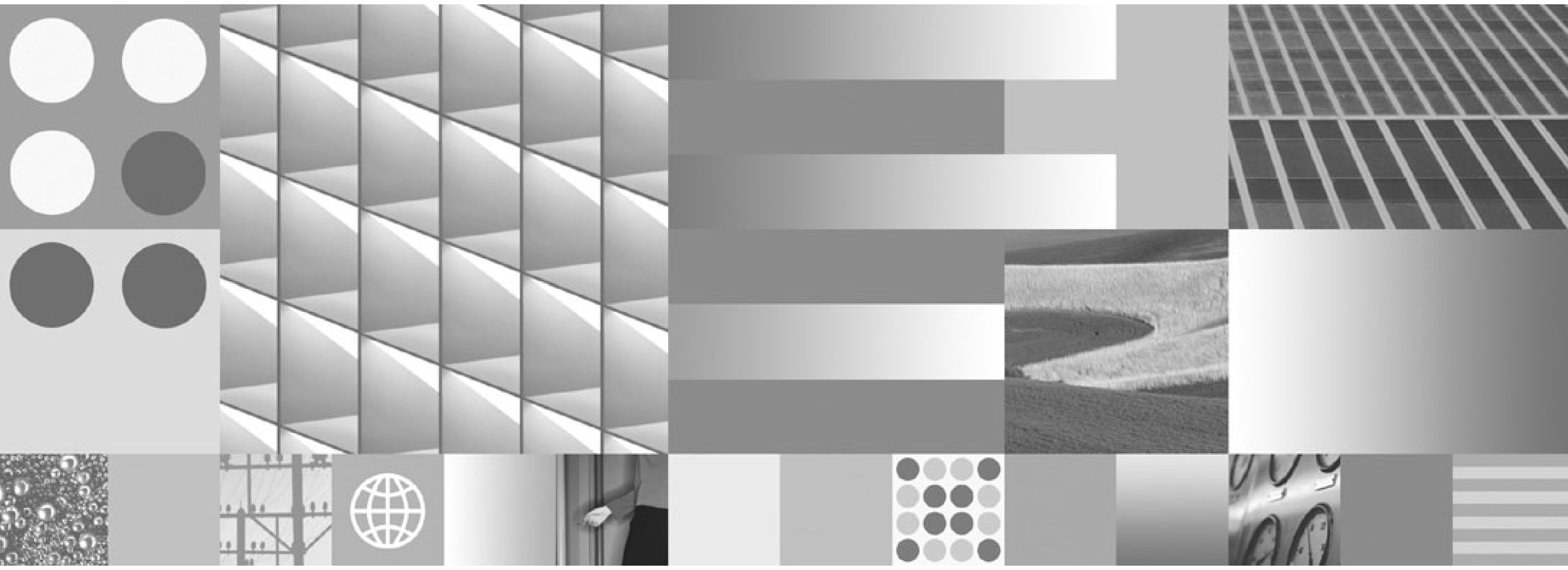
IBM OmniFind Enterprise Edition

Version 8.5

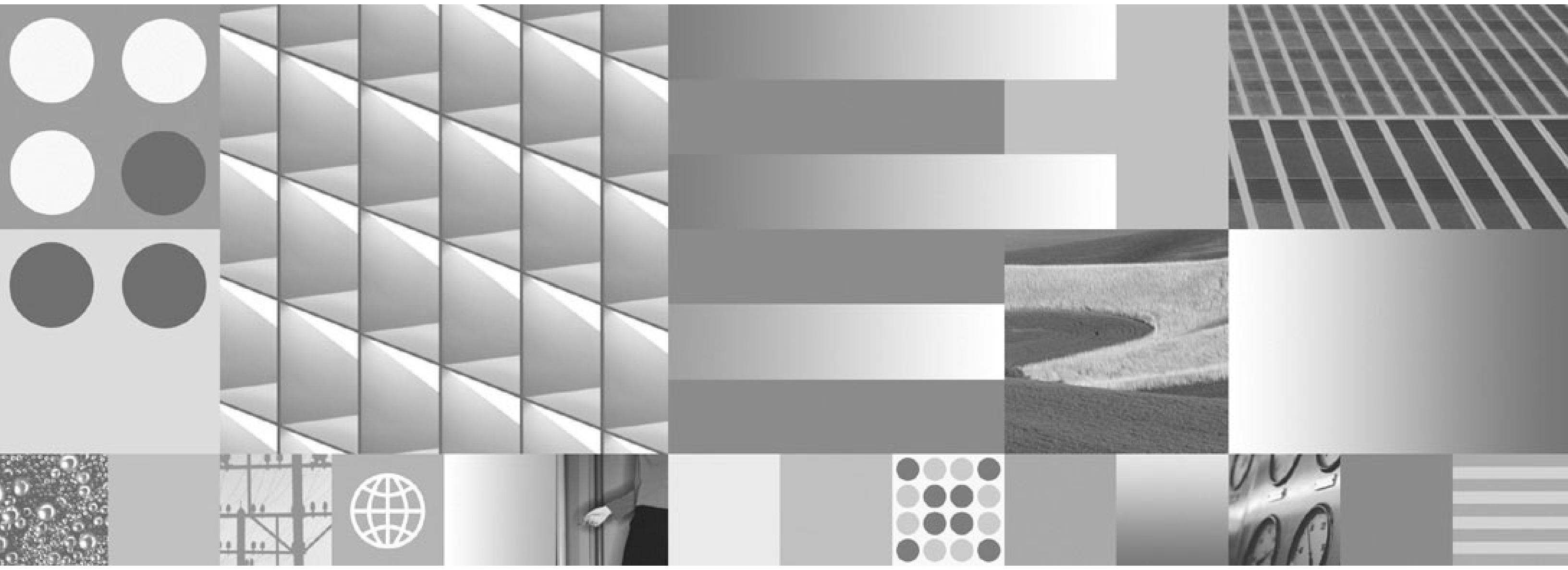


Programming Guide and API Reference for Enterprise Search

SC18-9284-04

IBM OmniFind Enterprise Edition

Version 8.5



Programming Guide and API Reference for Enterprise Search

SC18-9284-04

Note

Before using this information and the product it supports, read the information in “Notices and trademarks” on page 103.

Edition Notice

This edition applies to version 8, release 5, modification 0 of IBM OmniFind Enterprise Edition (product number 5724-C74) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

ibm.com and related resources

Product support and documentation are available from ibm.com®.

Support and assistance

Product support is available on the Web.

