Novell exteNd Director

5.0

www.novell.com

CONTENT SEARCH GUIDE

Novell®

Legal Notices

Novell, Inc.

1800 South Novell Place

Provo, UT 85606

www.novell.com

exteNd Director *Content Search Guide*

December 2003

Online Documentation: To access the online documemntation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Novell Trademarks

ConsoleOne is a registered trademark of Novell, Inc.
eDirectory is a trademark of Novell, Inc.
GroupWise is a registered trademark of Novell, Inc.
exteNd is a trademark of Novell, Inc.
exteNd Composer is a trademark of Novell, Inc.
exteNd Director is a trademark of Novell, Inc.
iChain is a registered trademark of Novell, Inc.
jBroker is a trademark of Novell, Inc.
NetWare is a registered trademark of Novell, Inc.
Novell is a registered trademark of Novell, Inc.
Novell eGuide is a trademark of Novell, Inc.

SilverStream Trademarks

SilverStream is a registered trademark of SilverStream Software, LLC.

Third-Party Trademarks

Acrobat, Adaptive Server, Adobe, AIX, Autonomy, BEA, Cloudscape, DRE, Dreamweaver, EJB, HP-UX, IBM, Informix, iPlanet, JASS, Java, JavaBeans, JavaMail, JavaServer Pages, JDBC, JNDI, JSP, J2EE, Linux, Macromedia, Microsoft, MySQL, Navigator, Netscape, Netscape Certificate Server, Netscape Directory Server, Oracle, PowerPoint, RSA, RSS, SPARC, SQL, SQL Server, Sun, Sybase, Symantec, UNIX, VeriSign, Windows, Windows NT

All third-party trademarks are the property of their respective owners.

Third-Party Software Legal Notices

The Apache Software License, Version 1.1

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Autonomy

Castor

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name "ExoLab" must not be used to endorse or promote products derived from this Software without prior written permission of Intalio Inc. For written permission, please contact info@exolab.org.

4. Products derived from this Software may not be called "Castor" nor may "Castor" appear in their names without prior written permission of Intalio Inc. Exolab, Castor and Intalio are trademarks of Intalio Inc.

5. Due credit should be given to the ExoLab Project (http://www.exolab.org/).

THIS SOFTWARE IS PROVIDED BY INTALIO AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTALIO OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Indiana University Extreme! Lab Software License

Version 1.1.1

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Indiana University" and "Indiana University Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/.

5. Products derived from this software may not use "Indiana University" name nor may "Indiana University" appear in their name, without prior written permission of the Indiana University.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

JDOM.JAR

Phaos

Sun

W3C

3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Contents

About This Book

Purpose

This book shows you how to use the Novell® exteNd Director™ Content Management and Search subsystems to implement Autonomy-based conceptual queries, keyword search capabilities, and SQL-based queries in your exteNd Director applications.

Audience

This book is for programmers who want to incorporate search capability into their Novell exteNd Director applications.

Prerequisites

This book assumes you are familiar with Java programming, XML, SQL, and content query formatting.

Organization

Here's a summary of the topics you'll find in this book:

Part and chapter		Description
1	About Searching	Overview of the range of searching techniques you can implement in exteNd Director applications. Compares conceptual searching, keyword searching, and SQL-based searching
PART I, "Conceptual Search Concepts", on page 19		
2	Configuring Your Environment for Conceptual Searching	How to set up your environment for implementing Autonomy-based conceptual searching, including platform-specific and server-specific configuration requirements
3	Setting Search Options	How to fine-tune the conceptual search process

Part and chapter		Description
4	Implementing Conceptual Search	How to implement Autonomy-based conceptual searching in exteNd Director applications using the Content Management and Search APIs
5	Fetching Content and Metadata	How to implement and trigger fetching in exteNd Director applications using the Search API
6	Querying Content and Metadata	How to construct and run Autonomy-based queries using the Content Management and Search APIs
7	Configuring the Dynamic Reasoning Engine for Specialized Searching	How to use Autonomy-based methods in exteNd Director to search for numbers and search content in other languages
8	Troubleshooting the Conceptual Search Process	How to diagnose and correct commonly encountered errors; includes techniques for debugging the Autonomy-based conceptual search process
PART II, “SQL-Based Search Concepts”, on page 103		
9	Implementing SQL-Based Searching	How to implement SQL-based searching in exteNd Director applications using the Content Management API; includes techniques for constructing and executing SQL query expressions
PART III, “Tools”, on page 111		
10	Administering the Dynamic Reasoning Engine	How to manage search behavior using the exteNd Director Dynamic Reasoning Engine (DRE) Administration console
PART IV, “Reference”, on page 123		
11	Search Options Reference	Complete set of options you can configure in exteNd Director for customizing Autonomy-based conceptual search
12	Search Query Types	Types of Autonomy-based queries supported by the exteNd Director Search subsystem and how to implement them

1 About Searching

This chapter provides an overview of the searching methods you can implement in exteNd Director applications.

The following topics are covered:

- Searching methods
- Overview of Autonomy-based conceptual searching
- Overview of SQL-based searching

Searching methods

You can implement the following types of searching in exteNd Director applications:

Type of search	Description	exteNd Director support
Conceptual and keyword	Matches concepts or keywords based on English-like queries to search document content and metadata. The underlying technology is built around Application Builder, a toolkit from Autonomy, Inc. consisting of application programming interfaces (APIs). These APIs provide access to conceptual query and index functionality of Autonomy's Dynamic Reasoning Engine (DRE).	The Search API wrappers the Autonomy API to provide classes and methods for searching data sources allowed under license agreements with Autonomy, Inc. The Content Management (CM) API wrappers the Search API to provide classes and methods for searching the exteNd Director CM repository. **IMPORTANT:** When you purchase exteNd Director you are licensed to search the exteNd Director CM repository.
SQL-based	Matches criteria specified in SQL queries to search document metadata.	The CM API provides classes and methods for searching the exteNd Director CM repository.

Overview of Autonomy-based conceptual searching

Autonomy-based search technology gives you the ability to implement conceptual and keyword searching in your exteNd Director components. Traditional *keyword searching* returns all documents that contain occurrences of a search string. By contrast, *conceptual searching* matches concepts, often returning more relevant results.

How conceptual searching works

NOTE: The information in this section is adapted from the *Autonomy Technology White Paper* from Autonomy, Inc.

The Autonomy Dynamic Reasoning Engine (DRE) uses sophisticated pattern-matching algorithms to analyze any type of unstructured information, including documents in text and binary formats. Using these algorithms, the DRE identifies the patterns that occur naturally in text, then looks for similar patterns in the data source and returns the most relevant results.

The DRE determines relevance by performing probabilistic analysis to determine what data is most important, then assigns weights to indexed terms based on their importance.

How conceptual searching differs from keyword searching

NOTE: The information in this section is adapted from the *Autonomy Technology White Paper* from Autonomy, Inc.

Recall that **traditional keyword searching** is the process of finding documents that contain text strings specified by a user. Keyword searches return all documents that contain one or more occurrences of the search string, regardless of the context in which it is used. Because context is ignored, the results frequently contain many irrelevant hits. To refine search results, users often must modify their queries by adding complex boolean expressions. Keyword searching is also known as *full-text searching*.

By contrast, **conceptual searching** does take into account the context in which search terms appear so that it can match concepts rather than simply finding literal text strings. The result set contains content that is related by meaning and ranked by relevance to the search criteria. In this way conceptual searching reduces the number of false hits by returning documents that contain the concept, whether or not they also contain the search string.

To further illustrate the difference between the two approaches, consider this example. A keyword search for the term **The+effect+of+the+recession+on+consumer+spending** would return only documents that contain occurrences of all of these terms, likely producing a number of irrelevant results. The identical conceptual search would return documents that match the concept underlying the search expression, even if the documents don't contain all the terms in the query.

Searching the CM repository

exteNd Director comes with a **data fetcher** for the exteNd Director CM repository. This CM fetcher **automatically** propagates document content and metadata from the CM repository into the exteNd Director DRE where it is indexed. The related processes of propagating and indexing data is often called *fetching*.

The exteNd Director CM subsystem communicates with the exteNd Director DRE through the Search subsystem. The CM API wrappers the Search API, providing classes and methods for constructing and running queries on content and metadata that reside in the CM repository and have been indexed by the exteNd Director DRE.

For more information on using the CM API for implementing conceptual searches against the CM repository, see Chapter 4, “Implementing Conceptual Search”, Chapter 5, “Fetching Content and Metadata”, and Chapter 6, “Querying Content and Metadata”.

Searching other data sources

The CM data fetcher that comes with exteNd Director allows you to use Autonomy technology **exclusively** with data from the exteNd Director CM repository. This fetcher automatically imports document content and metadata from the exteNd Director CM repository into the DRE for indexing, allowing you to subsequently conduct Autonomy-based searches over the indexed data.

To use Autonomy technology with exteNd Director to search other data sources, you must purchase additional data fetchers from Autonomy, Inc. For these licensed data sources, you use Search API classes directly to initiate the fetching process, and construct and run queries. Fetching occurs automatically only when you use the CM data fetcher.

What you can do with the Search subsystem

The Search API provides wrapper classes around the Autonomy APIs to give you access to the following capabilities programmatically:

- Fetch (import and index) content into the exteNd Director DRE
- Perform conceptual searches using a variety of query types, including fuzzy, proximity, and thesaurus searches
- Search both structured data (document metadata) and unstructured data (content) using a single query expression
- Use **Suggest More** queries to find documents similar in meaning
- Specify maximum number of hits to limit query results
- Page through the results of Autonomy-based conceptual and keyword queries
- Rank or limit the query results by relevance percentages or absolute weight

For more information about how to access and implement these capabilities, see Chapter 4, “Implementing Conceptual Search”, Chapter 5, “Fetching Content and Metadata”, and Chapter 6, “Querying Content and Metadata”.

Overview of SQL-based searching

The exteNd Director CM subsystem provides a built-in capability for SQL-based searching of metadata in the CM repository. You execute SQL search queries on document **metadata** only.

To search document **content**—or **both content and metadata**—use Autonomy-based searching, as described in "Overview of Autonomy-based conceptual searching" on page 14.

Why use SQL-based searching

SQL-based searching allows you to search metadata stored in relational databases. You might opt for this search method in exteNd Director to:

- Take advantage of the rich set of operators SQL provides, including IN and BETWEEN
- Search metadata that is available only through SQL-based document queries—and not via Autonomy-based search—such as category memberships or information about document links

What you can search

You can use SQL queries to search for the following metadata properties in the CM repository:

- Author
- Content size
- Creation date/time
- Document abstract
- Document type
- Expiration date/time
- Document folder
- **Locked by** field
- MIME type
- Document name
- Parent document
- Publish date and status
- Document status
- Subtitle
- Title
- Update time and user

For these properties, the CM API provides classes and methods for constructing and running SQL query expressions that search for values, ranges of values, words, phrases, or other patterns, as appropriate.

Support for SQL constructs

The CM API provides methods on the com.sssw.cm.api.**EbiDocQuery** object for defining SQL clauses that you use to construct search queries. In exteNd Director, you construct SQL-based queries by defining SELECT, WHERE, and ORDER BY clauses.

The com.sssw.cm.api.EbiDocQuery interface defines WHERE methods for setting search criteria. In addition, com.sssw.cm.api.EbiDocQuery extends the com.sssw.cm.api.**EbiDocMetaDataQuery** interface which defines SELECT and ORDER BY methods:

Method type	Description
SELECT	Lets you specify the properties to return if they meet the search criteria
WHERE	Lets you set search criteria by defining the subclauses of a SQL WHERE expression
ORDER BY	Lets you specify how to return the result set

For more information, see Chapter 9, “Implementing SQL-Based Searching”.

Conceptual Search Concepts

Describes the fundamentals of implementing Autonomy-based conceptual searching in exteNd Director applications

2 Configuring Your Environment for Conceptual Searching

This chapter explains how to set up your environment for implementing conceptual searching in your exteNd Director applications. You will learn about platform-specific and server-specific configuration requirements for using conceptual search capabilities with the Content Management (CM) subsystem.

The following configuration tasks are covered:

- Installing the exteNd Director Dynamic Reasoning Engine
- Adding the Autonomy Java Native Interface to your environment
- Determining your exteNd Director project configuration
- Creating the exteNd Director project
- Enabling conceptual search
- Setting security options
- Setting search options

Installing the exteNd Director Dynamic Reasoning Engine

As you learned in Chapter 1, "About Searching", conceptual searching uses a Dynamic Reasoning Engine (DRE) based on technology from Autonomy, Inc. Therefore, you must install the DRE on the platform where you will run exteNd Director applications that implement conceptual search.

This section describes how to install the DRE on platforms supported by exteNd Director. These platforms are listed in the *Release Notes*.

Installing the DRE on Windows

The exteNd Director DRE is automatically installed with exteNd Director on Windows and automatically launched when you start the Windows machine where you installed exteNd Director.

If you want to run the DRE on a different machine from where you installed exteNd Director, follow these steps:

➢ **To install the exteNd Director DRE on a remote Windows server:**

1. Run the exteNd Director installation program and proceed through the introductory screens.
2. On the Setup Type panel, choose the **Custom** option.
3. On the panel for selecting features, select **Autonomy DRE**.
4. Finish the installation wizard.

Installing the DRE on UNIX

➢ **To install the exteNd Director DRE on UNIX:**

1. Copy the installation files from the CD onto your machine:

For	Copy
AIX	aix-Director500Autonomy.tar.Z
HP-UX	hpux-Director500Autonomy.tar.Z
Linux	linux-Director500Autonomy.tar.Z
Solaris	sol-Director500Autonomy.tar.Z

2. Uncompress the copied file by entering this command:

   `uncompress filename.tar.Z`

   The file *filename*.**tar** is created.
3. Un-tar *filename*.**tar** by entering this command:

   `tar -xvf filename.tar`

   The directory `server-platformID` is created, as follows:

Source file	Directory created
aix-Director500Autonomy.tar	Director500Autonomy
hpux-Director500Autonomy.tar	Director500Autonomy
linux-Director500Autonomy.tar	Director500Autonomy

Source file	Directory created
sol-Director500Autonomy.tar	Director500Autonomy

4 Create the directories **Scripts** and **ns-home** by entering these commands:

```
mkdir Scripts
mkdir ns-home
```

5 Navigate to the directory **ns-home** and create two additional directories: **docs** and **cgi-bin**, by entering these commands:

```
cd ns-home
mkdir docs
mkdir cgi-bin
```

6 Navigate up one level and then down to the installer directory by entering these commands:

```
cd ..
cd server-platformID
```

7 In the installer directory, run the setup command, using this syntax:

```
./Setup.sh App_Name App_dir DRE_IP_Address DRE_
Port DRE_IndexerPort Root_html_dir CGI_dir_Map
```

where:

Argument	Description
App_Name	Name of your Autonomy server installation
App_dir	The full path of the location where the software should be installed **NOTE:** This directory will be created and must not already exist
DRE_IP_Address	The host IP address
DRE_Port	The port number for communicating with the DRE; this port must be different from the *DRE_IndexerPort*
DRE_IndexerPort	The port number for communicating with the indexing process of the DRE **NOTE:** This port must be different from the *DRE_Port*
Root_html_dir	The full path to the location of the root document directory for your Web server **NOTE:** The HTML files will be installed into a subdirectory with the same name as *App_Name*

Argument	Description
CGI_dir	The full path to the CGI directory on your Web server **NOTE:** This directory needs to be mapped for CGI files and to be accessible via the URL */Scripts* (see *CGI_Dir_Map*)
CGI_Dir_Map	The CGI mapping to be used

Here is a sample command:

```
./Setup.sh MyServer /Autonomy/Server 127.0.0.1 2000 2001
/opt/ns-home/docs /opt/ns-home/cgi-bin /Scripts
```

8 When prompted, choose a style.

9 Navigate to the *App_dir* directory and start the Autonomy services by entering these commands:

```
cd App_dir
StartQuery.sh
```

Adding the Autonomy Java Native Interface to your environment

You add the Autonomy Java Native Interface (JNI) to your environment by:

1 Adding autonomyJNI.jar to your application server classpath

2 Adding the Autonomy dynamic library to your environment

Adding autonomyJNI.jar to your application server classpath

AutonomyJNI.jar resides in:

On	JAR is installed at
Windows	The exteNd Director installation directory under the subdirectory **autonomy**
AIX	$HOME/Director500Autonomy
HP-UX	$HOME/Director500Autonomy3
Linux	$HOME/Director500Autonomy
Solaris	$HOME/Director500Autonomy

To add autonomyJNI.jar to your server classpath, follow server-specific instructions below.

Instructions for Novell server

Start the server from the command line with the classpath command. For example, assuming you install exteNd Director at **c:\xwb**, here is the syntax:

To	Use this command line
Prepend a JAR to the classpath	`silverserver +cp:p C:\xwb\exteNd Director\autonomy\autonomyJNI.jar`
Append a JAR to the classpath	`silverserver +cp:a C:\xwb\exteNd Director\autonomy\autonomyJNI.jar`

NOTE: If you install exteNd Director on your hard drive under **Program Files**, you must use the shortcut **Progra~1** in the path on the command line.

Instructions for BEA WebLogic server

Follow these instructions to add autonomyJNI.jar to your BEA WebLogic server classpath:

1. Open the server's startup file for editing.

   TIP: The default startup file on Windows NT is **startWebLogic.cmd**.

2. Edit the **set CLASSPATH** command by appending **autonomyJNI.jar** to the classpath.
3. Save and close the file.

Instructions for IBM WebSphere server

Copy **autonomyJNI.jar** to the WebSphere Classpath directory (typically **\WebSphere\AppServer\lib**).

Adding the Autonomy dynamic library to your environment

The Autonomy dynamic library is contained in the following files on each supported platform:

On this platform	Dynamic library is
Windows NT	autonomyJNI.dll
AIX	libautonomyJNI.so

On this platform	Dynamic library is
HP-UX	libautonomyJNI.sl
Linux	libautonomyJNI.so
Solaris	libautonomyJNI.so

You must add this dynamic library to your server environment—that is, to the machine on which you deploy your application. Follow the server-specific and platform-specific instructions below.

Instructions for Novell server

On Windows The file autonomyJNI.dll is installed in the exteNd Director installation directory under the subdirectory **autonomy**.

The directory containing autonomyJNI.dll is automatically added to the path of the machine where you install exteNd Director. If you are using this machine as your server, you are all set. Otherwise, add the path of the directory containing autonomyJNI.dll to your library path (the PATH environment variable of your server).

On UNIX Add the path of the directory containing libautonomyJNI.so or libautonomyJNI.sl to the library path as follows:

1. Open your Novell configuration file **.agprofile** in the Novell exteNd™ Application Server installation directory.
2. Edit . **agprofile** as follows:

For	Do this
AIX	Add the path to libautonomy.so to the LIBPATH variable
HP-UX	Add the path to libautonomy.sl to the SHLIB_PATH variable
Solaris	Add the path to libautonomy.so to the LD_LIBRARY_PATH variable

Instructions for other supported servers

Consult the documentation for BEA WebLogic and IBM WebSphere servers for instructions on how to edit the library path for those servers on Windows and UNIX platforms.

Determining your exteNd Director project configuration

Before you create your exteNd Director project, you need to determine which subsystems you need to include for implementing conceptual searching. Often, the configuration is dictated by the methodologies you choose for updating content in the CM repository.

