SYBASE®

Reference Manual

# OmniQ™ Enterprise Edition

3.0

[ Windows 2000/XP/2003 ]

DOCUMENT ID: DC00411-01-0300-01

LAST REVISED: December 2005

Copyright © 2003-2005 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase, the Sybase logo, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Warehouse, Afaria, Answers Anywhere, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Translator, APT-Library, AvantGo Mobile Delivery, AvantGo Mobile Inspection, AvantGo Mobile Marketing Channel, AvantGo Mobile Pharma, AvantGo Mobile Sales, AvantGo Pylon, AvantGo Pylon Application Server, AvantGo Pylon Conduit, AvantGo Pylon PIM Server, AvantGo Pylon Pro, Backup Server, BizTracker, ClearConnect, Client-Library, Client Services, Convoy/DM, Copernicus, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DataWindow .NET, DB-Library, dbQueue, Developers Workbench, DirectConnect, DirectConnect Anywhere, Distribution Director, e-ADK, E-Anywhere, e-Biz Impact, e-Biz Integrator, E-Whatever, EC Gateway, ECMAP, ECRTP, eFulfillment Accelerator, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, EWA, Financial Fusion, Financial Fusion Server, Gateway Manager, GlobalFIX, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, M2M Anywhere, Mach Desktop, Mail Anywhere Studio, Mainframe Connect, Maintenance Express, Manage Anywhere Studio, M-Business Channel, M-Business Network, M-Business Server, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, mFolio, Mirror Activator, MySupport, Net-Gateway, Net-Library, New Era of Networks, ObjectConnect, ObjectCycle, OmniConnect, OmniQ Enterprise Edition, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, PocketBuilder, Pocket PowerBuilder, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, QAnywhere, Rapport, RemoteWare, RepConnector, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Report-Execute, Report Workbench, Resource Manager, RFID Anywhere, RW-DisplayLib, RW-Library, S-Designor, SDF, Search Anywhere, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SOA Anywhere, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, STEP, SupportNow, S.W.I.F.T. Message Format Libraries, Sybase Central, Sybase Client/Server Interfaces, Sybase Financial Server, Sybase Gateways, Sybase IQ, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SybFlex, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, TradeForce, Transact-SQL, Translation Toolkit, UltraLite, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server, XcelleNet, and XP Server are trademarks of Sybase, Inc. 06/05

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

# Contents

# About This Book

**Audience**

This guide is for OmniQ™ Enterprise Edition system administrators and other professionals who are familiar with their system's environment, networks, disk resources, and media devices.

**How to use this book**

This book contains the following chapters:

- Chapter 1, "Overview," describes the features and architecture of the OmniQ Enterprise product.
- Chapter 2, "Configuring OmniQ Enterprise," describes how to configure the container XML, hub module, and remote modules.
- Chapter 3, "Configuring the Hyena Servlet Container," describes how to configure the Hyena servlet container.
- Chapter 4, "Document Management," describes how to use the system interface to create, manage, and edit document stores and document groups. This chapter also describes how to use search.
- Chapter 5, "Administration," describes how to administer HTTP handlers and indexing.
- Chapter 6, "Developing and configuring custom files," provides information on developing custom filters, parsers, and text splitters.
- Appendix A, "Generated Files," gives the names and locations of generated module files.

**Related documents**

**OmniQ Enterprise Edition documentation** The following OmniQ Enterprise Edition documents are available:

- The OmniQ Enterprise Installation Guide and Release Bulletin provides information on installation.

**Other sources of information**

Use the Sybase Product Manuals Web site to learn more about your product:

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

### Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click Certification Report.
3. In the Certification Report filter select a product, platform, and timeframe and then click Go.
4. Click a Certification Report title to display the report.

❖ **Finding the latest information on component certifications**

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
3. Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

### Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

**Conventions**

The syntax conventions used in this manual are:

| Key | Definition |
|---|---|
| commands and methods | Command names, command option names, utility names, utility flags, Java methods/classes/packages, and other keywords are in lowercase Arial font. |
| *variable* | Italic font indicates:<br>• Program variables, such as *myServer*<br>• Parts of input text that must be substituted; for example:<br>*Server*.log<br>• File names |
| File \| Save | Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File \| Save indicates "select Save from the File menu." |
| `package 1` | Monospace font indicates:<br>• Information that you enter in a GUI interface, a command line, or as program text<br>• Sample program fragments<br>• Sample output fragments |

*%OMNIQ_3.0%* refers to the OmniQ Enterprise installation directory; for example, *C:\%OMNIQ_3.0%\bin*.

**If you need help**

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

# Overview



## Managing unstructured information

In many organizations, employees keep their research, financial projections, and presentations on local PCs or team-shared space on the company network. Accessing, or even finding, such information often proves difficult when staff or storage rules and structures change.

Sybase OmniQ technology extracts and processes the text content from Web pages, e-mail messages, documents, and other forms of information where the content is unstructured. The ability to automatically process this content removes the need to index or describe information manually and allows organizations to automate such common business operations as data capture, retrieval, and linking. OmniQ technology offers an efficient and cost-effective solution for searching unstructured information, regardless of the format, and independent of the language in which the content is written.

The Internet offers familiarity with the most common type of search and retrieval technology—keyword search. Most people are familiar with the process of retrieving the information by typing one or two relevant keywords into a search engine. This process can be frustrating, ineffective, and time-consuming.

Keyword search technology requires a business to identify documents by associating keywords with the document, which are then used for subsequent retrieval. This process is known as document “tagging.” There are a number of problems with this approach:

- It is difficult to select a small number of keywords to represent all the concepts in a long document.

- Chosen keywords may be language-specific.
- The process of tagging information is expensive.

OmniQ provides the means to automatically capture and retrieve information based on concepts rather than keywords. Through the use of proprietary algorithms, OmniQ delivers a language-independent product capable of operating without the costly overhead associated with tagging. This provides an essential tool for managing the proliferation of unstructured information in today’s business environment.

OmniQ Enterprise Edition offers a number of features that allow today's organizations to ease or eliminate the time and cost required to support the demands of managing an organizations unstructured information. These features include:

- The support of more than 250 different formats of data, including most types of document, presentation, spreadsheet, and Web content formats.
- The automatic capture and aggregation all of unstructured data.
- The elimination of preprocessing or manual tagging of files, greatly improving the accuracy and efficiency of document retrieval.
- The extraction of paragraphs from matching documents.
- The ability to find similar documents by automatically providing a set of relevant content that is conceptually related to each document.
- The ability to scale to millions of documents using a fully distributed architecture
- The ability to query and process using natural language.
- Language independence.
- A well-defined Java and XML API that allows OmniQ Enterprise Edition to be integrated easily into other applications.

# Architecture

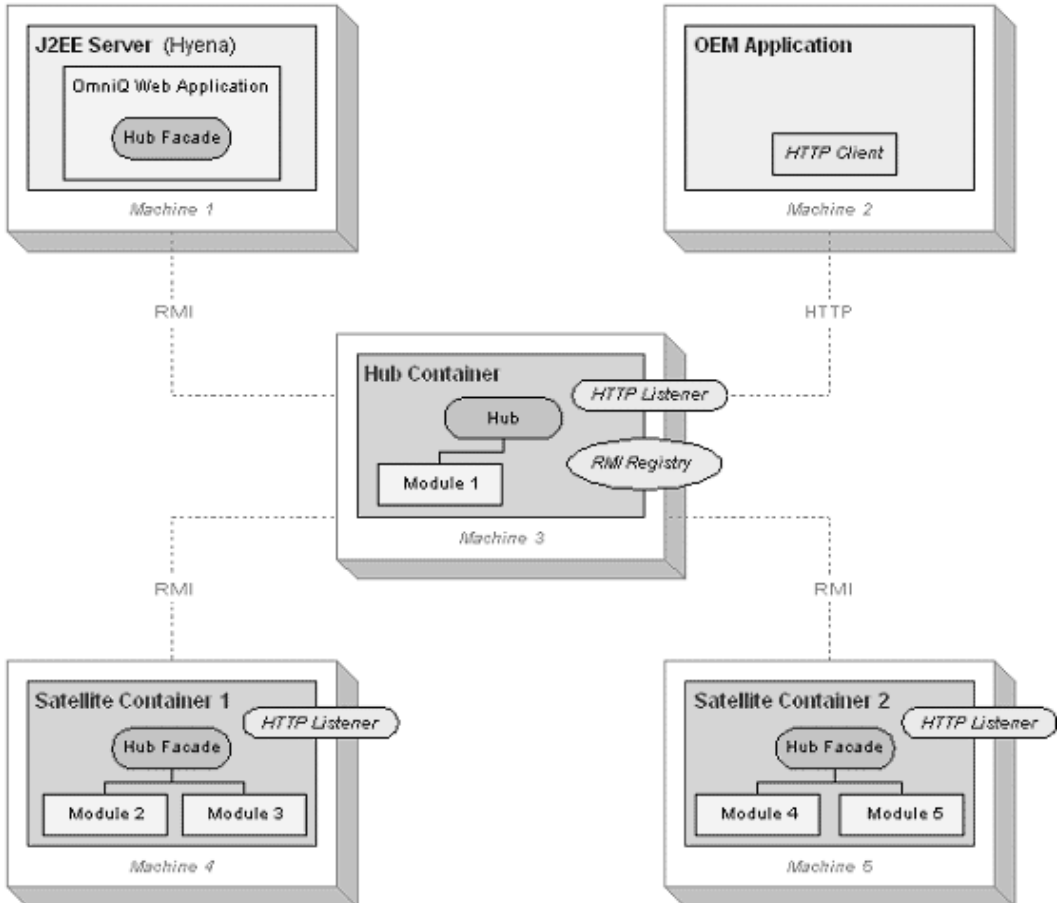
OmniQ Enterprise is a fully distributed system, with a central hub server and one or more satellite servers. Each server can contain one or many containers with one or more modules deployed in each container. The exact number of servers, containers, and modules is dependent on the needs of the OmniQ Enterprise installation.

The example architecture in Figure 1-1contains:

- A central hub
- Two satellite containers
- A J2EE server containing the Web application
- OEM application connecting to OmniQ Enterprise

See Chapter 2, “Configuring OmniQ Enterprise” for information on the various modules that comprise OmniQ Enterprise.

***Figure 1-1: Example OmniQ Enterprise architecture***



CHAPTER 2

# Configuring OmniQ Enterprise

This chapter describes the key configuration parameters for containers, the hub, and modules. It includes tips on using the configuration files, and changing parameters.



## Configuring the container XML file

Each container has an XML configuration file that determines if the container loads the hub and lists the modules to be loaded. The configuration files also allow you to set system properties for the Java virtual machine (JVM) in which the container runs. The hub and modules run in containers, and thus share some configuration parameters.