IBM® OmniFind™ Enterprise Edition
http://www.ibm.com/software/data/enterprise-search/omnifind-enterprise/support.html

IBM OmniFind Discovery Edition
http://www.ibm.com/software/data/enterprise-search/omnifind-discovery/support.html

IBM OmniFind Yahoo! Edition
http://www.ibm.com/software/data/enterprise-search/omnifind-yahoo/support.html

Information center

You can view the product documentation in an Eclipse-based information center with a Web browser. See the information center at http://publib.boulder.ibm.com/infocenter/discover/v8r5m0/.

PDF publications

You can view the PDF files online using the Adobe® Acrobat Reader for your operating system. If you do not have the Acrobat Reader installed, you can download it from the Adobe Web site at http://www.adobe.com.

See the following PDF publications Web sites:

Product	Web site address
OmniFind Enterprise Edition, Version 8.5	http://www.ibm.com/support/docview.wss?rs=63&uid=swg27010938
OmniFind Discovery Edition, Version 8.4	http://www.ibm.com/support/docview.wss?rs=3035&uid=swg27008552
OmniFind Yahoo! Edition, Version 8.4	http://www.ibm.com/support/docview.wss?rs=3193&uid=swg27008932

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

Send your comments by using the online reader comment form at https://www14.software.ibm.com/webapp/iwm/web/signup.do?lang=en_US&source=swg-rcf.

Contacting IBM

To contact IBM customer service in the United States or Canada, call 1-800-IBM-SERV (1-800-426-7378).

To learn about available service options, call one of the following numbers:

- In the United States: 1-888-426-4343
- In Canada: 1-800-465-9600

For more information about how to contact IBM, see the Contact IBM Web site at http://www.ibm.com/contact/us/.

Application development

Enterprise search API overview

IBM OmniFind Enterprise Edition provides several sets of Java™ application programming interfaces (APIs) for enterprise search so that you can create search and administration applications, modify crawled documents, or set up an identity management component to enforce document-level security.

IBM search and index APIs

Use the search and index application programming interfaces to create custom search applications. The enterprise search implementation of the search and index API (SIAPI) allows the search server to be accessed remotely. The search server stores the collection data for the enterprise search system. With these APIs, you can create applications that submit search requests, process and fetch search results, or browse taxonomy trees.

You can also use the search and index APIs to create applications to administer collections and enable indexes to be searched. With the administration APIs, you can administer collections from team rooms or portlets. You can also migrate indexing products to administer collections and indexes.

Enterprise search requires IBM WebSphere® Application Server, which you can install or which can be installed automatically by the OmniFind Enterprise Edition installation program. To ensure that your custom applications stay current with changes made to WebSphere Application Server, or to apply fixes for problems that might occur in WebSphere Application Server, periodically check the following Web site for information about interim fixes and cumulative fix packs for WebSphere Application Server: Recommended Fixes for WebSphere Application Server (http://www.ibm.com/support/docview.wss?uid=swg27004980).

Sample search application (ESSearchApplication)

You can use the sample search application that is provided with OmniFind Enterprise Edition as a base from which to develop your custom search applications. This sample application shows you how to do basic search and retrieval tasks with enterprise search collections, such as selecting collections for search, querying those collections, and configuring the display of search results.

Important: If you customize the sample search application, which is named ESSearchApplication, you must rename it to ensure that your changes are not overwritten when you install a fix pack or upgrade to a new version of OmniFind Enterprise Edition.

Crawler plug-in APIs

To modify documents after they are crawled, but before they are parsed and indexed for search, you can use the crawler plug-in APIs to add, change, or delete information in the document or the document metadata. You can also indicate that the document is to be ignored (skipped) and not indexed.

The crawler plug-in APIs are not part of the search and index APIs (SIAPI).

Identity management component APIs

Access to sensitive information that is contained in multiple repositories is typically controlled and enforced by the managing software. You identify yourself to the host system with a user ID and password. After the system authenticates your user ID and password, the managing software controls which documents you are allowed to see based on your access rights. Unless the enterprise has implemented a single sign-on policy, you must have several different user IDs and passwords for each repository.

OmniFind Enterprise Edition provides an identity management component that enables users to search multiple repositories with a single query and see only the documents that they are allowed to see. You can build this component into your applications so that users can sign on with only one user ID and password when searching secure collections.

See the Javadoc documentation for details about the APIs that can be used to create your own identity management component or customize the existing solution.

Installing the client toolkit

To build enterprise search applications, you must install the client toolkit (the `es.siapi.toolkit.jar` archive file).

About this task

The toolkit contains the required Java packages, sample applications, build scripts, and Javadoc documentation to build search applications and administration applications. The toolkit does not contain APIs for the identity management component.

The toolkit distributes the following sample applications:

- Search samples show how to use the search and index APIs to search collections, search categories, and retrieve search results.
- Administration samples show how to use the search and index APIs to create an application ID, create or destroy a collection, enable or disable a collection for indexing, enable or disable a collection for searching, add or remove documents, or build an index.
- Web service samples show how to use the Web service client proxy. The client proxy can be used to access the Web service that is hosted by the ESSearchServer application to process search requests.

Procedure

To install the client toolkit:

1. Find the client toolkit JAR file for your operating system or contact the search administrator to obtain the file. For a multiple server installation, the `es.siapi.toolkit.jar` archive file is on both search servers.
   - AIX®, Linux®, and Solaris: `ES_INSTALL_ROOT/lib/es.siapi.toolkit.jar`
   - Windows®: `ES_INSTALL_ROOT\lib\es.siapi.toolkit.jar`
2. Extract the JAR file by running the following command:
   ```
   jar -xvf es.siapi.toolkit.jar
   ```

3. Edit the `es.cfg` configuration file. This file is in the `ES_INSTALL_ROOT\lib\es.siapi.toolkit.jar` archive file. Add the following two lines at the end of the file:
   - `es_server_hostname=`*`fully qualified host name of the enterprise search index server`*

     This field is mandatory and specifies the index server host name, for example, omnifind.server.ibm.com.

     If you do not want to change the `es.cfg` file, the `es_server_hostname` parameter can be passed as a system argument to your Java application, for example: `-Des_server_hostname=omnifind.server.ibm.com`.
   - `.logFileName=`*`absolute path of a log file`*

     This field is optional and registers the trace information in a specific log file, for example, `c:\temp\siapi.log`. This field can be used to enhance the trace information from the APIs. The accepted values are INFO and ALL. The default value is INFO.

     By default a file called `siapi.log` is created in the directory where the custom applications are invoked.