Use the following table as a guide:

To update content	Include				
	Search subsystem	CM subsystem	WebDAV subsystem	Portal subsystem	DAC
Using the CMS Administration Console in the DAC	●	●	—	●	●
At runtime using the CM API	●	●	—	—	—
Using WebDAV to transfer updates from third-party applications into the CM repository	●	●	●	—	—

You can use any combination of these methodologies as long as you include the required subsystems in your exteNd Director project.

📖 For more information about the CMS Administration Console, CM API, and WebDAV, see the *Content Management Guide*.

Creating the exteNd Director project

Now that you have determined which subsystems you need for implementing conceptual search, you can create a new exteNd Director project that includes these subsystems and all others that your application will need.

You use the Project Wizard to create new projects. If you create a **custom** project, you can include subsystems individually. If you create a **typical** project, all exteNd Director subsystems are included automatically.

➢ **To create an exteNd Director project that supports searching:**

1. Follow the procedure for using the Project Wizard, described in the section on creating exteNd Director projects in *Developing exteNd Director Applications*.
2. Keep these guidelines in mind:
   - You must enable conceptual search for your project—either as you create a new project at design time or afterward in an existing project. See "Enabling conceptual search" on page 28 for details.
   - If you installed the DRE on UNIX, change the location of the binary document text filter directory, which defaults to a Windows path. You can set this directory at design time if you create a custom project in the Project Wizard and modify settings on the **Filters** tab in the **Content Management Search Configuration** panel.

     IMPORTANT: After creating your project, set read/write/execute permission on the designated binary document text filter directory.

Enabling conceptual search

Before you can implement search in your applications, you must set the option that enables the conceptual search capability.

You can set this option at design time when you create a **custom** exteNd Director project, or in an existing project.

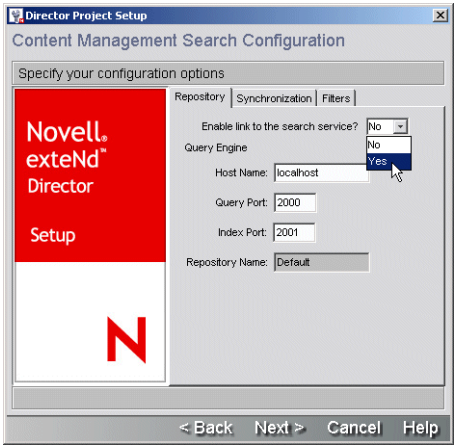
➢ **To enable search at design time:**

1. Begin creating your project using the Project Wizard.

   📖 For more information, see the section on creating exteNd Director projects in *Developing exteNd Director Applications*.
2. In the Setup Type panel, select **Custom**.

   IMPORTANT: *If you create a **typical** project instead of a custom project, search is disabled by default. You must then override this default after you create your project, using the technique described in "To enable search in an existing project:" on page 29.*

3 When you reach the Content Management Search Configuration panel, set **Enable link to the search service** to **Yes**:



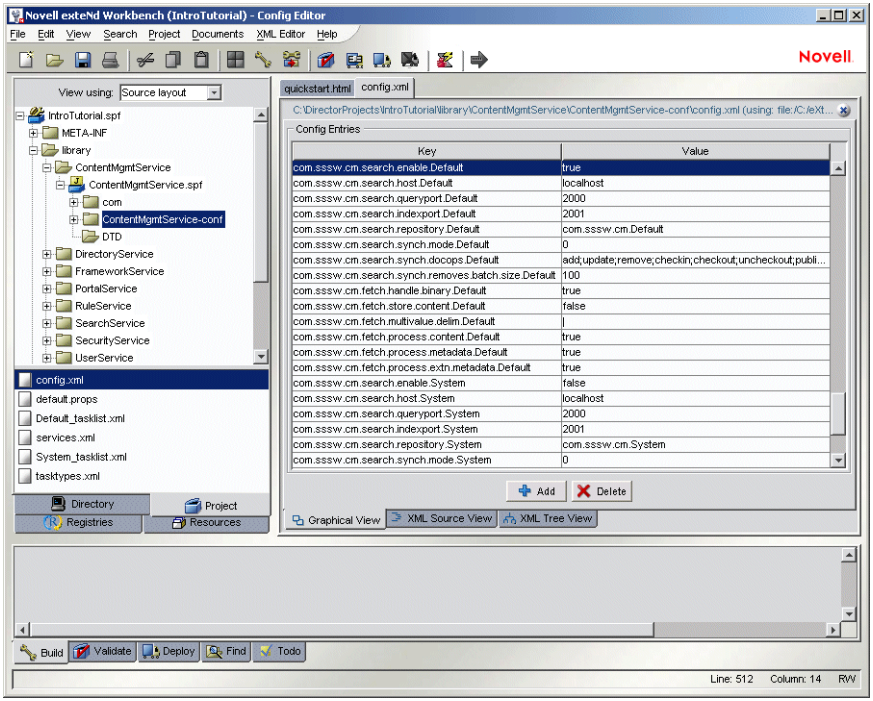
4 Complete the rest of the wizard panels to finish creating your project.

➢ **To enable search in an existing project:**

1 In exteNd Director, open **config.xml** for the CM subsystem in the project in which you want to implement conceptual search.

📖 For more information about where project files are located, see the section on exteNd Director project structure in *Developing exteNd Director Applications*.

2 In the configuration file, set the property **com.sssw.cm.search.enable.*repository name*** to **true**:



Make sure you enable search for the correct repository. For example, if you plan to search the default CM repository, enable this option:

```
com.sssw.cm.search.enable.Default
```

3 Configure other options in this file if necessary, as described in Chapter 3, "Setting Search Options".

4 Redeploy the project for the new settings to take effect.

 For more information about this option, see "Enable link to the Search subsystem?" on page 127.

Setting security options

Your system administrator needs to set security options so that when authorized users add custom metadata fields in the CM repository, these changes propagate to the Search service and are indexed correctly into the DRE. In this way, the CM subsystem is synchronized with the Search service to preserve the integrity of search results.

Here's what to do:

1. Assign authorized users to the **SearchAdmin** group.
2. Assign READ, WRITE, and PROTECT permissions to the SearchAdmin group.

 To learn how to set these options in the Portal Administration Console (PAC), see the chapter on managing security using the PAC in the *User Management Guide*.

Setting search options

The Search subsystem provides a comprehensive set of options that you can configure to customize search technology and behavior.

 To learn more about these options, see Chapter 3, "Setting Search Options" and Chapter 11, "Search Options Reference".

3 Setting Search Options

This chapter explains how to sort through and modify an extensive set of options for configuring how conceptual search operates in your exteNd Director applications.

The following topics are covered:

- About search options
- A decision matrix
- How to modify search options

About search options

exteNd Director provides a varied and powerful set of capabilities for implementing sophisticated Autonomy-based conceptual searching in your applications. Within this environment, you can shape search technology and behavior by modifying a number of configurable search options at design time and runtime. You can tailor:

- Types of data to search—content and/or metadata
- When and how often to import and index content from your data source as updates occur
- Types of document formats to search
- Operations that trigger the synchronization of the search database with your data source

For more information

Detailed information about search options is available from these sources:

For	See
Detailed descriptions of the complete set of options	“Search Options Reference” on page 125
Guidance on how to decide which options to modify	“A decision matrix” next
Instructions on how to configure search options	“How to modify search options” on page 36

A decision matrix

Before you configure search options, consider these key decision points about search capabilities required by your application:

Decision	Action
How do I want to configure the *exteNd Director* Dynamic Reasoning Engine (DRE)?	Use the DRE Administration console to set up a connection with a DRE running on a particular host using a specific port. See Chapter 10, “Administering the Dynamic Reasoning Engine”.
What types of documents do I want to search?	Specify which supported document MIME types and associated file extensions you want to index for searching by setting the options “Importable MIME types” on page 128 and “Importable file extensions” on page 127.
Do I want to search binary document formats or just text format?	Enable binary document search by setting the options “Support binary document formats?” on page 133 and “Install directory for binary document text filters” on page 130.
How do I synchronize the Content Management (CM) repository and the corresponding *exteNd Director* DRE database as updates occur in the CM repository?	Specify synchronization behavior by setting the options “Synchronization mode” on page 134, “Operations that trigger immediate synchronization” on page 132, and “Number of deleted documents to batch up” on page 132.

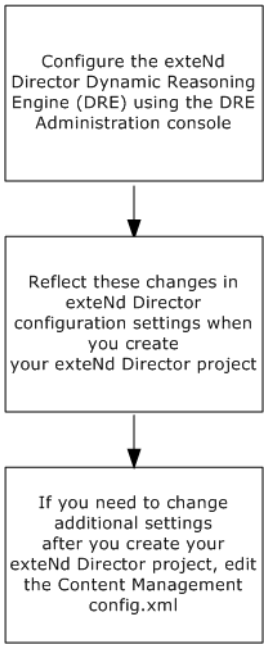
Decision	Action
What kind of information do I want to index for conceptual searching?	Specify the content and metadata you want to index by setting the options "Index document content?" on page 129, "Index standard document metadata?" on page 129, and "Index custom document metadata?" on page 128. **NOTE:** It is recommended that you enable at least one of these options so that the indexing process produces meaningful results.
Do I want detailed error tracing of the indexing process?	Set the option "Debug during import?" on page 126.
Do I want to search for numbers?	Set IndexNumbers = 1 in the Autonomy DRE configuration file, as described in "Searching for numbers" on page 77.
Do I want to search in other languages?	Set language-specific configuration parameters and specify sentence-breaking files, as described in "Searching in other languages" on page 78.

How to modify search options

There are several strategies for modifying search options:

- Using the exteNd Director Dynamic Reasoning Engine (DRE) Administration console
- At design time when you create an exteNd Director project
- In an existing exteNd Director project
- Programmatically at runtime
- By modifying the DRE configuration file

The diagram below illustrates the recommended order of operations for modifying search options:



IMPORTANT: You must mirror any changes you make in the DRE Administration console in the exteNd Director environment—either by modifying the associated settings in the exteNd Director Project Wizard at design time when you create your exteNd Director project or by modifying the config.xml file in an existing exteNd Director project.

Configuring the DRE using the exteNd Director DRE Administration console

You can configure your exteNd Director DRE by using the DRE Administration console, a graphical user interface for administering the exteNd Director DRE. The procedure is described in "Setting DRE search options" on page 118.

Setting search options at design time

A subset of search options can be modified at design time when you create a custom exteNd Director project:

- Enable link to the Search subsystem?
- Index port
- Install directory for binary document text filters
- Name of DRE database
- Name of DRE host
- Number of deleted documents to batch up
- Operations that trigger immediate synchronization
- Query port
- Synchronization mode

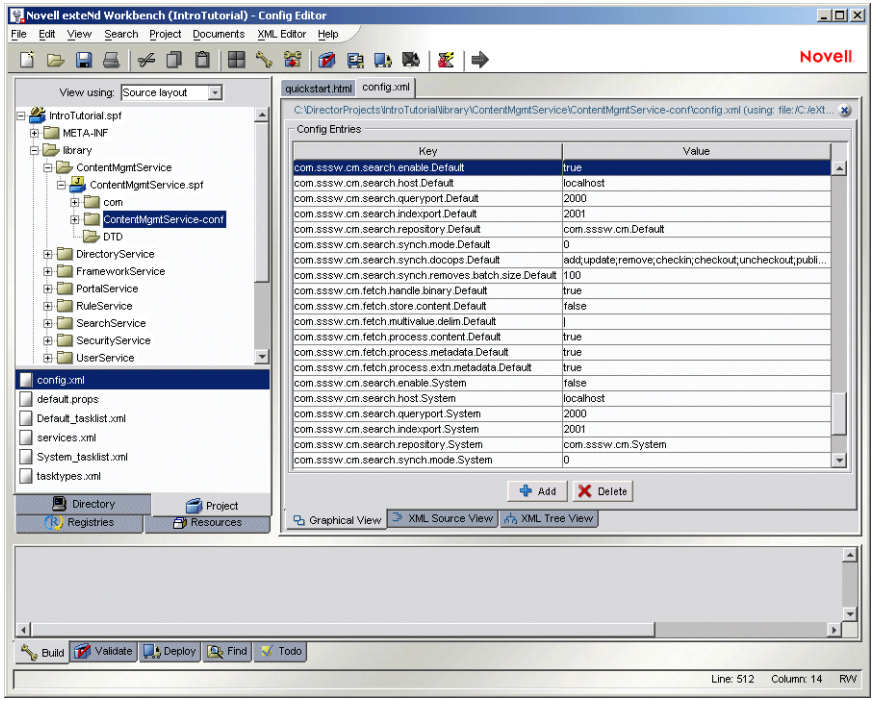
You can set these options on the Content Management Search Options panel in the exteNd Director Project Wizard during custom setup. This procedure is described in the chapter on configuring exteNd Director applications in *Developing exteNd Director Applications*.

You cannot modify these options when you create a standard exteNd Director project, because they are set to default values automatically. However, you can override these defaults after you create your project, as described in "Setting search options in an existing exteNd Director project" on page 37.

Setting search options in an existing exteNd Director project

You can modify settings for all search options in an existing exteNd Director project by editing the Content Management (CM) configuration file **config.xml**. When you modify options in this file, you must redeploy your project for the changes to take effect.

Here is a snapshot of this configuration file in graphical view:



As you can see, this configuration file contains a set of properties, each described by a key and a value.

Here is an example of the XML specification for the search option highlighted in the screen above. This option enables Autonomy-based conceptual search capability in the default CM repository:

```
<property>
<key>com.sssw.cm.search.enable.Default</key>
<value>true</value>
</property>
```

 For detailed descriptions of all search options, see Chapter 11, “Search Options Reference”.

Defining options for a specific Content Management repository

There is a subset of search options that must be defined for each Content Management (CM) repository that you use:

- Enable link to the Search subsystem?
- Index custom document metadata?

- Index document content?
- Index standard document metadata?
- Index port
- Name of DRE host
- Number of deleted documents to batch up
- Operations that trigger immediate synchronization
- Query port
- Support binary document formats?
- Symbol for concatenating multivalue custom metadata values before indexing
- Synchronization mode

If you scroll through the Content Management configuration file, you will see that these search options have repository names appended to their keys. For example, options defined for the Default CM repository have keys that end with the text string **Default**.

Setting search options programmatically at runtime

You can set a number of search options programmatically at runtime by writing exteNd Director components that call methods defined for the **EbiDataFetcherDelegate** class in the Search API. These settings remain in effect only for the duration of the runtime session.

These are the options that can be modified at runtime:

Option	EbiDataFetcherDelegate method
Name of DRE database	setDestRepository()
Name of DRE host	setHost()
Index port	setIndexPort()
Symbol for concatenating multivalue custom metadata values before indexing	setMultiValueDelim()
Index document content?	setProcessContent()
Index custom document metadata?	setProcessExtnMeta()
Index standard document metadata?	setProcessMeta()
Query port	setQueryPort()
Copy document contents into the DRE?	setStoreContent()

For more information about these options, see Chapter 11, "Search Options Reference".

Setting search options by modifying the DRE configuration file

You can set specific search options by modifying DRE parameters directly in the DRE configuration file located in autonomy\engine\DirectorDRE.cfg in the exteNd Director installation directory.

After changing settings in this file, you must restart the DRE (as described in "Resetting the DRE" on page 116) and reindex the data (as described in "Forcing indexing" on page 99).

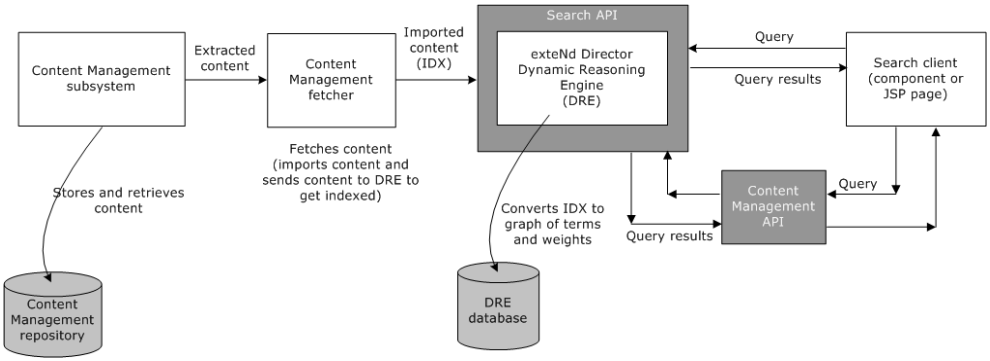
4 Implementing Conceptual Search

This chapter takes you through the steps of the process flow for implementing Autonomy-based conceptual searching in your exteNd Director applications.

The following topics are covered:

Searching the CM repository: how the Search and Content Management APIs are integrated

exteNd Director comes with a data fetcher that allows you to conduct Autonomy-based searches exclusively on content and metadata stored in the exteNd Director Content Management (CM) repository. The CM subsystem communicates with the exteNd Director Dynamic Reasoning Engine (DRE) through the Search API, as illustrated in this process flow diagram:



In this scenario, you implement conceptual search in your exteNd Director applications by using CM API classes that wrap the Search API. These wrapper classes provide methods for constructing and executing queries on content and metadata that reside in the CM repository and have been indexed by the exteNd Director DRE.

The CM fetcher performs the following search functions automatically:

- Imports data from the CM repository into the exteNd Director DRE for indexing—a process called **fetching**, as described in Chapter 5, "Fetching Content and Metadata". Data must be fetched before it becomes available for searching.
- Synchronizes the CM repository and the corresponding DRE database as you change content and metadata. You can control the mode and frequency of the synchronization process, as described in "Synchronization mode" on page 134 and "Operations that trigger immediate synchronization" on page 132.

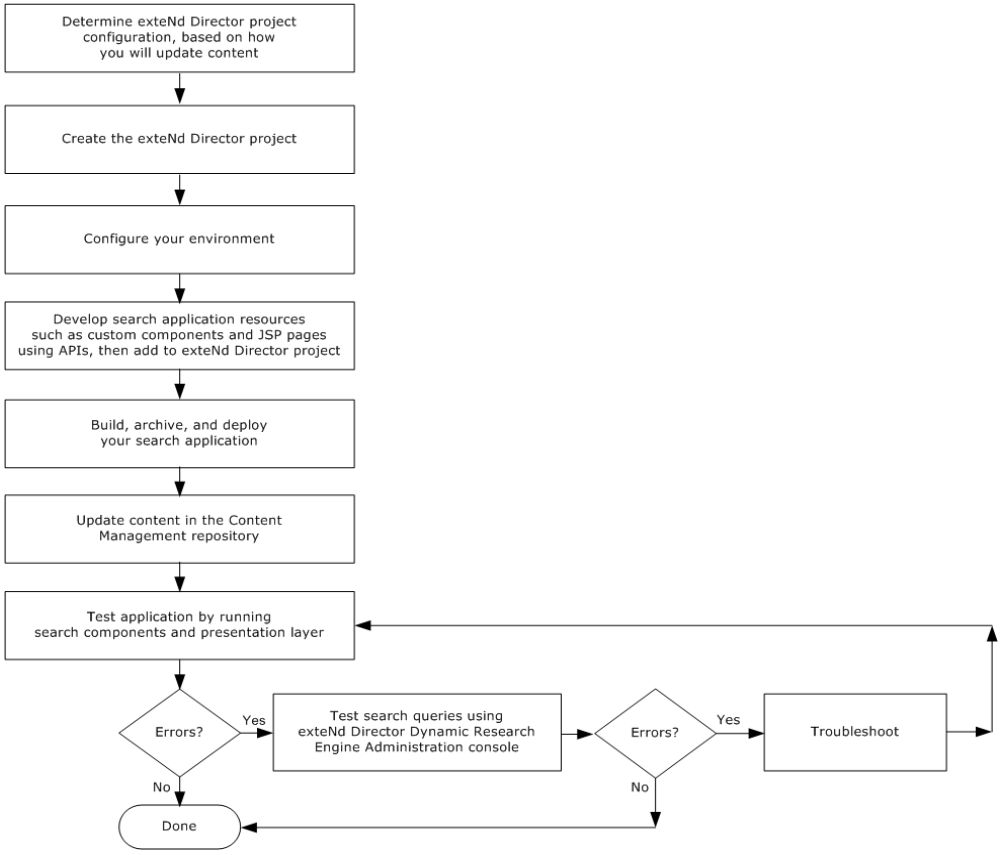
Searching data sources other than the CM repository

To use Autonomy technology with exteNd Director to search data sources other than the CM repository, you must work out legal and contractual issues to purchase additional fetcher products from Autonomy, Inc. You can then use the exteNd Director Search API to fetch and query content from the custom data source.

