageType` in the `ContentContext` class. This returns results in the specified `LanguageType` language, regardless of encoding:

```
params = new AutonomyLanguageParameterSet();
params.setLanguageType("englishASCII");
context.setParameter(FullTextSearchLanguageParameterSet.
   QUERY_LANGUAGE_PARAMETER_SET_KEY, params);
```

Query Across All Languages

If the query text is in one language and encoding and you want to query across all languages:

```
params = new AutonomyLanguageParameterSet();
params.setLanguageType("englishASCII");
params.setAnyLanguage(true);
context.setParameter(FullTextSearchLanguageParameterSet.
QUERY_LANGUAGE_PARAMETER_SET_KEY, params);
```

Query Multiple Specific Languages

If the query text is in one language and encoding and you want to query multiple specific languages:

```
params = new AutonomyLanguageParameterSet();
params.setLanguageType("englishASCII");
params.setAnyLanguage(true);
params.setMatchLanguageType("frenchASCII+germanUTF8");
context.setParameter(FullTextSearchLanguageParameterSet.
   QUERY_LANGUAGE_PARAMETER_SET_KEY, params);
```

Enterprise Search for Microsoft Word, Excel, and PowerPoint Files in Multibyte Languages

You can configure search and indexing for Microsoft Word (.doc), Excel (.xls), and PowerPoint (.ppt) files in Content Management and GroupSpace communities. In these cases, you need to use the default configuration for an Autonomy server, that is, one language and one encoding across all repositories using that server. For more information, see "One Language per Autonomy Server" on page 4-2.

In addition to using the default Autonomy server configuration, you need to set the encodings for indexing and searching on the file names as described in this section. Without these encoding settings, search cannot find the file names based on multibyte encodings. These encoding are set in the following files:

- omnislave.cfg
- AutonomyIDOLServer.cfg
- web.xml—required only for GroupSpace

Note: The supported language types and encodings are listed in `[LanguageTypes]` section in the AutonomyIDOLServer.cfg file, which is located in the `<BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/<os>/IDOLserver/IDOL` directory.

Settings in omnislave.cfg

You must specify the system's default encoding in the omnislave.cfg file. This file is located in the `BEA_HOME>/wlserver10.0/cm/thirdparty/autonomy-wlp10/<OS>/filters` directory.

To specify the encoding:

1. In the omnislave.cfg file, remove any `FileNameFromCharSet=<encoding>` settings from any sections in which they appear.
2. In the [Configuration] section, add the system's default encoding. For example:

```
FileNameFromCharSet=SHIFTJIS
```

Settings in AutonomyIDOLServer.cfg

You must specify the DefaultLanguageType and DefaultEncoding settings in the AutonomyIDOLServer.cfg file. The DefaultEncoding must be same encoding as specified in omnislave.cfg. And the DefaultLanguageType must be in the corresponding language type to the encoding specified in omnislave.cfg and the language of document. For example for the Japanese language with Shift-JIS encoding, you would specify:

```
[LanguageTypes]

DefaultLanguageType=japaneseSHIFT_JIS
DefaultEncoding=SHIFTJIS
```

GroupSpace Encoding

You need to also update the groupspace encoding in web.xml to use the same encoding that you specified in omnislave.cfg. (The web.xml file is located in the WEB-INF directory of the portal web project.) For example:

```
<context-param>
<param-name>com.bea.apps.groupspace.search.enterprise.outputEncoding</para
m-name>
<param-value>SHIFTJIS</param-value>
</context-param>
<context-param>
<param-name>com.bea.apps.groupspace.search.enterprise.connectionEncoding</
param-name>
<param-value>shift_jis</param-value>
</context-param>
```

Note: The setting in AutonomyIDOLServer.cfg and web.xml are not confined to matters of file name search, but are required to handle multibyte characters in Enterprise Search.

After modifying these files, you must re-index the existing content for the multibyte characters in filenames. For information on how to do this, see "Re-Indexing BEA Content" on page 7-1.

CHAPTER 5

Metadata Searching

Content Management supports metadata search as well as full-text search. This chapter describes searching for both published and versioned metadata. Published content is either content from a repository that does not have library services enabled or content that has completed a workflow and is a Published state. Versioned content is content in a library services-enabled repository that is not yet published. Searching for published content returns `Nodes` while searching for unpublished content return `Versions`. For more information about enabling library services, see Enabling Library Services for a BEA Repository in the *Content Management Guide*.

Note: Searching for published content and searching for versioned content are mutually exclusive because these search types use different APIs. and You use `ISearchManager.search` to search for published content and `IVersionManager.search` to search for versioned content. This means that the search queries go against different data sources to return the data.

This chapter contains the following sections:

Introduction

The metadata search feature uses the content expression language. You use different properties when searching for metadata in published content and versioned content. The differences are

discussed in the relevant sections. A full list of supported properties is available in `com.bea.content.expression` in the WebLogic Portal Javadoc.

Searching for Metadata in Published Content

This section describes searching for metadata in published content. The following sections provide more detail:

- Properties
- The Search Object
- Limitations
- Examples

Properties

The following tables describe commonly used properties and operators A full list of supported properties is available in `com.bea.content.expression` in the WebLogic Portal Javadoc.

Table 5-1 System Properties

System Property	Description and Example
`cm_path`	The Virtual Content Repository path to the content item. `cm_path = 'BEA Repository/books/Ulysses'`
`cm_uid`	The unique ID for a content item. `cm_uid = '0003456'`
`cm_parent_uid`	The ID of the parent for a content item. `cm_parent_uid = '87543'`
`cm_createdBy`	The user who created the content item. `cm_createdBy = 'jjoyce'`
`cm_modifiedDate`	The date the content item was last modified. `cm_modifiedDate = '04/01/2008'`
`cm_nodeName`	The name of the content item. `cm_nodeName != 'abc'`
`cm_isHierarchy`	Deprecated.

Table 5-1 System Properties

System Property	Description and Example
`cm_isContent`	Deprecated.
`cm_objectClass`	The content type associated with a content item. `cm_objectClass = 'simpleType'`
`cm_binaryName`	The file name of the binary value of a content item. `cm_binaryName = 'foo.gif'`
`cm_binarySize`	The size of the binary value of a content item (in bytes) `cm_binarySize = '188'`
`cm_contentType`	The MIME type for content item binary properties. `cm_contentType = 'image/gif'`
`cm_objectClassInstance`	Finds all instances of a given object class and all of its children. `cm_objectClassInstance = 'Product'`
`cm_value`	Find any property value on any nodes of a particular value. `cm_value = 'red'`

Table 5-2 Operators

Operator	Purpose
`like`	Syntax textual like operator.
`likeignorecase`	Syntax textual like (case insensitive) operator.
`=`	Syntax textual equals operator.
`!=`	Syntax textual not equals operator.
`&&`	Syntax textual and logical operator.
`in()`	Syntax textual in operator.
`! [negate]`	Syntax textual not operator.
`||`	Syntax textual or logical operator.
`>`	Syntax textual greater than operator.