   The configuration file required by the APIs to access the enterprise search server. Pass the absolute path of the configuration file as a system argument to the Java executable files.

Javadoc documentation

Javadoc documentation is available for the search and index APIs (for both search and administration applications), crawler plug-ins, and the identity management component.

The Javadoc documentation is installed in these default locations:

Javadoc documentation	Installation directory
Search and index APIs	`ES_INSTALL_ROOT/docs/api/siapi`
Fetch API	`ES_INSTALL_ROOT/docs/api/fetch`
Non-Web crawler plug-ins	`ES_INSTALL_ROOT/docs/api/crawler`
Web crawler plug-ins	`ES_INSTALL_ROOT/docs/api/wc`
Identity management APIs	`ES_INSTALL_ROOT/docs/api/imc`

Related concepts

"Search and index APIs"

"Administration applications" on page 44

Related reference

"Search applications" on page 7

Search and index APIs

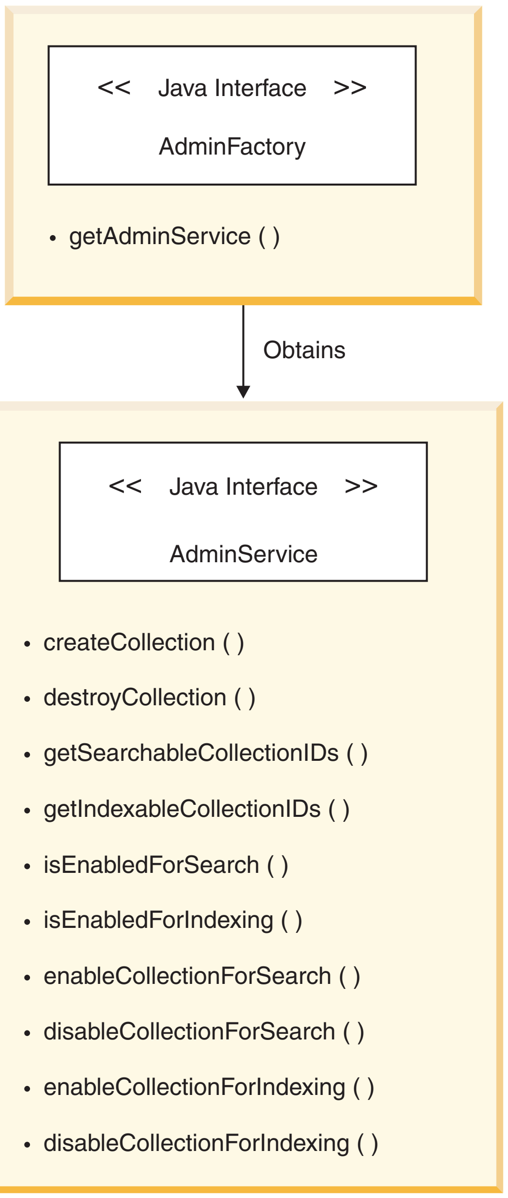
The IBM search and index API is a programming interface that enables you to search, browse, and administer collections and taxonomies.

The search and index API is a factory based interface that allows for different implementations of the search engine. By using search and index APIs, your search application can use different search engines that are provided by IBM without changing your search and index API application. For example, if you create a search and index API application in WebSphere Portal that uses the portal search engine, you can use the OmniFind Enterprise Edition enterprise search engine without the need to change your search application.

The search and index APIs support the following types of search and administration tasks:

- Search application tasks:
  - Searching indexes
  - Customizing the information that is returned in search results sets
  - Searching and browsing taxonomies
  - Searching over several collections as if they were one collection (search federation)
  - Viewing results with URIs that you can click and viewing scoring information (ranking)
  - Searching and retrieving documents from a broad range of enterprise data sources, such asWebSphere Information Integrator Content Edition repositories and Lotus Notes® databases
- Administration application tasks:
  - Administering user application IDs
  - Creating or destroying collections
  - Enabling or disabling collections for search and indexing
  - Building and reorganizing collections
  - Adding documents to a collection or removing documents from collections

The following figures show the relationships among the search and index APIs.



<< Java Interface >>
AdminFactory
• getAdminService ( )
Obtains
<< Java Interface >>
AdminService
• createCollection ( )
• destroyCollection ( )
• getSearchableCollectionIDs ( )
• getIndexableCollectionIDs ( )
• isEnabledForSearch ( )
• isEnabledForIndexing ( )
• enableCollectionForSearch ( )
• disableCollectionForSearch ( )
• enableCollectionForIndexing ( )
• disableCollectionForIndexing ( )