Alternatively, you can import your custom data into the CM repository and use the CM wrapper classes for implementing Autonomy-based conceptual search.

The process flow for implementing conceptual searching

The diagram below presents the recommended process flow for implementing support for Autonomy-based searching of content in the CM repository. Subsequent sections describe each of these tasks in detail.



Configuring your project and search environment

To use Autonomy-based conceptual search capabilities with the CM subsystem, you must configure your environment by:

- Installing the exteNd Director Dynamic Reasoning Engine
- Installing the exteNd Director Dynamic Reasoning Engine
- Adding the Autonomy Java Native Interface to your environment
- Enabling conceptual search
- Setting security options
- Setting search options

See Chapter 2, "Configuring Your Environment for Conceptual Searching" for detailed information about these configuration tasks.

Developing application resources

The Search and CM subsystems provide APIs that allow you to develop application resources such as search components, JSP pages, and servlets for implementing Autonomy-based search functionality in your exteNd Director applications.

The Search API provides wrapper classes around the Autonomy API that you can use to fetch content from custom data sources, then construct and execute conceptual-style queries against this data.

The CM API wrappers the Search API to provide classes and methods for searching the CM repository in particular.

Implementing search operations

This section describes search operations you can implement in exteNd Director applications.

Types of operations

Here are the key operations you can implement in your exteNd Director applications using the Search API:

Operation	Description	For more information
Fetch	Import structured and unstructured data into the query engine where it is indexed for querying	See Chapter 5, "Fetching Content and Metadata"

Operation	Description	For more information
Query	Create and execute queries on indexed content and process results	See Chapter 6, "Querying Content and Metadata"and Chapter 12, "Search Query Types"

📖 For in-depth descriptions of how to use the Search classes and methods—along with illustrative code examples—see Chapter 5, "Fetching Content and Metadata", Chapter 6, "Querying Content and Metadata", and the *API Reference*.

The relationship between indexing and querying

The exteNd Director DRE queues its indexing jobs and executes them asynchronously. As a result, certain documents may take longer to index than others and, consequently, may not be available for querying immediately. Therefore, when implementing Autonomy-based searching, leave a time window that allows the indexing process to finish before you issue queries.

Programming practices

This section describes the best practices for using the Search API and CM API to develop application resources that implement Autonomy-based searching in your exteNd Director applications.

Using delegates

A *delegate* is a wrapper that hides the location of a service. The delegate model follows the J2EE Business Delegate pattern.

When you use a delegate, you do not need to know whether the service is using a local manager object or an EJB. The delegate initially attempts to instantiate a local manager. If this fails, it attempts to use an EJB instead. This approach allows developers to use the same code on clients and servers to instantiate services.

📖 For more information about delegates, see the chapter on coding Java for exteNd Director applications in *Developing exteNd Director Applications*.

The CM API provides a delegate interface for managing search operations in the CM repository; the Search API provides delegates for managing search activities for custom data sources. When implementing Autonomy-based searching in exteNd Director applications, it is recommended that you use these delegates as follows:

When the data source is	Use
CM repository	◆ com.sssw.cm.api.EbiContentMgmtDelegate
Custom	◆ com.sssw.search.api.EbiDataFetcherDelegate ◆ com.sssw.search.api.EbiQueryEngineDelegate

 To learn how to use these objects, see Chapter 5, "Fetching Content and Metadata" and Chapter 6, "Querying Content and Metadata".

Logic flow

Here is the recommended logic flow for implementing search in your exteNd Director application. Add this logic to the getComponentData() method of your component:

1 Fetch data—that is, import content and metadata from your data source into the exteNd Director DRE for indexing.

exteNd Director comes with a data fetcher for the CM repository. When you use this repository as your data source, the fetching process is done automatically.

If you license other fetchers from Autonomy to work with outside data sources, you need to initiate the fetch process programmatically. Follow these guidelines:

If you want to search	Do this
CM repository	Specify how to synchronize the CM repository with the associated exteNd Director DRE database to ensure that all updates are imported and indexed in a timely manner. You can: ◆ Select one of two synchronization modes—immediate or batch—as described in "Synchronization mode" on page 134. ◆ Specify triggers, as described in "Operations that trigger immediate synchronization" on page 132.

If you want to search	Do this
Data sources other than the CM repository	1 Work out legal and contractual issues with Autonomy, Inc. One option is to purchase additional fetcher products from Autonomy, Inc. 2 Create a descriptor for each data fetcher, as described in "Data fetcher descriptors" on page 56. 3 For each data source, instantiate an object that implements the EbiDataFetcherDelegate interface. 4 Call the fetchData() method on each EbiDataFetcherDelegate object to import data from the data source into the associated exteNd Director DRE for indexing. 📖 For more information, see Chapter 5, "Fetching Content and Metadata".

2 Instantiate a blank query object:

If you want to search	Use
CM repository	`com.sssw.cm.factory.EboFactory.getQuery()`
Custom data sources	`com.sssw.search.factory.EboFactory.getQuery()`

3 Set the query type.

📖 For descriptions of Autonomy-based query types, see Chapter 12, "Search Query Types".

4 Specify the query string.

📖 For syntax, see Chapter 12, "Search Query Types".

5 Set other parameters such as **maximum number of results to return** and **relevance cut**.

TIP: You need to call methods on com.sssw.search.api.**EbiQuery**.

6 Get an object for running the query:

If you want to search	Do this
CM repository	Get an object that implements the EbiContentMgmtDelegate interface, as described in "Querying the CM repository" on page 60.
Custom data sources	Get an object that implements the EbiQueryEngineDelegate interface, as described in "Querying custom data sources" on page 60.

7 Run the query.

8 Process the results.

 For details on implementing these steps, see Chapter 5, "Fetching Content and Metadata" and Chapter 6, "Querying Content and Metadata".

Code example: querying the CM repository

The following code segment demonstrates how to construct and execute an Autonomy-based search query against the CM repository:

```
...
//Instantiate a blank query object
com.sssw.search.api.EbiQuery query = com.sssw.cm.factory.EboFactory.getQuery();

//Specify the query type
query.setQueryType(query.QUERY_TYPE_TEXT);
query.setText("animal+mammal");

//Select all columns
query.selectAll();

//OR ... Select individual columns, like doc id and title
//query.select(com.sssw.cm.core.EbiCmConstants.DOCID);
//query.select(com.sssw.cm.core.EbiCmConstants.TITLE);

//Ask for a maximum of 50 results
query.setMaxNumResults(50);

//Ask for results that are at least 80% relevant
query.setRelevanceCut(80);

//Get the content manager delegate
EbiContentMgmtDelegate contentMgr =
com.sssw.cm.client.EboFactory.getDefaultContentMgmtDelegate();

//Run the query
Iterator iterResults = contentMgr.runQuery(context, query, null, true).iterator();
```

```
//Process results
while (iterResults.hasNext()) {
   com.sssw.cm.api.EbiQueryResult res = (EbiQueryResult)iterResults.next()
   System.out.println("DOCID:" + res.getID());
   System.out.println("TITLE:" + res.getTitle());
   String content = (res.getData() != null) ? new String(res.getData()) : "none";
   System.out.println("CONTENT:" + content);
   System.out.println("RELEVANCE:" + res.getIntegerProperty(res.PROP_DOC_WEIGHT));
   System.out.println("QUICK SUMMARY:" + res.getProperty(res.PROP_DOC_QUICK_SUMMARY));
}
...
```

Interacting with the CM repository

If you plan to add and update content in the CM repository programmatically using the CM API, you also need to write components and related resources that implement this logic. To learn about all the ways to interact with the CM repository, see "Updating content in the CM repository" on page 50.

Packaging application resources

As you begin to develop your search application resources, you must make decisions about how to package them inside your exteNd Director project. Follow the guidelines in the chapter on using resource sets in *Developing exteNd Director Applications*.

After you have incorporated your custom resources in the exteNd Director project, you are ready to deploy the application to your application server, as described in "Building, archiving, and deploying your application" next.

Building, archiving, and deploying your application

You build, archive, and deploy your application in exteNd Director just as you would any J2EE application.

📖 For server-specific guidelines, see the chapter on deploying exteNd Director applications in *Developing exteNd Director Applications*.

Updating content in the CM repository

There are several ways to add and update content in the CM repository, each described in this section. You can use any combination of these methods as long as you include the appropriate subsystems in your exteNd Director project, as described in "Determining your exteNd Director project configuration" on page 27.

Updating content in the CMS Administration Console

The CMS Administration Console is part of the exteNd Director Web tier, a prebuilt Web application that provides a graphical user interface for creating, updating, and publishing content in the CM repository. When you create content in the CM Administration Console, you can take advantage of the exteNd Director CM features designed to facilitate searching—including the ability to define and associate custom metadata with documents.

📖 For more information, see the chapter on the CMS Administration Console in the *Content Management Guide*.

Updating content using the CM API

You use classes and methods in the CM API to write exteNd Director components, servlets, and JSP pages to create, update, and publish content in the CM repository.

📖 For more information, see the chapter on managing documents in the *Content Management Guide*.

Creating and updating content in third-party applications

You can create and update content in third-party applications. Before you can perform conceptual searches on this content, you must take additional steps:

- Ensure that your third-party application produces content in a format that can be searched by the Search subsystem.

  📖 For information about configuring search formats, see "Importable MIME types" on page 128 and "Importable file extensions" on page 127.

- Import and publish the third-party content in the CM repository.

If your third-party application is a WebDAV-enabled client, you can use the WebDAV subsystem to transfer content into the CM repository, as described in the chapter on using WebDAV clients with exteNd Director for collaborative authoring in the *Content Management Guide*. In this case, make sure you include the WebDAV subsystem when you create your exteNd Director project.

IMPORTANT: Third-party content imported into the CM repository via WebDAV is saved as **system resources**. A limitation of system resources is that you cannot define custom metadata for them. However, you can still search their content and standard metadata. This limitation does not apply if you use the exteNd Director content import utility or create your own WebDAVclient using the client API provided in the exteNd Director WebDAV subsystem. You can associate custom metadata with content created in this type of custom-designed WebDAV client.

📖 For more information, see the chapters on importing content and building your own WebDAV client in the *Content Management Guide*.

Testing queries

You can use the exteNd Director DRE Administration console to test your queries in isolation before you deploy your application.

📖 For more information, see "Testing queries" on page 117.

Troubleshooting the search application

exteNd Director provides several techniques for debugging your search application and correcting commonly encountered problems.

📖 For more information, see Chapter 8, "Troubleshooting the Conceptual Search Process".

5 Fetching Content and Metadata

This chapter describes fetching and explains how to implement it in your exteNd Director search applications.

The following topics are covered:

- About fetching
- Implementing fetching in your applications
- Data fetcher descriptors

About fetching

Fetching content involves two processes:

- Importing data from the Content Management (CM) repository
- Indexing the data in the exteNd Director DRE

The default fetcher

exteNd Director comes with a data fetcher for the exteNd Director CM repository. The CM fetcher allows you to perform Autonomy-based conceptual searches **only** on data stored in the CM repository.

The CM fetcher initiates the following processes automatically:

Process	Description
Fetching	Imports data from the CM repository into the exteNd Director DRE database for indexing

Process	Description
Synchronization	Propagates updates made in the CM repository to the corresponding DRE database so they can be indexed

Fetching is initiated whenever synchronization occurs. You schedule synchronization to run as a real-time or batch process, as described in "Synchronization mode" on page 134.

For real-time—or immediate—synchronization, you can specify the actions that trigger the process, as described in "Operations that trigger immediate synchronization" on page 132.

By default, synchronization occurs as a real-time process in immediate mode and is triggered when any of the following CM operations occur:

- Add
- Update
- Remove
- Check in
- Check out
- Publish
- Uncheckout
- Unpublish
- Unlock
- Roll back

Fetchers for custom data sources

If you want to use a data source other than the CM repository, you must purchase additional data fetchers from Autonomy, Inc.

When you purchase other data fetchers from Autonomy, you must initiate the fetch process programmatically using the Search API, as demonstrated in "Implementing fetching in your applications" next. You must also create a descriptor for each data fetcher you use, as described in "Data fetcher descriptors" on page 56.

Implementing fetching in your applications

The exteNd Director Search API provides wrapper classes around Autonomy APIs that provide methods for fetching content.

Key fetcher classes and interfaces

Key classes and interfaces for fetching content include:

Class or interface	Description	Package
EbiDataFetcherDelegate	Delegate for accessing objects that implement the EbiDataFetcher interface, which provides methods for importing content from a specified data source into the exteNd Director DRE, where it is indexed	com.sssw.search.api
EboFactory	Factory class that provides methods for getting Search subsystem delegates such as EbiDataFetcherDelegate	com.sssw.search.client

Fetcher methods

This section describes Search API methods that you can use to perform data fetching in your exteNd Director applications.

Getting a fetcher delegate

Here is the method for getting a fetcher delegate:

```
com.sssw.search.client.EboFactory.getDataFetcherDelegate()
```

This method returns an object that implements the EbiDataFetcherDelegate interface. Methods on this object can be used to invoke and manage the data fetching process.

For information about why to use delegates, see "Programming practices" on page 45.

Initiating the fetch process

Here is the method for fetching data:

```
com.sssw.search.API.EbiDataFetcherDelegate.fetchData()
```

This method fetches document data from the source repository into the destination query engine database.

Code example: fetching data

The following code segment shows how to initiate the fetch process using the Search API:

```
...
//Instantiate a data fetcher delegate
com.sssw.search.api.EbiDataFetcherDelegate fetcher =
com.sssw.search.client.EboFactory.getDataFetcherDelegate(whichFetcher);

//Fetch the data
fetcher.fetchData(context, false);
...
```

The getDataFetcherDelegate() method takes a string argument that specifies which data fetcher to instantiate. Each data fetcher requires a descriptor, as described in "Data fetcher descriptors" next.

Data fetcher descriptors

You must create a descriptor for each data fetcher you purchase from Autonomy. exteNd Director provides a descriptor for the CM data fetcher.

You add data fetcher descriptors in the services configuration file **services.xml** for the Search subsystem.

 For more information about where project files are located, see the section on exteNd Director project structure in *Developing exteNd Director Applications*.

Syntax

The syntax of data fetcher descriptors in services.xml sets up a mapping between the data fetcher interface in the Search API and the data fetcher implementation class.

This mapping is illustrated in the descriptor for the CM data fetcher, which is supplied with exteNd Director:

```
<service>
<interface>com.sssw.search.api.EbiDataFetcher.CM</interface>
```

```
<impl-class>com.sssw.cm.core.EboDataFetcher</impl-class>
<description>Data fetcher object for the Content Management
subsystem</description>
<max-instances>0</max-instances>
<startup>M</startup>
<namespaced>false</namespaced>
</service>
```

The string **CM** in the **<interface>** element is an example of a data fetcher specifier that you pass as a string argument to the getDataFetcherDelegate() method to initiate the fetch process, as described in "Code example: fetching data" on page 56.

This is the syntax to use for any data fetcher descriptor. For example, if you purchase or create a custom data fetcher called MYCO that is defined by the class com.sssw.myco.Fetcher, you need to add a descriptor that looks like this:

```
<service>
<interface>com.sssw.search.api.EbiDataFetcher.MYCO</interface>
<impl-class>com.sssw.myco.Fetcher</impl-class>
<description>Data fetcher object for MYCO</description>
<max-instances>0</max-instances>
<startup>M</startup>
<namespaced>false</namespaced>
</service>
```

Associating data fetchers with query result objects

For each data fetcher you add, you can optionally add an associated query result object descriptor. Here is the descriptor in services.xml for the query result object already defined for the CM data fetcher:

```
<service>
<interface>com.sssw.search.api.EbiQueryResult.CM</interface>
<impl-class>com.sssw.cm.core.EboQueryResult</impl-class>
<description>Query result object for the Content Management
subsystem</description>
<max-instances>0</max-instances>
<startup>M</startup>
<namespaced>false</namespaced>
</service>
```

To associate the CM query result object with your MYCO data fetcher, you need to add a descriptor that looks like this:

```
<service>
<interface>com.sssw.search.api.EbiQueryResult.MYCO</interface>
<impl-class>com.sssw.cm.core.EboQueryResult</impl-class>
<description>Query result object for MYCO</description>
<max-instances>0</max-instances>
<startup>M</startup>
<namespaced>false</namespaced>
</service>
```

Use this same syntax to create a descriptor for a custom query result object, as follows:

```
<service>
<interface>com.sssw.search.api.EbiQueryResult.MYCO</interface>
<impl-class>com.sssw.myco.webapp.impl.MycoQueryResult</impl-class>
<description>Query result object for MYCO</description>
<max-instances>0</max-instances>
<startup>M</startup>
<namespaced>false</namespaced>
</service>
```

6 Querying Content and Metadata

This chapter describes the content query process and explains how to implement querying in your exteNd Director search applications.

The following topics are covered:

About querying

Queries are structured expressions that you can use to search content from a data source. Autonomy-based search capabilities in exteNd Director allow you to query both content and metadata **using a single query expression**, rather than requiring you to write separate queries for each type of data.

Querying metadata In exteNd Director you can query two types of metadata:

- **Standard (basic) metadata** is descriptive information about content that is automatically attached to every document. Examples of standard metadata are title, author, and creation date.

- **Custom (extension) metadata** is application-specific information about content that you define in the Content Management (CM) subsystem as fields in document types.

Querying content You can query content only if it has been published.

Querying the CM repository

exteNd Director comes with a data fetcher that allows you to conduct Autonomy-based searches exclusively on content and metadata stored in the exteNd Director CM repository.

To query the CM repository, you use the CM subsystem in conjunction with the Search subsystem. The CM API provides classes that wrapper relevant search functions associated with the CM repository, as described in "Implementing querying for the CM repository" on page 61.

Querying custom data sources

To use Autonomy technology with exteNd Director to search data sources other than the CM repository, you must purchase additional data fetchers from Autonomy, Inc.

To query custom data sources, you must use Search API classes to instantiate a query object and run the query against the other data sources you are licensed to use, as described in "Implementing querying for custom data sources" on page 66.