The XML is formed with a root container tag enclosing zero or more SystemProperty tags, exactly one Hub tag, zero or more Module tags, and zero or one Data tag. The format is as follows:

```
<Container id="1" port="8001">
   <SystemProperty name="exampleName" value="exampleValue" />
   <Hub local="true" host="127.0.0.1" port="7000" bindName="Hub"
   logEvents="true" />
   <Module id="101" class="com.omniq.xmp.ExampleModule"
   name="Example Module" />
   <Data directory="G:\example\data" />
</Container>
```

The modules can contain zero or more HttpHandler tags, which in turn can contain zero or more Param tags. For example:

```
<Module id="101" class="com.omniq.xmp.ExampleModule" name="Example Module">
   <HttpHandler class="com.omniq.xmp.ExampleHandler"
resourceURI="/handler/example">
       <Param name="exampleName" value="exampleValue" />
   </HttpHandler>
</Module>
```

## The hub

The hub is the core component of the system. It is a special module that is the global coordinator of OmniQ Enterprise. The container, which loads the real hub (sometimes referred to as the hub container), also runs a Java RMI registry to listen for remote requests. Satellite containers load a hub facade to handle communication with the real hub. All queries and administration requests are negotiated by the hub.

## Configuration and ID conventions

OmniQ Enterprise is installed with two example configurations:

- Single-server configuration – the files for this configuration are located in *%OMNIQ_3.0%\OmniQ\config\Container.0.xml*.
- Two-server configuration – this configuration requires two files—one for the hub container and one for the satellite container. The files for this configuration are respectively located in:
  - *%OMNIQ_3.0%\OmniQ\config\Container.1.xml*
  - *%OMNIQ_3.0%\OmniQ\config\Container.2.xml*

Containers, hub facades, and modules are not automatically assigned unique IDs (UIDs)—you must configure them manually. The UIDs must be within the range of 1 to the UID Generator's seed value, which is 1,000 by default. See "Unique ID (UID) Generator" on page 10.

If a container or module is assigned an ID greater than the seed value, it may conflict with an internally generated ID and cause an unexpected error later, therefore, avoid doing this.

Because these UIDs are split across several files, you must employ a numbering convention. The example two-server configuration files use the following conventions:

- Container ID – a value from 1 – 9.
- Container XML – includes the container ID in its name, for example, `Container.1.xml`.
- HTTP listener – the container's HTTP listener binds to the port number 8000 plus the container's ID. For example, the port is 8001 for Container 1 and 8002 for Container 2.
- Hub container – always binds the RMI Registry on port 7000.
- Hub facade ID – on satellite containers this is 10 times the container ID. For example, the hub facade ID for Container 2 is 20.

**Note** An exception is that the default Web application always allocates its hub facade ID as 999 as it does not need to follow the other conventions.

- Modules – each module has the ID of 100 times the container ID + *N*. For example, the first module ID on Container 1 is 101, the second is 102, the third is 103 and so on.

The single server configuration does not strictly follow this convention as all IDs are visible in the same file, making the assigning of duplicate IDs less likely. If the OmniQ Enterprise installation requires more than nine servers, a different convention is necessary.

Table 2-1 shows the attributes for the container tag.

***Table 2-1: Container tag***

| Attribute | Default value | Description |
|---|---|---|
| id | None | The unique ID of the container. This value is used to identify the container when it registers itself with the hub. |
| port | None | The TCP/IP port on which the container's embedded HTTP server listens. |

Table 2-2 shows the attributes for the SystemProperty tag. The system properties include JVM settings, Stellent SearchML settings, and global indexing and querying parameters for modules loaded within the container.

***Table 2-2: SystemProperty tag***

| Attribute | Default value | Description |
|---|---|---|
| name | None | The name of the Java system property to set. In other words, the name you use within the Java process when using the java.lang.System.getProperty (java.lang.String) method. |
| value | None | The string value to associate with the property name. |

Table 2-3 shows the attributes for the hub tag.

***Table 2-3: Hub tag***

| Attribute | Default value | Description |
|---|---|---|
| local | false | When set to true, the real hub is loaded into the current container. Otherwise, the container loads a hub facade. |
| id | None | This is the unique ID of the hub facade, which is used when the hub facade registers itself with the real hub. If the hub is local, this attribute is not required. |
| host | 127.0.0.1 | If the hub is not local, the facade hub uses this value to contact the real hub on the RMI registry. |
| port | None | The TCP/IP port on which the RMI registry started by the hub container is bound. When the hub is local, the port is used when starting the RMI registry. When the hub is not local, the port is used to connect to the RMI registry to access the real hub. |
| bindName | Hub | The name by which the hub is bound on the RMI registry. When the hub is local, bindName is used to bind the hub. When the hub is not local, bindName is used to look up the hub. |
| logEvents | false | This value indicates whether or not the event log should be enabled. The location of the hub is irrelevant. |
| logDirectory | <*data.directory*>*log* | |

Table 2-4 shows the attributes for the module tag.

***Table 2-4: Module tag***

| Attribute | Default value | Description |
|---|---|---|
| id | None | The unique ID of the module. id is used to identify the module when it is registered with the hub. |
| name | None | The name of the module. |
| class | None | The name of the Java class that is the module. |
| enabled | true | If set to false, the module is not loaded. |

Table 2-5 shows the attributes for the HttpHandler tag.

***Table 2-5: HttpHandler tag***

| Attribute | Default value | Description |
|---|---|---|
| class | None | The name of the Java class that is the HTTP handler (the resource). |
| resourceURI | None | The HTTP URI of the HTTP handler resource. This is used to complete the URL. For example, http://<*container.host*>:<*container.port*>/<*resourceURI*>. |
| name | None | The name of the parameter to pass to the HTTP handler. |
| value | None | The string value to associate with the parameter name. |

# Modules

This section describes the OmniQ Enterprise modules and their configurations. Each module runs in a container and, with a few restrictions, can either be run in its own separate container on different servers, or grouped with other modules within a single container.

The available modules are:

- Unique ID (UID) Generator
- Document Group Manager
- Text Manager

- Term Lexicon Manager
- Term Lexicon Manager Delegate
- Metadata Manager
- Metadata Manager Delegate
- Query Manager
- Repository Manager
- Filter Factory

## Unique ID (UID) Generator

The Unique ID Generator settings are loaded through the *UIDGeneratorModule.default.xml* configuration file. Table 2-6 shows the parameters in this file.

***Table 2-6: UIDGeneratorModule.default.xml parameters***

| Parameter | Default | Description |
|---|---|---|
| filename | uid.dat | The file that stores the next unique ID. |
| alwaysOpen | false | If set to true, the underlying Java class leaves the file handle open to the filename above. |
| seed | 1,000 | The UID Generator seed starts from 1,000, as numbers less than 1,000 are reserved by OmniQ Enterprise as module IDs. |

## Document Group Manager

The Document Group Manager module is contained in the hub container and does not have a configuration file associated with it, as initially the system contains no predefined document groups.

## Text Manager

The Text Manager module settings are loaded through the *TextModule.default.xml* configuration file. Table 2-7 shows the parameters in this file.

***Table 2-7: TextModule.default.xml parameters***

| Parameter | Default | Description |
|---|---|---|
| min.term.length | 2 | The minimum term length deemed valid for indexing. This is not taken into account in the list of preserved terms and does not apply to single-digit terms. |
| max.term.length | 20 | The maximum term length deemed valid for indexing. This value must match the Term Lexicon Manager parameter term.length.max. |
| stopwords.filename | *stopwords_en.txt* | This file contains a list of stopwords to remove during the indexing and querying processes to improve system performance. See “Stopwords” on page 20. |
| preserved.terms.filename | *preserved_terms_en.txt* | This file contains the list of preserved terms that are not stemmed during indexing. The list can also include terms less than the minimum term length defined in the min.term.length parameter. See “Preserved terms” on page 21. |
| term.splitter.class | *com.isdduk.text.BreakIteratorSplitter* | Specifies the Java class used for breaking text into separate words. The default BreakIteratorSplitter handles all double-byte character sets. |
| term.stemmer.class | *com.isdduk.text.Porter2Stemmer* | Specifies the Java class used for term stemming. The default Porter2Stemmer is for English text. |
| query.augmentor.filename | *query_aug_en.txt* | This file contains a list of synonyms and acronyms. See “Synonyms and acronyms (query augmentation)” on page 21. |
| parsers.filename | *Parsers.xml* | The name of the file in the *config* folder that contains the list of text parsers. |

You can set the term splitter and stemmer classes to language-independent classes or to language-specific classes. Language-specific stemmers allow an increase in system performance when OmniQ Enterprise is going to index documents in one language only.

## Term Lexicon Manager

The Term Lexicon Manager settings are loaded through the *TermLexiconModule.default.xml* configuration file. Table 2-8 shows the parameters for the Term Lexicon Manager.

***Table 2-8: Term Lexicon Manager parameters***

| Parameter | Default | Description |
|---|---|---|
| term.length.max | 20 | The maximum term length deemed valid for indexing. This value must match the Text Manager parameter max.term.length. See “Text Manager” on page 10. |
| cache.capacity | 131,072 | The number of terms stored in memory to improve indexing and querying performance. |
| cache.useRootChildrenCache | true | If set to true, the underlying term lexicon data structures cache some of their structure in memory to improve indexing and querying performance. |
| unify.size.threshold | 10,000 | Determines how many terms in each term lexicon segment are stored in memory before being written to disk. |
| unify.idle.threshold | 120,000 | The time, in milliseconds, that the Term Lexicon Manager remains idle before unifying the pending terms. The idle time is restarted when a new term is added, or when an existing term is looked up. |
| number.of.segments | 20 | The number of term lexicon segments. For maximum efficiency, the value should be equal to the term.length.max. |
| minimization.factor | 50 | The branching factor of the underlying term lexicon segments. This parameter affects the lookup performance of the Term Lexicon Manager. Do not change this value without consulting Sybase Technical Support. |

## Term Lexicon Manager Delegate

The Term Lexicon Manager Delegate allows terms and their unique IDs to be cached locally. This saves the remote module from excessive communication with the hub’s Term Lexicon Manager. The Term Lexicon Manager Delegate settings are loaded through the *TermLexiconModuleDelegate.default.xml* configuration file.

The only parameter in the *config* file is cache.capacity, which represents the number of terms that can be cached locally.

## Metadata Manager

The Metadata Manager settings are loaded through the *MetadataModule.default.xml* configuration file. Table 2-9 shows the parameters in this file.

***Table 2-9: MetadataModule.default.xml parameters***

| Parameter | Default | Description |
|---|---|---|
| metadata.filename | *Metadata.ser.gz* | The name of the file to where the metadata fields are serialized. Do not change this parameter. |
| uid.filename | uid.dat | The name of the file that stores the next unique ID, which is used when creating new metadata fields. Do not change this parameter. |

## Metadata Manager Delegate

The Metadata Manager Delegate allows metadata fields and their unique IDs to be cached locally. This saves the remote module from excessive communication with the hub's Metadata Manager. The Metadata Manager Delegate settings are loaded through the *MetadataModuleDelegate.default.xml* configuration file.