Figure 1. Administration APIs

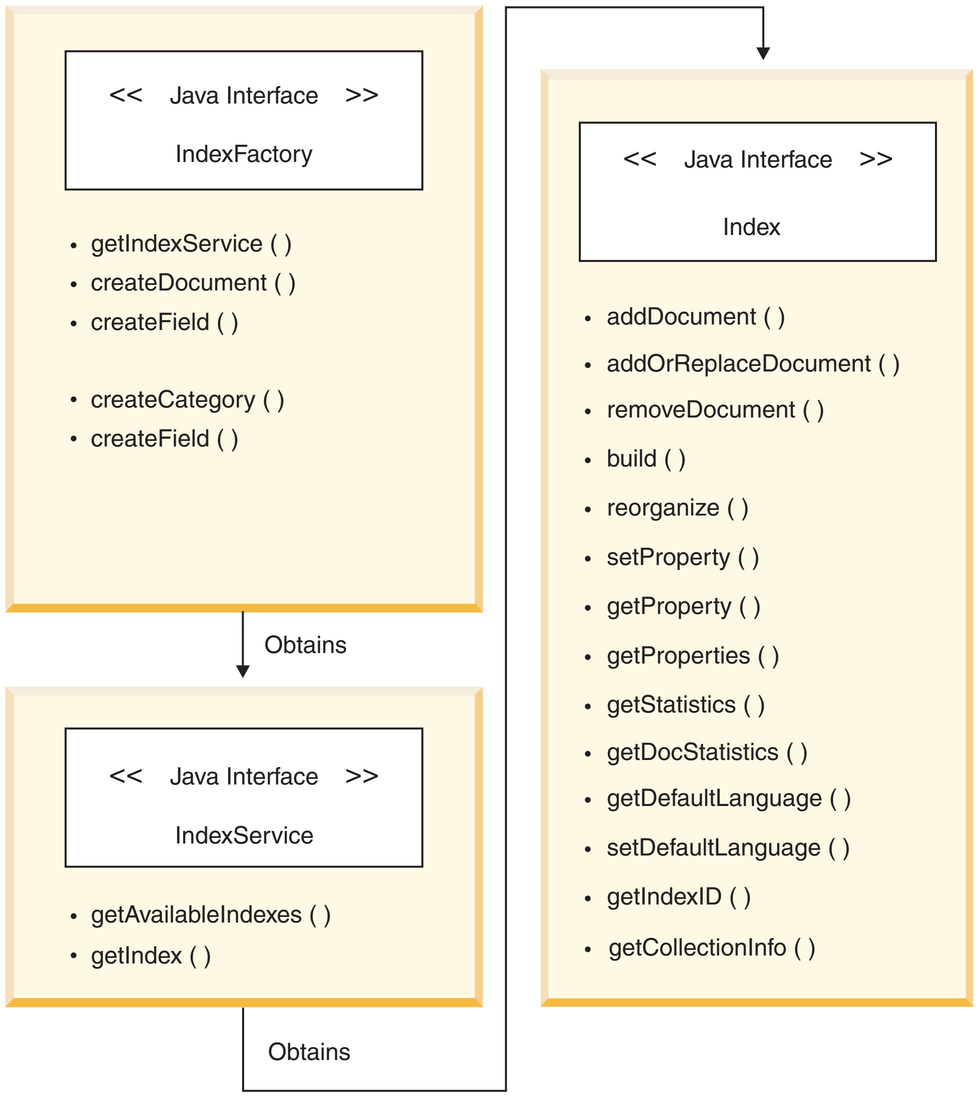


Figure 2. Index APIs

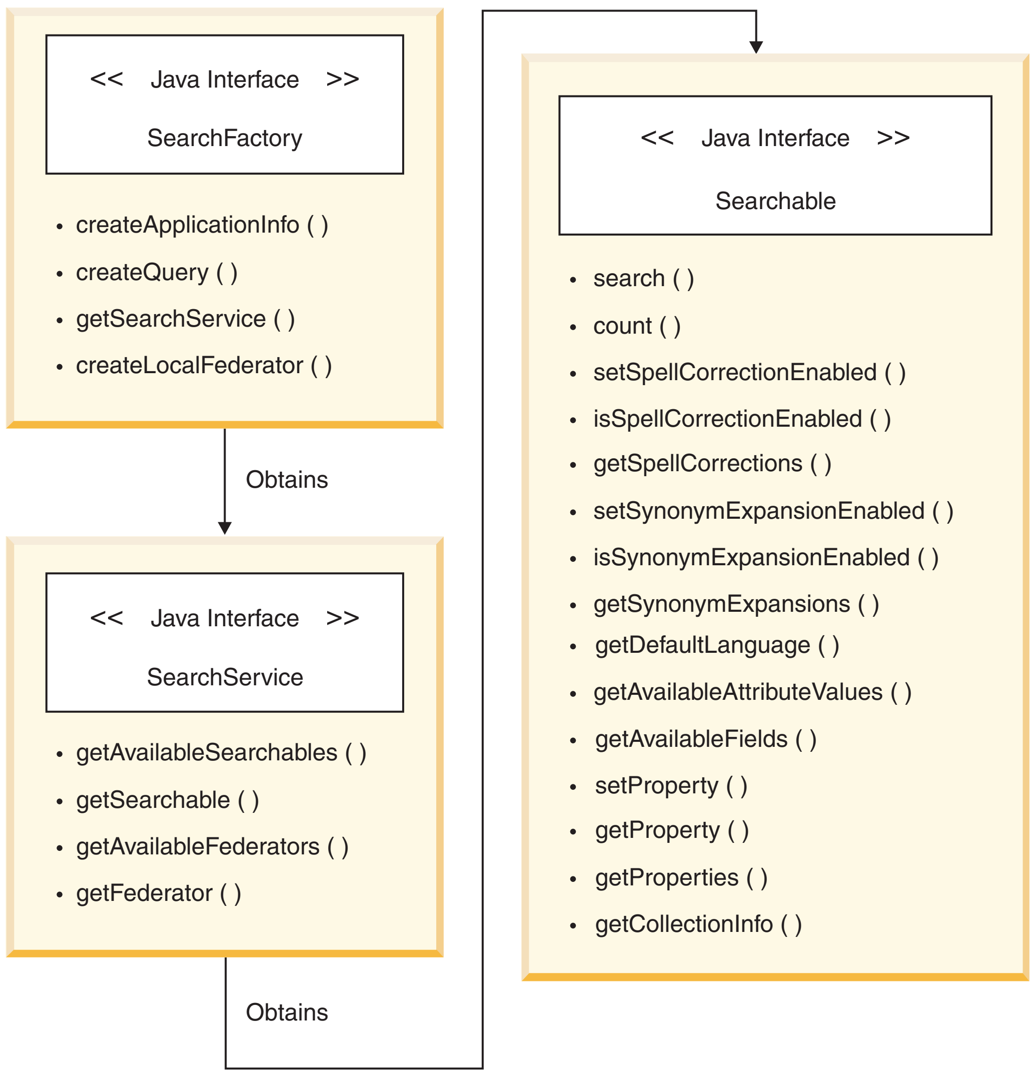


Figure 3. Search APIs

Related concepts

"Administration applications" on page 44

"Javadoc documentation" on page 3

Search applications

Search applications can access enterprise search collections, issue queries, and process query results.

See the Javadoc documentation for examples of the search and index APIs.

To create a search application with the search and index APIs, follow these general steps:

1. Instantiate an implementation of a `SearchFactory` object.

   The `SearchFactory` can then be used to obtain a `SearchService` object.
2. Use the `SearchFactory` object to obtain a `SearchService` object.

   The `SearchService` object is configured with the connection information that is necessary to communicate with the search engine. With the `SearchService` object, you can access searchable collections. Configure the `SearchService` object with the host name, port, and, if WebSphere global security is enabled, a valid WebSphere user name and password for the search server. Configuration parameters are set in a `java.util.Properties` object. The parameters are then passed to the `getSearchService` factory method that generates the `SearchService` object.

Enterprise search applications support Secure Sockets Layer (SSL) version 3. However, applications that use SSL must include a reference to an existing keystore. WebSphere Application Server provides a utility called `iKeyman.exe` in the Java Runtime Environment `bin` directory that can be used for working with keystores.

With SSL, you can establish a security-enabled Web site on the Internet or on your private intranet. A browser that does not support HTTP over SSL cannot request URLs that use HTTPS.

When you request a search and index API service, such as `SearchService searchService = factory.getSearchService(Properties);`, you can use any of the following properties for a service object. The property names are case sensitive.