Table 5-2 Operators

Operator	Purpose
<	Syntax textual less than operator.
contains	Syntax textual contains operator.
containsall()	Syntax textual contains all operator.
containsany()	Syntax textual contains any operator.
toProperty('<propertyName>')	Use for special characters in property names such as spaces, backslash, and so on.

Table 5-3 Wildcards

Wildcard	Description	Example
*	Supports any characters.	cm_NodeName like “ab*” matches “abcde”, “ab”, “ab234”, “abc”, and so on.
_	Supports one character.	cm_nodeName like “ab_” matches “abc”, “abd”, “ab2”, and so on.

The Search Object

You can search for metadata in a content repository using the com.bea.content.search object. The search object takes in an expression that the API processes and return search results as com.bea.content.Node objects. Listing 5-1 shows an example of a simple search.

Listing 5-1 Simple Search Example

```
ISearchManager searchManager = ContentManagerFactory.getSearchManager();
Search search = new Search(“cm_nodeName != null”);
search.setSortCriteria("cm_objectClass, cm_nodeName");
ISortableFilterablePagedList<Node> nodes =
   searchManager.search(context, search);
```

The search manager API also supports various other types of searches like `idSearch`. For more information, see the WebLogic Portal Javadoc.

Limitations

When using the BEA repository, metadata search results on implicit properties cannot be sorted. To sort the returned data you must use `postSearchSort` method in the `ISearchManager` API. That method is slower than native sorting, which is done using the search.`setSortCriteria` method.

Examples

The expression language supported by search is based on a simple, easy to use grammar. It supports a rich set of operations, and easily allows building complex queries using nested expressions. To look at the various data types, such as strings, dates, numbers, and booleans supported by the expression language, For more information, see the WebLogic Portal Javadoc.

Table 5-4 Examples

Example	Description
`cm_nodeName = 'hello'`	Returns all the nodes matching the name “hello”.
`cm_nodeName = 'hello' || cm_nodeName = 'world'`	Finds any node name that has the name “hello” or “world”.
`cm_nodeName = 'hello' && city = 'boston'`	Find any node name of name “hello” and city of “boston”.
`cm_nodeName = 'hello' && (color = 'red' || color = 'blue')`	Use the brackets (and) to find nested expressions. Nesting follows a left to right order of evaluation. This expression finds all nodes named hello, and whose color property is either “red” or “blue”.
Complex Nesting Examples	
`cm_nodeName = 'hello' && ( (city = 'boston' || city = 'boulder' ) && (color = 'red') )`	Finds any node named “hello” and whose color property is “red” and whose city property is either “boulder” or “boston”.
`!(cm_nodeName = 'hello' && city = 'boston')`	The negation ! operator returns an inverse result. In this example, all the nodes that do not have the name “hello” and the city “boston” are returned.

Searching for Metadata in Versioned Content

This chapter describes searching for metadata search in versioned content. It contains the following sections:

Specific Properties for Versions

Property	Description and Example
`cm_version`	The version number of the content. `cm_version = '6'`
`cm_versionComment`	Text describing the changes to the version. `cm_versionComment = 'Updates from Marketing'`
`cm_checkedOut`	Version is checked out. `cm_checkedOut = 'false'`
`cm_assignedToUser`	The user to which the node is assigned. `cm_assignedToUser = 'rjordan'`
`cm_role`	The role to which the node is assigned. `cm_role = 'Admin'`
`cm_latestVersion`	The latest version of the content. `cm_latestVersion = 'true'`

Limitations

Search for versioned content has some limitations because the tables that store versioned data can be in a different data store than the ones that store published data. This means that you cannot search for both published and versioned metadata with SQL queries. Therefore some properties are not supported when searching on versioned content. Here's a list of unsupported properties:

- `cm_isContent`
- `cm_isHierarchy`

- `cm_objectClass`
- `cm_path`
- `cm_createdBy`
- `cm_objectClassInstance`
- `cm_parent_uid`
- `cm_createdDate`
- Paths set via `setSearthPaths` in the `Search` object throw an error when that search object is used in a versioned system search.

Searching for versioned content and searching for published content are mutually exclusive because these search types use different APIs.

Supported Attributes

Attributes that retain their meaning from searching on versioned content are:

Table 5-5 Supported Attributes for Versioned Content Search

Attribute	Description
`cm_nodeName`	Search for node name.
`cm_uid`	Search for node by UID (User Identifier).
`cm_value`	Search for any value in versioned data
`cm_lifeCycleStatus`	Searching for nodes in any workflow state. For example, `cm_lifeCycleStatus = '3'` (Ready for Approval).

Examples

Table 5-6 shows some examples for searching versioned content (content used in a workflow).

Table 5-6 Searching Versioned Content

Example	Description
`name = 'matt' && (age = 33 || age = 13)`	Returns node versions matching the name “matt” and age of “13” or “33”.
`cm_version = '1' && city = 'calcutta'`	Returns the first version of nodes and city of “calcutta”.
`cm_versionComment = 'DevUpdates' && product = 'portal'`	Returns node versions that equal comments with “DevUpdates” for the “portal” product.
`cm_modifiedBy = 'weblogic' && product = 'portal'`	Finds node versions modified by “weblogic” for the “portal” product.
`cm_lifeCycleStatus > 0 && article = 'development'`	Returns node versions in any workflow state and articles about “development”.
`cm_nodeName = 'matt_2' && city = 'calcutta'`	Returns node versions matching the name “matt_2” and city of “calcutta”.
`cm_checkedOut = true`	Finds node versions that are checked out.
`cm_assignedToUser != null`	Returns versioned nodes that are assigned to users.
`cm_value > 30 || cm_binaryName = null`	Finds node versions for any property with a value greater than “30” or whose file name of a binary value is unknown or undefined.
`cm_role in ('weblogic','Admin') && cm_value > 30`	Finds node versions that have roles in “weblogic” and “Admin” and any property with a value greater than “30”.
`cm_version = '5'`	Returns node versions with a version number of “5”.
`age > 10 && (city in ('calcutta','boulder') || cm_role = 'Admin')`	Finds node versions with an age value greater than “10” and city of “calcutta” and “boulder” or role assigned to “Admin”.
`cm_uid = '2051' && city = 'boulder'`	Finds node versions with node ID “2051” and city of “boulder”
`cm_nodeName = 'foo' && cm_latestVersion = true`	Returns the latest version of nodes named “foo”.