***Table 2-10: Metadata Manager Delegate parameter***

| parameter | Default | Description |
|---|---|---|
| metadata.filename | *Metadata.ser.gz* | The name of the file to where the metadata fields are serialized to. Do not change this parameter. |

## Query Manager

The Query Manager settings are loaded through the *QueryModule.default.xml* configuration file. Table 2-11 shows the Query Manager parameters.

***Table 2-11: Query Manager parameters***

| parameter | Default | Description |
|---|---|---|
| cache.termStats.maxSize | 131,072 | The number of term statistics stored in memory to improve querying performance. |

| parameter | Default | Description |
|---|---|---|
| queryRunnerPool.size | 20 | The number of concurrent threads used to run queries. |
| queryParsers.filename | *QueryParsers.xml* | The name of the file in the *config* folder that contains the list of query parsers. |

## Repository Manager

The Repository Manager has no configuration settings and is used to allow other containers to pass on the text from documents located in other containers.

## Filter Factory

The Filter Factory settings are loaded through the *FilterFactory.default.xml* configuration file.

The list of default filters in the configuration file are:

- Default HTML filter
- Default EML filter
- SearchML export multi-filter
- SearchML filter

Each filter specifies a number of settings, which determine which class is loaded for the filter, which paragraph extractor is used, and the MIME types to which the filter applies. Table 2-12 shows the filter setting parameters.

***Table 2-12: Filter settings***

| Parameter | Default | Description |
|---|---|---|
| className | None | The Java class that defines the filter. |
| extractorClassName | None | The Java class used for extracting paragraphs from the filtered text |
| mimeTypes | None | The list of MIME types that are associated with the filter. |
| timeout | 45,000 | Indicates the time in milliseconds the filter waits while filtering a document. If the filter exceeds the given time, the filter aborts. This parameter is used mainly by the Stellent filter. |
| keepTempFiles | false | If set to true, the filter keeps any temporary files produced during the filtering process. This is used mainly by the Stellent filter. |

In addition to the filter-specific settings, there are a number of general filter settings that help the extractors determine the paragraphs. The filter ensures that each paragraph is between the minimum and maximum lengths and aims for the ideal paragraph length.

Table 2-13 shows the paragraph length settings.

***Table 2-13: Paragraph settings***

| Parameter | Default | Description |
|---|---|---|
| default.minParaLen | 250 | The minimum number of characters in a paragraph |
| default.idealParaLen | 500 | The ideal number of characters in a paragraph |
| default.maxParaLen | 1,000 | The maximum number of characters in a paragraph |

## Default HTML filter

The default HTML filter is a custom filter used to parse HTML files.

Table 2-14 shows a list of the parameter settings used by the default HTML filter.

***Table 2-14: Default HTML filter parameters***

| Parameter | Value |
|---|---|
| className | com.omniq.filter.html.HTMLFilter |
| extractorClassName | com.omniq.filter.SegmentOrientedExtractor |
| mimeTypes | text/html |
| timeout | N/A |
| keepTempFiles | N/A |

## Default EML filter

The default EML filter is a custom filter used to parse EML files that are used by some e-mail systems.

Table 2-15 shows the parameter settings used by the default EML filter.

***Table 2-15: Default EML filter parameters***

| Parameter | Value |
|---|---|
| className | com.omniq.filter.eml.EMLFilter |
| extractorClassName | com.omniq.filter.eml.EMLExtractor |
| mimeTypes | message/rfc822 |
| timeout | N/A |
| keepTempFiles | N/A |

## SearchML export multi-filter

The SearchML export multi-filter is the default filter used to parse all other MIME types that do not have their own specific filter. The output from the SearchML export multi-filter is an XML document that contains the raw text and associated document metadata.

Table 2-16 shows a list of the parameter settings used by the SearchML export multi-Filter.

***Table 2-16: SearchML multi-filter parameters***

| Parameter | Value |
|---|---|
| className | com.omniq.filter.stellent.SearchMLFilterExport |
| extractorClassName | com.omniq.filter.SegmentOrientedExtractor |
| mimeTypes | * |
| timeout | 45000 |
| keepTempFiles | false |

## SearchML filter

The SearchML filter is an internal filter used to parse the XML output from the SearchML export multi-filter as given above.

Table 2-17 shows a list of the parameter settings used by the default SearchML filter.

***Table 2-17: SearchML parameters***

| Parameter | Value |
|---|---|
| className | com.omniq.filter.stellent.SearchMLFilter |
| extractorClassName | com.omniq.filter.SegmentOrientedExtractor |
| mimeTypes | text/x-searchml |
| timeout | N/A |
| keepTempFiles | N/A |

# Language-specific configuration

You can optimize OmniQ Enterprise if you know that all the source documents are in one language. This optimization includes using stemming algorithms, stopwords, and preserved terms.

Configure these settings using the Text Manager and Query Manager.

## Parsers

Parsers are used for processing metadata values, which are generally received as string key/value pairs. While document body text is processed by the system term splitter and stemmer, metadata often must be handled differently (as metadata values can be not only strings, but also numeric and date types). The parsers loaded by the Text Manager are referenced in the metadata field parser and query parser XML configuration files.

There are four different types of parsers:

- String
- Numeric decimal
- Numeric integer
- Date (time)

A string parser is always handled by internal classes. You can build custom numeric and dates parsers and plug them into the system if necessary. Table 2-18 shows the attributes for the Parser tag.

***Table 2-18: The Parser tag***

| Attribute | Default | Description |
|---|---|---|
| identifier | None | The Parser instance's identifier. This must be a name and a unique ID separated by an underscore (_). |
| class | None | The Java implementation class. |

Table 2-19 shows the attributes for the Param tag.

***Table 2-19: The Param tag***

| Attribute | Default | Description |
|---|---|---|
| name | None | The name of the parameter to pass to the parser. |
| value | None | The string value to associate with the parameter name. |

OmniQ Enterprise comes with the preconfigured parsers, shown in Table 2-20, which are adequate for most common metadata types.

***Table 2-20: Preconfigured parsers***

| Name | float_1 |
|---|---|
| Class | **com.isdduk.text.SimpleFloatParser**<br>This class parses strings representing decimal numbers into actual decimal numbers. For instance, the string "3.142" is parsed into Java float 3.142. |

| Name | integer_2 |
|---|---|
| Class | **com.isdduk.text.IntegerParser**<br>This class parses strings representing an integer number into an actual integer number; any floating-point information is discarded. For instance, both "3" and "3.142" are parsed into Java int 3. |

| Name | dateUK_3 |
|---|---|
| Class | **com.isdduk.text.DateFormatParser** |
| Name | **dateMs1970_4** |
| Class | **com.isdduk.text.Ms1970DateParser** |

| | |
|---|---|
| Parameter | Name – roundTo.<br>Value – choose a year, month, day, hour, minute, second, or any other value to denote no rounding should take place.<br>This class is date parser, which effectively parses strings representing long integer (64-bit) numbers, which themselves represent dates as the number of milliseconds since 1 January 1970. The preconfigured instance rounds dates to the nearest day (UTC). |

| **Name** | **intB2KB_5** |
|---|---|
| **Class** | **com.isdduk.text.B2KBIntParser**<br>This class parses strings representing byte-size numbers and converts them into kilobyte-size numbers. For instance, the string “2048” (bytes) is parsed as Java int 2 (kilobytes). |

| **Name** | **datePDF_6** |
|---|---|
| **Class** | **com.isdduk.text.PDFDateParser** |
| Parameter | Name – roundTo.<br>Value – choose a year, month, day, hour, minute, second, or any other value to denote no rounding should take place.<br>This class handles the PDF date format, in which dates are formatted “D:20030602143803+01'00'”. The preconfigured instance rounds dates to the nearest day (UTC). |

## Query parsers

The query parsers’ settings are loaded through the *config* file as specified in the queryParsers.filename parameter in the Query Manager configuration file.

This enables the OmniQ Enterprise administrator to configure parsers for querying metadata fields. In some cases, you might want to search metadata fields using formats different from those that were indexed. For instance, a date metadata field might be indexed in the YYYY-MM-DD format yet searched on using a DD/MM/YYYY format. Table 2-21 shows the metadata field tag attributes.

***Table 2-21: Metadata field tag attributes***

| Attribute | Default value | Description |
|---|---|---|
| name | None | The internal name of the metadata field to which this setting applies. |
| parser | None | The identifier of the parser used to parse the query metadata values. |

**Note** You can change the query parser configuration at any time, however, changes take effect only after you restart the container hosting the Query Module.

## Stopwords

Stopwords are common words such as “I,” “a,” “an,” “the,” and so on, that are ignored during the indexing or querying process. Removing the most common words during the indexing process keeps index sizes smaller, which enhances performance.

You can change the list of stopwords in one of two ways:

- Edit the list of words in the default stopwords file located in *%OmniQ_3.0%\OmniQ\config\stopwords_en.txt*, or
- Create a new stopwords file and configure the Text Manager to read from the new file, by editing *%OmniQ_3.0%\OmniQ\config\TextModule.default.xml* and changing the value of the stopwords.filename parameter to point to the new file.

**Note** The stopword list must be UTF-8 encoded. Because the words on the stoplist are ignored when you index documents, (in other words, the document is indexed as if the words on the stoplist did not exist), you must make any changes to the stoplist before you index. If you have already indexed your documents, and add new stopwords, the words are not included in your query but the disk space consumed by that word’s associated data is not reclaimed until you reindex your documents.

Removing stopwords after you have already indexed your documents has no affect until you reindex your documents.

## Preserved terms

You can use preserved terms to ensure that some terms are *not* removed as part of the indexing and querying processes. For example, the term “US” would be removed from any extracted text if the term “us” was entered in the list of stopwords. The case-sensitive list of preserved terms ensures that “us” will be removed, but “US” is indexed and made available to the query calculations.

You can change the list of preserved terms in one of two ways:

- Edit the list of words in the default preserved terms file, preserved_terms_en.tx, located in *%OMNIQ_3.0%\OmniQ\config*, or
- Create a new preserved terms file and configure the Text Manager to read from the new file by editing *%OMNIQ_3.0%\OmniQ\config\TextModule.default.xml* and changing the value of the preserved.terms.filename parameter to point to the new file.

---

**Note** The preserved term list must be UTF-8 encoded and changed before you index any documents, as preserved terms require special handling during indexing. If you have already indexed documents, changing the preserved terms has no effect, as the terms must still be queried exactly as before to produce matches (as the terms are fixed in the indexes).

---

## Synonyms and acronyms (query augmentation)

In OmniQ Enterprise, the use of synonyms and acronyms is collectively called query augmentation. Synonyms are implemented as lists of words that are considered to have the same meaning: For example:

- drowsy, lethargic, listless, sleepy
- holiday, vacation

When a term featured in a list is used as a query parameter, all the other words in the list are appended to the query. For example, the query `The medicine made me drowsy`, when augmented using the synonym examples above, becomes `The medicine made me drowsy, lethargic, listless, sleepy`.