Table 1. Property values for service API objects

Property name	**Expected value**
`protocol`	HTTP or HTTPS for SSL. The default is HTTP. If the protocol is HTTPS, the host name must be fully qualified and the port must be the SSL port. The default port is 443.
`trustStore`	The fully qualified path to the keystore. If the operating system is Windows, the backslashes must be escaped with a double backslash, for example, `c:\\temp\\WASWebContainer.jks`. **Restriction:** If the protocol is HTTPS, the `trustStore` value must not be empty. An exception is thrown if the `trustStore` property is null.
`trustPassword`	The password to access the keystore. **Restriction:** If the protocol is HTTPS, then the `trustPassword` value must not be empty. An exception is thrown if the `trustPassword` property is null.
`proxyHost`	The host name of the proxy server.
`proxyPort`	The port number for the proxy server.
`proxyUser`	If the proxy server requires HTTP basic authentication, this is the user name for that login request.
`proxyPassword`	The password for the user that is specified by the `proxyUser` parameter.

3. Obtain a Searchable object.

   After you obtain a `SearchService` object, you can use it to obtain one or more Searchable objects. Each search and index API searchable object is associated with one enterprise search collection. You can also use the `SearchService` object to obtain a `federator` object. A `federator` object is a special kind of `Searchable` object that enables you to submit a single query across multiple `Searchable` objects (collections) at the same time.

   When you request a `Searchable` object, you need to identify your application by using an application ID. Contact your enterprise search administrator for the appropriate application ID.

4. Issue queries.

The search application passes search queries to the search runtime on the search server.

After the `Searchable` object is obtained, you issue a query to that `Searchable` object. To issue a query to the `Searchable` object:

a. Create a `Query` object.
b. Customize the `Query` object.
c. Submit the `Query` object to the `Searchable` object.
d. Get the query results, which are specified in a `ResultSet` object.

5. Process query results.

   Process queries with the `ResultSet` interface object and the `Result` interface object. The search and index APIs have a variety of methods for interacting with the `ResultSet` interface and individual `Result` interface objects.

The search and index APIs are a factory-based Java API. All of the objects that are used in the search application are created by calling search and index API object-factory methods or are returned by calling methods of factory-generated objects. You can easily switch between search and index API implementations by loading different factories.

The search and index API implementation in OmniFind Enterprise Edition is provided by the `com.ibm.es.api.search.RemoteSearchFactory` class.

Use the following search and index API packages to create a search application:

com.ibm.siapi
Root package

com.ibm.siapi.browse
Contains taxonomy browsing interfaces

com.ibm.siapi.common
Common SIAPI interfaces

com.ibm.siapi.search
Interfaces for searching collections

Obtaining a `SearchFactory` object

To create a search and index API search application, obtain the implementation of the `SearchFactory` object as in the following example:

```
SearchFactory factory =
SiapiSearchImpl.createSearchFactory
    ("com.ibm.es.api.search.RemoteSearchFactory");
```

Obtaining a `SearchService` object

Use the `SearchFactory` object to obtain a `SearchService` object. With the `SearchService` object, you can access searchable collections.

Configure the `SearchService` object with the host name, port, and, if WebSphere global security is enabled, a valid WebSphere user name and password for the search server.

Configuration parameters are set in a `java.util.Properties` object. The parameters are then passed to the `getSearchService` factory method that generates the `SearchService` object. The following example shows how to obtain a `SearchService` object:

```
Properties configuration = new Properties();
configuration.setProperty("hostname", "es.mycompany.com");
configuration.setProperty("port", "80");
config.setProperty("username", "websphereUser");
config.setProperty("password", "webspherePassword");
SearchService searchService =
 factory.getSearchService(config);
```

Obtaining a `Searchable` object

Use the `SearchService` object to obtain a `Searchable` object. A `Searchable` object is associated with a searchable collection. With a `Searchable` object, you can issue queries and get information about the associated collection. Each enterprise search collection has an ID.

When you request a `Searchable` object, you need to identify your application by using an application ID. Contact your enterprise search administrator for the appropriate application ID.

The following example shows how to obtain a Searchable object:

```
ApplicationInfo appInfo = factory.createApplicationInfo
("my_application_id","my_password");
Searchable searchable =
 searchService.getSearchable(appInfo, "some_collection_id");
```

Call the `getAvailableSearchables` method to obtain all of the `Searchable` objects that are available for your application.

```
Searchable[] searchables =
 searchService.getAvailableSearchables(appInfo);
```

Issuing queries

After the `Searchable` object is obtained, you issue a query to that `Searchable` object. To issue a query to the `Searchable` object:

1. Create a `Query` object.
2. Customize the `Query` object.
3. Submit the `Query` object to the `Searchable` object.
4. Get the query results, which are specified in a `ResultSet` object.

The following example shows how to issue a query:

```
String queryString = "big apple";
Query query = factory.createQuery(queryString);
query.setRequestedResultRange(0, 10);
ResultSet resultSet = searchable.search(query);
```

Processing query results

With the `ResultSet` interface and `Result` interface, you can process query results, as in the following example:

```
Result[] results = resultSet.getResults();
for ( int i = 0 ; i < results.length ; i++ ) {
 System.out.println
( "Result " + i + ": " + results[i].getDocumentID()
  + " - " + results[i].getTitle() );
}
```

Related concepts

"Administration applications" on page 44

"Javadoc documentation" on page 3

Controlling query behavior

With the methods and properties that belong to the `Query` interface, you can control many aspects of query behavior, including how the query is processed, how results are returned, and what metadata is returned with each result.

See the Javadoc documentation for more details about each method and property.

Related concepts

"Query syntax" on page 19

Related reference

"Query syntax structure" on page 31

"Detailed Web service client application" on page 97

Creating secure searches with access control list constraints

You can set the access control list constraints for a query for secure searches by using the `setACLConstraints(java.lang.String aclConstraints)` method.

The `setACLConstraints(java.lang.String aclConstraints)` method supports the following XML query string:

```
@SecurityContext::'<XML query string>
```

Setting query properties

You can control query processing by using the `setProperty` method.

The `setProperty` method for query object has the following format:

```
query.setProperty("String name", "String value");
```

You can set the following query properties:

- HighlightingMode: Enables query terms to be highlighted in several areas of the search result details. Values are:
  - DefaultHighlighting: Highlights query terms in the summary only. This is the default if your search application does not set the HighlightingMode property.
  - ExtendedHighlighting: Extends the highlighting of query terms to other areas of the search result, for example, title, URL, and other fields.
- FuzzyNGramSearch: Fuzzy search enables a non-strict search in n-gram collections to be performed. This property is Boolean and its values are:
  - false: A strict search will be performed. This is the default if your search application does not set the `FuzzyNGramSearch` property.
  - true: Fuzzy search will be performed.
- ProximityWindowSize: When two query terms fall in a window of this size, the document score is boosted by a proximity boost. The value of this property is an unsigned integer. The default value is 5.