Alternatively, you can import your custom data into the CM repository and use the CM wrapper classes for implementing Autonomy-based conceptual queries.

Types of queries you can run

The exteNd Director Search subsystem supports the following types of queries:

- Boolean queries
- Conceptual queries
- Field queries
- Fuzzy queries
- Get-all queries
- Keyword search
- Proper name search
- Proximity queries
- Suggest similar documents
- Thesaurus queries

📖 For detailed descriptions of each type of query—including syntax definitions and code examples showing how to specify each query type—see Chapter 12, "Search Query Types" .

Implementing querying for the CM repository

To implement Autonomy-based conceptual and keyword search in your exteNd Director applications, you use CM API functionality that wrappers the relevant Search APIs:

This CM class	Does this	To this search class
`com.sssw.cm.api.EbiContentMgmtDelegate.runQuery()`	Wrappers	`com.sssw.search.api.EbiQueryEngineDelegate.runQuery()`
`com.sssw.cm.api.EbiQueryResult`	Extends	`com.sssw.search.api.EbiQueryResult`

The wrapper classes provide methods for constructing and running queries on content and metadata that reside in the CM repository and have been indexed by the exteNd Director (Dynamic Reasoning Engine) DRE.

In addition, you can configure your environment to manage the processes of document fetching and querying, as described in Chapter 3, "Setting Search Options".

Key classes and interfaces for querying the CM repository

Key classes and interfaces for querying the CM repository include:

Class or interface	Description	Package
EbiContentMgmtDelegate	Delegate for accessing objects that implement the EbiContentManager interface **NOTE:** EbiContentManager is an interface that provides methods for accessing standard metadata, custom metadata, and content in the CM repository	com.sssw.cm.api

Class or interface	Description	Package
EbiQuery	Interface that provides methods for constructing various types of Autonomy-based queries and setting query properties	com.sssw.search.api
EbiQueryResult	Interface that provides methods for processing the results of Autonomy-based queries of content and metadata in the CM repository	com.sssw.cm.api
EboFactory	Factory class that provides methods for getting content manager delegates	com.sssw.cm.client
EboFactory	Server-side factory class that provides methods for instantiating objects used by the CM subsystem	com.sssw.cm.factory

Methods for querying the CM repository

This section describes the CM API methods you can use to query the CM repository in your exteNd Director applications.

Getting a content manager delegate

Here is the method for getting a content manager delegate:

```
com.sssw.cm.client.EboFactory.getDefaultContentMgmtDelegate()
```

This method returns a content manager delegate associated with the default CM repository. The content manager delegate provides methods for running Autonomy-based queries on document content and metadata in this repository.

📖 For information about why to use delegates, see "Programming practices" on page 45.

Instantiating query objects for the CM repository

Autonomy-based queries are based on the EbiQuery interface—an interface that resides in the Search API. To search content and metadata in the CM subsystem, you must instantiate a query object that not only implements this interface but also is associated with the CM repository. The CM API provides the method to use:

```
com.sssw.cm.factory.EboFactory.getQuery()
```

Using this query object, you can call Search API methods to construct Autonomy-based queries and fine-tune search results, as described in "Constructing queries for the CM repository" on page 63.

Constructing queries for the CM repository

Here are key methods for constructing queries for the CM repository:

Method	Description
com.sssw.search.API.EbiQuery.setQueryType()	Specifies the type of query you want to run 📖 For more information, see Chapter 12, "Search Query Types"
com.sssw.search.API.EbiQuery.setQueryText()	Specifies the query string
com.sssw.search.API.EbiQuery.setMaxNumResults()	Sets the maximum number of results to return
com.sssw.search.API.EbiQuery.setRelevanceCut()	Sets the minimum relevance criteria for query results

NOTE: You use the same methods for constructing Autonomy-based queries for custom data sources. The difference is that you call these methods on a query object instantiated from a factory in the Search API, as described in "Implementing querying for custom data sources" on page 66.

Issuing queries against the CM repository

Here is the method for querying the CM repository:

```
com.sssw.cm.api.EbiContentMgmtDelegate.runQuery()
```

This method runs a query that you construct using the com.sssw.search.api.EbiQuery interface and returns the results as a collection of objects that implements the com.sssw.cm.api.EbiQueryResult interface.

Code example: issuing an Autonomy-based query against the CM repository

The following code segment demonstrates how to instantiate a query object and run a query against the default CM repository, called **Default**:

```
...
public void getComponentData( EbiPortalContext context, java.util.Map params ) throws
com.sssw.fw.exception.EboUnrecoverableSystemException
{
   //Declare a string buffer
   StringBuffer sb = new StringBuffer();

   //Set the query string
   String queryString = "The+effect+of+the+recession+on+consumer+spending";

   try
   {
      //Create a blank query object
      com.sssw.search.api.EbiQuery query = com.sssw.cm.factory.EboFactory.getQuery();

      //Set query type to text
      query.setQueryType(query.QUERY_TYPE_TEXT);

      //Specify the query string; this is a conceptual query
      query.setText(queryString);

      //Ask for a maximum of 50 results
      query.setMaxNumResults(50);

      //Ask for results that are at least 80% relevant
      query.setRelevanceCut(80);

      //Ask to return all available document properties in the results
      query.selectAll();

      //Get the content manager delegate
      EbiContentMgmtDelegate contentMgr =
com.sssw.cm.client.EboFactory.getDefaultContentMgmtDelegate();

      //Run the query
      //The boolean argument in runQuery indicates whether results should be filtered
      Iterator iterResults = contentMgr.runQuery(context, query, true).iterator();

      //Process the results
      while (iterResults.hasNext())
      {
         com.sssw.cm.api.EbiQueryResult res =
(com.sssw.cm.api.EbiQueryResult)iterResults.next();
         //Get document metadata
         String docTitle = res.getTitle();
         java.sql.Timestamp dateCreated = res.getDateCreated();
```

```
            //Get document content
            String docAbstract = res.getAbstract();

            //Add query result to the string buffer returned by the component
            sb.append("\n").append(docTitle).append(dateCreated).append(docAbstract).append("
\n");
         }
      }
      catch (Exception _E)
      {
         System.out.println ("Query failed");
         _E.printStackTrace();
      }
      //Set content type
      context.setContentType(com.sssw.portal.api.EbiComponentConstants.MIME_TYPE_HTML_UTF8);

      //Place the content into the context
      context.setComponentContent( sb.toString() );
}
...
```

As you can see, this component retrieves both standard metadata—the document title and the date created—and content from the query results. By default, the exteNd Director DRE is configured to index both types of information. This behavior is controlled by two search options that are enabled by default in the CM subsystem configuration file:

- `com.sssw.cm.fetch.process.content.`*`repository name`*
  See the description in "Index document content?" on page 129.
- `com.sssw.cm.fetch.process.metadata.`*`repository name`*
  See the description in "Index standard document metadata?" on page 129.

The CM subsystem provides many other options that you can configure to customize your search environment, as described in Chapter 3, "Setting Search Options".

Implementing querying for custom data sources

The exteNd Director Search API provides wrapper classes around Autonomy APIs that provide methods for querying content in data sources other than the CM repository.

IMPORTANT: To use Autonomy technology with exteNd Director to search other data sources, you must purchase additional data fetchers from Autonomy, Inc.

Key query classes and interfaces for querying custom data sources

Key classes and interfaces for querying custom data sources include:

Class or interface	Description	Package
EbiQueryEngineDelegate	Delegate for accessing objects that implement the EbiQueryEngine interface, which provides methods for configuring the query engine and processing queries **NOTE:** EbiQueryEngine is an interface that provides methods for interacting with the DRE	com.sssw.search.api
EbiQuery	Interface that provides methods for constructing various types of queries and setting query properties	com.sssw.search.api
EbiQueryResult	Interface that provides methods for processing the results of queries executed by the Search subsystem	com.sssw.search.api
EboFactory	Factory class that provides methods for getting Search subsystem delegates such as EbiQueryEngineDelegate	com.sssw.search.client

Class or interface	Description	Package
EboFactory	Server-side factory class that provides methods for instantiating objects used by the Search subsystem—such as an EbiQuery object	com.sssw.search.factory

Query methods

This section describes Search API methods that you can use for querying custom data sources in your exteNd Director applications.

Getting a query engine delegate

Here is the method for getting a query engine delegate:

```
com.sssw.search.client.EboFactory.getQueryEngineDelegate()
```

This method returns an object that implements the EbiQueryEngineDelegate interface. Methods on this object can be used to configure the query engine and run queries.

📖 For information about why to use delegates, see "Programming practices" on page 45.

Instantiating query objects for custom data sources

Autonomy-based queries are based on the EbiQuery interface that resides in the Search subsystem API. To search content and metadata in custom data sources, you must instantiate a query object that implements this interface. Here is the method to use:

```
com.sssw.search.factory.EboFactory.getQuery()
```

Using this query object, you can call Search API methods to construct Autonomy-based queries and fine-tune search results.

Constructing queries for custom data sources

Here are key methods for constructing Autonomy-based queries for custom data sources:

Method	Description
com.sssw.search.API.EbiQuery.setQueryType()	Specifies the type of query you want to run 📖 For more information, see Chapter 12, "Search Query Types"

Method	Description
com.sssw.search.API.EbiQuery.setQueryText()	Specifies the query string
com.sssw.search.API.EbiQuery.setMaxNumResults()	Sets the maximum number of results to return
com.sssw.search.API.EbiQuery.setRelevanceCut()	Sets the minimum relevance criteria for query results

NOTE: You use the same methods for constructing Autonomy-based queries for the CM repository. The difference is that you call these methods on a query object instantiated from a factory in the CM API, as described in "Implementing querying for the CM repository" on page 61.

Issuing queries against custom data sources

Here is the method for issuing queries:

```
com.sssw.search.api.EbiQueryEngineDelegate.runQuery()
```

This method runs a query that you construct using the com.sssw.search.api.EbiQuery interface and returns the results as a collection of objects that implements the com.sssw.search.api.EbiQueryResult interface.

Code example: issuing an Autonomy-based query against a custom data source

The following code segment presents the getComponentData() method of an exteNd Director component that implements the logic for issuing an Autonomy-based conceptual query against a custom data source:

```
...
public void getComponentData( EbiPortalContext context, java.util.Map params ) throws
com.sssw.fw.exception.EboUnrecoverableSystemException
{
   //Declare a string buffer
   StringBuffer sb = new StringBuffer();

   //Set the query string, using syntax for a conceptual query
   String queryString = "physician+specialty+orthopaedics";

   try
   {
      //Create a blank query object
      com.sssw.search.api.EbiQuery query = com.sssw.search.factory.EboFactory.getQuery();

      //Set query type to text
      query.setQueryType(query.QUERY_TYPE_TEXT);
```

```
      //Specify the query string; this is a conceptual query
      query.setText(queryString);

      //Ask for a maximum of 50 results
      query.setMaxNumResults(50);

      //Ask for results that are at least 80% relevant
      query.setRelevanceCut(80);

      //Ask to return all available document properties in the results
      query.selectAll();

      //Get the query engine delegate
      EbiQueryEngineDelegate qe =
com.sssw.search.factory.EboFactory.getQueryEngineDelegate();

      //Run the query
      Iterator iterResults = qe.runQuery(context, query, null, true).iterator();

      //Process the results
      while (iterResults.hasNext())
      {
         com.sssw.search.api.EbiQueryResult res =
(com.sssw.search.api.EbiQueryResult)iterResults.next();

         //Get document metadata
         String docTitle = res.getTitle();
         java.sql.Timestamp dateCreated = res.getDateCreated();

         //Get document abstract
         String docAbstract = res.getAbstract();

         //Add query result to the string buffer returned by the component
         sb.append("\n").append(docTitle).append("\n").append(dateCreated).append("\n").ap
pend(docAbstract).append("\n");
      }
   }
   catch (Exception _E)
   {
      System.out.println ("Query failed");
      if (m_log.isError())
         m_log.error(_E);
   }
   //Set content type
   context.setContentType(com.sssw.portal.api.EbiComponentConstants.MIME_TYPE_HTML_UTF8);

   //Place the content into the context
   context.setComponentContent( sb.toString() );
}
...
```

Search query descriptors

You can construct search query descriptors as XML files that can be used to initialize the search query object. The XML for a search query definition must conform to the rules specified in **search-query-def_4_0.dtd**, a file that resides in the **DTD** folder within the **SearchService.jar** of your exteNd Director project **library** folder.

Advantages of using query descriptors

There are several advantages to initializing a query object programmatically from an XML query descriptor:

- Query descriptors are reusable.
- You set all desired options with one method call, rather than making individual calls to define specific query properties.

Query type element

Every search query definition contains an element for specifying the query type:

```
<!ELEMENT search-query-def (text-query | fuzzy-query | get-all-
query | suggest-query | name-search-query)>
```

In turn, each query type element provides properties for refining the query. For example, consider the text query element:

```
<!ELEMENT text-query (query-text?, field-spec?, query-options?,
selected-props?)>
```

Using this definition, you can construct a field query by defining a field specifier list in the **field-spec** property that indicates which metadata to search for the text defined in **query-text**.

Here are the XML definitions for other query types:

- Fuzzy queries:

  ```
  <!ELEMENT fuzzy-query (query-text?, field-spec?, query-options?,
  selected-props?)>
  ```

- Get-all queries:

  ```
  <!ELEMENT get-all-query (field-spec?, query-options?, selected-
  props?)>
  ```

- Suggest similar documents:

  ```
  <!ELEMENT suggest-query (doc-list, suggest-options?, field-
  spec?, query-options?, selected-props?)>
  ```

- Proper name search:

  ```
  <!ELEMENT name-search-query (query-text?, field-spec?, query-
  options?, selected-props?)>
  ```

The Search API provides a method for setting query type at runtime—setQueryType()—that you call on the EbiQuery object.

 For a detailed description of each type of query—including syntax definitions and code examples showing how to specify each query type—see Chapter 12, “Search Query Types” .

Query options property

Each query type includes a **query-options** property that allows you to fine-tune query behavior. Here is the XML definition for query-options:

```
<!ELEMENT query-options (
    batch-options?,
    date-range?,
    exclusions?,
    generate-quick-summary?,
    thesaurus-options?,
    max-num-results?,
    relevance-cut?,
    sort-by-date?,
    sort-by-relevance?,
    use-abs-weight?
)>
```

For each of these options, the Search API provides methods that you can call on the EbiQuery object for setting options individually at runtime. Here is a description of each option:

Query option	Description	Associated method
batch-options	Return the result set in batches of a particular size	setBatchOptions()
date-range	Search within the specified range of document creation dates	setDateRange()
exclusions	Exclude the specified documents from the query results	setExclusions()
generate-quick-summary	Generate quick summaries for each item in the result set	setGenerateQuickSummary()
thesaurus-options	Set up a thesaurus repository for thesaurus queries	setThesaurus()
max-num-results	Set the maximum number of results to return	setMaxNumResults()

Query option	Description	Associated method
relevance-cut	Set the relevance cut (the minimum similarity score) for query results	setRelevanceCut()
sort-by-date	Sort query results by date	setSortByDate()
sort-by-relevance	Sort query results by relevance	setSortByRelevance()
use-abs-weight	Return relevance scores as absolute weights rather than percentages	setUseAbsWeight()

Selected properties

The **selected-props** property for query types allows you to specify the document properties to return in the query results. Here is the XML definition for selected-props:

```
<!ELEMENT selected-props (prop-name* | select-all)>
```

Using this definition, you can specify that your query return individual document properties or all available document properties.

In addition, you can call the following Search API methods on the EbiQuery object to specify document properties at runtime:

Method	Description
select()	Select the specified document property to return in the query results
selectAll()	Return all available document properties in the query results
selectAlways()	Always return the specified document property in the query results
removeSelect()	Remove the specified property from the list of selected properties

Example: defining a text query in XML

Here is an example of a text query defined in XML:

```
<search-query-def>
    <text-query>
        <query-text><![CDATA[clinical+trials+diabetes+research]]></query-text>
        <field-spec>
```

```
            <field-spec-list><![CDATA[fnameTITLE=*report*+fnameCountry=*USA*]]></field-
spec-list>
            <field-boolean-expr><![CDATA[fnameTITLE+AND+fnameCountry]]></field-boolean-
expr>
        </field-spec>
        <query-options>
            <date-range>
                <date-from><![CDATA[11/01/2002]]></date-from>
                <date-to><![CDATA[11/02/2002]]></date-to>
                <date-pattern><![CDATA[dd/MM/yyyy]]></date-pattern>
            </date-range>
            <generate-quick-summary/>
            <max-num-results><![CDATA[50]]></max-num-results>
            <relevance-cut><![CDATA[70]]></relevance-cut>
            <sort-by-date/>
            <sort-by-relevance/>
        </query-options>
        <selected-props>
            <prop-name><![CDATA[AUTHOR]]></prop-name>
            <prop-name><![CDATA[TITLE]]></prop-name>
            <prop-name><![CDATA[CREATED]]></prop-name>
        </selected-props>
    </text-query>
</search-query-def>
```

Based on this definition, this query is not merely a simple text query—or keyword search—but instead has the following characteristics:

Characteristic	Description
Performs a conceptual search	The **<query-text>** element specifies a search string in the form of a conceptual query: *word1+word2+word3+... wordN*
Searches content and metadata (field query)	The **<field-spec>** element specifies that the application search all documents that contain the string *report* in their **title** fields and the string *USA* in their **country** fields
Searches within a specified time period	The **<date-range>** element restricts the search to documents created between November 1 and 2, 2002
Specifies a maximum number of results	The **<max-num-results>** element specifies that a maximum of 50 results be returned
Specifies a relevance threshold	The **<relevance-cut>** element requests results that are at least 70% relevant
Returns specific document properties	The **<selected-props>** element indicates that the author, title, and date created should be returned for each document in the result set

This sample XML query definition resides in **search-query-def_4_0_sample.xml**, located in the DTD folder within the SearchService.jar of your exteNd Director project library folder.

Example: initializing a query object from an XML descriptor

Here is sample code that initializes a query object from an XML query descriptor:

```
...
//Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();

//Read in your query XML descriptor
Document queryDesc =
com.sssw.fw.util.EboXmlHelper.getDocumentFromString(myInputStream);

//Initialize the blank query object with data from the XML
//descriptor
query.fromXML(queryDesc.getDocumentElement());
...
```

The getDocumentFromString() method returns a DOM document, converted from a string that represents an XML document—in this case the input argument **myInputStream**.

Sorting query results

This section describes how to sort the results of Autonomy-based conceptual queries.

Sorting by date and then relevance

You can sort query results by date, relevance, or both. When you sort by both properties, the results are first sorted by date, then by relevance.

➢ **To sort by date and then relevance:**

1. Define a com.sssw.search.api.**EbiQuery** object.
2. Call any of the following methods on that object:

Sorting factor	To	Call
Relevance	**Enable** sorting by relevance	setSortBy**Relevance**(**true**)
	Disable sorting by relevance	setSortBy**Relevance**(**false**)
Date	**Enable** sorting by date	setSortBy**Date**(**true**)
	Disable sorting by date	setSortBy**Date**(**false**)

Sorting field query results

You can also sort results of field queries in ascending or descending order by a single parameter. The parameter can be the value of a standard metadata field or custom metadata field.