Acronyms are implemented as a list of keys (or a single key) with a corresponding list of values. In the following example the keys “HTML” and “HTM” have the values “Hypertext Markup Language”:

- HTML, HTM = Hypertext Markup Language
- USA, US = United States of America

Acronyms can augment a query in two ways:

- Acronym expansion – when a term featured in an acronym key list is found in a user's search terms, all the corresponding values are added to the original query. For example, the query `How to write HTML documents`, when augmented with the acronym examples above, becomes `How to write HTML Hypertext Markup Language documents`.
- Acronym resolution – when a list of terms featured as an acronym values list is found in a user's search terms, all the corresponding keys are added to the original query. For example, the query `How to write Hypertext Markup Language documents`, when augmented with the acronym examples above, becomes `How to write Hypertext Markup Language HTML HTM documents`.

You can change the list of synonyms and acronyms in one of two ways:

- Edit the list of words in the default synonyms and acronyms file located in *%OmniQ_3.0%\OmniQ\config\query_aug_en.txt*, or
- Create a new query augmentation file and configure the Text Manager to read from the new file, by editing *%OmniQ_3.0%\OmniQ\config\TextModule.default.xml* and changing the value of the query.augmentor.filename parameter to point to the new file.

---

**Note** The query augmentation list must be UTF-8 encoded. Synonyms and acronyms are processed at runtime, so you can edit the augmentation list without having to reindex any documents. You must, however, restart OmniQ Enterprise to load the new synonym and acronym lists.

---

# Metadata

OmniQ Enterprise supports four types of metadata fields:

- TEXT
- DATE
- FLOAT

- INT

Each of these types supports a number of different parsers, which format or modify the metadata in different ways (see “Parsers” on page 17). For example, different DATE parsers parse the date value differently depending on the date format specified.

Each metadata field is configured to be one of these four types and use one of the valid parsers.

OmniQ Enterprise cannot automatically extract new metadata fields from documents, as the metadata type and parser must be specified by the OmniQ Enterprise administrator in advance.

## TEXT metadata

TEXT metadata corresponds to any metadata field that contains words or characters. If any date or numeric information is treated as text then each digit or number is treated simply as another character.

You can set the TEXT metadata parser to one of the internal text parsers that are set by the Text Manager module. These are set as either TEXT_STANDARD or TEXT_FILENAME.

TEXT metadata fields do not support range searching.

## DATE metadata

DATE metadata corresponds to any metadata field that can be parsed into a date. The exact parsing of the date depends on the date parser’s settings. For example, the parser may have the format DD/MM/YYYY or YY-MM-DD.

DATE metadata fields support range searching.

## FLOAT metadata

FLOAT metadata corresponds to any metadata field that can be parsed into a numeric value. The exact parsing of the numeric value depends on the parser’s settings.

FLOAT metadata fields support range searching.

## INT metadata

INT metadata corresponds to any metadata field that can be parsed into an integer value. The exact parsing of the numeric value depends on the parser's settings. For example, the com.isdduk.text.IntegerParser parser simply attempts to convert the metadata to an integer, while the com.isdduk.text.B2KBIntParser parser attempts to convert the metadata to an integer and then divide the result by 1024 to get the value expressed in terms of kilobytes (KB) instead of bytes (B).

INT metadata fields support range searching.

## Configuring metadata

The list of metadata fields are loaded through the configuration file as specified in the metadata.filename parameter (*Metadata.xml*) in the Metadata Manager configuration file. Each metadata field in *Metadata.xml* are specified by the parameters shown in Table 2-22.

***Table 2-22: Metadata field parameters***

| parameter | Default | Description |
|---|---|---|
| name | None | The internal name of the metadata field; one word with no spaces. This name is used in XML queries over HTTP. |
| displayName | None | The human-readable name for the metadata field. |
| type | None | The metadata field type—can be one of TEXT, DATE, FLOAT and INT. |
| parser | None | The parser used to format the metadata field into the format OmniQ Enterprise will use. The parser must be defined in the *Parsers.xml* file. |
| indexable | false | If set to true, OmniQ Enterprise indexes any document data found for this metadata field. |

### Adding new metadata fields

When source documents contain metadata fields that are not listed in the default set, you can add them:

1 Using a text editor, open the *Metadata.xml* located in *%OMNIQ_3.0\OmniQ\config*.

2 Anywhere within the XML Metadata tag, add a new field tag specifying the following attributes:

- Name – the name of the metadata field inside the document.

- DisplayName – specifies the way the metadata field displays on the search page.
- Type – specifies whether the metadata type is TEXT, DATE, FLOAT, or INT.
- Parser – the name of the parser used to parse the metadata field for indexing (for TEXT, use one of the internal parsers—TEXT_STANDARD or TEXT_FILENAME. Otherwise, use a parser listed in the *Parsers.xml* file).
- Indexable – set this property to true to index this metadata field.

3 Save and close the file.

For example, if the new metadata field is Customer ID, the new field tag would look similar to this:

```
<Field name="custId" displayName="Customer ID" type="INT"
parser="integer_2" indexable= "true" />
```

**Note** You must add the new metadata field before indexing any documents.

If the metadata field requires parsing from a nonstandard string, for example, to change the customer ID portion of the string "CUST-98334" to an INT, refer to "Developing and configuring custom parsers" on page 62.

If you need to search the new metadata field in a different format from that in which it was indexed, you must configure a new query parser. For example, if INT metadata field values have been parsed from strings such as "CUST-98334," but you do not want to type the "CUST-" prefix for searching, you can add a new query parser configuration as follows:

1 Using a text editor, open *QueryParsers.xml*, which is located in *%OMNIQ_3.0%\OmniQ\config*.

2 Anywhere within the XML QueryParsers tag, add a new MetadataField tag, specifying the following attributes:

- Name – the internal name of the metadata.
- Parser – the name of any parser listed in the *Parsers.xml* configuration file.

3 Save and close the file.

4 Restart OmniQ Enterprise.

Using the above example, the new MetadataField tag might look similar to this:

```
<MetadataField name="custId" parser="integer_2" />
```

**Note** You can change the query parser configuration at any time.

# MIME types

The list of MIME types that OmniQ Enterprise can index is in the *MimeTypeMap.default.xml* configuration file. In the configuration file, each MIME type is assigned a type and is marked as to whether or not it is supported.

A MIME type might have several extensions, each of which may or may not be indexable. The list of MIME types allows OmniQ Enterprise to index only those document types that may contain valid text data. Common formats such as plain text, Microsoft Office documents, Adobe PDF documents, and HTML files are indexable by default, whereas executable MIME types are not.

You can add custom MIME types and the appropriate text filter in the *FilterFactory.default.xml* file.

# Configuring modules using system parameters

Many shared module settings are configured using SystemProperty tags in the container XML configuration file. These properties are set as JVM system properties and are accessible to all classes loaded in the container. Properties set in this manner are “container-global.”

You can enter numeric parameters using several formats:

- Plain integers – for example, 20.
- K – for example, 20K = 20 x 1000.
- M – for example, 20M = 20 x 1000K.
- KB – for example, 20KB = 20 x 1024 bytes.
- MB – for example, 20MB = 20 x 1024KB.
- GB – for example, 20GB = 20 x 1024MB.

These formats allow high values to be entered as parameters while keeping the parameters easy to read. They also avoid the “missing zero” problem that can sometimes occur when entering parameters that have a high number of trailing zeros.

## Indexing processes

OmniQ Enterprise stores its data in a number of proprietary data structures, generically called indexes. See “Index information” on page 45 and “Indexing” on page 57 for more information.

Indexing involves three different processes—the first two are fundamental indexer processes and might occur numerous times during one indexing session. The third process occurs as a maintenance operation. These processes are:

1 Filtering, or parsing, documents and extracting data in memory

2 Writing processed data to index stripes on disk

3 Unifying index stripes on disk

### Extracting data into memory

The first indexing process has a threshold for restricting the amount of memory the extracted data buffer can consume before the data is written to disk (process 2). The greater the memory allocation, the more efficient is the entire indexing process, as more data can be handled at once.

Parameters that affect the extraction process are shown in Table 2-23.

***Table 2-23: General upload parameters***

| Parameter | Default | Description |
|---|---|---|
| omniq.index.buffer.maxMemory | 10MB | The indexing process is more efficient if many documents are indexed in a batch. The buffer's maximum memory allocation determines how many documents are processed in each batch. |
| omniq.indexer.maxDocumentSize | 10MB | Sets the maximum document size to be indexed by OmniQ Enterprise.<br>**Note** Very long documents have an adverse effect on the query results. |

## Writing data to disk

The second process is when the buffered data is written to the indexes. There are two main sets of parameters that affect this stage—the rate at which the data is written to the indexes (to reduce CPU and disk contention), and the index settings themselves.

Parameters that affect the write process are shown in Table 2-24.

***Table 2-24: Index parameters***

| Parameter | Default | Description |
|---|---|---|
| omniq.indexer.sleepDurationMillis | 20 | The time, in milliseconds, the indexer thread sleeps during indexing to allow other CPU-intensive applications to run. |
| omniq.indexer.sleepFrequency | 20 | Indicates the number of omniq.indexer.sleepFrequency cycles the indexer thread will sleep. |
| omniq.index.term.numSegments | 5 | The number of segments helps to distribute the indexed data across a number of files, reducing the “seek” times of large files. |
| omniq.index.term.minimizationFactor | 20 | The branching factor of each index segment. This parameter affects the lookup performance of the index segment. |
| omniq.index.term.useRootChildrenCache | true | If set to true, the index segments cache some of their structure in memoryto improve indexing and querying performance. |
| omniq.index.metadata.numSegments | 2 | The number of segments helps to distribute the indexed data across a number of files, reducing the seek times of large files. |
| omniq.index.metadata.minimizationFactor | 10 | The branching factor of each metadata index segment. This parameter affects the lookup performance of the metadata index segment. |
| omniq.index.metadata.useRootChildrenCache | true | If set to true, the metadata index segments cache some of their structure in memory to improve indexing and querying performance. |
| omniq.lexicon.document.maxKeyLength | 256 | The maximum document file path length deemed valid for indexing. |
| omniq.lexicon.document.minimizationFactor | 20 | The branching factor of each document lexicon segment. This parameter affects the lookup performance of the document lexicon segment and should not be changed without consulting with technical support. |

| Parameter | Default | Description |
|---|---|---|
| omniq.lexicon.document.useRootChildrenCache | true | If set to true, the document lexicon segments will cache some of their structure in RAM to improve indexing and querying performance. |
| omniq.lexicon.reverseDocument.numSegments | 4 | The number of segments helps to distribute the indexed data across a number of files, reducing the seek times of large files. |

## Unifying index stripes

The unifying process is for maintenance and optimization. This can take place only after a document store has been reindexed, which in turn produces new Index Stripes. See “Index unification” on page 46 for more information.

Parameters which affect the unifying process are shown in Table 2-25.