- AllowStopwordRemoval: Determines whether stop words are removed during query parsing. If this property is not set, the engine removes or does not remove stop words according to the policy of the search engine. This property is Boolean and its values are:
  - false: Stop words are not removed during query parsing.
  - true: Stop words are removed during query parsing.
- NearDuplicateDetection: Specifies whether documents that are nearly identical are to be suppressed when search results are displayed. The default value is No.
  - Yes: Enables documents with similar titles and summaries to be suppressed when a user views search results. For nearly duplicate document analysis to be performed, an enterprise search administrator must ensure that the `config.properties` file for the search application specifies the property **preferences.nearDuplicateDetection=Yes**.
  - No: Search results are not filtered to suppress documents that have similar titles and summaries to documents that are already displayed in the search results.

Exposing ranking information

You can add the `ExposeRankInfo` property to the query object in the search application.

When the search application associates with the `Query` object the Boolean property `ExposeRankInfo` with a value of true, enterprise search returns ranking information as properties that are associated with the `ResultSet` object.

The following properties are associated with the `ResultSet` object, and they are accessible through the `ResultSet.getProperty(property-name)` object:

- `TextScoreWeightInFinalScore` property: This value, which is also known as `a`, is the coefficient of the text score in the final document score. The final score of the document is represented as `a*(text score) + (1-a)*(static score)`.
- `BoostsVector` property: This value is a string of 16 decimal integers separated by spaces. These are the boost values of the 16 boost classes to which fields and token attributes of the document are associated by the search administrator. The 16 values match the boost classes in the following order: Content class A, ..., Content class H, Metadata class A, ..., Metadata class H
- `QueryWordBoosts` property: This value is a string made of query words. Each word is followed by its boost and the boost value is a decimal integer. The words and the integers are separated by spaces. The boosts for the query words are determined by dictionaries.

The following properties are associated with the `Result` object, and they are accessible through the `Result.getProperty(property-name)` object:

- `TermHitsPerBoostClass` property: This property has 16 decimal integer values that are separated by spaces. The `i-th` integer specifies the number of query term occurrences in the document as tokens of the token attribute that is associated to boost class `i`. The 16 values match the boost classes in the following order: Content class A, ..., Content class H, Metadata class A, ..., Metadata class H
- `StaticScore` property: This property, whose value is an integer, is a string that represents the static score of the document.
- `NormalizedStaticScore` property: This property, whose value is a double precision number (Double), represents the normalized static score (to a number from 0 to 1) .

- `TextScore` property: This property, whose value is a double precision number (Double), represents the text score of the document.
- `KeywordCount` property: This property, whose value is a decimal integer, represents the number of words in the document.
- `UniqueKeywordCount` property: This property, whose value is a decimal integer, represents the number of unique (distinct) words in the document.

Enabling fuzzy searches

A fuzzy search query searches for character sequences that are not only the same but similar to the query term. All possible n-grams are treated as search terms, and the query returns documents including specified n-grams. However, it does not always mean that the documents have character sequences that are similar to the query term.

In this example, the capital letters ABC and the at sign (@) refer to n-gram characters. For example, if the query term is ABCDE, a typical n-gram fuzzy search returns a document that includes character sequence such as @@AB@@@BC@@@@@@CD@@@@@DE because this document has all the n-grams that are generated from the specified query. However, for some languages, this query result is not preferable because it often means completely different meanings if those n-grams are far apart.

To improve fuzzy search results, you can control the level of ambiguity in the query by specifying the `FuzzyNGramAmbiguity` property and optionally the `FuzzyNGramAmbiguityCondition` property.

The `FuzzyNGramAmbiguity` property returns documents with the most (not necessarily all) n-grams that are more closely related such as @@ABC@DE@@@ by using an ambiguity calculation that is based on each query term.

`FuzzyNGramSearch` property

Fuzzy search performs a non-strict search in n-gram collections. This property is Boolean:

- false: A strict search is performed. This is the default if your search application does not set the `FuzzyNGramSearch` property.
- true: Fuzzy search is performed.

`FuzzyNGramAmbiguity` property

This property is activated only if the `FuzzyNgramSearch` property is set to true. These properties are configured by the `Query.setProperty` method.

The ambiguity must be greater than 0.0 and less than or equal to 1.0. If the ambiguity is set to 1.0, it is equivalent to an exact match. The lower the number that the ambiguity is set to, the more it allows ambiguity determines whether each document has similar character sequences to the search term. Thus, the search query retrieves more documents.

Ambiguity is similar to the ratio of characters appearing in the same position and the same order to the search query.

- Format: *ambiguity*
- Ambiguity: float value, 0.0< ambiguity <= 1.0, to specify ambiguity

This property is used to set the ambiguity that is applied to all search terms of the query except for the terms that are specified by the `FuzzyNGramAmbiguityCondition` property. The higher the ambiguity is, the more similar the returned document will be. In other words, the document includes character sequences closer to the original search term if the higher ambiguity is specified.

`FuzzyNGramAmbiguityCondition` property

Optional: Activated only if the `FuzzyNgramSearch` property is set to true. These properties are configured by the `Query.setProperty` method.

- Format: term1=ambiguity1[,term2=ambiguity2]...[,termn=ambiguityn]
- Term: character string, to specify the term which overrides the ambiguity. (Note that *Term* does not include operands such as +, -, ~, but it does include field names such as tablename)
- Ambiguity: float value, 0.0< ambiguity <= 1.0, to specify ambiguity of the Term

This property is an optional property to specify ambiguity to each term. When a term includes a comma (,), it must be two commas (,,) to escape the delimiter. This rule does not apply to the equal sign (=) because the last equal sign before each comma is treated as the term end.

In the following example, enterprise search searches for all terms except for tablename: DATA_TBL with ambiguity 0.8, and search for tablename: DATA_TBL with ambiguity 1.0 (exact):

```
q.setProperty("FuzzyNGramAmbiguity", "0.8");
q.setProperty("FuzzyNGramAmbiguityCondition",
  "tablename:DATA_TBL=1.0");
```

Specifying query languages

You can use the `setQueryLanguage(java.lang.String lang)` method to specify a language other than the collection default language on the search server.