Part III Staging

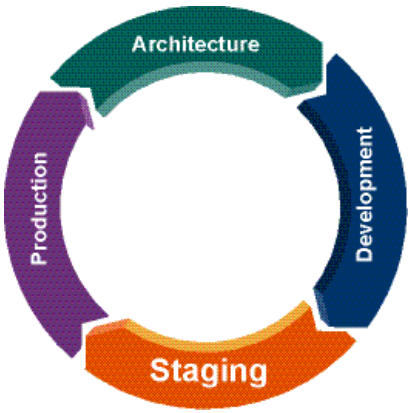
During the staging phase of the portal life cycle, you configure your search engine according to the configuration of your portal cluster where you will deploy your portal application.

Part III includes the following chapter:

- Chapter 6, "Staging Search Capabilities"

If you moved on to the Production phase and then go back to make changes that affect the development phase, you must redeploy your portal application in order to view your changes. The BEA Propagation Utility performs the redeployment; see the *BEA WebLogic Portal Production Operations User Guide* for more information.

For a detailed description of the staging phase of the portal life cycle, see the *BEA WebLogic Portal Overview*. The portal life cycle is shown in the following graphic:



CHAPTER 6

Staging Search Capabilities

When you move to a production environment, you must configure Autonomy to match the portal environment you use.

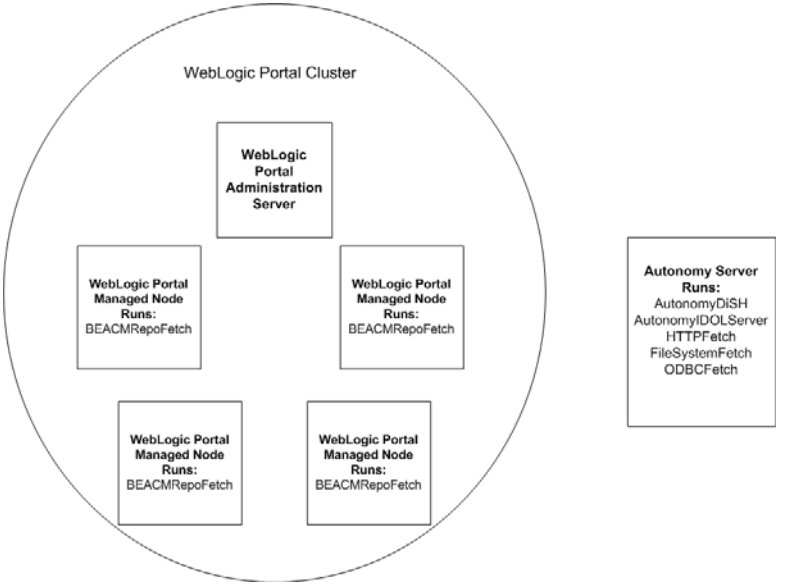
This involves editing the configuration files for the search components you are using, deploying the Autonomy Service Dashboard, and configuring Autonomy fetches to search for information according to parameters you set.

Note: The Autonomy documentation is included in your WebLogic Portal installation directory at `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

The tasks covered in this chapter assume a typical Autonomy configuration running in a WebLogic Portal cluster. Consult the Autonomy documentation if you want to create a more complex configuration, such as running HTTPFetch on a separate server.

Figure 6-1 provides an example of a typical production environment.

Figure 6-1 Example of WebLogic Portal Cluster Using Autonomy



This chapter discusses the following topics:

- Installing Autonomy on Your Target Server
- "Updating the Autonomy License"
- Configuring Autonomy on Your Target Server
- Setting up BEA Content Management Search
- Staging File System Fetch within a WebLogic Cluster
- Starting the Autonomy Services
- Installing Autonomy Service Dashboard

Installing Autonomy on Your Target Server

You need to install the appropriate Autonomy engines for your server's operating system.

When you installed WebLogic Portal, the Autonomy engine for the target operating system was included. If you need a version of Autonomy for a different operating system than the operating system on which you installed WebLogic Portal, you will need to download and install WebLogic

Portal onto the operating system for which you need Autonomy. You can then retrieve the respective operating system files for Autonomy.

Note: Remember that due to licensing restrictions, you can only run one Autonomy IDOL Server.

For example, if you downloaded and installed WebLogic Portal on a Windows server, the Windows version of Autonomy was included in the download. If you want to install Autonomy on a Linux server, you need to download and install the Linux version of WebLogic Portal in order to have the correct version of Autonomy for a Linux machine.

Supported Operating Systems

WebLogic Portal makes available versions of Autonomy that are compatible with operating systems that WebLogic Portal also supports. However, you may run Autonomy on separate server that uses an operating system that WebLogic Portal does not support. For more information about the supported configurations for Autonomy, see the Autonomy documentation.

Note: To obtain Autonomy files for a non-WebLogic Portal supported operating system, contact your Autonomy representative. For more information about configuring Autonomy on a non-WebLogic supported operating system, see "Configuring the BEA Content Management Fetch When Using a Non-WebLogic Portal Supported Operating System" on page 6-14.

Installing Autonomy

To install Autonomy:

1. Create a directory on your target server for the Autonomy components.
2. From your WebLogic Portal installation, navigate to the Autonomy distribution for your target operating system. For example, `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/<operatingsystem>`.
3. Copy the entire directory from your installation directory to the target directory on your target server.

Updating the Autonomy License

This section explains how to update the default Autonomy license with a production license. The default license allows 10,000 documents to be indexed, while the production license allows 500,000 documents to be indexed.

To update your Autonomy license:

1. Stop all Autonomy Services. To do this, call the `autonomy.cmd` script with the `stop` parameter. See "Starting the Autonomy Services" on page 6-17.
2. In the `IDOLserver/DiSH` directory, do the following:
   a. Delete the `uid` and `license` directories.
   b. Run the command: `AutonomyDiSH.exe -revokelicense`
3. In the `IDOLserver/IDOL` directory, do the following:
   a. Delete the `uid` directory.
   b. In the `content`, `agentstore`, `category`, and `community` directories under `IDOLserver/IDOL`, delete the `uid` and `license` directories.
4. Create a backup of the existing `IDOLserver/DiSH/licensekey.dat` file to prevent overwriting it.
5. Copy the production license file (`licensekey.dat`) into `IDOLserver/DiSH`.
6. Restart the Autonomy Services. See "Starting the Autonomy Services" on page 6-17.
7. Verify that the new license is accepted by reviewing the `license.log` files in the above directories and verify that the license now allows 500,000 documents, rather than the evaluation license limit of 10,000 documents. To do this, search in `IDOLserver\IDOL\logs\content_application.log` for the string "This license allows 500000 documents to be indexed".

Configuring Autonomy on Your Target Server

You need to modify your Autonomy configuration to match your production environment and the parameters of your cluster. This includes modifying the respective configurations of the search tools you are using to account for security concerns and their network location.

You also need to configure the types of information you want to include in your searches.

Note: For out-of-the-box implementation, you can set `CONTENT_SEARCH_OPTION=full` in the startWebLogic script and add the necessary properties to the repository.