***Table 2-25: unifying parameters***

| Parameter | Default | Description |
|---|---|---|
| omniq.unifier.sleepDurationMillis | 20 | The time, in milliseconds, that the unifier thread sleeps during unifying to allow other CPU-intensive applications to run. |
| omniq.unifier.sleepFrequency | 100 | Indicates the number of omniq.unifier.sleepFrequency cycles the unifier thread will sleep. |
| omniq.unifier.termMapSizeSoftLimit | 40K | The limit of the number of terms processed in each unifying batch. |
| omniq.unifier.termMapSizeInBytesSoftLimit | 32MB | The memory limit used for processing the terms in each unifying batch. |
| omniq.unifier.metadataMapSizeSoftLimit | 40K | The limit to the number of metadata processed in each unifying batch. |
| omniq.unifier.metadataMapSizeInBytesSoftLimit | 32MB | The memory limit used for processing the metadata in each unifying batch. |

**Warning!** The index and lexicon parameters are critical to how the system performs—do not modify them without consulting with a Sybase OmniQ Enterprise support engineer.

## Query parameters

All queries run against OmniQ Enterprise are affected by the query parameters. The document scores can be scaled up or down using the confidence parameter, and the linking parameters affect all “find similar” queries.

Table 2-26 shows the query parameters.

***Table 2-26: Query parameters***

| Parameter | Default | Description |
|---|---|---|
| omniq.query.termLimit | 30 | The maximum number of terms in a query. If the user’s query exceeds this number, OmniQ Enterprise selects the most important omniq.query.termLimit number of terms from the query to use as the internal query. |
| omniq.query.confidence | 125 | OmniQ Enterprise generates its own scaling factor when converting internal document relevance scores to a more user-friendly percentage score. This scaling can be influenced by the omniq.query.confidence and has the effect that a higher confidence lowers the overall scores, while a lower confidence raises the overall scores. |
| omniq.query.linking.default.minDocRel | 5 | The minimum document relevance for a linking query can be specified on a per-query basis, but this value is used when the minimum document relevance is not specified. |
| omniq.query.linking.minTerms | 5 | The minimum number of terms that are generated automatically by OmniQ Enterprise to be used as a linking query. |
| omniq.query.linking.maxTerms | 10 | The maximum number of terms that are generated automatically by OmniQ Enterprise to be used as a linking query. |
| omniq.query.linking.confidence | 50 | The omniq.query.linking.confidence parameter works in the same way as omniq.query.confidence does, except for linking queries instead of normal queries.<br>Linking queries tend to generate lower document scores, as the generated linking query can cover many different topics. To compensate, the confidence parameter is low to raise the overall linking query scores. |

## Metadata paragraph files

The metadata paragraph files (MPFs) are where OmniQ Enterprise stores the metadata and text of the files it has indexed. This data is used in constructing result sets and for generating plain-text versions of the indexed documents.

The MPF is a custom data structure—each contains the metadata and body text of a number of indexed documents (“Configuring MPFs” on page 31) in a compressed format. The first group of MPFs is created in the 0 (zero) directory, and subsequent groups are numbered sequentially beginning with 1.

## Configuring MPFs

The MPF classes utilize a strategy to best compress all the paragraphs from documents, favoring documents of average length (where the average length is implied from the MPF configuration). Each paragraph is written to disk in one of two ways:

1 The paragraph is entered into a paragraph group that is compressed as a whole and written to disk, or

2 The paragraph is compressed and written to disk individually.

The first technique is employed initially, as the compression scheme works better with more data—thus the paragraphs take up less space on disk. The second technique is employed when the paragraph group allocation is exhausted.

The paragraphs are not all written together, as it is often necessary to read individual paragraphs from disk (and compressing all the paragraphs together forces the application to read and decompress all paragraphs to access the sole paragraph required). The grouping provides a balance between data compression and disk I/O.

The number of paragraphs in any one paragraph group is not fixed; groups accept new paragraphs until the data buffer’s soft limit is reached. “Soft” indicates that a limit can be exceeded, but the group is then closed. The ideal scenario is when all the paragraphs from a document fit exactly within the allocated number of paragraph groups. Unused paragraph groups result in redundancy.

You can configure the paragraph grouping using the MPF parameters shown in Table 2-27. The MPF parameters are defined for all document stores in a container and are set in the main container file Container.<uid>.xml file.

***Table 2-27: MPF parameters***

| Parameter | Default | Description |
|---|---|---|
| omniq.index.mpf.docsPerFile | 20 | The number of documents stored in each MPF. |
| omniq.index.mpf.filesPerFolder | 250 | The number of MPFs stored in each folder. |
| omniq.index.mpf.foldersPerFolder | 50 | The number of MPF folders stored per folder. |

| Parameter | Default | Description |
|---|---|---|
| omniq.index.mpf.maxParagraphGroups | 5 | The maximum number of paragraph groups to allocate per document. |
| omniq.index.mpf.maxTotalGroupEntries | 50 | The maximum number of paragraphs from any one document that can be in a paragraph group. |
| omniq.index.mpf.bufferSoftLimit | 8192 | The ideal number of bytes an uncompressed paragraph group can consume before it is closed, compressed, and written to disk. This limit is usually slightly exceeded by design. |

CHAPTER 3

# Configuring the Hyena Servlet Container

This chapter describes the key configuration parameters for the Hyena servlet container provided with OmniQ Enterprise. The Hyena servlet container is a standalone lightweight HTTP server for use only with OmniQ Enterprise. You can use the Hyena servlet container, or you can integrate OmniQ Enterprise with any J2EE application server, such as Apache Tomcat.



## Configuring Hyena

The configuration file for the Hyena servlet container is *server.xml*, which is located in *%OMNIQ_3.0%\Hyena\config*. Use a text editor, such as Wordpad to edit the file.

Table 3-1shows the attributes for the HTTP server tag.

***Table 3-1: HTTP Server tag***

| Attribute | Default value | Description |
|---|---|---|
| port | None | The TCP/IP port on which Hyena listens for connections. |
| host | *localhost* | The name or IP address of the host on which the Hyena servlet container resides. |
| stdOutput | false | All standard output (for example, printed to java.lang.System.out and java.lang.System err) is always redirected to the Hyena log file.<br>When set to true, the output is sent to the original standard output (usually the console) as well. |

Table 3-2 shows the attributes for the Request-Handler tag.

***Table 3-2: Request-Handler tag***

| Attribute | Default value | Description |
|---|---|---|
| minThreads | 10 | The minimum number of server threads that Hyena uses to serve connections. |
| maxThreads | 75 | The maximum number of server threads that Hyena uses to serve connections. |
| maxIdleTime | 10000 | The number of milliseconds an idle server thread is kept alive before being destroyed. This parameter applies only when the current number of server threads exceeds the minimum. |
| debug | false | If set to true, request handling debug information is written to standard output for every HTTP connection received. |

Table 3-3 shows the attributes for the Request-Parser tag.

***Table 3-3: Request-Parser tag***

| Attribute | Default value | Description |
|---|---|---|
| maxHeaderLength | None | The maximum number of characters accepted in any one HTTP request header (including GET parameters). Requests using headers longer than this are denied.<br>Requests that send large parameter values should use the POST method. |
| maxNumberOfHeaders | None | The maximum number of request headers accepted as part of any single request. Requests formed using more headers than this are denied. |

Table 3-4 shows the attributes for the Request-Keep-Alive tag.

***Table 3-4: Request-Keep-Alive tag***

| Attribute | Default value | Description |
|---|---|---|
| enabled | false | When set to true, HTTP keep-alive is used with all HTTP clients that support it. |
| maxRequests | None | The maximum number of requests that are served by any one connection. |
| timeout | None | The number of milliseconds the server waits for further requests on an open connection before breaking it. |

Table 3-5 shows the attributes for the Remote-Admin tag.

***Table 3-5: Remote-admin tag***

| Attribute | Default value | Description |
|---|---|---|
| enabled | false | When set to true, authorized stop and start commands sent via HTTP are accepted. |

| Attribute | Default value | Description |
|---|---|---|
| authCode | None | The authorization code required by the remote administration listener. |

Table 3-6 shows the attributes for the Logging tag.

***Table 3-6: Logging tag***

| Attribute | Default value | Description |
|---|---|---|
| enabled | false | If set to true, HTTP requests are logged. |
| directory | None | Designates the directory in which log files are written. |
| prefix | None | The standard prefix for all log file names (appears before the date). |
| suffix | None | The standard suffix to use for all log file names (appears after the date). |
| timestamp | false | If set to true, time of the HTTP request is logged. |

Table 3-7 shows the attributes for the container tag.

***Table 3-7: Container tag***

| Attribute | Default value | Description |
|---|---|---|
| debug | false | When set to true, servlet/JSP debug information is written to standard output for each request received. |

Table 3-8 shows the attributes for the JSP-handler tag.

***Table 3-8: JSP-handler tag***

| Attribute | Default value | Description |
|---|---|---|
| vigilance | None | When set to true, servlet/JSP debug information is written to standard output for each request received. |

Table 3-9 shows the attributes for the error-template tag.

***Table 3-9: Error-template tag***

| Attribute | Default value | Description |
|---|---|---|
| path | *%Hyena%\config\error_template.htm* | Defines the path to an HTML template with which error messages are formatted to display to clients when an application error is encountered. |

Table 3-10 shows the attributes for the context tag.

***Table 3-10: Context tag***

| Attribute | Default value | Description |
|---|---|---|
| name | None | All context resource URIs implicitly start with this value; it must begin with a forward slash. For example, the context named */omniq* might have its home page at */omniq/index.html*. |
| path | None | |

## The MIME-mapping tag

The MIME-mapping tag defines no attributes but has two other tags nested within each opening and closing pair:

- Extension – its node value represents a file extension, for example, “htm” for HTML documents.
- MIME-type – its node value represents a MIME type, for example, “text/html” for HTML documents.

Use MIME configuration when setting the content-type HTTP response header for requested files.

CHAPTER 4

# Document Management



## Using the System interface

This section describes the functions of the System interface and how to use them. The System interface provides a view of the distributed OmniQ Enterprise installation.

### Administration

OmniQ Enterprise is administered through a J2EE Web application, therefore you can administer OmniQ Enterprise from any machine that can run a Web browser. The administration console consists of a Welcome window and three additional main windows:

- Search – shows the demonstration search page.
- System – allows you to view the distributed setup. This includes memory reports and the system configuration of all containers within the OmniQ Enterprise installation.
- Document Management – allows you to add, update, and remove documents from OmniQ Enterprise indexes and organize them into groups. See “Document groups” on page 48.

## System

This window shows each separate OmniQ Enterprise container, listing the host name and port on which each runs and which modules are loaded therein. The various data and configuration directories are also listed.

### System properties

Shows the system properties of the JVM in which the chosen container runs (those returned by java.lang.System.getProperties() on the remote JVM).

### Memory usage

This window shows the memory consumption of the OmniQ Enterprise containers, including the JVM allocation and consumption, as well as the memory-hungry resources loaded within the loaded modules (such as data caches).

## Events

There are three main types of event; information, warning, and error. All events are recorded through a Hub Manager, one of which is always present within a participating OmniQ Enterprise container.

The Events window displays pages of recorded events that have occurred within the distributed OmniQ Enterprise installation. Events can be selected by Hub Manager (container), filtered by type and be returned sorted in chronological or reverse order.