Use Unicode identifiers for languages to set a specific language. For example, for English, the query language parameter is `en`. For Chinese, use `zh-CN` for simplified Chinese and `zh-TW` for traditional Chinese.

Setting linguistic modes

Use the `setLinguisticMode(int mode)` method to specify how you want the search engine to match query terms.

The `setLinguisticMode(int mode)` method sets the linguistic mode for a query. You can set one of the following modes:

- LINGUISTIC_MODE_ENGINE_DEFINED: Unmodified terms are matched according to the engine's best-effort policy. This is the default mode.
- LINGUISTIC_MODE_EXACT_MATCH: Unmodified terms are matched as entered without undergoing linguistic processing.
- LINGUISTIC_MODE_BASEFORM_MATCH: Unmodified terms are matched by their base form after undergoing linguistic processing. For example, the query term *jumping* matches documents that contain *jump*, *jumped*, *jumps*, and so on.
- LINGUISTIC_MODE_EXACT_AND_BASEFORM: Unmodified terms are matched by their base form and their exact form after undergoing linguistic processing. For example, the query term *jumping* matches documents that contain *jump*, *jumped*, *jumps*, and so on. The difference from the LINGUISTIC_MODE_BASEFORM_MATCH mode is that although linguistic base form matching relies on the query language that matches the identified

languages of the result documents, the LINGUISTIC_MODE_EXACT_AND_BASEFORM mode assures that documents that contain the exact form *jumping* are returned regardless of their identified language.

Returning metadata fields

You can use the `setReturnedFields(String[] fieldNames)` method to control which metadata fields are returned in the `Result` object.

By default, enterprise search does not return any metadata fields, so you must use this method to return metadata fields.

Returning ranking information in queries

When the search application associates with the `Query` object the Boolean property `ExposeRankInfo` with a value of true, enterprise search returns ranking information as properties that are associated with the `ResultSet` object. You can add the `ExposeRankInfo` property to the query object in the search application.

The following three properties are associated with the `ResultSet` object, and they are accessible through the `ResultSet.getProperty(property-name)` object:

- `TextScoreWeightInFinalScore` property: This value, which is also known as `a`, is the coefficient of the text score in the final document score. The final score of the document is represented as `a*(text score) + (1-a)*(static score)`.
- `BoostsVector` property: This value is a string of 16 decimal integers separated by spaces. These are the boost values of the 16 boost classes to which fields and token attributes of the document are associated by the search administrator. The 16 values match the boost classes in the following order: Content class A, ..., Content class H, Metadata class A, ..., Metadata class H
- `QueryWordBoosts` property: This value is a string made of query words. Each word is followed by its boost and the boost value is a decimal integer. The words and the integers are separated by spaces. The boosts for the query words are determined by dictionaries.

The following properties are associated with the `Result` object, and they are accessible through the `Result.getProperty(property-name)` object:

- `TermHitsPerBoostClass` property: This property has 16 decimal integer values that are separated by spaces. The `i-th` integer specifies the number of query terms occurrences in the document as tokens of the token attribute that is associated to boost class `i`. The 16 values match the boost classes in the following order: Content class A, ..., Content class H, Metadata class A, ..., Metadata class H
- `StaticScore` property: This property, whose value is an integer, is a string that represents the static score of the document.
- `NormalizedStaticScore` property: This property, whose value is a double precision number (Double), represents the normalized static score (to a number from 0 to 1) .
- `TextScore` property: This property, whose value is a double precision number (Double), represents the text score of the document.
- `KeywordCount` property: This property, whose value is a decimal integer, represents the number of words in the document.
- `UniqueKeywordCount` property: This property, whose value is a decimal integer, represents the number of unique (distinct) words in the document.

Enabling predefined result attribute values

You can use the `setReturnedAttribute(int attributeType, boolean isReturned)` method to enable or disable any of the predefined result attribute values that are returned with each `Result` object.

By default, enterprise search returns all the predefined result attribute values except for the metadata fields attribute RETURN_RESULT_FIELDS.

The following values are valid for the `attributeType` object:

- RETURN_RESULT_TITLE: The `Result.getTitle` object returns null if the `isReturned` object is set to false.
- RETURN_RESULT_DESCRIPTION: The `Result.getDescription` object returns null if the `isReturned` object is set to false.
- RETURN_RESULT_FIELDS: The `Result.getFields` object and the `Result.getFields(String)` object return null if the `isReturned` object is set to false.
- RETURN_RESULT_CATEGORIES: The `Result.getCategories` object returns null if the `isReturned` object is set to false.
- RETURN_RESULT_TYPE: The `Result.getDocumentType` object returns null if the `isReturned` object is set to false.
- RETURN_RESULT_SOURCE: The `Result.getDocumentSource` object returns null if the `isReturned` object is set to false.
- RETURN_RESULT_LANGUAGE: The `Result.getLanguage` object returns null if the `isReturned` object is set to false.
- RETURN_RESULT_DATE: The `Result.getDate` object returns null if the `isReturned` object is set to false.
- RETURN_RESULT_SCORE: The `Result.getScore` object returns 0.0 if the `isReturned` object is set to false.
- RETURN_RESULT_URI: The `Result.getDocumentURI` object returns null if the `isReturned` object is set to false.

Specifying the range of results

You can use the `setRequestedResultRange(int fromResult, int numberOfResult)` method to specify the range of returned results.

The **fromResult** value controls which ranked document your result set starts from. For example, a value of 0 means that you are requesting the first document in the query results.

The **numberOfResults** value controls how many results to return in the current page of results. The maximum is 500.

Setting category details

You can specify the required category detail level for query results by using the `setResultCategoriesDetailLevel(int detailLevel)` method.

The `setResultCategoriesDetailLevel(int detailLevel)` method is used if the categories attribute `RETURN_RESULT_CATEGORIES` is enabled. The default value is `RESULT_CATEGORIES_ALL`.

- `RESULT_CATEGORIES_ALL`: Each result category is returned with its complete path (starting at the root path) information.
- `RESULT_CATEGORIES_NO_PATH_TO_ROOT`: Each result category is returned without the full path information; that is, `ResultCategory.getPathFromRoot()` will return

null. Use the `setReturnedAttribute(RETURN_RESULT_CATEGORIES, false)` attribute to stop the retrieval of result categories completely.

Enabling site collapse

You can use the `setSiteCollapsingEnabled(boolean value)` method to specify whether the top results contain more than two results from the same Web site or data source.