NOTE: Standard metadata field names are listed in the [Fields] section of the DRE configuration file **DirectorDRE.cfg**, located at autonomy\engine in your exteNd Director installation directory.

➢ **To sort field query results:**

1. Before issuing a field query, make sure you configure your search environment to specify the types of metadata you want to search—standard metadata and/or custom metadata—as described in Chapter 3, "Setting Search Options".
2. In a text editor, open the DRE configuration file **DirectorDRE.cfg**.
3. Enable field sorting by setting the parameter **FIELDSORT=1** in the [Server] section.

   NOTE: If this parameter does not appear, add it to the file.
4. Save and close the configuration file.
5. Reset the DRE, as described in "Resetting the DRE" on page 116.
6. Reindex the data, as described in "Programming practices" on page 45.
7. Specify the sort parameter by appending one of these expressions to the field specifier list you created for your field query:

Sort expression	Description
`&fsort=`*`FIELDNAME`*	Sort in ascending order by the value of the field *FIELDNAME*

Sort expression	Description
`&fsort=-`*`FIELDNAME`*	Sort in descending order by the value of the field *FIELDNAME*

For example, suppose that in your CM repository you define a document type called **Colleges**, and two custom fields—**Ranking** and **Location**. If you want to find all colleges located in Massachusetts, sorted in descending order by rank, your field specifier should look like this:

```
...
String fieldSpecList =
   "fnameDOCTYPENAME=*Colleges*+fnameLocation=*Massachusetts*+
   &fsort=-Ranking";
String fieldBooleanExpr = "fnameDOCTYPENAME+AND+fnameLocation";
query.setFieldSpecList(fieldSpecList, fieldBooleanExpr);
...
```

For more information about constructing and implementing field queries, see “Field queries” on page 137.

7 Configuring the Dynamic Reasoning Engine for Specialized Searching

This chapter explains how to configure your exteNd Director Dynamic Reasoning Engine (DRE) to perform specialized search tasks.

The following topics are covered:

- Searching for numbers
- Searching in other languages

Searching for numbers

This section describes how to enable number searches using the DRE configuration file.

➢ **To enable searching for numbers:**

1. In a text editor, open the DRE configuration file **DirectorDRE.cfg**, located at autonomy\engine in your exteNd Director installation directory.
2. Set the parameter **INDEXNUMBERS=1**.

   NOTE: If this parameter does not appear, add it to the file.
3. Delete the parameter **DONTINDEXNUMBERS=1**.
4. Save and close the configuration file.
5. Reset the DRE, as described in "Resetting the DRE" on page 116.
6. Reindex the data, as described in "Programming practices" on page 45.

Searching in other languages

This section describes how to configure the DRE for searching in other languages, including those that use a multibyte character set (MBCS). By default, the DRE is configured to process English-language data.

➢ **To search in other languages:**

1. Configure your search environment to import multibyte character set (MBCS) and other binary formats, as described in "Importing MBCS and other binary formats" below.
2. Set language-specific configuration parameters, as described in "Modifying language-specific configuration parameters" on page 80.
3. (Optional) Copy sentence-breaking files into the directory where the DRE resides, as described in "Providing sentence-breaking files (optional)" on page 82.
4. Reset the DRE, as described in "Resetting the DRE" on page 116.
5. Reindex the data, as described in "Forcing indexing" on page 99.

Importing MBCS and other binary formats

This section describes how to configure your search environment to import multibyte character set (MBCS) and other binary formats into the DRE for indexing.

You enable MBCS support by configuring Autonomy Omnislave, a plug-in module that converts data from binary file formats so it can be indexed in the DRE.

The Omnislave configuration file is called **omnislave.cfg** and resides in autonomy\OmniSlaves in your exteNd Director installation directory. The omnislave.cfg file contains two types of sections:

Section type	Description
`[CONFIGURATION]`	Provides general configuration settings that apply to all `[`*<file_format>*`]` sections that appear below it
`[`*<file_format>*`]`	Defines settings for specific file formats that you want OmniSlaves to extract

In the following example, formats are defined for Word, RTF, and Microsoft PowerPoint files:

```
[Configuration]
OmniConvertExtns0=*.doc
OmniConvertLibraryCsvs0=wpconvdll.dll,wordconv.dll,rtfconv.dll
OmniConvertConfigSectionCsvs0=WordPerfect,MSWord,Rtf
OmniConvertExtns1=*.rtf
```

```
OmniConvertLibraryCsvs1=rtfconv.dll
OmniConvertConfigSectionCsvs1=Rtf
OmniConvertExtns2=*.ppt
OmniConvertLibraryCsvs2=pptconv.dll
OmniConvertConfigSectionCsvs2=Ppt

Logging=0
LogAppend=TRUE
LogMaxKBytes=500

[MSWord]
OutputCharSet=ASCII

[Rtf]

[Ppt]
OutputCharSet=ASCII
StopList=pptconv.dat
```

➢ **To enable support for MBCS and other binary formats:**

1. Open the **omnislave.cfg** in your favorite text editor.
2. Create `[<file_format>]` sections for each of the file formats you want Omnislave to convert for indexing.
3. In each `[<file_format>]` section, add a parameter **OutputCharSet** and set it to the character set to which you want to convert the file format.

   Choose one of these character set constants:

   - ASCII
   - UTF8
   - UCS2
   - GREEK
   - GREEK_ISO
   - HEBREW
   - HEBREW_ISO
   - ARABIC
   - ARABIC_ISO
   - CYRILLIC
   - CYRILLIC_KO18
   - CYRILLIC_ISO
   - THAI
   - EASTERNEUROPEAN
   - EASTERNEUROPEAN_ISO
   - TURKISH

- CHINESESIMPLIFIED
- CHINESETRADITIONAL
- KOREAN
- SHIFTJIS
- JIS
- EUC

For example, if you want to search a Word document in traditional Chinese, add the following lines of code under the appropriate `[CONFIGURATION]` section in the Omnislave configuration file:

```
[MSWord]
OutputCharSet=CHINESETRADITIONAL
```

Modifying language-specific configuration parameters

You modify language-specific search parameters in the DRE configuration file **DirectorDRE.cfg**, located at autonomy\engine\ in your exteNd Director installation directory.

➢ **To modify language-specific parameters in the DRE:**

1. Open **DirectorDRE.cfg** in your favorite text editor.
2. Set the **CharConv** parameter to the language you want the DRE to use:

Language	Value
European	0 (default)
Japanese	1
Korean	2
Simplified Chinese	4
Traditional Chinese	5
Traditional Chinese indexed as Simplified Chinese	6
Eastern European	7
Russian WINANSI	8
Russian KO18	9
Hebrew	10
Greek	11
Swedish	12

3 Set the **TermSize** parameter to specify the maximum number of characters for any term in the DRE:

Language	Value
English and European languages	10 (default)
German	30
Japanese	30
Korean	40

4 (Optional) Set the **StripLanguage** parameter to select which language to use when stripping terms to their stems (for example, stripping **running** to **run**):

Option	Value
English	0
Conversion from UK to US English	1
no stripping	2
German	3
Italian	4
Russian	5
Advanced English	6
Spanish	7
Dutch	8
Advanced German	9
French	10
Greek	11
Swedish	12
Danish	13
Portuguese	14
Advanced Spanish	15
Norwegian	16

NOTE: Use the advanced settings for English (6) and German (9) when possible. Exception: if you set the StripLanguage to 0 or 1 for English or 3 for German when you indexed content into the DRE, you must use those same settings when you send queries to the DRE.

Providing sentence-breaking files (optional)

When you use languages that do not separate words with spaces, you must specify appropriate delimiters. exteNd Director provides language-specific sentence-breaking files on your product CD that you must copy into the directory where the DRE resides—**autonomy\engine** in your exteNd Director install directory. The following sections describe the sentence-breaking files and associated DRE configuration settings required for languages that do not delimit words with spaces.

Traditional Chinese

The required sentence-breaking files are:

Platform	Sentence-breaking files	Location on CD
NT	◆ chinesebreaking.dll ◆ big5togb.txt ◆ wordlist.txt ◆ chineseconvlist.txt	Autonomy\MBCS\chinese_nt_1_0_3.zip
UNIX	◆ chinesebreaking.so ◆ big5togb.txt ◆ wordlist.txt ◆ chineseconvlist.txt	Autonomy/MBCS/chinese_aix_1_0_2.tar.Z Autonomy/MBCS/chinese_hpux11_1_0_3.tar.Z Autonomy/MBCS/chinese_solaris_1_0_3.tar.Z

The required language-specific configuration settings are:

DRE configuration parameter	Value
CharConv	5
TermSize	40
StripLanguage	2

Simplified Chinese

The required sentence-breaking files are:

Platform	Sentence-breaking files	Location on CD
NT	◆ chinesebreaking.dll ◆ big5togb.txt ◆ wordlist.txt ◆ chineseconvlist.txt	Autonomy\MBCS\chinese_nt_1_0_3.zip
UNIX	◆ chinesebreaking.so ◆ big5togb.txt ◆ wordlist.txt ◆ chineseconvlist.txt	Autonomy/MBCS/chinese_aix_1_0_2.tar.Z Autonomy/MBCS/chinese_hpux11_1_0_3.tar.Z Autonomy/MBCS/chinese_solaris_1_0_3.tar.Z

The required language-specific configuration settings are:

DRE configuration parameter	Value
CharConv	4
TermSize	40
StripLanguage	2

Japanese

The required sentence-breaking files are:

Platform	Sentence-breaking files	Location on CD
NT	◆ japanesebreaking.dll ◆ \dic\tag.attr ◆ \dic\tag.counter ◆ \dic\tag.index ◆ \dic\tag.mrph ◆ \dic\tag.string ◆ \dic\tag.table ◆ jtag.dll ◆ jtag.ini ◆ jtag_at.dll ◆ japaneseconvlist.txt	Autonomy\MBCS\japanese_nt_2_0_5.zip
UNIX	◆ japanesebreaking.sl ◆ /dic/system/jtag.attr ◆ /dic/system/jtag.hash ◆ /dic/system/jtag.id ◆ /dic/system/jtag.mrph ◆ /dic/system/jtag.offset ◆ /dic/system/jtag.table ◆ /dic/system/jtag.trie ◆ jtag.ini ◆ libcodeconv.sl ◆ libjtag_at.sl ◆ libjtag.sl ◆ japaneseconvlist.txt	Autonomy/MBCS/japanese_aix_2_0_5.tar.Z Autonomy/MBCS/japanese_hpux11_2_0_5.tar.Z Autonomy/MBCS/japanese_solaris_2_0_5.tar.Z

The required language-specific configuration settings are:

DRE configuration parameter	Value
CharConv	1
TermSize	30
StripLanguage	2

Korean

The required sentence-breaking files are:

Platform	Sentence-breaking files	Location on CD
NT	◆ koreanbreaking.dll ◆ koreanconvlist.txt ◆ Koma.dll ◆ HanTag.dll ◆ main.dat ◆ prob.dat ◆ main.fst ◆ prob.fst ◆ pos.nam ◆ tag.nam ◆ tagout.nam ◆ connection.txt ◆ stopposnam.txt ◆ tagname.txt	Autonomy\MBCS\korean_nt_1_0_1.zip
UNIX	◆ koreanbreaking.so ◆ koreanconvlist.txt ◆ main.dat ◆ prob.dat ◆ main.fst ◆ prob.fst ◆ pos.nam ◆ tag.nam ◆ tagout.nam ◆ connection.txt ◆ stopposnam.txt ◆ tagname.txt	Autonomy/MBCS/korean_aix_1_0_1.tar.Z Autonomy/MBCS/korean_hpux11_1_0_1.tar.Z Autonomy/MBCS/korean_solaris_1_0_1.tar.Z

The required language-specific configuration settings are:

DRE configuration parameter	Value
CharConv	2
TermSize	40
StripLanguage	2

8 Troubleshooting the Conceptual Search Process

This chapter provides troubleshooting tips to help you implement conceptual searching successfully in your exteNd Director applications. You will learn how to diagnose and correct commonly encountered errors.

The following topics are covered:

- Commonly encountered problems
- General debugging techniques

Commonly encountered problems

This section diagnoses commonly encountered problems and suggests corrective actions. The following issues are covered:

- Unable to retrieve results exception
- Class not found exception for Autonomy JNI when accessing the Content Management (CM) subsystem
- UnsatisfiedLinkError for autonomyJNI.dll
- Search results become invalid after restarting the DRE service
- Documents do not appear to be indexed
- Queries return no results or too few results
- Document content does not appear to be stored in the DRE
- java.lang.Exception for Autonomy JNI when publishing documents on UNIX

Unable to retrieve results exception

This section explains why you might encounter this exception and describes how to correct the problem.

Diagnosis—DRE is not running

The Autonomy Java Native Interface (JNI) usually throws this exception if the exteNd Director Dynamic Resource Engine (DRE) is not running.

➢ **What to do:**

1. Determine whether the exteNd Director DRE is running:
   - On Windows:

     Look for the process **DirectorDRE.exe** in the Task Manager or for the service **exteNd Director DRE** in the Services Manager.
   - On UNIX:

     Issue the following command from your browser:

     ```
     http://server name or machine name:2000/qmethod=v
     ```
2. If the DRE is not running, start it:
   - On Windows:

     From the Start menu, select **Programs>Novell>exteNd Director 4.1>exteNd Director DRE**.
   - On UNIX:

     Run the command **./StartQuery.sh**. This command resides in the directory where you installed the DRE.

Class not found exception for Autonomy JNI when accessing the Content Management (CM) subsystem

This section explains why you might encounter this exception and describes how to correct the problem.

Diagnosis—JNI classes not on classpath

This exception is thrown if the Autonomy Java Native Interface (JNI) classes are not on the classpath of your application server. These classes are stored in **autonomy\autonomyJNI.jar** in theexteNd Director installation directory.

➢ **What to do:**

- Add Autonomy JNI classes to the classpath of your application server, as described in "Adding autonomyJNI.jar to your application server classpath" on page 24.

Diagnosis—Autonomy DLL not on library path

This exception is thrown if the autonomyJNI.dll is not on your library path. This dynamic link library is located at **autonomy\autonomyJNI.dll** in the exteNd Director installation directory.

➢ **What to do:**

- Add the directory containing autonomyJNI.dll to the Path environment variable of the machine where you installed exteNd Director, as described in "Adding the Autonomy dynamic library to your environment" on page 25.

UnsatisfiedLinkError for autonomyJNI.dll

You may see the following error message when you redeploy the exteNd Director project:

```
java.lang.UnsatisfiedLinkError: Native Library autonomyJNI.dll
already loaded in another classloader
```

This section explains why you might encounter this error and describes how to correct the problem.

Diagnosis—Mismatch of revision levels

The error occurs when autonomyJNI.jar and autonomyJNI.dll are not at the same revision level.

➢ **What to do:**

- Make sure you have the correct revisions of these files. You can check revision numbers programmatically by calling the method getApiVersion() on com.sssw.search.api.EbiQueryEngineDelegate.

Search results become invalid after restarting the DRE service

This section explains why you might encounter this behavior and describes how to correct—and prevent—the problem.

Diagnosis—Autonomy handles custom fields incorrectly

This problem occurs when you add new custom fields in the Content Management (CM) repository **after** creating documents that use the preexisting set of custom fields. Because of the way Autonomy handles custom fields, you must reinitialize the DRE to read in the new field set. Otherwise, search results are invalid.

➢ **What to do:**

1. Remove all documents from the DRE, as described in "Removing content from the DRE" on page 116.
2. Reconfigure the DRE by issuing a reset from the DRE Administration console, as described in "Resetting the DRE" on page 116.
3. Restart the DRE, as described in "Diagnosis—DRE is not running" on page 88.
4. Reindex your contents back into the DRE, as described in "Forcing indexing" on page 99.

CAUTION: *You must perform these steps **every time** you add new custom fields **after** creating documents that use custom metadata. To avoid this problem, see the preventive action below.*

➢ **To prevent this problem from occurring:**

Add all custom fields before adding any documents in the CM repository.

Documents do not appear to be indexed

You can check whether or not documents have been indexed by using any of the methods described in "Examining DRE content" on page 120.

This section explains why you might encounter this behavior and describes how to correct the problem.

Diagnosis—Search is disabled

The integration between the CM subsystem and the Search subsystem is disabled.

➢ **What to do:**

1. Make sure the exteNd Director DRE is running, as described in "Diagnosis—DRE is not running" on page 88.
2. Enable the option **com.sssw.cm.search.enable.*repository name***.

   For example, if you are working with the default CM repository—named **Default**—the property name will look like this:

   ```
   com.sssw.cm.search.enable.Default
   ```

   TIP: You set this option in the CM config.xml file, as described in "Setting search options in an existing exteNd Director project" on page 37. For more information about this option, see "Enable link to the Search subsystem?" on page 127 and "Defining options for a specific Content Management repository" on page 38.
3. Redeploy your exteNd Director project for the new setting to take effect.

Diagnosis—Search options are mismatched

The values of search options do not correspond to exteNd Director DRE settings.

➢ **What to do:**

1. Check your exteNd Director DRE settings in the DRE Administration console, as described in "Setting DRE search options" on page 118.
2. Configure the following search options to match the DRE settings:

   ```
   com.sssw.cm.search.host.repository name
   com.sssw.cm.search.queryport.repository name
   com.sssw.cm.search.indexport.repository name
   com.sssw.cm.search.repository.repository name
   ```

   TIP: You set these options in the CM config.xml file, as described in "Setting search options in an existing exteNd Director project" on page 37. If you are using the default CM repository, *repository name* = `Default`.
3. Redeploy your exteNd Director project for the new settings to take effect.

Diagnosis—Synchronization is scheduled as a batch process

There are two modes for synchronizing changes in the CM repository with DRE indexing:

Mode	Description
Immediate	Changes in the CM repository are propagated to the DRE as soon as they occur—so you see your documents indexed in real time

Mode	Description
Batch	Changes in the CM repository are propagated to the DRE as a scheduled or periodic background task. If you have scheduled synchronization to run in batch mode, you will not see indexing occur until the synchronization is triggered.

➢ **What to do:**

- Determine which synchronization mode is enabled by checking the value of the following search option:

  `com.sssw.cm.search.synch.mode.`*`repository name`*

  If the value is 1, synchronization occurs in batch mode and you should not expect to see your documents indexed immediately.

  TIP: You view this option in the CM config.xml file, as described in "Setting search options in an existing exteNd Director project" on page 37. For more information about this option, see "Synchronization mode" on page 134.

Diagnosis—Invalid document type

The document type of the content you are trying to index is invalid or unsupported. The Search subsystem supports the following MIME types for indexing content:

- Plain text
- HTML
- SGML
- XML
- Microsoft Word for Windows Version 3.x and higher
- Microsoft Excel Version 3.x and higher
- Microsoft PowerPoint Version 4 and higher
- Adobe Acrobat PDF

➢ **What to do:**

Make sure the MIME type of your document is supported by the Search subsystem. You can check document MIME types in the CMS Administration Console by following these steps:

1 Select the document of interest in the CMS Administration Console.

The content Property Inspector opens.

2 In the property inspector, select the **Versions** tab.

The MIME type of the document is displayed, along with other properties.

📖 For information on how to use the CMS Administration Console, see the chapter on the CMS Administration Console in the *Content Management Guide*.

Diagnosis—Documents have not been published

You may not have published the documents you are searching for in the CM subsystem. Only published content can be imported and indexed in the DRE.