## Search

The Search window allows you to search across all the documents that have been indexed by OmniQ Enterprise. In the Search window, you can enter:

- Search Terms – enter your natural language query in the text field. The more information you provide, the more accurate your results are. See “Search strategies” on page 49.

  After entering your query, click Search.

- Not Terms – enter terms to indicate concepts dissimilar to those for which you are searching. Unlike the Boolean NOT operator, documents that contain these terms are still considered for retrieval. However, the number of Not Terms a document contains is considered by the scoring algorithm and its relevance score is downgraded accordingly (based on the weight of the Not Terms it contains).

  For example, a search for “operating systems” with Not Terms “Windows XP,” would not discount a document for containing the phrase, “opens in a new window.”

- Document Groups – limit your search to one or more predefined document groups. Only documents from the chosen document groups are included in the search results.
- Number of Results – number of document results to display per page.
- Number of Paragraphs – number of document paragraphs to display per result document.

**Note** Paragraph scoring is not enabled in version 3.0 of OmniQ Enterprise. Result paragraphs are always the head of the document.

- Score Unknown Terms – when true, search terms unknown to the system (for example, terms that do not exist in any indexed document) are considered by the scoring algorithm.
- Metadata – three general types of metadata are supported; text, numbers, and dates.

**Note** Some metadata fields are document-specific. For example, a Microsoft Word document will have a Word Count, whereas a plain text document will not, and an HTML document will most likely not. Metadata fields that are guaranteed to be searchable for all documents are reliable. When the field searched on is not supported or not present in a document, it is automatically excluded from the results.

Each metadata parameter consists of:

- Field – select from the drop-down list of these predefined fields:

***Table 4-1: Predefined metadata fields***

| Name | Type | Reliable |
|---|---|---|
| Author | TEXT | No |
| Character Count | INT | No |
| Client | TEXT | No |
| Comment | TEXT | No |
| Company | TEXT | No |
| Creation Date | TEXT | No |
| Document Name | TEXT | Yes |
| Document Origin | TEXT | Yes |
| Document Path | TEXT | Yes |
| Document Size (KB) | INT | Yes |
| Document Type | TEXT | No |
| Editor | TEXT | No |
| File Type | TEXT | Yes |
| Keyword | TEXT | No |
| Language | TEXT | No |
| Last Modified | DATE | Yes |
| Page Count | INT | No |
| Project | TEXT | No |
| Publisher | TEXT | No |
| Reference | TEXT | No |
| Second Author | TEXT | No |
| Status | TEXT | No |
| Subject | TEXT | No |
| Title | TEXT | No |
| Word Count | INT | No |

- Operator – select from the drop-down list. All metadata types support the “equal to” (=) operator, and the number and date types also support “greater than or equal to'”(>=), and the “less than or equal to”(<=) operators.
- Value – enter the values for the selected field each document must contain.The values for text types are processed as search text. In other words, search terms are stemmed and augmented through synonyms and acronyms (except file path fields).

The numeric type values are simply numbers and the date type values should be in the format configured by the OmniQ Enterprise System Administrator, for example `dd/mm/yyyy`. See “Query parsers” on page 19.

These parameters are primarily to be used in conjunction with the Search Term and Not Term parameters to refine the search, but you can also use them independently for a simple metadata search (results from a pure metadata search have no meaningful relevance scores).

- Metadata Combination Operators – there are two combination operators:
  - Within Fields – this operator is used when there is at least one metadata parameter with a value that consists of more than one term. When the operator is AND, every term must be present in the document metadata for the match to succeed. When the operator is OR, only one of the terms need exist.

    For example, when the parameter is: `Author = "John Smith"`, the Within Fields operator differentiates the two possible interpretations, which are: `Author = "John AND Smith"` or `Author = "John OR Smith"`.

    **Note** OmniQ Enterprise 3.0 supports only one Within Fields operator, so you cannot perform a metadata search for `Author = "John AND (Smith OR Roberts)"`.

  - Across Fields – this operator is used when there are at least two metadata parameters. When the operator is AND, both parameters must succeed for the match to succeed; if the operator is OR, only one of the parameters need succeed.

    For example, when the parameters are Author = “Smith,” Title = “Algebra,” the Across Fields operator differentiates the two possible interpretations, which are:

    - Author = “Smith” AND Title = “Algebra” or
    - Author = “Smith” OR Title = “Algebra”

    **Note** OmniQ Enterprise 3.0 supports only one Across Fields operator, so you cannot perform a metadata search for multiple Across Fields operators.

- Minimum Document Relevance – the minimum relevance ranking a document must score for it to be included within the search results. Documents with scores lower than this minimum threshold are not returned.
- Minimum Paragraph Relevance – this feature is not enabled in version 3.0 of OmniQ Enterprise.

# Document Store Managers

A Document Store Manager manages zero or more Document Stores.

The Document Store Manager pages are for managing the OmniQ Enterprise indexed resources:

The left navigation menu displays three subsections:

- Document Stores – lists all document stores and the functions for adding, updating, and removing them.
- Document Store Managers – lists the Document Store Managers in the installation. This is useful for planning where to place new document stores.
- Document Groups – lists all document groups and the functions for adding, updating, and removing them.

Table 4-2 shows the Document Store Manager properties and their values.

***Table 4-2: Document Store Manager properties***

| Property | Value |
|---|---|
| ID | The unique ID of the Document Store Manager module |
| Name | The name of the manager (to assist differentiation) |
| Document Stores | A list of all the document stores the Document Store Manager manages |

# Document stores

A document store represents one or more collection of documents related by their physical location. The file system document store accepts one or more directory roots (for example *D:\documents\office*), the contents of which OmniQ Enterprise indexes.

Although documents from different physical locations (*C:\docs* and *\\network-share\docs*) can coexist in the same document store, internally, all documents found in all root directories of a document store are indexed together. This means they share the same data structures as well as being updated and removed together.

## Creating document stores

There are no document stores until you create them. To create a new document store:

1 Click Create New.

2 In the Document Store window, enter:

- Name – the name of the document store.
- Manager – the document store Manager in which the document store should exist.
- Member of – the document groups of which the document store is a member. See “Document groups” on page 48.
- Not a Member of – the document groups of which the document store is not a member.

- Directories – specify one or more root directory whose contents will be indexed and available for searching.
- File Type Filter – use this to include or exclude documents by file extension or MIME type, for example:
  - Include text/html – indexes only HTML documents.
  - Include doc – indexes only Microsoft Word documents.
  - Exclude text/xml, txt – indexes all documents except XML and text documents.
- Index Now – indicates whether to proceed with indexing immediately or to save the configuration without indexing at this time.

3 Click Create. The document store is created and you return to the main window.

The new document store appears in the list of document stores, ordered by name, alphabetically. Beneath the document store name is the number of searchable documents the document store contains and a list of the indexed document roots. There is also a summary of the last indexing session and, if the store is being indexed, the current indexing session information. Beside the document store name are four buttons:

- Incremental Index – this button re-runs the indexing process over the saved document store configuration. All new documents are indexed; all updated documents are reindexed; and all deleted documents are removed from the indexes.
- Part Index – the Part Index function is primarily for use within OEM applications.

The Part Index is a shortcut for adding new documents to the document store and/or for notifying the document store of document updates (candidates for re-indexing). This function saves time by not checking the directory trees for new, modified or deleted documents, which can be time saving for large document stores. Only the document parameters are considered.

**Note** The new document must exist within one of the document store's root directories. Documents not located in a valid root directory are ignored.

- Edit – you can edit most attributes of a document store. For example, you can rename a document store; add or remove document roots; add or remove File Type Filters; and move the document store in and out of document groups.
- Remove – use this button to remove document stores. When you remove a document store, all settings and indexes are permanently removed from the disk. All documents indexed under the removed document store are no longer returned in searches.

## Index information

Each document store has a link to its Index Information window. The Index Information window displays high-level technical information about the internal index structures of the document stores.

The index information shows some of the internal workings of OmniQ Enterprise as shown in Table 4-3.

***Table 4-3: Index information properties***

| Property | Value |
|---|---|
| Documents Indexed | The total number of live and deleted documents in the indexes of all index stripes |
| Deleted Documents | The total number of documents in the indexes of all index stripes that reference deleted documents |
| Live Documents | The total number of documents in the indexes of all index stripes that reference live documents |
| Number of Stripes | The number of index stripes the indexed data is split across |

## Index unification

Each indexing session's data is stored in a separate stripe—having too many stripes eventually causes a bottleneck, so the stripes should be unified into a single stripe. As an additional benefit, the unification process also purges data marked for deletion and defragments the indexed data structures.

When index unification is taking place, a progress bar displays on the screen.

## Index stripes

Each index stripe is listed showing details of its internal data structures. These include the generic term and metadata indexes plus the data structures that are instrumental in tracking file system documents.

***Table 4-4: Index stripe properties***

| Property | Value |
|---|---|
| Root | Where the index stripe stores its data. The root property creates directories and data files in here as necessary |
| Term Index Segments | The number of segments into which the term indexes are divided |
| Metadata Index Segments | The number of segments into which the metadata indexes are divided |
| Deleted Documents | The number of deleted documents for which this stripe still holds data (data which is purged on unification) |
| Live Documents | The number of live documents for which this stripe holds data |
| **Document lexicon** | |
| **Property** | **Value** |

| Property | Value |
|---|---|
| Segments | The number of segments into which the document lexicon is divided |
| Documents | The number of documents in the lexicon. |
| ID Range | The ID range of the document IDs (first to last) |
| Last Indexed | The name of the last document indexed and the time it was added |

### Session information

Both the previous and any current indexing session details display on the Session Information window.

Table 4-5 summarizes indexer activity.

***Table 4-5: Indexer activity***

| Property | Value |
|---|---|
| Total | The total number of documents found |
| Indexable | The number of documents eligible for indexing |
| Selected | The number of documents selected for indexing |
| Skipped | The number of documents purposefully ignored |
| Deleted | The number of documents that have been indexed but no longer exist |
| New | The number of new unindexed documents found |
| Updated | The number of updated (changed since indexing) documents found |
| Unchanged | The number of indexed documents that have not changed |
| Failed | The number of documents that should have been indexed but were not, due to a problem |

# Document groups

This section describes creating, editing, and removing document groups.A document group is a virtual collection of documents, based on the document store in which they reside. For example, an OmniQ Enterprise installation might include three Document Store Managers on three separate machines, each having a “CV” document store. You can create a “CV” document group to include all three CV document stores; you can then use the document group as a search parameter to indicate that search results should only come from the “CV” document group.

Table 4-6 shows the document group properties.

***Table 4-6: Document group properties***

| Property | Value |
|---|---|
| Name | The name of the document group |
| ID | The unique document group ID assigned to the group |
| Document store members | The document stores that are members of this group |

## Creating document groups

Choose a name for the group, then add and remove the document store members as desired.

## Editing document groups

Rename the group, or add and remove document store members as desired.

## Removing document groups

Click Remove to remove the document group from the system.