Table 6-1 lists the location of the configuration files you need to modify.

Table 6-1 Autonomy Components and Their Respective Configuration Files

Autonomy Component	Configuration File You Need to Modify
DiSH Server	`//IDOLserver/DiSH/AutonomyDiSH.cfg`
IDOL Server	`//IDOLserver/IDOL/AutonomyIDOLServer.cfg`
Agent Stores for the IDOL Server	`//IDOLserver/IDOL/agentstore/agentstore.cfg`
HTTP Fetch	`//HTTPFetch/HTTPFetch.cfg`
File System Fetch	`//FileSystemFetch/FileSystemFetch.cfg`

This section includes the following topics:

- Configuring the Autonomy IDOL Server
- Configuring the Autonomy DiSH
- Configuring Agentstore
- Configuring HTTP Fetch
- Configuring File System Fetch

Configuring the Autonomy IDOL Server

The Intelligent Data Operating Layer (IDOL) server is responsible for indexing content as well as processing queries. For more information about the IDOL server, see the Autonomy IDOL Server documentation.

To configure the IDOL server for your production environment modify the `AutonomyIDOLServer.cfg` file. The file is located `//autonomy/IDOLserver/IDOL/AutonomyIDOLServer.cfg`.

To configure the Autonomy IDOL server:

1. Open the `AutonomyIDOLServer.cfg` file in a text editor.

2. In the [License] section, edit the LicenseServerACIPort to match the port on which DiSH is running, if you changed this port.
3. In the [Service] section, enter the port number by which service commands can be sent to DiSH. By default this is 20003. Note that this port must not be used by any other service.
4. In the [Server] section,
   – Edit the client list settings (IndexClients, AdminClients) for security as required.
   – Edit the IndexPort and Port (ACI) settings as needed.
5. In the [Paths] section, edit the Modules and TemplateDirectory to point to the location of these directories on the target system. These must be absolute paths.
6. Locate and edit all other directory or file path settings and adjust to point to the new location (for example, the [NT_V4] Library). These must be absolute paths.
7. In the [Database] section, create and remove Autonomy databases as required for your needs. Consult the Autonomy documentation for managing databases.
8. When finished, save your changes.

Configuring the Autonomy DiSH

The Distributed Services Handler (DiSH) is used to manage Autonomy components. You can access DiSH functions through the Autonomy Service Dashboard or use Autonomy's ACI interface. For more information about the Autonomy DiSH, see the Autonomy DiSH documentation.

To configure DiSH, you can use a text editor to modify the autonomyDiSH.cfg file. The autonomyDiSH.cfg file is located in the *//autonomy*/IDOLserver/IDOL/AutonomyIDOLDiSH.cfg directory.

To configure Autonomy DiSH:

1. Open the autonomyDiSH.cfg file in a text editor.
2. In the [Service] section, edit the ServicePort setting if needed to avoid port conflicts.
3. In the [Server] section, edit the following:
   – Modify the AdminClients as required for establishing security as needed.
   – Modify the Port setting if needed to avoid port conflicts.
4. In the [Email] section, make modifications as defined by your company's SMTP setup.

5. In the [ChildServices] section, remove the setting for the BEACMRepoFetch service.
6. Remove the [BEACMRepoFetch] section.
7. In the [IDOLServer], [HTTPFetch] and [FileSystemFetch] sections, modify each path to ensure the executable files use the location on the target server. These paths must be absolute.
8. If you changed the Service Port or ACI port (or plan on doing so in the agentstore.cfg file), you need to adjust these settings to match.
9. When finished, save your changes.

Configuring Agentstore

Agents provide the facilities to find and monitor information from a configurable list of internet and intranet sites, news feeds, chat streams and internal repositories that you want to enable your portal users to search.

For more information about using agents, see the Autonomy IDOL Server Guide.

To configure the Agentstore for your cluster, edit the agentstore.cfg file. The file is located //IDOLserver/IDOL/agentstore/agentstore.cfg.

To configure agentstore:

1. Open the agentstore.cfg file in a text editor.
1. Modify [License], [Service] and [Server] settings as required for port conflicts and security.
2. Locate and replace all file and directory settings and adjust to point to the new location. The paths must be absolute.
   - In the [Paths] section, change the TemplateDirectory to point to the new location.
   - In the [Logging] section, change the LogDirectory and the LogArchiveDirectory to point to the new location.
   - In the [LanguageTypes] section, change the LanguageDirectory to point to the new location.
3. When finished, save your changes.

Configuring HTTP Fetch

HTTP Fetch is responsible for crawling specified websites and passes the content to the IDOLServer for indexing.You need configure this fetch and create HTTP fetch jobs that you need.

You do this by editing the `HTTPFetch.cfg` file. It is located `//autonomy/HTTPFetch/HTTPFetch.cfg`

To configure the HTTP Fetch:

1. Open the `HTTPFetch.cfg` in a text editor.
1. The `[Service]` section determines which machines are permitted to use and control the HTTPFetch service via the service port. Modify the port and client security control as required.

   Note: If you modify the port settings in this file, you need to update the HTTPFetch port settings in the `AutonomyDiSH.cfg` file.
2. The `[Default]` section contains the default settings that apply to all the jobs that you define in `[Spider]`section. If you changed the `IndexPort` in the `AutonomyIDOLserver.cfg` file, you need to modify the `IndexPort` setting to match.
3. When finished, save your changes.
4. Create HTTP Fetch jobs as required (to spider and index the websites you want to search). For information about creating fetch jobs, see the Autonomy HTTPFetch documentation.

Configuring File System Fetch

File System Fetch polls specified areas of a filesystem and, when content changes are found, imports the content and passes the content to the IDOLServer for indexing.

To control how files are imported from an internal location (for example, from a computer on your network), you need to configure File System Fetch and then create the fetch jobs you need.

The File System Fetch configuration file is located: `//autonomy/FileSystemFetch/FileSystemFetch.cfg`

Use a text editor to edit the `FileSystemFetch.cfg` file to match your production environment.

To configure File System Fetch:

1. Modify the `[Server]` and `[Service]` sections to change ports, if needed, and to control security. If you modify the port information in this file, you need to also update the settings related to File System Fetch in the `AutonomyDiSH.cfg` file.
2. In the `[Default]` section, modify the `IndexPort` to match the `IndexPort` set in `AutonomyIDOLserver.cfg`, if necessary.
3. When finished, save your changes.
4. Create File System Fetch jobs as required (to spider and index certain file system locations) for your needs. For information about creating fetch jobs, see the Autonomy File System Fetch documentation.

If deploying WebLogic Portal in a cluster environment, you need to ensure that each machine in your cluster can access the content indexed by File System Fetch, see "Staging File System Fetch within a WebLogic Cluster" on page 6-16.