➢ **What to do:**

In the CMS Administration Console, determine whether the documents of interest have been published by following these steps:

1 Select the document of interest.

The content Property Inspector opens.

2 In the Property Inspector, select the **Versions** tab.

If the document has been published, one of its version icons appears with a green border:

3 Publish documents as necessary.

📖 For more information about publishing documents, see the section on administering version control in the chapter describing the CMS Administration Console in the *Content Management Guide*

Diagnosis—DRE cannot find OmniSlave files

If you are indexing binary documents, you must specify the correct path to Autonomy's OmniSlave binary document filtering technology. By default, the OmniSlave files are stored at:

```
exteNd Director installation directory\exteNd
Director\autonomy\OmniSlaves
```

➢ **What to do:**

1 Make sure the path to OmniSlave files is specified correctly in the following search option:

```
com.sssw.cm.fetch.binary.filters.dir
```

TIP: You set this option in the CM config.xml file, as described in "Setting search options in an existing exteNd Director project" on page 37. For more information about this option, see "Install directory for binary document text filters" on page 130.

2 If you change the path, redeploy your exteNd Director project for the new setting to take effect.

Queries return no results or too few results

This section explains why you might encounter this behavior and describes how to correct the problem.

Diagnosis—No documents match your search criteria

Your search criteria may be too narrow or incorrectly specified, or your query terms may be misspelled.

➢ **What to do:**

Examine your query and take any of the following corrective steps as necessary:

- Broaden your search criteria.
- Add documents that fit the search criteria.
- Correct misspelled query terms or try a fuzzy query.

  📖 For more information, see "Fuzzy queries" on page 138.

Diagnosis—Documents may not have been indexed

You can check whether or not documents have been indexed by using any of the methods described in "Examining exteNd Director DRE content" on page 99.

➢ **What to do:**

- See the troubleshooting tips in "Documents do not appear to be indexed" on page 90.

Diagnosis—You may have configured the DRE incorrectly

If you change parameters in the DRE configuration file without reinitializing the DRE and reindexing the data, the DRE produces no results or erroneous results.

NOTE: The DRE configuration file is located at autonomy\engine\DirectorDRE.cfg in the exteNd Director installation directory.

➢ **What to do:**

1 Reconfigure the DRE by issuing a reset from the DRE Administration console, as described in "Resetting the DRE" on page 116.

2 Restart the DRE, as described in "Diagnosis—DRE is not running" on page 88.

3 Rerun the query.

Diagnosis—You may have issued the wrong type of query

A common scenario is to issue a conceptual query when you really intend to run a keyword search. In a keyword search, the DRE finds documents that contain occurrences of the desired keyword. By contrast, the conceptual query is an intelligent search that matches concepts rather than literal text strings.

📖 For more information, see "How conceptual searching differs from keyword searching" on page 15.

➢ What to do:

- Make sure you are using the correct syntax for the type of query you want to run. For example, if you want to search for documents that contain the words **silk** and **worm**, use the query notation for **keyword search**:

  ```
  silk:+worm:
  ```

  Notice that this syntax is different from **conceptual search** notation:

  ```
  silk+worm
  ```

📖 For more information, see "Overview of Autonomy-based conceptual searching" on page 14 and Chapter 6, "Querying Content and Metadata".

Diagnosis—Document content is not copied into the DRE

You may not have enabled the option to copy the content of documents you are searching into the exteNd Director DRE. If you are issuing a keyword query, you must make sure the content of the target documents is stored in the DRE when they are indexed.

➢ What to do:

1. Enable the following search option:

   ```
   com.sssw.cm.fetch.store.content.repository name
   ```

   You set this option in the CM config.xml file, as described in "Setting search options in an existing exteNd Director project" on page 37. For more information about this option, see "Copy document contents into the DRE?" on page 126.
2. Redeploy your exteNd Director project for the new setting to take effect.

Diagnosis—Your relevance threshold may be too low

If the relevance cutoff threshold is too low, the DRE will drop some results that you actually want to see.

➢ What to do:

- Bump up the threshold by calling the method setRelevanceCut() on com.sssw.search.api.EbiQuery.

Document content does not appear to be stored in the DRE

This section explains why you might encounter this behavior and describes how to correct the problem.

Diagnosis—Document content is not copied into the DRE

You may not have enabled the option to copy the content of documents you are searching into the exteNd Director DRE. This option is disabled by default to avoid incurring the overhead of storing content in both the CM repository and the DRE.

➢ **What to do:**

1. Enable the following option:

   `com.sssw.cm.fetch.store.content.`*`repository name`*

   You set this option in the CM config.xml file, as described in "Setting search options in an existing exteNd Director project" on page 37. For more information about this option, see "Copy document contents into the DRE?" on page 126.
2. Redeploy your exteNd Director project for the new setting to take effect.

java.lang.Exception for Autonomy JNI when publishing documents on UNIX

This section explains why you might encounter this behavior and describes how to correct the problem.

Diagnosis—You do not have write permission for the binary document text filter directory

The binary document text filter directory—which resides in the directory where you installed the DRE—contains executables required for importing data from the CM repository into the DRE for indexing. By default, publishing is an operation that triggers immediate synchronization, an event that involves importing updated content from the CM repository into the DRE. You must have read/write/execute permission for the binary document text filter directory so that the import process can proceed to completion.

➢ **What to do:**

1. Find your binary document text filter directory:
   - **1a** In exteNd Director, open **config.xml** for the CM subsystem in your exteNd Director project.
   - **1b** Find the following key:

     `com.sssw.cm.fetch.binary.filters.dir`

     The value of this key is the path for the binary document text filter directory.

NOTE: You set the binary document text filter directory when you created your exteNd Director project, as described in the section on creating a new exteNd Director project in *Developing exteNd Director Applications*. In this section, look for information about setting parameters on the **Filters** tab of the Content Management Search Configuration panel.

2 Set read/write/execute permission on the binary document text filter directory.

General debugging techniques

This section describes techniques you can use to determine whether search processes are running as expected and whether you have constructed your queries correctly. Some of these techniques require you to run the exteNd Director DRE Administration console, which is described in Chapter 10, "Administering the Dynamic Reasoning Engine". These topics are included:

- Logging
- Examining exteNd Director DRE content
- Forcing indexing
- Getting the list of terms indexed for a document
- How to test queries

Logging

This section describes how to monitor the indexing process by generating and examining exteNd Director and Autonomy logs.

Configure and examine the import log

The import log records the activity of the Autonomy importer at runtime. By default, this log resides in **autonomy\OmniSlaves\import.log** in the directory where you installed exteNd Director.

You configure the behavior of the import log in the file **importslave.cfg**, located in the same directory as import.log. You can specify the following options:

- Logging level
- Location of log file
- Documents to be excluded from import, based on size in bytes and words

Enable debugging during import

When enabled, this option writes content to the server console for debugging purposes as documents are imported and indexed.

➢ **To log information about indexing in exteNd Director:**

1. Raise the logging levels of the CM subsystem and the Search subsystem to 5, preferably on a small prototype document set.

   Level 5 logging records debugging messages and information about application progress on the server console as you interact with the CM repository and search its content.

   Use the Director Administration Console (DAC) to adjust logging levels for EboSearchLog and EboCmLog, as described in the section on logs in the chapter about application configuration using the DAC in the *Content Management Guide*.

2. Enable the following search option:

   ```
   com.sssw.cm.fetch.dump.imported.data
   ```

   You set this option in the CM config.xml file, as described in "Setting search options in an existing exteNd Director project" on page 37. For more information about this option, see "Debug during import?" on page 126.

3. Redeploy your exteNd Director project for the new setting to take effect.

4. Monitor the index process on your server console as you run your search application.

📖 For more information, see the chapter on logging information in the *Developing exteNd Director Applications*.

Look at Autonomy's activity log through your browser

You can view the log of activities performed by Autonomy by entering the following URL in your browser:

```
http://DRE host:DRE-query-port/qmethod=v
```

For example, if your host name is **localhost** and your DRE-query-port is **2000** (the default), the URL should look like this:

```
http://localhost:2000/qmethod=v
```

Examining exteNd Director DRE content

This section describes several ways to examine the content of the exteNd Director DRE.

➢ **To examine DRE contents through your browser:**

- Issue the following command from your browser:

  `http://DRE host:DRE-query-port/qmethod=g`

  For example, if your host name is **localhost** and your DRE-query-port is **2000** (the default), the URL should look like this:

  `http://localhost:2000/qmethod=g`

  This command lists the documents that have been indexed. You can identify any document of interest by looking up its **Doc_id** property value (the unique identifier of the document within the DRE). This value appears in the results generated by the **qmethod=g** command.

➢ **To examine DRE contents by backing up the DRE:**

- See "Examining DRE content" on page 120.

Forcing indexing

There are situations in which you need to force the exteNd Director DRE to reindex data —for example, when you reconfigure the search environment, as described in Chapter 3, "Setting Search Options".

The following procedure shows how to configure the exteNd Director DRE to reindex all content as a batch process.

➢ **To force indexing:**

1. Open **config.xml** for the CM subsystem in your exteNd Director project.
2. Change the synchronization mode to batch by setting com.sssw.cm.search.synch.mode.*repository name* to **1**.

   TIP: By default you use the CM repository—named **Default**. So you would set synchronization mode on **com.sssw.cm.search.synch.mode.Default**.
3. Save and close **config.xml**.
4. In the same location as config.xml, open the task list configuration file for your CM repository—*repository name*_**tasklist.xml**.

   TIP: For the default CM repository, the configuration file is **Default_tasklist.xml**.
5. Look for the definition of the **synch** task—the task that synchronizes the CM subsystem with the Search service engine.

   TIP: The task definition appears as either **<periodic-synch>** (default) or **<scheduled-synch>**.

6 Set the **interval** (for periodic-synch) or the **schedule** (for scheduled-synch) as desired.

7 Specify that all content should be reindexed, by adding the following element to the synch task definition:

```
<since-last>false</since-last>
```

NOTE: If you do not disable this property, the DRE indexes only the content that hasn't been processed in the previous run of the task (the default setting).

Here is a sample synchronization task definition that meets these requirements:

```
<periodic-synch>
  <task-name>Default Repository Synchronization</task-name>
  <description>The Default Repository Synchronization
Task</description>
  <since-last>false</since-last>
  <enabled>true</enabled>
  <interval>
    <millis>86400000</millis>
    <exact>false</exact>
  </interval>
</periodic-synch>
```

8 Save and close **Default_tasklist.xml**.

9 Redeploy the application.

TIP: After reindexing the content, it is recommended that you set the **<since-last>** property back to **true** to avoid reindexing all content again unnecessarily.

10 Start the synch task from the CMS Administration Console, as described in the section on administering automated tasks in the *Content Management Guide*.

📖 For more information about tasks in the CM repository, see the chapter on managing tasks in the *Content Management Guide*.

Getting the list of terms indexed for a document

You may want to examine the list of terms indexed for a specific document to verify that the correct information is in the DRE. You can retrieve the 40 most important terms from a document using this command:

```
http://IPAddress:QueryPort/qmethod=t&querytext=docid
```

NOTE: The value **docid** is the Doc_id property for the document of interest. You can look up Doc_id values as described in "Examining exteNd Director DRE content" on page 99.

How to test queries

You can use the exteNd Director DRE Administration console to test your queries in isolation to validate whether your queries return the expected results.

➢ **To test queries in the DRE Administration console:**

- See "Testing queries" on page 117.

II SQL-Based Search Concepts

Describes the fundamentals of implementing SQL-based search in exteNd Director applications

9 Implementing SQL-Based Searching

This chapter describes how to implement SQL-based searching in exteNd Director applications.

The following topics are covered:

- Logic flow for implementing SQL-based search
- Building the search criteria
- Example: searching standard document metadata

Logic flow for implementing SQL-based search

You can write search components to implement SQL-based searching in exteNd Director applications. Use the following logic flow in the getComponentData() method of the exteNd Director component:

1. Instantiate a content manager delegate, as follows:

```
com.sssw.cm.api.EbiContentMgmtDelegate contentMgr =
com.sssw.cm.client.EboFactory.getDefaultContentMgmtDelegate();
```

NOTE: A delegate is a wrapper that hides the location of a service. The delegate model follows the J2EE Business Delegate pattern. For more information about delegates, see the chapter on coding Java for exteNd Director applications in *Developing exteNd Director Applications*.

2. Instantiate a document metadata query object, as follows:

```
com.sssw.cm.api.EbiDocQuery query = (EbiDocQuery)
contentMgr.createQuery(EbiDocQuery.DOC_QUERY);
```

3 Select the fields to be returned by calling SELECT methods on the EbiDocQuery object.

EbiDocQuery inherits SELECT methods from com.sssw.cm.api.EbiDocMetaDataQuery.

4 Specify the search criteria by calling WHERE methods on the EbiDocQuery object.

Each method returns a com.sssw.fw.api.EbiQueryExpression object, which represents a SQL WHERE subclause.

5 Concatenate the WHERE subclauses by calling methods on EbiQueryExpression to combine terms using logical operators and parentheses.

📖 For more details, see "Building the search criteria" on page 106.

6 Specify the order in which to return results in ORDER BY clause(s) by calling ORDERBY methods on the com.sssw.cm.api.EbiDocMetaDataQuery object.

7 Execute the search by calling findElements() on the EbiContentMgmtDelegate object.

The findElements() method executes the query and returns a collection of EbiDocument objects. The collection contains the IDs and data for only those fields you designated using SELECT methods in Step 3. As such, the objects in the collection are not complete representations of the documents in the CM repository.

Building the search criteria

You specify search criteria for metadata by calling WHERE methods on the document metadata query object com.sssw.cm.api.EbiDocQuery. WHERE methods match values against data and include a NOT parameter that can negate the clause.

- For **standard** metadata, use WHERE methods that correspond to the property of interest. For example, if you want to find documents whose **author** is Smith, use the whereAuthor() method.
- For **custom** (extension) metadata, use the **whereField*()** methods, as described in "Defining criteria for searching custom metadata" on page 108.

Using operators to match values against data

WHERE methods can match values against data by using SQL, relational, or string operators defined in com.sssw.fw.api.EbiMetaDataQuery, a generic interface for defining SQL queries over metadata.

Here is a summary of available operators:

Constant	Definition
SQL operators	
OP_BETWEEN	SQL BETWEEN operator
OP_IN	SQL IN operator
OP_IS_NULL	SQL IS NULL operator (the value argument in the WHERE method is not ignored)
Relational operators	
ROP_EQUAL	Is equal to (=)
ROP_GEQ	Is greater than or equal to (>=)
ROP_GREATER	Is greater than (>)
ROP_LEQ	Is less than or equal to (<=)
ROP_LESS	Is less than (<)
String operators	
SOP_ENDS_WITH	Whether the target string ends with the specified character sequence
SOP_EQUALS_IGNORE_CASE	Whether two strings are equivalent without considering case
SOP_LIKE	SQL LIKE operator
SOP_LIKE_IGNORE_CASE	SQL LIKE operator, where the case of the operands is ignored
SOP_STARTS_WITH	Whether the target string begins with the specified character sequence

Concatenating WHERE expressions

When you need to connect WHERE expressions for several metadata fields, you can concatenate them using com.sssw.fw.api.EbiQueryExpression methods. Here are some examples illustrating patterns that assign the result to expression1:

Example	Pattern
`expression1.andExpression(expression2);`	Join two expressions using AND
`expression1.parenthesize();`	Enclose expression in parentheses before joining it to another expression
`expression1.orExpression(expression3);`	Join two expressions using OR

Defining criteria for searching custom metadata

To define criteria for searching custom (extension) metadata fields, use **whereField*()** methods defined on com.sssw.cm.api.EbiDocQuery.

Here are the steps to follow:

1 Construct an expression that identifies the field of interest, by either ID or name:

For	Use
ID	whereFieldID*() methods
Name	whereFieldName*() methods

2 Construct a second expression that specifies the desired value of the field.

Use any whereFieldValue*() method.

3 Concatenate the WHERE expressions you just created using the andExpression() method and the parenthesize() method.

The resulting expression restricts the search to the field identified in Step 1 and the values specified in Step 2.

TIP: To search another field, set up another pair of field identifier/value expressions in the same way and concatenate the result with the rest of the WHERE clause.

Example: searching standard document metadata

The following example illustrates how to perform a SQL-based search on standard document metadata. In this example, a method called **executeDocMetaSearch()** finds documents of type **Movie Review** that meet the following search criteria:

- Movie was released between 1990 and 2000
- Author is **JSmith** or title starts with **A**

```
public void executeDocMetaSearch(EbiContentMgmtDelegate cmgr, EbiContext context, String
mrDocTypeID, String yorFieldID)
        throws EboUnrecoverableSystemException, EboSecurityException,
EboItemExistenceException
    {
        // Search for all the Movie Review documents where
        // (Author is 'JSmith' or Title starts with 'A')
        // AND
        // where YearOfRelease is between the year 1990 and the year 2000.

        EbiDocQuery docQuery = (EbiDocQuery)cmgr.createQuery(EbiDocQuery.DOC_QUERY);
        docQuery.selectAll();

        EbiQueryExpression expr = docQuery.whereDocTypeID(mrDocTypeID,
EbiDocQuery.ROP_EQUAL, false);
        EbiQueryExpression expr2 = docQuery.whereAuthor("JSmith", EbiDocQuery.ROP_EQUAL,
false);
        EbiQueryExpression expr3 = docQuery.whereTitle("A", EbiDocQuery.SOP_STARTS_WITH,
false);

       EbiQueryExpression expr4 = docQuery.whereFieldValueBetween(yorFieldID,
EboMisc.getInteger(1990), EboMisc.getInteger(2000), false);

        // (Author is 'JSmith' or Title starts with 'A)
        expr2.orExpression(expr3);
        expr2.parenthesize();

        // YearOfRelease was between the year 1990 and the year 2000
        expr4.andExpression(expr5);

        // (Author is 'JSmith' or whose Title starts with 'A')
        // AND
        // YearOfRelease was between the year 1990 and the year 2000
        expr.andExpression(expr2);
        expr.andExpression(expr4);

        docQuery.setWhere(expr);

        // Sort results by creation date/time, in ascending order
        docQuery.orderByCreateDate(true);

        // Execute the search and filter the results based
        // on security constraints
        Collection results = cmgr.findElementsFiltered(context, docQuery);
    }
```

In this example, the com.sssw.cm.api.EbiDocQuery.whereFieldValueBetween() method allows you to specify your custom metadata query in a single line of code. Other similar methods include:

- com.sssw.cm.api.EbiDocQuery.whereFieldValue()
- com.sssw.cm.api.EbiDocQuery.whereFieldValue_ByName()
- com.sssw.cm.api.EbiDocQuery.whereFieldValueBetween_ByName()
- com.sssw.cm.api.EbiDocQuery.whereFieldValueIn()
- com.sssw.cm.api.EbiDocQuery.whereFieldValueIn_ByName()

 For more information on these methods, see the javadoc for EbiDocQuery.

The EbiDocument objects in the returned collection contain all the properties that have been defined for the movie reviews in ascending order by creation date.

The executeDocMetaSearch() method accesses the following objects as input arguments:

- Content manager—EbiContentMgmtDelegate
- Context object—EbiContext
- Document type
- Field of interest

III Tools

Describes how to use the exteNd Director Dynamic Reasoning Engine Administration console to manage conceptual searching

10 Administering the Dynamic Reasoning Engine

This chapter explains how to manage search behavior using the exteNd Director Dynamic Reasoning Engine (DRE) Administration console.