# Search strategies

A concept search engine performs at its best when you enter queries with search words in context (for example, in short phrases rather than as isolated words). In addition, if you know that more than one language is in use, repeating the concepts using different vocabulary generally improves results. Searching is often an iterative activity with queries being expanded and refined based on the results returned.

This section provides tips for optimizing a concept-based search engine, which provides greater flexibility than traditional approaches to free text searching, such as the Boolean combination of keywords.

For example, a user receives an e-mail message that says:

Following the incident close to Watford railway station in July, we need to assess the damage being done by tree branches tangling in overhead power lines or falling onto the tracks.

The user then wants to locate documents matching the e-mail message. Using a traditional search method, he or she might enter something similar to:

```
branches AND lines AND tracks
```

In this query, the user is using the Boolean operator AND to filter the information. This type of query is very precise and is helpful when:

- The user knows exactly what information is required, and it can be expressed in just a few words.
- There is no ambiguity in the words used in the query.
- The vocabulary of the target documents is known precisely.

In practice, this is rarely the case. It is more common that users are unsure of how to formulate their query precisely, thus introducing ambiguity within the query. Differing vocabulary used in documents to describe similar concepts can also result in important documents being missed altogether, and too many irrelevant documents being returned.

If the user is searching a large database of documents, a query like the one in the above example is likely to retrieve a large number of items, many of which are not relevant to the specific query due to the query searching for a small number of specific, isolated words. Words like “branches” and “lines” are ambiguous and are common in a database of documentation concerning the railway system.

A probabilistic search engine like the OmniQ Enterprise Inference Engine is better suited to a query that contains a number of concepts and is expressed using ambiguous language, thus increasing the likelihood the user retrieves results that are relevant to the query.

Using the e-mail example from above, isolate the key concepts, which are:

- Damage being done by tree branches
- Tangling of overhead power lines
- Falling trees and tree branches
- Obstruction or damage to tracks

Irrelevant concepts might include:

- Watford Railway Station

- July

Inclusion of irrelevant concepts unfocuses the search and may introduce some unwanted documents. So, a more effective query is:

```
damage being done by tree branches, tangling of overhead
power lines, falling tree branches, obstruction and
damage to tracks
```

**Note** You need not delimit concepts using a comma.

This is a better query because it contains all of the key concepts in the original query and expresses them using words in context. Results returned by this query are likely to produce significantly better results than the first attempt. The best documents are likely to achieve very high relevance scores with the scores falling off rapidly for documents that do not include all of these concepts.

However, it is likely that some relevant documents will still be missed, due to differing vocabulary. Therefore, we could use our knowledge of the environment and expand the original concepts to include variations that we know from experience tend to occur and this may produce a query similar to:

```
damage being done by tree branches, tangling of overhead
power lines, falling tree branches, obstruction and
damage to tracks, forestry, wind damage, storm damage,
damage to rails, lines being pulled down by trees blown
over
```

At first this may seem more confusing and less precise than the previous examples, but in fact it contains additional ways of defining the original concepts. You may find that no documents achieve a 100% relevance score with this query because no document includes all of these combinations. However, the most relevant documents are at the top of the list.

Often, you can improve search results by feeding back information from documents discovered by the system. For example, if a search produces a document that is relevant but the terminology used in the extracted summary is different from the search text, you may want to expand the original query by appending words or phrases from the document “hit list.” In this way, the search becomes more accurate as you provide additional information.

CHAPTER 5

# Administration



## HTTP handlers

An HTTP handler is a Java object designed to service HTTP requests, similar to a simplified Java servlet. OmniQ Enterprise allows allocation of any number of HTTP handlers to modules at configuration time. This means you can develop and plug in custom HTTP handlers as necessary. OmniQ Enterprise ships with five HTTP handlers, four XML handlers, and a generic file serving handler (for the Document Type Definitions).

See “Configuring the container XML file” on page 5.

### XML Document Groups HTTP handler

This handler returns a list of Document Groups in an XML format compliant with its Document Type Definition, which can be found at *<container-home>/config/dtd/DocumentGroups.dtd*.

It lists each Document Group’s ID for use as a search parameter, as well as its name and the names and addresses of each of the document stores’ members for display and integration purposes.

In a default installation of OmniQ Enterprise, you can find the XML handler and its Document Type Definition (DTD) handler respectively at:

- *http://<container-host>:<container-port>/xml/documentgroups*, and
- *http://<container-host>:<container-port>/dtd/documentgroups*

## XML metadata HTTP handler

This handler returns a list of all indexable metadata Fields in an XML format compliant with its DTD, which can be found at *<container-home>/config/dtd/Meta-data.dtd*.

This handler lists each Metadata Field internal name (for use as a search parameter) as well as its display name and type for display and integration purposes.

In a default installation of OmniQ Enterprise, you can find this XML handler and its DTD handler respectively at:

- *http://<container-host>:<container-port>/xml/metadata*, and
- *http://<container-host>:<container-port>/dtd/metadata*

## XML query HTTP handler

This handler takes query parameters over HTTP (GET or POST) and returns a result set in XML compliant with the result set DTD, which can be found at *<container-home>/config/dtd/ResultSet.dtd*.

In a default installation of OmniQ Enterprise, you can find this XML handler and its DTD handler respectively, at:

- *http://<container-host>:<container-port>/xml/query*, and
- *http://<container-host>:<container-port>/dtd/resultset*

The XML query HTTP handler parameters are shown in Table 5-1.

***Table 5-1: XML query HTTP handler parameters***

| | |
|---|---|
| **Normal query parameters** | |
| terms | A natural language query string describing the concepts that all documents should contain. |
| notTerms | A natural language query string describing the concepts documents should not contain. |
| **Linking query parameters** | |
| linkingDocAddr | The address of the document to use to create a `find-similar` query. |
| **Linking query with external document parameters** | |
| targetDSA | The target document store address, for example, the exact document store from which OmniQ obtains its initial term statistics. |
| targetDSMID | The target Document Store Manager ID. The exact document store from which OmniQ obtains its initial term statistics resides here and is chosen by the system. |
| linkingDocPath | The full path to the external document (from the Document Store Manager's perspective) used to create a `find-similar` query. |
| **Common parameters** | |
| DocumentGroupIds | A comma-delimited list of document group IDs. When present, only documents that are members of these groups are returned. |
| metadata | A metadata search expression, that takes the form:<br>< *name*><*operator*><*value*><br>where <*name*> is the internal name of the metadata field; <*operator*> is "=," ">=," or "<=" (the latter two are only supported by numeric and date types) and <*values*> is the criteria of the metadata search. |
| metadataOperatorWithinFields | Valid values are AND and OR. When this parameter is AND and two or more values are presented for any one metadata field, all must match for the query to succeed.<br>When this parameter is OR, only one of any number of values needs to match for the query to succeed. |
| metadataOperatorAcrossFields | Valid values are AND and OR. When this parameter is AND and two or more metadata parameters are present, all must succeed for the query to succeed.<br>When this parameter is OR, only one of any number of metadata parameters needs to succeed for the query to succeed. |
| minDocRel | The minimum relevance, expressed as a percentage, a document must achieve to be included within a result set. |
| numParas | The number of document excerpts to return for each document returned in the page of results. |
| scoreUnknownTerms | When set to true, terms present in a query yet unknown to OmniQ (for example, terms not present in any indexed document) are represented in the scoring algorithm.<br>When set to false, unknown terms are ignored |

| Normal query parameters | |
|---|---|
| resultsOffset | An integer value that represents the place in the result set the page of results should begin. The first document in the result-set (for example, the top scoring document) is at offset zero (0). If the offset is greater than the number of results found, OmniQ Enterprise returns an empty page of results. |
| resultsLength | An integer value that represents the number of documents to include in the page of results. |
| maxResultsNeeded | An integer value that represents the maximum number of results required by the caller (the minimum value is implicitly resultsOffset + resultsLength). This yields performance benefits when queries are cached for returning on a page-by-page basis. |

## XML document HTTP handler

This handler returns the text from an indexed document in an XML format compliant with its DTD, which can be found at *<container-home>/config/dtd/Document.dtd*.

In a default installation of OmniQ Enterprise, you can find this XML handler and its DTD handler respectively, at:

- *http://<container-host>:<container-port>/xml/document*, and
- *http://<container-host>:<container-port>/dtd/document*

The handler accepts the following parameters shown in Table 5-2.

***Table 5-2: XML document HTTP handler parameters***

| Parameter | Description |
|---|---|
| address | The document address of the document to fetch as XML. The document address format is <DSM-ID>-<DS-ID>-<DOC-ID>(Document Store Manager ID, document store ID, and Document ID). |
| useParagraphs | When this parameter is true, the body text of the document is broken into paragraphs and formatted between extra paragraph XML tags.<br>If set to false, the entire body text is returned in a large, unbroken block. |

# Indexing

This section describes how the indexing of documents works using index sessions and index stripes.

## Indexing session

Iindexing session describes all data collected during the pass of a document store's indexer. Data for all documents is collected during the first indexing session; subsequent indexing sessions collect data for new documents, modified documents, and deleted documents. Thus, the amount of data collected during two different indexing sessions can vary dramatically (from everything to nothing).

All data collected by the indexing session is stored in the indexing session's data buffer. The data buffer is a RAM-oriented data structure, where data is aggregated, ready to be written to an index stripe. This buffer is flushed when the maximum memory threshold has been exceeded (specified in the system property omniq.index.buffer.maxMemory). The buffer shares this memory allocation with the document store's active index stripe. See "Active index stripes" on page 58.

The data is transferred from the data buffer to one of two types of index stripes—active or static. See "Index stripes" below.

## Index stripes

Whether data is written to an active or a static index stripe is decided during the indexing session. The current active stripe stores all the data collected during the indexing session if it can accommodate it, otherwise, the active index stripe is emptied into a new static stripe, and all data collected during the indexing session is stored in the new static index stripe. When the active index stripe is emptied, it is discarded. A new active stripe is created the next time an indexing session collects a sufficiently small amount of data to fit into an active index stripe.

The index stripe can also contain a number of **.gz* compressed serialized files.

### Active index stripes

Each document store's collection of index stripes contain exactly zero or one active index stripe. An active index stripe is a collection of RAM-oriented data structures–all of its data is stored in RAM while it keeps a copy on disk for persistence. An active index stripe is always writable, thus may contain data collected over numerous indexing sessions.

The active index stripe persists its data in three compressed files–the *<ais-version-ID>* is incremented each time the active index stripe is updated:

- *ais-dat-<ais-version-ID>.ser.gz*
- *ais-del-<ais-version-ID>.ser.gz*
- *ais-len-<ais-version-ID>.ser.gz*

When an active index stripe is emptied into a static index stripe, these files are deleted.

### Static index stripes

Each document store's collection of index stripes contain zero or more static index stripes. A static index stripe is a collection of disk-oriented data structures, which you cannot change once written.