Setting up BEA Content Management Search

To set up full-text search for your BEA repositories, you must configure the BEA Content Management fetch and then enable your BEA repositories for full-text search.

This section includes the following topics:

Configuring the BEA Content Management Fetch

The BEA content management fetch enables full-text search for BEA repositories. For each managed server in your WebLogic Portal cluster, you need to configure the BEA content management fetch.

To configure the content management fetch:

1. Set an environment variable called `CONTENT_SEARCH_OPTION` and assign it a value of `minimal`.

2. Edit the `BEACMRepoFetch.cfg` file located in //*operating_system_directory*/`internal/BEACMRepoFetch/BEACMRepoFetch.cfg`. This file configures the settings for the full-text search of BEA repositories.
   - Modify `[Server]` and `[Default]` settings to change port numbers and client security as required.
   - Modify `[Default]` DreHost settings to point to the hostname or IP address of the server which is running the IDOLServer.
   - Modify `[Default]` IndexPort to match the IndexPort setting in the `AutonomyIDOLServer.cfg` file on the remote server.
3. When finished, save your changes.

WARNING: Do not modify any other settings within this file.

Configuring BEA Repositories for Full-Text Search

After you have configured the BEA Content Management Fetch, you need to enable your BEA repositories to take advantage of full-text search. This ensures your BEA repositories can locate the Autonomy IDOL server.

This section includes the following topics:

- Adding Autonomy Properties to your BEA Repository
- Editing Full-Text Search Properties

Adding Autonomy Properties to your BEA Repository

You need to define Autonomy properties for your BEA repositories within the Virtual Content Repository using the Portal Administration Console. These properties ensure that your BEA repositories can locate the Autonomy services.

Table 6-2 lists the Autonomy properties you need to add.

Table 6-2 Autonomy Properties for BEA Repositories

Property	Definition

`search.staging.area`	You need to set this property ONLY if you are using using a shared drive to index content. For more information, see "Configuring the BEA Content Management Fetch When Using a Non-WebLogic Portal Supported Operating System" on page 6-14.
	
	When setting this property, you must use the system default file delimiter or the data will not be properly indexed, as shown in the following examples:
	
	• Windows:
	`search.staging.area=\..\..\cm\thirdparty\autonomy-wlp10\internal\BEACMRepoTemp`
	
	• UNIX:
	`search.staging.area=/../../cm/thirdparty/autonomy-wlp10/internal/BEACMRepoTemp`
	
	Note: This path is appended to the <*BEA_HOME*> directory and therefore needs to be relative to that directory. When you set this path, be sure to start the path with a file separator character and use slashes appropriate for your operating system. Also be sure all directories in the path from `wlserver_10.0` exist.
	
	The default directory is:
	
	`/cm/thirdparty/autonomy-wlp10/internal/BEACMRepoTemp`

`search.engine.host`	This is the hostname for the machine on which the IDOL server resides.
`search.index.port`	This is the Autonomy index port. This value needs to match the `[Server]IndexPort` setting in the `AutonomyIDOLServer.cfg` file. See "Configuring the Autonomy IDOL Server" on page 6-5 for information about this file.

Table 6-2 Autonomy Properties for BEA Repositories

Property	Definition
`search.query.port`	This is the port setting that is used by the IDOL server. This value needs to match the `[Server]Port` setting in the `AutonomyIDOLServer.cfg` file. See "Configuring the Autonomy IDOL Server" on page 6-5 for information about this file.
`search.urlconnection.timeout`	When Autonomy database commands are issued using HTTP to the search indexing port and the search engine port, this time-out setting is specifies the HTTP connection time-out, in milliseconds. The default time-out is 180000 (180 seconds).

Note: After you make any changes to repository properties, Portal Administration Console users must log out and log back in to view the changes.

To add a property to a repository:

1. From the main menu of the Portal Administration Console, select **Content > Content Management**.
2. In the resource tree, click **Repositories** to view the Manage | Repositories tree.
3. In the Manage | Repositories resource tree, select the BEA Repository to which you want to add a property.
4. In the Properties section on the Summary tab, click **Add Property**.
5. In the Add Property dialog, enter the name and value for your property. Enter each property included in Table 6-2.
6. Click **Save**.

A summary of the new repository information is displayed in the Summary tab.

Editing Full-Text Search Properties

You need to ensure that all full-text functions are enabled for each BEA repository you want to enable to use full-text search.

Table 6-3 lists the advanced full-text search repository properties and how they are used.

Table 6-3 Required Settings for Full-Text Search

Advanced Property	What it does:
Search Enabled	Enables users to search the repository using metadata.
Search Indexing Enabled	Allows content to be indexed for portal search. This enables portal developers to include full-text content search or metadata search in any portlets that they develop.
Full-Text Search Enabled	Enables users to search the repository using the full-text of the content within the repository.

To edit full-text search repository properties:

1. Select **Content > Content Management** from the navigation menu at the top of the console.
2. Select **Manage | Repositories**.
3. In the resource tree, click the repository you want to modify to view its Summary tab.
4. In the Advanced section, click **Advanced** to view the Edit Advanced Properties for Repository dialog.
5. In the Edit Advanced Properties for Repository dialog, ensure that each property in Table 6-3 is enabled.
6. When finished making changes, click **Save**.

Your modifications display in the Advanced section of the Summary page.

Note: After you disconnect a repository or make any changes to repository properties, Portal Administration Console users must log out and log back in to view the changes.

Troubleshooting Full-Text Search for BEA Content Management Repository

Use the following to check full-text search in your Autonomy configuration:

- Verify that the Autonomy processes are running: AutonomyDiSH.exe, content.exe, and so on.
- Verify that the data is indexed in Autonomy. To view all data in Autonomy, use http://localhost:9014/action=list.

- Verify that the repository configuration settings are enabled:
  - `cm_fireRepositoryEvents=true`
  - `search-is-enabled=true`
  - `search-indexing-is-enabled=true`
  - `fulltext-search-is-enabled=true`
- Verify that the `ObjectClass` is marked searchable.
- Verify that `ObjectClass` property definitions are marked searchable.