The following topics are covered:

- exteNd Director DRE Administration console functions
- Starting the exteNd Director DRE Administration console
- Resetting the DRE
- Removing content from the DRE
- Testing queries
- Setting DRE search options
- Examining DRE content
- Restoring DRE content
- Getting help on how to use the DRE Administration console

NOTE: Information in this chapter is adapted from the *Autonomy Server 2.2* manual from Autonomy, Inc.

exteNd Director DRE Administration console functions

The exteNd Director Search subsystem gives you access to the exteNd Director DRE Administration console, a graphical user interface for administering the exteNd Director DRE. You can use the DRE Administration console to:

- Reset the DRE after changing configuration settings
- Remove documents and other content from the DRE
- Test content queries in isolation locally before deployment

- Set DRE search options
- Examine DRE content
- Restore DRE content

Starting the exteNd Director DRE Administration console

You can start the exteNd Director DRE Administration console from the Windows Start menu or by invoking the executable file.

➢ **To start the DRE Administration console:**

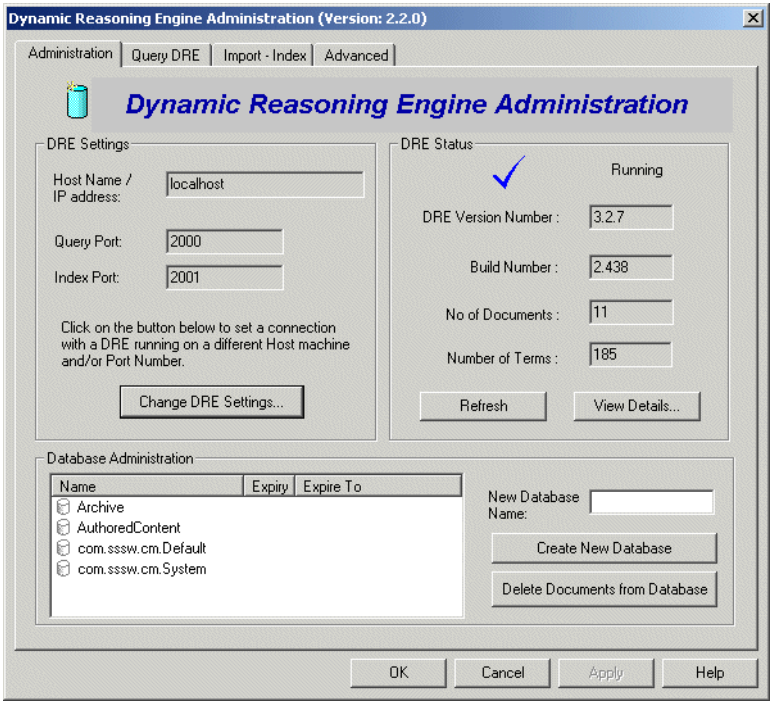
1. Make sure the exteNd Director DRE is running.

   On Windows, the exteNd Director DRE is invoked automatically when you start your server. On UNIX, you must start the exteNd Director DRE manually, as described in "Installing the exteNd Director Dynamic Reasoning Engine" on page 21.

2. Invoke the DRE administrator using one of these methods:

From	Do this
Windows Start menu	Select **Programs>Novell>exteNd Director 4.1>exteNd Director DRE Administrator**
Executable file	Double-click **autonomy\engine\DirectorDREadmin.exe** in your exteNd Director installation directory

The DRE Administration console tries to connect with the DRE, then displays its main Administration panel on your desktop:



Notice that there is a symbol in the upper-right panel that gives you an immediate visual cue about the status of the DRE:

Symbol	Means
✓	DRE is running and has connected to the DRE Administration console
⊘	DRE Administration console cannot connect to the DRE

If the administrator cannot connect to the DRE, see Chapter 8, “Troubleshooting the Conceptual Search Process”.

Resetting the DRE

When you change configuration parameters, you must reset the DRE to reinitialize its settings.

➢ **To reset the DRE:**

1. Start the Administration console, as described in "Starting the exteNd Director DRE Administration console" on page 114.
2. Select the **Advanced** tab.
3. Click **Reset DRE**.

Removing content from the DRE

You can use the DRE Administration console to remove documents from the DRE or to remove all content—including documents, terms, and probability weightings. After removing any of these resources, you must reindex the remaining content into the DRE before you can search the Content Management (CM) repository again. This section describes the recommended procedures.

➢ **To remove documents from the DRE:**

1. Start the Administration console, as described in "Starting the exteNd Director DRE Administration console" on page 114.

   The console opens with the Administration tab selected.
2. Select a DRE database from the list in the Database Administration panel.

   TIP: To specify the DRE database for the default CM repository, select **com.sssw.cm.Default**.
3. Click **Delete Documents from Database**.

   A confirmation window appears.
4. Click **Yes** to confirm the deletion.
5. Reindex your content into the DRE, as described in "Forcing indexing" on page 99.

➢ **To remove all content from the DRE:**

1. Start the Administration console, as described in "Starting the exteNd Director DRE Administration console" on page 114.
2. Select the **Advanced** tab.
3. Click **Initialize DRE**.
4. Reindex your contents into the DRE, as described in the section on how to force indexing.

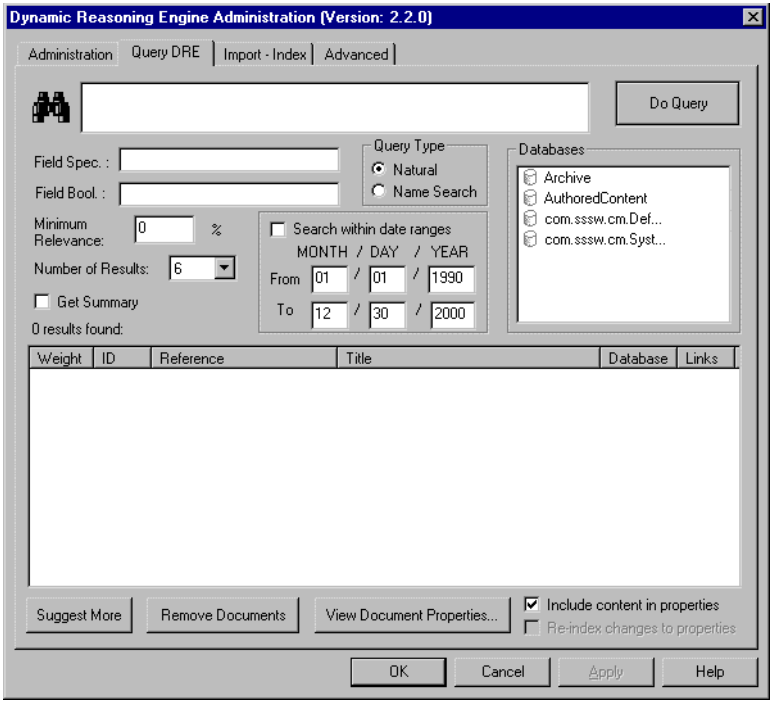
Testing queries

You can use the DRE Administration console to test your queries in isolation to validate whether they return the expected results.

➢ **To test queries in the DRE Administration console:**

1. Start the Administration console, as described in "Starting the exteNd Director DRE Administration console" on page 114.
2. Select the **Query DRE** tab.

   The DRE Administration dialog opens with the Query DRE tab selected:



3. Make sure the DRE database you want to search appears in the Databases list.

   By default, the Search subsystem designates com.sssw.cm.Default as the DRE database to search.
4. Enter your search query in the text box next to the search symbol:

5 Click the **Do Query** button.

The query results appear in the results table at the bottom of the dialog.

📖 For more information about querying in the DRE Administration console, click the **Help** button at the bottom of the dialog. To learn about supported query types and how to construct queries, see Chapter 12, "Search Query Types" and Chapter 6, "Querying Content and Metadata".

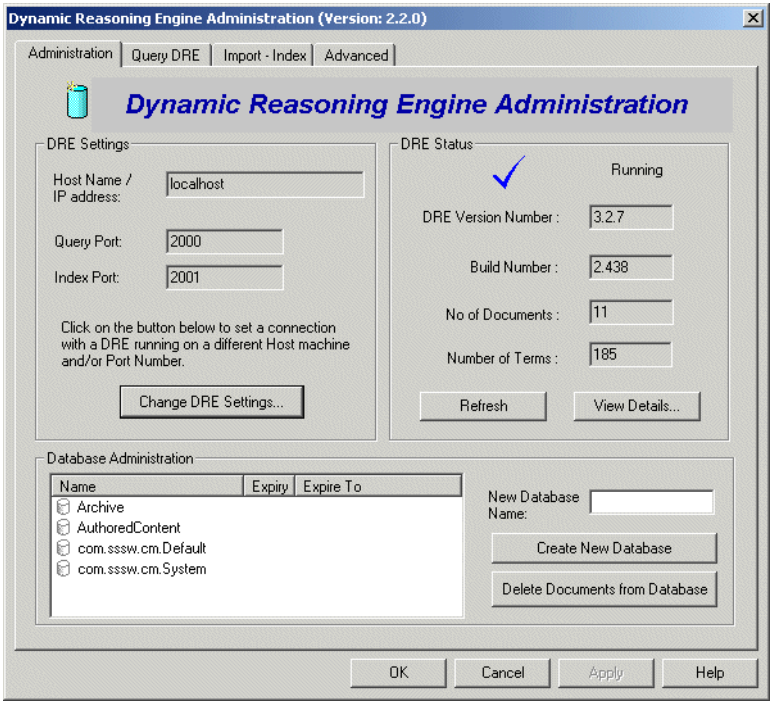
Setting DRE search options

This section describes how to configure DRE settings in the exteNd Director DRE Administration console.

➢ **To modify search options in the DRE Administration console:**

1 Start the Administration console, as described in "Starting the exteNd Director DRE Administration console" on page 114.

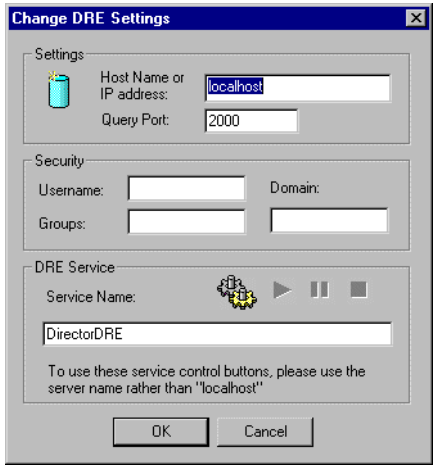
2 Select the **Administration** tab if it is not already displayed:



Note that the DRE Administration console has successfully connected to the DRE running on localhost at query port 2000, as indicated by the check mark symbol:

3 Click **Change DRE Settings**.

The Change DRE Settings dialog opens:



4 Enter a new DRE host and/or query port and click **OK**.

The DRE Administration console attempts to connect to the DRE running on the new host and/or port number. If the connection fails, the check mark symbol changes to an alert symbol:

5 Propagate your changes to the exteNd Director environment using techniques described in "Setting search options at design time" on page 37 and "Setting search options in an existing exteNd Director project" on page 37.

Examining DRE content

This section explains how you can examine DRE content by creating a backup file using the exteNd Director DRE Administration console.

➢ **To examine DRE contents by backing up the DRE:**

1. Start the Administration console, as described in "Starting the exteNd Director DRE Administration console" on page 114.
2. Select the **Advanced** tab.
3. Specify a path to the file that will hold the backup data.

   For example:

   ```
   c:\Autonomy\backups\mybackup.idx
   ```

4. Click **Backup Now!** for the LOCAL IDX BACKUP.

   The DRE administrator backs up DRE data in the IDX file you specified. You can open the IDX file in a text editor to examine content, standard metadata, and custom fields associated with each document that has been imported and indexed in the DRE. Here is an excerpt from an IDX file showing information about one document:

   ```
   #DREENDDOC
   #DREREFERENCE Production/Employee Classifieds/For Sale: Electric
   Lawnmower
   #DRETITLE
   #DREFIELD summary=""
   #DREFIELD CHECKSUM=""
   #DREFIELD DREDOCTYPE=""
   #DREFIELD BIAS=""
   #DREFIELD AuthorTemplate=""
   #DREFIELD AUTHOR="B. Greene"
   #DREFIELD CONTENTSIZE="307"
   #DREFIELD CREATED="1002060735000"
   #DREFIELD DOCABSTRACT="Selling 5-year-old electric lawnmower."
   #DREFIELD DOCID="07fce6f50a8e94ef39c96a8164ae0000"
   #DREFIELD DOCNAME="For Sale: Electric Lawnmower"
   #DREFIELD DOCTYPEID="07feb46c1d4e94d3561a6a8164ae0000"
   #DREFIELD DOCTYPENAME="Employee Classified"
   #DREFIELD EXPIRATIONDATE=""
   #DREFIELD FOLDERID="07ffc46c1d4e94d3561a6a8164ae0000"
   #DREFIELD LOCKEDBY="administrator"
   #DREFIELD MIMETYPE="text/html"
   #DREFIELD PARENTDOCID=""
   #DREFIELD PUBLISHDATE=""
   #DREFIELD PUBLISHSTATUS="1"
   #DREFIELD REPOSITORYID=""
   #DREFIELD STATUS=""
   #DREFIELD SUBTITLE=""
   #DREFIELD TITLE="Selling Electric Lawnmower"
   #DREFIELD UPDATETIME=""
   #DREFIELD UPDATEUSER=""
   ```

```
#DREDOCID 8
#DRESECTION 0
#DRETYPE TEXT
#DREDATE 1002060735
#DREDBNAME com.sssw.cm.Default
#DRESTORECONTENT y
#DRECONTENT
I'm selling my 5-year-old electric lawnmower. We recently moved
to a house with a much larger yard, so using the cord has become
problematic. Regrettably, we must revert to a gas-powered unit.
Our electric unit is in excellent condition and has never needed
any repairs. Asking $100.
```

TIP: You can restore DRE content from this backup IDX file, as described in "Restoring DRE content" next.

Restoring DRE content

This section explains how you can restore DRE content from a backup IDX file.

➢ **To restore DRE content:**

1. Start the Administration console, as described in "Starting the exteNd Director DRE Administration console" on page 114.
2. Back up DRE content to an IDX file, as described in "Examining DRE content" on page 120.
3. Select the **Import-Index** tab.
4. Click **Add IDX Files**.

   An Open dialog appears.
5. Select your backup IDX file and click **Open**.

   The IDX file is added to the list on the Import-Index panel in the DRE administrator.
6. Select the IDX file from the list and click **Index Into DRE now!**.

Getting help on how to use the DRE Administration console

- Click the **Help** button at the bottom of the DRE Administration console to invoke Autonomy online help.

IV Reference

Describes the options for configuring a conceptual search environment and the types of queries you can implement

11 Search Options Reference

This chapter describes the search options you can configure in exteNd Director:

- Copy document contents into the DRE?
- Debug during import?
- Enable link to the Search subsystem?
- Importable file extensions
- Importable MIME types
- Index custom document metadata?
- Index document content?
- Index standard document metadata?
- Index port
- Install directory for binary document text filters
- Name of DRE database
- Name of DRE host
- Number of deleted documents to batch up
- Operations that trigger immediate synchronization
- Query port
- Support binary document formats?
- Symbol for concatenating multivalue custom metadata values before indexing
- Synchronization mode

For background information, see Chapter 3, “Setting Search Options”.

Copy document contents into the DRE?

Indicates whether to copy document contents into the exteNd Director Dynamic Reasoning Engine (DRE) when the document is imported and indexed.

Key in config.xml	Default	Tips	Set method
com.sssw.cm.fetch.store.content.*repository name*	false	◆ Set to **false** to avoid the overhead of storing the same information in two places: the Content Management (CM) repository and the DRE database. ◆ Set to **true** to allow you to extract and back up document content using the DRE Administration console and execute keyword searches. **IMPORTANT:** This option must be set to true to enable keyword searching.	EbiDataFetcherDelegate.setStoreContent()

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37
- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39

Debug during import?

Indicates whether to dump document contents to the console as documents are indexed for debugging purposes.

Key in config.xml	Default	Tips
com.sssw.cm.fetch.dump.imported.data	false	◆ Set to **true** to get a greater degree of detail than tracing CM and Search logs when documents are imported and indexed. ◆ Set to **false** to avoid performance overhead.

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

Enable link to the Search subsystem?

Indicates whether to activate Autonomy and Search subsystem interactions with the CM subsystem—including importing, indexing, and searching.

Key in config.xml	Default	Tips
com.sssw.cm.search.enable.*repository name*	false	◆ Set to **true** to enable interaction between the CM repository and the Search subsystem. ◆ Set to **false** to disable all interactions between Autonomy and the CM subsystem, including conceptual search. **IMPORTANT:** You must set the value to **true** to use Autonomy and the Search subsystem.

You can configure this option:

- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37
- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

Importable file extensions

Specifies the extensions of binary formats that you can import into the DRE.

Key in config.xml	Default	Tips
com.sssw.cm.fetch.extensions	.html;.sgml;.xml;.txt;.rtf;.pdf;.xls;.xls;.ppt;.ppt	The number and order of importable extensions must match the number and order of importable MIME types.

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

Importable MIME types

Specifies the MIME types of binary formats that you can import and index in the DRE. The following MIME types are supported:

- Plain text
- HTML
- SGML
- XML
- Microsoft Word for Windows Version 3.x and higher
- Microsoft Excel Version 3.x and higher
- Microsoft PowerPoint Version 4 and higher
- Adobe Acrobat PDF

Key in config.xml	Default	Tips
com.sssw.cm.fetch.mime.types	text/html;text/sgml;text/xml;text/plain; application/msword;application/pdf; application/msexcel; application/xmsexcel; application/powerpoint; application/mspowerpoint	The number and order of importable MIME types must match the number and order of importable file extensions.

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

Index custom document metadata?

Indicates whether custom document metadata (extension metadata) field values should be indexed. Custom metadata is application-specific information about content that you define in the CM subsystem as fields in document types. Custom metadata helps to categorize content, making it easier to search.

Key in config.xml	Default	Tips	Set method
com.sssw.cm.fetch.process.extn.metadata.*repository name*	true	◆ Set to **true** to index custom metadata. ◆ Set to **false** to ignore custom metadata and index only standard metadata and/or content.	EbiDataFetcherDelegate.setProcessExtnMeta()

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37
- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39

Index document content?

Indicates whether document content should be indexed.

Key in config.xml	Default	Tips	Set method
com.sssw.cm.fetch.process.content.*repository name*	true	• Set to **true** to index document content. • Set to **false** to ignore document content and index only standard and/or custom document metadata.	EbiDataFetcherDelegate.setProcessContent()

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37
- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39

Index standard document metadata?

Indicates whether standard document metadata should be indexed. Standard metadata is descriptive information about content that is automatically attached to every document. Examples of standard metadata are title, author, and creation date.