Each static index stripe directory contains the form IS-#, where # is a unique stripe ID. Each index stripe contains the following directories and files:

- *doc* – directory that contains a proprietary data structure used to store document information used in query calculations.
- *metadata* – directory that contains a proprietary data structured used to store metadata information used in query calculations.

- *term* – directory that contains a proprietary data structure used to store body text information used in query calculations.
- *common-docroot.txt* – absolute path to the directory root common to all indexed directories. For example, *C:\office\sales* and *C:\office\marketing* have a common document root of *C:\office*.
- *del.lex* – contains statistical data on deleted documents not yet purged.
- *deleted.ser.gz* – contains deleted document identifiers.
- *di-inf.ser.gz* – the document index meta information.
- *lengths.ser.gz* – the document length (or term count) for each indexed document.

CHAPTER 6

# Developing and configuring custom files

This chapter contains information for developing, configuring, and using custom filters, parsers, and text splitters.



## Developing and configuring custom filters

OmniQ Enterprise uses a third-party solution, Stellent, for parsing many document formats. The Stellent document filter is a multi-filter—in other words, the same filter instance handles all supported MIME types. Thus, the Stellent filter is configured to handle the MIME type */*, indicating that it can filter text from documents of any MIME type presented to it.

When OmniQ Enterprise obtains a filter for a document, it first identifies its MIME type from the file extension. For example, *C:\document.pdf* has the MIME type “application” and the subtype “pdf” (application/pdf). OmniQ Enterprise then requests a filter from the Filter Factory to handle documents with the identified MIME type.

The filter lookup is performed in this order:

1 If a filter is configured to handle a specific MIME type, that filter instance is returned.

2 If a multi-filter (*/*) is configured, that filter instance is returned.

3 No filter is returned, denoting “not indexable.”

You can add additional filters by editing the XML configuration file located in *%OMNIQ_3.0%\OmniQ\config\FilterFactory.default.xml*. See “Modules” on page 9 for information about the *FilterFactory.default.xml* file.

# Developing and configuring custom parsers

This section provides information for Java developers about developing, configuring, and using custom parsers.

## Parser classes

You can create custom date, int, and float parsers and plug them into the system.

All parsers implement this common base interface:

```
com.isdduk.text.Parser
     getId() : short
     setId(short) : void
     getName() : java.lang.String
     setName(java.lang.String) : void
     init(com.isdduk.util.map.FastMap): void
```

This interface defines methods that facilitate tracking and displaying information about parser instances loaded in OmniQ Enterprise, mainly simple GET and SET methods. There is also an initialization method which takes a map of parameters should the parser require any—this method is guaranteed to be called before parsing commences. You can extend the convenience base class com.isdduk.text.AbstractParser.

Irrespective of whether the convenience base class is utilized, each custom parser class must provide a no arguments constructor and implement the appropriate one of these three specialized parser interfaces:

```
com.isdduk.text.DateParser
     parse(java.lang.String, com.isdduk.util.set.LongSet) : boolean
     parse(java.lang.String) : com.isdduk.util.set.LongSet
     format(long) : java.lang.String

com.isdduk.text.IntParser
     parse(java.lang.String, com.isdduk.util.set.IntSet) : boolean
     parse(java.lang.String) : com.isdduk.util.set.IntSet
     format(int) : java.lang.String

com.isdduk.text.FloatParser
     parse(java.lang.String, com.isdduk.util.set.FloatSet) : boolean
     parse(java.lang.String) : com.isdduk.util.set.FloatSet
     format(float) : java.lang.String
```

The parse method, which returns a Boolean result, should contain the parsing logic; the other parse method should simply create a suitable set object and delegate the call, as it is a convenience method for when there is no suitable set object in its scope. The format method should reverse the parse process and return the date, int, or float value as a string (although this is not always possible).

**Note** The date parser turns date strings into long values—the number of milliseconds that have passed since the 1st of January 1970 in Coordinated Universal Time (UTC).

## Adding the new parser

Once you have compiled the new parser class, you must make it available to the system. The easiest way to do this is by adding the class to a Java Archive (JAR) file and placing the JAR file in the *%OMNIQ_3.0%\OmniQ\lib* directories.

**Note** You must place the JAR file into every container's library folder.

You must add the new parser to the internal set of parsers. Edit the *%OMNIQ_3.0%\OmniQ\config\Parsers.xml* configuration file and add a new Parser tag.

For example, a parser that parses user IDs from strings might have a configuration similar to this:

```
<Parser identifier="intUserId_8" class="com.mycompany.IntParserImpl">
     <Param name="base" value="16" />
</Parser>
```

You can load multiple instances of the same class. This example assumes the parser class can parse different integer bases (octal, decimal, hexadecimal, and so on), but the configured instance expects the hexadecimal format.

**Note** As OmniQ Enterprise is a distributed system, it is important the new parsers are configured for the container instance that loads the Text Manager.

## Using the new parser for metadata indexing

You must configure each metadata field that requires the new parser. To do this:

1 Using a text editor, open the *Metadata.xml* configuration file located in *%OMNIQ_3.0%\OmniQ\config*.

2 Locate the appropriate fields and change the parser attribute to the value of the new parser identifier. For example:

```
<Field name="userid" displayName="User ID" type="INT"
parser="intUserId_8" indexable="true" />
```

If it is a new metadata field and does not exist in this file, add it. See Adding New Metadata types.

---

**Note** Since OmniQ Enterprise is a distributed system, it is important to perform metadata changes for the container instance which loads the Metadata Manager.

---

## Using the new parser for querying

If the metadata field requires metadata query values to be handled by the new parser, it is necessary to override the default behavior by editing the *QueryParsers.xml* file located in *%OMNIQ_3.0%\OmniQ\config*. If we assume the querying format is the same as it is for indexing, then the query parser would have the following entry:

```
<MetadataField name="userid" parser="intUserId_8" />
```

---

**Note** OmniQ Enterprise is a distributed system, therefore it is important that the metadata changes are performed for the container instance which loads the Query Manager.

---

# Developing and configuring custom text splitters

This section provides information for Java developers about developing, configuring, and using custom text splitters.

All document body text and textual metadata (excluding file paths) values are passed through the configured term splitter to be broken into individual terms. Each term that is not preserved (see “Preserved terms” on page 21), not a stopword (see “Stopwords” on page 20), and is neither too short nor too long, is passed to the configured term stemmer to be reduced to its root form. Both the term splitter and term stemmer can be reimplemented and reconfigured where necessary.

## Configuring the term splitter

The term splitter interface defines numerous methods, many of which must be the same, regardless of the splitting algorithm employed. To simplify implementing new term splitters, OmniQ Enterprise includes an abstract base class that you can extended to inherit much of the required functionality:

com.isdduk.text.AbstractTermSplitter

The convenience base class does not implement any splitting algorithms. The various split methods defined by the term splitter interface are as follows (see the Javadocs for the full interface method listing):

```
com.isdduk.text.TermSplitter

     split(java.lang.String source) :
com.isdduk.util.set.FastSet<java.lang.String>
     split(java.lang.String source, boolean validate) :
com.isdduk.text.StringList
     splitFrequencies(char[] source,
          com.isdduk.util.map.FastTermMap insertInto) : void
     splitFrequencies(java.lang.CharSequence source,
          com.isdduk.util.map.FastTermMap insertInto) : void
```

## Configuring the term stemmer

The term stemmer interface is much simpler than its splitter counterpart. It defines only three methods:

```
com.isdduk.text.TermStemmer
     stem(com.isdduk.text.Term term) : com.isdduk.text.Term
     hasNormalize() : boolean
     normalize(com.isdduk.text.Term term) : com.isdduk.text.Term
```

The stem method takes a term argument and returns a stemmed version of it, which is in many cases the same object, although perhaps with a different length. The normalize method caters for terms that are not sent through the stem method (which should incorporate normalization as part of its routine)—it ensures the term conforms to a single standard of representation (for example, a German stemmer may normalize the sharp S “ß” to its equivalent “ss” or vice versa). Terms may bypass the stem method occasionally, when their lengths exceed the maximum allowed (and are therefore “force stemmed” to fit).

## Replacing the system text and term splitters

Once you have compiled the new term splitter class, you must make it available to the system. The easiest way to do this is by adding the class to a JAR file and placing the JAR file in the *%OMNIQ_3.0%\OmniQ\lib* directories.

---

**Note** You must place the JAR file into every container’s library folder since OmniQ Enteprise is a distributed system.

---

The module responsible for loading the term splitter and stemmer is the Text Manager module. Edit the *TextModule.default.xml* configuration file located in *%OMNIQ_3.0%\OmniQ\config* and change the term.splitter.class property and the term.stemmer.class property as necessary.

---

**Note** You must perform the metadata changes for the container instance that loads the Text Manager.

---

APPENDIX A

# Generated Files

Each module contains its own directory where it stores files. These can be serialized Java object files or proprietary data structures. The format of each directory is the short name of the module followed by its unique module ID.

## Module files

***Table A-1: Module generated file locations***

| Module | File location |
|---|---|
| Document Group Manager | *$OMNIQ_HOME\OmniQ\Data\DGM-<uid>*<br>where *<uid>* is its unique module ID. |
| Document ID generator | *$OMNIQ_HOME\OmniQ\Data\DocIdGenerator-<uid>*<br>where *<uid>* is its unique module ID. |
| Filter factory | *$OMNIQ_HOME\OmniQ\Data\FilterFactory-<uid>* |
| Metadata Manager | *$OMNIQ_HOME\OmniQ\Data\MetadataModule-<uid>*<br>where *<uid>* is its unique module ID. |
| Metadata Manager Delegate | *$OMNIQ_HOME\OmniQ\Data\MetadataModuleDelegate-<uid>*<br>where *<uid>* is its unique ID. |
| Query Manager | *$OMNIQ_HOME\OmniQ\Data\QueryModule-<uid>*<br>where *<uid>* is its unique ID. |
| Repository Module | *$OMNIQ_HOME\OmniQ\Data\RepositoryModule-<uid>*<br>where *<uid>* is its unique ID. |
| Term Lexicon Manager | *$OMNIQ_HOME\OmniQ\Data\TermLexiconModule-<uid>*<br>where *<uid>* is its unique ID. |
| Term Lexicon Manager Delegate | *$OMNIQ_HOME\OmniQ\Data\TermLexiconModuleDelegate-<uid>*<br>where *<uid>* is its unique ID. |
| Text Manager | *$OMNIQ_HOME\OmniQ\Data\TextModule-<uid>*<br>where *<uid>* is its unique ID. |

| Module | File location |
|---|---|
| Unique ID generator | *$OMNIQ_HOME\OmniQ\Data\UID-<uid>*<br>where *<uid>* is its unique ID. |
| Document Store Manager | *$OMNIQ_HOME\OmniQ\Data\DSM-<uid>*<br>where *<uid>* is its unique ID. |
| Document Stores | *$OMNIQ_HOME\OmniQ\Data\DSM-<uid1>\DS-<uid2>*<br>where *<uid1>* is the unique ID of the Document Store Manager and *<uid2>* is the ID of the Document Store. |

# Index

## T



## U



## X