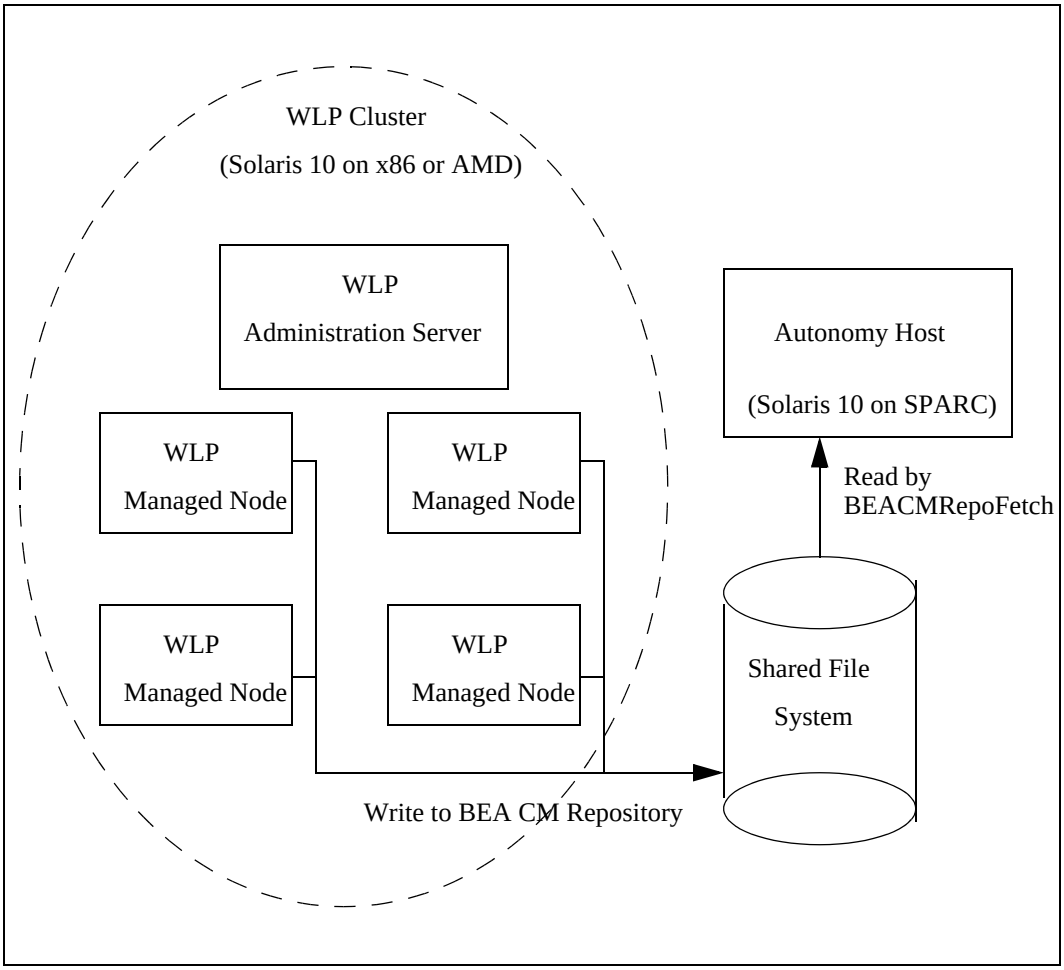
  Note: For more information about `ObjectClass`, see the *WebLogic Portal JavaDoc*.
- Verify that the `CONTENT_SEARCH_OPTION` is configured to start Autonomy. This option is in the `domain/bin/startWebLogic` script.
- Scan the Autonomy log files (files ending in .log) for warnings and errors. These files are under `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/...` directory.
- Verify that indexed data is being written to the FileSystemFetch directory specified via the search.staging.area repository configuration property. The default tree is under `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/internal/BEACMRepoTemp`.

Configuring the BEA Content Management Fetch When Using a Non-WebLogic Portal Supported Operating System

If you run Autonomy on an operating system that is not also supported by WebLogic Portal, you must configure BEA content management search differently. You must create a shared filesystem that can be written to by WebLogic Portal and also accessed by Autonomy's server.

Figure 6-2 provides an example of a remote Autonomy installation using a shared file system.

Figure 6-2 Example Remote Autonomy Installation on a non-WebLogic Portal supported operating system.



To configure BEA Content Management search:

1. Stop all Autonomy services.
1. Create a shared directory where *shared_drive* is the name of your shared drive.
2. On the Autonomy host, mount *shared_drive*.
3. For each managed server in your cluster, mount *shared_drive*.

Note: Mount *shared_drive* with the same exact mapping on each managed server.

4. On each managed server, set the CONTENT_SEARCH_OPTION environment variable to none. This prevents Autonomy from starting the content management search.
5. Using the WebLogic Portal Administration Console, define a repository property called search.staging.area with a value of shared_drive. For more information on setting other Autonomy properties, see "Staging File System Fetch within a WebLogic Cluster" on page 6-16.
6. Use a text editor to modify the <*BEA_HOME*>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/<*operating_system_directory*>/internal/BEACMRepoFetch/BEACMRepoFetch.cfg file to point to the *shared_drive* you have created. This will ensure that BEA content gets indexed.
   - In the [BEACMImport] section, set the DirectoryPathCSVs variable to match the directory of your *shared_drive*/binary
   - In the [BEACMRepoIDXImport] section, set the DirectoryPathCSVs variable to match the location of your *shared_drive*/nonbinary.
7. Restart Autonomy services. See the autonomy.sh or autonomy.cmd file in <*BEA_HOME*>/wlserver_10.0/cm/thirdparty/autonomy-wlp10 for a sample start script.

Staging File System Fetch within a WebLogic Cluster

When you deploy WebLogic Portal in a cluster environment, each machine in the cluster must be able to access information and content that is indexed by Autonomy fetches. For example, both BEA repositories and Autonomy's File System Fetch use filesystems to store indexed content. You should configure each machine in your cluster to be able to access these filesystems.

Note: The BEA Content Management Fetch does not require these steps, see "Setting up BEA Content Management Search" on page 6-9

File System Fetch is used to index content that resides in a filesystem. When indexing, unless otherwise configured, the DREREFERENCE property is set to the complete path of the file. Therefore, with default queries, the link to return the actual content (file) will be the path to the file. Within a server cluster, each node in the cluster must have access to the filesystem on which the document resides.

1. Create a shared filesystem that is accessible by both the host machine upon which File System Fetch resides and also accessible by each node in your WebLogic Portal cluster. The mapping

to the path where the files reside must be the same for each node in the cluster and the FileSystemFetch host.

2. Place the files to be imported/indexed into the shared drive as required.

3. Configure the FileSystemFetch job to import/index the contents of the shared drive using the mapping from the above step. For more information about configuring File System Fetch, see the Autonomy File System Fetch documentation.

Note: When returning query results to the browser and displaying a link to access/download the file, pass the `DREREFERENCE` property (which will contain the fully qualified path/file name) through a servlet which will stream the file to the browser. For more information about indexing and queries, see the *Autonomy IDOL Server Guide* and the Autonomy JavaDoc.

Starting the Autonomy Services

You must configure the start script that is used to start Autonomy services on your server. You can either copy these to your target server and modify as required, based on your target directory or you can create similar scripts to meet your needs.

The Autonomy start script depends on two environment variables that should be set on your portal domain server: `WL_` and `CONTENT_SEARCH_OPTION`.