Key in config.xml	Default	Tips	Set method
com.sssw.cm.fetch.process.metadata.*repository name*	true	• Set to **true** to index document standard metadata. • Set to **false** to ignore standard document metadata and index only document content and/or custom document metadata.	EbiDataFetcherDelegate.setProcessMeta()

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37
- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39

Index port

Specifies the port number used by the DRE for indexing:

Key in config.xml	Default	Set method
com.sssw.cm.search.indexport.*repository name*	2001	EbiDataFetcherDelegate.setIndexPort()

You can configure this option:

- Using the DRE Administration console, as described in "Configuring the DRE using the exteNd Director DRE Administration console" on page 37
- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37
- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37
- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39
- In the DRE configuration file, as described in "Setting search options by modifying the DRE configuration file" on page 40

Install directory for binary document text filters

Specifies where Autonomy OmniSlave binary document text filters are installed.

Key in config.xml	Default	Tips
com.sssw.cm.fetch.binary.filters.dir	C:\exteNd\exteNd Director\Autonomy \OmniSlaves	Specify the directory where the Autonomy OmniSlave technology is installed

You can configure this option:

- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

Name of DRE database

Specifies the name of the DRE database that fetches documents from the Default CM repository.

Key in config.xml	Default	Tips	Set method
com.sssw.cm.search.repository.Default	com.sssw.cm.Default	If you override the default, make sure all search options that are specified for this database reflect the new name.	EbiDataFetcherDelegate.setDestRepository()

You can configure this option:

- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37
- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37
- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39

Name of DRE host

Specifies the host name or IP address of the DRE that fetches documents from the default CM repository.

Key in config.xml	Default	Set method
com.sssw.cm.search.host.Default	localhost	EbiDataFetcherDelegate.setHost()

You can configure this option:

- Using the DRE Administration console, as described in "Configuring the DRE using the exteNd Director DRE Administration console" on page 37
- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37
- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39

Number of deleted documents to batch up

Specifies the number of deleted documents to batch up before removals are reflected in the DRE.

Key in config.xml	Default	Tips
com.sssw.cm.search.synch.removes.batch.size.*repository name*	100	This setting is relevant only if batch synchronization is enabled (com.sssw.cm.search.synch.mode is set to 1), as described in "Synchronization mode" on page 134.

You can configure this option:

- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37
- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

Operations that trigger immediate synchronization

Specifies the operations on documents that trigger immediate synchronization to reflect changes to metadata or content.

Key in config.xml	Default	Tips
com.sssw.cm.search.synch.docops.*repository name*	add; update; remove; checkin; checkout; publish; uncheckout; unpublish; unlock; rollback	This setting is relevant only if immediate synchronization is enabled (com.sssw.cm.search.synch.mode is set to 0), as described in "Synchronization mode" on page 134.

You can configure this option:

- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37
- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

Query port

Specifies the port number on which the DRE expects to receive queries.

Key in config.xml	Default	Set method
com.sssw.cm.search.queryport.*repository name*	2000	EbiDataFetcherDelegate.setQueryPort()

You can configure this option:

- Using the DRE Administration console, as described in "Configuring the DRE using the exteNd Director DRE Administration console" on page 37
- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37
- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37
- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39
- In the DRE configuration file, as described in "Setting search options by modifying the DRE configuration file" on page 40

Support binary document formats?

Indicates whether to enable support for indexing documents in binary formats.

Key in config.xml	Default	Tips
com.sssw.cm.fetch.handle.binary.*repository name*	true	◆ Set to **true** to index documents developed in applications that produce binary formats, such as Adobe Acrobat and Microsoft Word and PowerPoint. ◆ Set to **false** if you are indexing documents in text formats only—including XML, HTML, and other text documents.

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

Symbol for concatenating multivalue custom metadata values before indexing

Specifies the delimiter symbol to use for concatenating multivalue custom metadata values before they are indexed into the DRE. You must specify a delimiter, because Autonomy does not support multivalue properties.

Key in config.xml	Default	Set method
com.sssw.cm.fetch.multivalue.delim. *repository name*	/	EbiDataFetcherDelegate.setMultiValueDelim()

You can configure this option:

- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37
- Using its Set method, as described in "Setting search options programmatically at runtime" on page 39

Synchronization mode

Specifies how the DRE propagates changes to documents in the CM repository.

Key in config.xml	Default	Tips
com.sssw.cm.search.synch.mode. *repository name*	0 (immediate mode)	◆ Set to **0** for immediate mode—Propagate changes as soon as they occur. You can specify events to trigger synchronization in this mode, as described in "Operations that trigger immediate synchronization" on page 132. ◆ Set to **1** for batch mode—Propagate changes as a scheduled or periodic background task. You can specify the number of deleted documents to batch up before removals are reflected in the DRE, as described in "Number of deleted documents to batch up" on page 132.

You can configure this option:

- When you create your exteNd Director project, as described in "Setting search options at design time" on page 37
- In an existing project, as described in "Setting search options in an existing exteNd Director project" on page 37

12 Search Query Types

This chapter describes the types of Autonomy-based queries supported by the exteNd Director Search subsystem and how to implement them.

The following queries are covered:

- Boolean queries
- Conceptual queries
- Field queries
- Fuzzy queries
- Get-all queries
- Keyword search
- Proper name search
- Proximity queries
- Suggest similar documents
- Thesaurus queries

For background information, see Chapter 6, "Querying Content and Metadata".

Boolean queries

Description A *boolean query* uses logical operators to refine search criteria.

Syntax

```
word1+LOGICAL OPERATOR+word2+LOGICAL OPERATOR+word3+LOGICAL
OPERATOR+... wordN
```

This is the order of precedence of logical operators (in descending order):

1. `NOT`
2. `AND`, `WNEAR`, `NEAR`
3. `OR`, `XOR`, `EOR`

You can use parentheses to group operators and operands.

Examples

```
effect+recession+AND+economic+AND+slowdown+AND+consumer+AND+spending+OR
+buying
```

In this example parentheses are not needed, because AND has a higher precedence than OR. To change the order of evaluation use parentheses, as in this example:

```
((effect+recession+AND+economic)+OR+(+slowdown+AND+consumer+AND+
spending))+OR+buying+OR+(effect+AND+economic+NOT+depression)
```

Implementation In the getComponentData() method of your exteNd Director component, set up the boolean query like this:

```
...
// Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();
// Set the query type to be "text"
query.setQueryType(query.QUERY_TYPE_TEXT);
// Specify the query
query.setText("effect+AND+recession+AND+economic+AND+slowdown+AND+c
onsumer+AND+spending+OR+buying");
...
```

Conceptual queries

Description A *conceptual query* returns content that is related by meaning and ranked by relevance to the search criteria. The query string should be as specific as possible—usually a paragraph or at least a sentence.

Syntax *word1+word2+word3+... wordN*

Example `The+effect+of+the+recession+on+consumer+spending`

Implementation In the getComponentData() method of your exteNd Director component, set up the conceptual query like this:

```
...
// Instantiate a blank query object
```

```
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();
// Set query type to be "text"
query.setQueryType(query.QUERY_TYPE_TEXT);
// Specify the query
query.setText("The+effect+of+the+recession+on+consumer+spending");
...
```

Field queries

Description

A *field query* allows you to search metadata in documents, or metadata and content in a single method call. Metadata refers to standard metadata and/or custom metadata. Before issuing a field query, make sure you configure your search environment to specify the types of metadata you want to search—standard metadata and/or custom metadata—as described in Chapter 3, “Setting Search Options”.

Syntax

To specify the metadata to search, you must set up a field specifier list by calling the method setFieldSpecList() on an object that implements the EbiQuery interface. If you are searching more than one field, you must define a field boolean expression that specifies how the fields should be searched, then pass this expression as the second argument to setFieldSpecList(). If you are searching only one field, this argument should be **null**.

Field specifier list The syntax for the **field specifier list** is:

```
fname<Fieldname1=*value1*+fname<Fieldname2=*value2*+...
fname<FieldnameN=*value3*
```

You can issue field queries using the names of standard or custom metadata fields, as long as the metadata has been indexed.

Standard metadata field names are listed in DirectorDRE.cfg, located in:

```
exteNd Director installation directory\autonomy\engine
```

Field boolean expression The syntax for the **field boolean expression** is:

```
fname<Fieldname1+LOGICAL OPERATOR+fname<Fieldname2+LOGICAL
OPERATOR+... fname<FieldnameN
```

Logical operators can be the keywords **AND**, **OR**, and **NOT**.

Example

Suppose that in your Content Management (CM) repository you define a document type called **Research Study** and a custom field called **Topic**. If you want to find all research studies whose topic is **Economy**, follow these steps:

1. Define a field specifier list, as follows:

```
String fieldSpecList = "fnameDOCTYPENAME=*Research
Study*+fnameTopic=*Economy*";
```

2. Define a boolean expression to direct the search, as follows:

```
String fieldBooleanExpr = "fnameDOCTYPENAME+AND+fnameTopic";
```

3. Pass these two strings as arguments to the setFieldSpecList() method:

```
query.setFieldSpecList(fieldSpecList, fieldBooleanExpr);
```

Implementation

In the getComponentData() method of your exteNd Director component, set up the field query like this:

```
...
// Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();
// Set query type to be "text"
query.setQueryType(query.QUERY_TYPE_TEXT);
// Specify the query
query.setText("effect+recession+economic+slowdown+consumer+spending
");
// Provide field specifiers (to find research studies about the
// economy)
 String fieldSpecList = "fnameDOCTYPENAME=*Research
Study*+fnameTopic=*Economy*"
 String fieldBooleanExpr = "fnameDOCTYPENAME+AND+fnameTopic";
 query.setFieldSpecList(fieldSpecList, fieldBooleanExpr);
...
```

Notice that this example defines criteria for searching content (using the setText() method) as well as criteria for searching metadata (using the setFieldSpecList() method) in a single query expression. This is a powerful feature, because it allows you to search both content and metadata with a single call to the runQuery() method.

Fuzzy queries

Description

A *fuzzy query* tries to match terms even when they are misspelled.

Implementation

Fuzzy queries are implemented in the same way as conceptual queries except that you must define the query type to be **fuzzy**. In the getComponentData() method of your exteNd Director component, set up the fuzzy query like this:

```
...
// Instantiate a blank query object
```

```
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();
// Set query type to be "fuzzy"
query.setQueryType(query.QUERY_TYPE_FUZZY);
// Specify the query
query.setText("The+efect+of+the+recession+on+consumer+spending");
...
```

Get-all queries

Description A *get-all query* returns all documents.

Implementation You do not specify query strings for get-all queries, but you must define the query type to be **get all**. In the getComponentData() method of your exteNd Director component, set up the get-all query like this:

```
...
// Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();
// Set query type to be "get all"
query.setQueryType(query.QUERY_TYPE_GETALL);
...
```

NOTE: The get-all query is provided for debugging purposes, but not recommended for production use—because returning all documents can impact performance.

Keyword search

Description A traditional *keyword search* looks for occurrences of a search string.

Syntax `word1:+word2:+word3:+... wordN:`

Example `effect:recession:+economic:+slowdown:+consumer:+spending:+buying:`

Implementation These are the steps for implementing keyword searches:

1. Make sure that content can be indexed **and** stored in the DRE, by enabling the property **com.sssw.cm.fetch.store.content.*repository name*** in **config.xml** for the CM subsystem.

   📖 For more information about where project files are located, see the section on exteNd Director project structure in *Developing exteNd Director Applications*.

   📖 For more information about this property, see "Copy document contents into the DRE?" on page 126.

2. In the getComponentData() method of your exteNd Director component, set up the keyword query like this:

```
...
// Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();
// Set query type to be "text"
query.setQueryType(query.QUERY_TYPE_TEXT);
// Specify the query
query.setText("effect:+recession:+consumer:+spending:+buying:")
;
...
```

Proper name search

Description As its name suggests, a *proper name search* looks for proper names.

Syntax `ProperName1+ProperName2+ProperName3+... ProperNameN`

Example `Ralph+Waldo+Emerson`

Implementation Proper name searches are implemented in the same way as conceptual queries, except that you must define the query type to be a **name search**. In the getComponentData() method of your exteNd Director component, set up the proper name search like this:

```
...
// Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();
// Set query type to be "name search"
query.setQueryType(query.QUERY_TYPE_NAMESEARCH);
// Specify the query
query.setText("Ralph+Waldo+Emerson");
...
```

Proximity queries

Description A *proximity query* specifies that certain words in the query must occur close to each other.

Syntax Put single quotes around the words that should occur close together.

```
word1+‘word2+word3’+... wordN
```

Example

```
effect+recession+‘economic+slowdown’+‘consumer+spending’+‘consumer+
buying’
```

Usage In the getComponentData() method of your exteNd Director component, set up the proximity query like this:

```
...
// Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();
// Set the query type to be "text"
query.setQueryType(query.QUERY_TYPE_TEXT);
// Specify the query
query.setText("effect+recession+‘economic+slowdown’+‘consumer+spend
ing’+‘consumer+buying’");
...
```

Suggest similar documents

Description A *suggest-similar* query tries to find documents similar to other documents that were found to be relevant to your search criteria.

Syntax

```
document-identifier1+document-identifier2+document-identifier3+...
document-identifierN
```

The document identifier is either the document ID assigned to the document by the query engine (exteNd Director DRE) or its URL (DRE reference).

Example In the query expression, you can specify document identifiers as document IDs or document URL references.

Implementation To implement the **suggest-similar** query, follow these steps:

1. Set the query type to **suggest**, using the setQueryType() method.
2. Define suggest options, using the setSuggestOptions() method:
   - **2a** Indicate whether the document identifiers you specify in the query expression should be treated as document IDs (generated by the DRE) or document references.
   - **2b** Indicate whether the documents you specify in the query expression should be included or excluded from the query results.
3. Construct the query expression.

In the getComponentData() method of your exteNd Director component, set up the **suggest similar** query like this:

```
...
//Get the query engine delegate
EbiQueryEngineDelegate qe =
com.sssw.search.factory.EboFactory.getQueryEngineDelegate();

//Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();

query.selectAll();

//Set query type to "text" to get documents whose references
//or IDs can be passed on to the suggest-similar query
query.setQueryType(com.sssw.search.api.EbiQuery.QUERY_TYPE_TEXT);

//Set query string
query.setText("economic+recession+effects");

String ids = "";

//Search for documents that meet the original search criteria

try
{
   Iterator results = qe.runQuery(context, query, null,
false).iterator();
    //Put together the list of engine document IDs that constitute
    //the suggest-similar query
   boolean first = true;
   while (results.hasNext())
   {
      com.sssw.search.api.EbiQueryResult res =
(com.sssw.search.api.EbiQueryResult) results.next();
      String engineDocID =
res.getProperty(EbiQueryResult.PROP_ENGINE_DOC_ID);
      if (!first)
         ids += "+";
      else
         first = false;
```

```
            ids += engineDocID;
        }
    }
    catch (Exception e)
    {
        if (m_log.isError())
            m_log.error(e);
    }

    if (!"".equals(ids))
    {
        //Set query type to "suggest"
        query.setQueryType(EbiQuery.QUERY_TYPE_SUGGEST);
        query.setSuggestOptions(true, true);
        query.setText(ids);
        //Run the suggest-similar query
        Iterator results = qe.runQuery(context, query, null, false);
        …
    }
```

In this example, the setSuggestOptions() method specifies that:

- Identifiers in the query expression be interpreted as document IDs
- Documents referenced in the query expression **suggestQuery** be excluded from the query results

To use document URL references (instead of document IDs) in this example, modify the following statements:

- Change:

  ```
  String engineDocID =
  res.getProperty(EbiQueryResult.PROP_ENGINE_DOC_ID);
  ```

  to:

  ```
  String engineDocID =
  res.getProperty(EbiQueryResult.PROP_ENGINE_DOC_REF);
  ```

- Change:

  ```
  query.setSuggestOptions(true, true);
  ```

  to:

  ```
  query.setSuggestOptions(false, true);
  ```

Thesaurus queries

Description

A *thesaurus query* analyzes not only the terms in the query expression, but also associated terms (or synonyms) that you define in a separate thesaurus.

To issue a thesaurus query, you follow these basic steps:

1. Decide what terms you want to search for.
2. For each term, create a thesaurus document that contains synonyms or words you want to associate with that term.
3. Construct your query expression.
4. Mark the query to run as a thesaurus query by calling the setIsThesaurusQuery() method on an object that implements the EbiQuery interface.
5. Get a repository descriptor object for each thesaurus DRE you create.
6. Set thesaurus options by calling the setThesaurus() method on an EbiQuery object.
7. Run the query.

Thesaurus DRE

For each term you plan to search, you need to create a thesaurus DRE (Step 2 above). This section explains how.

NOTE: This section is based on information adapted from the *Autonomy Server 2.2* manual from Autonomy, Inc.

➢ **To create a thesaurus DRE:**

1. Create a folder called **ThesaurusDRE** in the folder where the Autonomy integration files are installed.

   By default, the Autonomy integration files are stored in the exteNd Director installation directory at:

   ```
   exteNd Director\autonomy
   ```

2. Copy all files from the existing exteNd Director DRE to the **ThesaurusDRE** folder.

   These files are stored in the exteNd Director installation directory at:

   ```
   exteNd Director\autonomy\engine
   ```

3. In the **ThesaurusDRE** folder, rename the following files:

Change this	To this
DirectorDRE.exe	**Thesaurus**DRE.exe
DirectorDREadmin.exe	**Thesaurus**DREadmin.exe
DirectorDRE.cfg	**Thesaurus**DRE.cfg

4 Edit **ThesaurusDRE.cfg** to point to different ports, as follows:

Change this	To this
The QUERYPORT setting	QUERYPORT=**8000**
The INDEXPORT setting	INDEXPORT=**8001**

5 In your favorite text editor, create a thesaurus document.

The thesaurus document should contain a list of associated words, separated by carriage returns, as in this example:

6 Save the thesaurus document.

7 Use the DRE Administration console to import and index your thesaurus document in the thesaurus DRE. Follow these steps:

7a Double-click **ThesaurusDRE.exe**.

7b Connect the DRE Administration console to the thesaurus DRE by double-clicking **ThesaurusDREadmin.exe** in the **ThesaurusDRE** folder.

Make sure the connection was successful by checking for the check mark symbol on the DRE Administration console window:

7c Create a new DRE database by entering **ThesaurusDB** in the New Database Name text box and clicking the **Create New Database** button.

7d If an alert box appears, click **Yes** to confirm that you want to create the database.

7e Select the **Import-Index** tab.

7f Click the **Import Files into IDX format** button.

The Main Import Settings dialog opens.

7g Click the **Add Files** button.

7h Browse to your thesaurus document, select it, and click **Open**.

The thesaurus document should appear in the list of input files to import.

7i Select **ThesaurusDB** as the destination database and click **OK**.

7j Click **Yes** to confirm that you want to add the thesaurus document's IDX file to the list of documents to index.

7k Click **Index into DRE now!** to index the thesaurus document.

Now you are ready to implement a thesaurus query.

Implementation

In the getComponentData() method of your exteNd Director component, set up the **thesaurus** query like this:

```
...
//Instantiate a blank query object
com.sssw.search.api.EbiQuery query =
com.sssw.search.factory.EboFactory.getQuery();

//Construct your query expression
query.setText("feline");

//Mark the query to run as a thesaurus query
query.setIsThesaurusQuery(true);

//Get a repository descriptor for the thesaurus DRE
EbiRepositoryDesc thesaurus =
com.sssw.search.factory.EboFactory.getRepositoryDesc("141.155.166.1
81", 8000, 8001, "ThesaurusDB");

//Set thesaurus options
query.setThesaurus(thesaurus);

//Run the query
Collection results = queryEngine.runQuery(context, query,
repositories, true);
...
```

Index

S

W