Oracle® WebLogic Portal

Integrating Search

10*g* Release 3 (10.3)

September 2008

ORACLE®

Contents

Introduction

Oracle 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
- Search in the Portal Life Cycle
- Getting Started
- Disabling Search/ Search Indexing
- Autonomy Documentation

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.

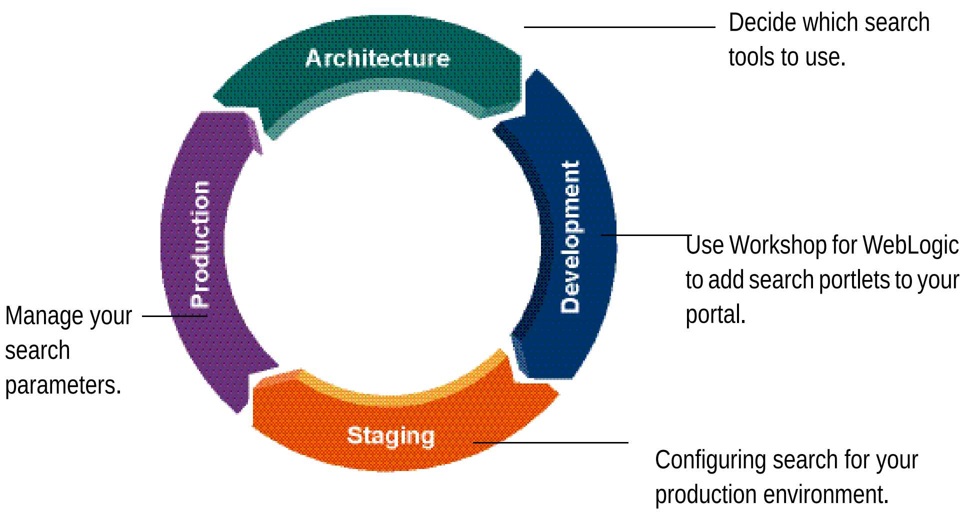
WebLogic Portal utilizes Autonomy® search for its search functionality. See "Getting Started" on page 1-3 for an introduction to the Autonomy components.

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
Decide which search tools to use.
Development
Use Workshop for WebLogic to add search portlets to your portal.
Staging
Configuring search for your production environment.
Production
Manage your search parameters.

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.

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.

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.

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.

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.
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.

Table 1-1 Autonomy Search Components Used with WebLogic Portal

Autonomy Component	What It Does:
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 contact Oracle Support. 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 `<`*`WLPORTAL_HOME`*`>/content-mgmt/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, contact Oracle Support.

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 Oracle 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, contact Oracle Support.

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

Oracle provides front-line support for Autonomy components—contact the Oracle Support Department. Oracle 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 `<`*`WLPORTAL_HOME`*`>/content-mgmt/thirdparty/autonomy-wlp10/common/docs`.

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:

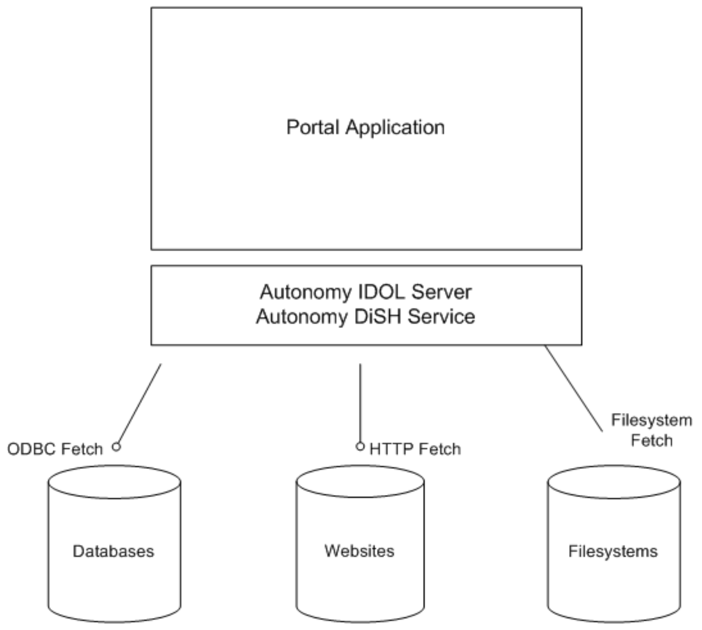
Understanding How Search is Implemented

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, contact Oracle Support.

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

Oracle 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. For more information, contact Oracle Support.

For more information about installing and deploying Autonomy services, contact Oracle Support.

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 `<WLPORTAL_HOME>/content-mgmt/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 `<WLPORTAL_HOME>/content-mgmt/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 `<WLPORTAL_HOME>/content-mgmt/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 WLP 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 3-2.

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 `<WLPORTAL_HOME>/content-mgmt/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 `<WLPORTAL_HOME>/content-mgmt/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 WLP Repository 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 3-2.

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 `<`*`WLPORTAL_HOME`*`>/content-mgmt/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 `<`*`WLPORTAL_HOME`*`>/content-mgmt/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 WLP Repository 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 WLP 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
- Searching for Metadata in Published Content
- Searching for Metadata in Versioned Content

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 = 'WLP 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 WLP 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:

- Limitations
- Supported Attributes
- Examples

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”.

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 `<WLPORTAL_HOME>/content-mgmt/thirdparty/autonomy-wlp10/common/docs`.

This chapter includes the following sections:

- Using the Autonomy Service Dashboard
- Re-Indexing WLP Repository Content

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 WLP Repository Content

You can re-index WLP repository content at any time. For example, you may need to re-index WLP 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 WLP repository 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 `<`*`WLPORTAL_HOME`*`>/content-mgmt/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.

Table 7-1 Command Line Arguments for the cm_index_data Script

Argument	Description
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`.