- `WL_` must be set to the weblogic directory in your BEA WebLogic installation
- `CONTENT_SEARCH_OPTION` is used to indicate that level to which Autonomy will run.
  - `full` indicates to run the Autonomy DiSH engine and have it start all configured children
  - `minimal` indicates to only run the BEACMRepoFetch engine for use within the BEA repositories (this is the minimal production configuration when using BEA repositories)
  - `none` disables all Autonomy services.

You call this script with either `start` or `stop` as a parameter.

1. Review the `autonomy.cmd/sh` files that reside in the `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10` directory. Listing 6-1 shows an example script.

2. Modify as necessary. Be sure to map the Autonomy shared library directories and ensure that the `AutonomyDiSH.exe` is started.

Listing 6-1 Example Autonomy Start Script

```
setlocal

if "%WL_%" == "" goto :NO_WL_
if "%1" == "" goto :USAGE

cd %WL_%\cm\thirdparty\autonomy-wlp10\win32

if "%1" == "start" (
  echo Cleaning up license and uid files
  rmdir /s /q HTTPFetch\license >nul 2>&1
  rmdir /s /q IDOLserver\DiSH\license >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\content\license >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\agentstore\license >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\category\license >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\community\license >nul 2>&1

  rmdir /s /q HTTPFetch\uid >nul 2>&1
  rmdir /s /q internal\BEACMRepoFetch\uid >nul 2>&1
  rmdir /s /q IDOLserver\DiSH\uid >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\uid >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\content\uid >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\agentstore\uid >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\category\uid >nul 2>&1
  rmdir /s /q IDOLserver\IDOL\community\uid >nul 2>&1
  rmdir /s /q FileSystemFetch\uid >nul 2>&1

  echo Starting Autonomy with CONTENT_SEARCH_OPTION =
%CONTENT_SEARCH_OPTION%
  if "%CONTENT_SEARCH_OPTION%" == "full" (
    if not exist IDOLserver\DiSH\AutonomyDiSH.exe (
       @echo Unable to locate the Autonomy DiSH executable.  Cannot start
       the search engine.
       goto :_the_end
    )
    cd IDOLserver\DiSH
    %WL_%\server\bin\beaexecg.exe -hidewindow -command:"AutonomyDiSH.exe"
    @echo Autonomy Distributed Search Handler engine started.
  )
```

```
  if "%CONTENT_SEARCH_OPTION%" == "minimal" (
    if not exist internal\BEACMRepoFetch\BEACMRepoFetch.exe (
       @echo Unable to locate the BEACMRepoFetch executable.  Cannot start
       the search engine.
       goto :_the_end
    )
    cd internal\BEACMRepoFetch
   %WL_%\server\bin\beaexecg.exe -hidewindow -command:"BEACMRepoFetch.exe"
    @echo Autonomy BEACMRepoFetch engine started.
  )
  goto :_the_end
)
if "%1" == "stop" (

  @REM taskkill depends on the path to WBem.  Adding it here
  @REM just to ensure that it exists on the system path.
  set PATH=%SystemRoot%\System32\Wbem;%PATH%

  if "%CONTENT_SEARCH_OPTION%" == "minimal" (
    taskkill /F /T /IM BEACMRepoFetch* >nul
    @echo Autonomy BEACMRepoFetch engine stopped.
  )
  if "%CONTENT_SEARCH_OPTION%" == "full" (
    taskkill /F /T /IM AutonomyDiSH* >nul
    @echo Autonomy processes stopped.
  )
  goto :_the_end
)
goto :USAGE

:NO_WL_
@echo The environment variable WL_ is not set.  Cannot start Autonomy DiSH.
goto _the_end

:USAGE
@echo Usage: "autonomy.cmd [start|stop]"
pause
goto _the_end
```

```
:_the_end
endlocal
```

3. Run your script.
4. Verify that the services are running. On Windows, use the TaskManager application and view the Processes tab. If using Unix, use the `ps` command to view a list of services that are currently running. The following services should be running:
   - `content.exe`
   - `category.exe`
   - `community.exe`
   - `agentstore.exe`
   - `AutonomyIDOLserver.exe`,
   - `AutonomyDiSH.exe`
   - `HTTPFetch.exe`
   - `FileSystemFetch.exe`.
5. Inspect the log files for each of the services to verify that there were no errors such as port conflicts, license restrictions, and so on.

Stopping Autonomy Services in Windows 2000

On a Windows 2000 host, content management full-text search requires the PsKill utility to be installed and available in the PATH. PsKill is required to properly shut down the full-text search processes when the domain is shut down. The PsKill utility can be found at: http://www.microsoft.com/technet/sysinternals/ProcessesAndThreads/PsKill.mspx.

After installing it, run the PsKill command once to accept the license. Make sure it exists on your system PATH, and then restart the server. The scripts can now properly shut down the full-text search processes.

Installing Autonomy Service Dashboard

The Autonomy Service Dashboard is an stand-alone front-end web application that allows administrators to manage all Autonomy modules and child services running locally or remotely.

The Dashboard communicates with one or more Autonomy Distributed Service Handler (DiSH) modules that provide the back-end process for monitoring and controlling all the Autonomy child services, such as fetches.

You deploy the Autonomy Service Dashboard as a portal application within your enterprise application using the WebLogic Server Console. Before deploying the dashboard, you must modify the configuration to match your production environment.

For complete documentation on how to use the Autonomy Service Dashboard, see the Autonomy DiSH documentation.

This section includes the following topics:

- Prepare the Dashboard for Installation
- Deploy the Autonomy Service Dashboard

Prepare the Dashboard for Installation

Before you deploy the Autonomy Service Dashboard, you need to edit the location configuration to match the new deployment location. The default location that is used is: `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/common/lib/`.

To prepare the Autonomy Service Dashboard for installation:

1. Copy the `autonomyservicedashboard.cfg` to its new location.
2. Edit the configuration information to match the new location by editing the `web.xml` file for the Autonomy Service Dashboard.
   a. Create a temporary directory to use when completing your edits. For example, `c:/temp/working`
   b. Copy the `autonomyservicedashboard.war` to your working directory.
   c. Using a compression utility such as WinZip or JavaJar, unzip the `autonomyservicedashboard.war`.
   d. Configure the `WEB-INF/web.xml` file by editing the `<context-param>` value to match the new location of the `autonomyservicedashboard.war` file. The default location is `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/common/lib`.
   e. Save the `C:/working/META-INF/web.xml` file.

3. In the `C:/temp/working` directory, re-zip or re-compress the `autonomyservicedashboard.war` file. Overwrite the existing `autonomyservicedashboard.war` file with the new file of the same name that contains your modified `web.xml` file. Be sure to keep the files in their original directory structure, including the `META-INF` directory.
4. Copy the `autonomyservicedashboard.war` file you just created back to the location where you want to locate the Autonomy Service Dashboard. For example, `<BEA_HOME>/wlserver_10.0/portal/thirdparty/autonomy-wlp10/common/lib/`.

Deploy the Autonomy Service Dashboard

You should deploy the Autonomy Service Dashboard in the same domain as is used by your portal cluster. However it needs to be deployed as a stand-alone application rather than part of your portal application.

Note: You can also deploy the Autonomy Service Dashboard into another web application container, such as Tomcat.

After you deploy the Autonomy Service Dashboard, you can use the default username of `admin` and the default password of `admin` to log in.

To deploy the Autonomy Service Dashboard:

1. From the WebLogic Portal domain where you wish to deploy the Autonomy Service Dashboard, run the WebLogic Server Console.
2. In the Domain Structure section, select **Deployments**. This displays a list of the deployed components.
3. In the Change Center section, click **Lock & Edit.**
4. In the Summary of Deployments section, click **Install**.
5. Navigate to the location of the `autonomyservicedashboard.war` file and select it.
6. Click **Next**.
7. Select **Install this deployment as an application** and click **Next**.
8. Make changes as required
9. Click **Finish**.
10. In the Summary of Deployments section, review any messages or errors displayed.

11. In the Change Center section, click **Activate Changes**.
12. Verify that the **autonomyservicedashboard deployment** is prepared.
13. Mark the check box next to the **autonomyservicedashboard deployment** and click **Start > Servicing all requests**.
14. In the Start Application Assistant, click **Yes**.
15. Navigate to the Autonomy Service Dashboard to verify that is it deployed. The default URL is: `http://localhost:7001/autonomyservicedashboard`.
16. Using the default login (`admin`) and password (`admin`), log in to the Autonomy Service Dashboard.
17. Configure the Autonomy Service Dashboard to point to your DiSH implementation (use the same server and port settings that you used when you edited the `autonomyDiSH.cfg` file in "Configuring the Autonomy DiSH" on page 6-6.

    For complete documentation on how to use the Autonomy Service Dashboard, see the Autonomy DiSH documentation.
18. Ensure that the IP address of the computer(s) running the Autonomy Service Dashboard are configured in the AdminClient settings of the services. For information on how to configure these settings, see the Autonomy IDOL Server documentation.

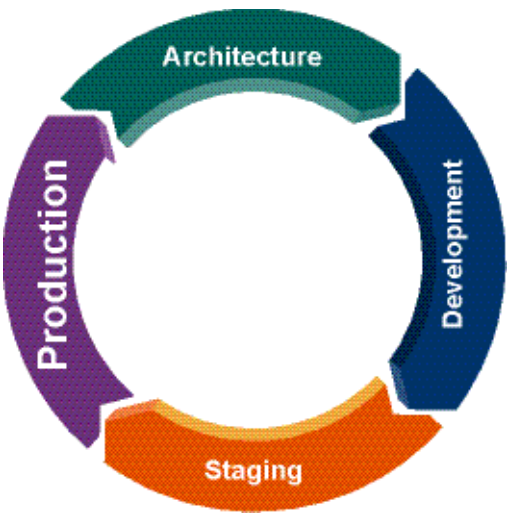
Part IV Production

This section contains guidelines and procedures to manage search configurations after you have deployed your portal and are running in production mode.

Part IV contains instructions for tasks you should accomplish in the production phase and includes the following chapter:

- Chapter 7, "Using Search in Production"

For a description of the Production phase of the portal life cycle, see the *BEA WebLogic Portal Overview*. The portal life cycle is shown in the following graphic:



CHAPTER 7

Using Search in Production

After you have deployed your portal, you can manage your search services.

Note: The Autonomy documentation is included in your WebLogic Portal installation directory at `<BEA_HOME>/wlserver_10.0/cm/thirdparty/autonomy-wlp10/common/docs`.

This chapter includes the following sections:

Using the Autonomy Service Dashboard

You can either use using the Autonomy Service Dashboard. The Autonomy Service Dashboard allows you to access the Autonomy DiSH server which monitors the performance of Autonomy's search services.

You can also monitor services using Autonomy's ACI interface. For more information about monitoring search services, see the Autonomy DiSH documentation.

Re-Indexing BEA Content

You can re-index BEA content at any time. For example, you may need to re-index BEA content if your indexes get corrupted (power outage, hardware problems, and so on).

WebLogic Portal provides a script that you can use to re-index BEA content. You can either use command line arguments or a `.properties` file to indicate the content you want to index.

Note: WebLogic Server must be running when re-indexing content.

To re-index content, do the following:

1. From any managed server in your cluster, navigate to the `index_cm_data.cmd/sh` script. It is located in the `<BEA_HOME>/wlserver_10.0/cm/bin` directory.

Tip: You can view help for re-indexing content by typing `index_cm_data -help`.

2. Optionally, if using the `cm_indexer.properties` file, modify the properties file to match the parameters of your configuration. Listing 7-2 provides an example of the `cm_indexer.properties` file.
3. Run the script. Table 7-1 provides a complete listing of the command line arguments and their descriptions. If no arguments are set, the `cm_indexer.properties` is assumed and used. See Listing 7-2 for an example of a `cm_indexer.properties` file. If using command line arguments, see Listing 7-3 for an example.

Listing 7-1 Example of Using the cm_indexer.properties File

```
C:\bea\weblogic100\cm\bin\index_cm_data
```

Listing 7-2 Sample cm_indexer.properties File

```
# Set verbose to true if you want to view any error messages.
verbose=true
#Use username and passowrd that is used to access the portal application.
user=weblogic
password=weblogic
#Use the t3 protocol to refer to the WebLogic Server URL
url=t3://localhost:7001
#Indicate the name of the repository
repository=Shared Content Repository
#Indicate the name of the portal application
application=portalApp
#Optionally, indicate which content types you want to index.
```

```
type=
#Indicate the repository path of the content you want to index.
path=/Shared Content Repository
```

Listing 7-3 Example of Using Command Line Arguments:

```
C:\bea1204\weblogic100\cm\bin\index_cm_data -verbose -user weblogic
-password weblogic -url t3://localhost:7001 -repository myRepo -application
myPortalApp -path \myRepo
```

Table 7-1 Command Line Arguments for the cm_index_data Script

Argument	Description
verbose	Set verbose to `true` to view error messages.
user	The user name.
password	The user password.
url	The URL of the WebLogic Server. For example, when running the CM_INDEX_DATA script for the local machine, this URL should be: `t3://localhost:7001`. Note that you should use the T3 protocol, not HTTP.
repository	Name of the repository you want to index.
application	Name of the portal application that uses the repository.
type	Optional. Indicate which content type you want to index.
path	The path of the repository or repository folders you want to index. For example, if you want to index the entire repository use `path=/RepositoryName`. If you want to index a particular folder within the repository, use `path=/RepositoryName/FolderName`.