For example, if a particular query returned 100 results from http://www.ibm.com and site collapsing was enabled, the `ResultSet` object contains only two of those results in the top results. The other results from that site appear only after results from other sites are listed.

To retrieve more results from that same site, use the samegroupas:*result URL* query syntax or re-issue the same query with the site http://www.ibm.com added to the query string. See "Query syntax" on page 19 for more information.

Sorting by relevance, date, numeric fields, or text fields

You can use the `setSortKey` method to specify a sort key to help you sort results.

Any field that is defined for the collection and declared as ″sortable″ (for text fields) or ″parametric″ (for numeric fields) can be named in the call to the `setSortKey` method. Textual keys are sorted lexicographically and numeric keys are sorted arithmetically.

The collating sequence (the order of characters in the alphabet for the purpose of sorting) is by default the one that is used by the search server. You can specify a different sequence by providing a locale name as a second argument to the `setSortKey` method. For example, the method `setSortKey("title", "de_AT")` sorts results by the value of their title field by using the alphabetic order that is common in German as used in Austria. Use the standard five character locale format xx_XX. For example, the locale for American English is en_US. The locale for Japanese is ja_JP.

Sorting order (ascending or descending) is specified by a call to the method `setSortOrder`. Two constants, SORT_ORDER_DESCENDING and SORT_ORDER_ASCENDING are defined in com.ibm.siapi.search.BaseQuery and can be used as arguments to the method. For example, the following method causes sorting to be done in ascending order.

```
setSortOrder(com.ibm.siapi.search.BaseQuery.SORT_ORDER_ASCENDING)
```

The default sort order is descending: The first results to be output are those at the top of the order. For example, the most relevant results appear at the top if they are sorted by relevance, the most recent results appear at the top if they are sorted by date, and so on.

Results whose sort key value is missing, undefined, or unavailable are sorted to the end of the results list regardless of their sort order.

Several reserved field names can be used as an argument to the `setSortKey` method to indicate sort by relevance, date, or no sort. These predefined values are defined in `com.ibm.siapi.search.BaseQuery`:

SORT_KEY_NONE
Specifies that results are not to be sorted.

SORT_KEY_DATE
Sorts results by date.

SORT_KEY_RELEVANCE
Sorts results by relevance. This is the default value.

Typically, when the collection is scanned for results, only a pool of the most relevant results is retained. If sorting is indicated, it applies only to the results in the pool, that is, the most relevant to the query. The maximum and default pool size is 500. It can be set to any smaller value by calling the `setSortPoolSize` method and specifying pool size. If the pool size is larger than 500, you receive an exception.

However, the search administrator can set up the system with a larger maximum or default pool.

If you want to sort all possible results (not only the most relevant), call the `setSortPoolSize` method with the value `SORT_ALL_RESULTS`, which is defined in `com.ibm.siapi.search.BaseQuery`. After calling this method, the system retains a pool of results that supersede all others in the collection when they are sorted by the values of the indicated field. Pool size is ignored if no sort or sort by date is requested.

The following restrictions apply to sorting:

- In sortable fields (text fields that can be used as sort keys) only the first 256 characters of the field are typically used when comparing two results for sorting. Documents whose sort key values identical up to the 256th position cannot be distinguished when they are sorted, and they might appear in the result stream in any order. The search administrator can change this value up to a maximum of 2000 characters.
- Unlike other text, sortable fields are used as is with no tokenization, normalization, or stop word removal.
- Sorting is incompatible with streaming: when the engine is used in streaming mode, results are output in the order in which they are encountered in the collection without any sorting.
- In a collection, if you define a field as numeric for some documents and textual for other documents, specifying that field as a sort key uses only the numeric values for sorting. Documents with a textual value in the field appear at the end of the result list with documents that do not define the field.

To indicate the results from a query should be sorted by a field, use the following methods:

```
BaseQuery.setSortKey(<field_name>) or BaseQuery.setSortKey
(<field_name><locale>)
BaseQuery.setSortOrder({SORT_ORDER_ASCENDING | SORT_ORDER_DESCENDING})
BaseQuery.setSortPoolSize({<int> | SORT_ALL_RESULTS})
```

Setting the sort order for results

You can use the `setSortOrder(int sortOrder)` method to specify a sort order.

Specify the sort order as SORT_ORDER_ASCENDING or SORT_ORDER_DESCENDING.

The sort order is ignored if the sort key is SORT_KEY_RELEVANCE or SORT_KEY_NONE.

Specifying the number of relevant results

You can control how many of the top relevant results will be sorted and returned in the result set by using the `setSortPoolSize(int sortPoolSize)` method.

Values for the number or returned results range from 1 to 500. The default sort pool size is 500. Any other values cause the search server to throw a `SiapiException` object.

The sort pool size is ignored if the `sortKey` is SORT_KEY_RELEVANCE or SORT_KEY_NONE.

To sort all possible results (not only the most relevant), call the `setSortPoolSize` method with the value `SORT_ALL_RESULTS`, which is defined in `com.ibm.siapi.search.BaseQuery`. After such a call, the system retains a pool of results that supersede all others in the collection when they are sorted by the values of the indicated field. The pool size is ignored if no sort or sort by date is requested.

Enabling spelling correction

You can specify whether suggested spelling corrections are to be provided with the search results,

Use `setSpellCorrectionEnabled(boolean enable)` method to specify that spelling corrections for terms in the query are to be included in the search results. Spelling correction is disabled by default.

Setting query expansion

You can set the synonym expansion mode for a query by using the `setSynonymExpansionMode (int mode)` method.

You can use one of the following modes:

- SYNONYM_EXPANSION_OFF: Pass this constant to the setSynonymExpansionMode method to prevent synonyms from being expanded even if the query contains the synonym operator.
- SYNONYM_EXPANSION_MANUAL: Pass this constant to the setSynonymExpansionMode method to expand synonyms only for the query terms that are affected by the synonym operator.
- SYNONYM_EXPANSION_AUTOMATIC: Pass this constant to the setSynonymExpansionMode method to do a best effort to expand all applicable query terms.

Determining query evaluation times and query timeouts

You can use `ResultSet` methods to see how much time it takes to evaluate queries and whether a query timeout occurred.

The `ResultSet.getQueryEvaluationTime` method returns the amount of time, in milliseconds, that it takes to evaluate queries.

The `ResultSet.isEvaluationTruncated` method can show whether a query timed out before it was completely processed.

Query syntax

You can refine search results by using specific characters in a query.

Simple query syntax characters

The following list describes the characters that you can use in search applications to refine query results.

Free style query syntax
Free style query syntax is used to describe queries that do not have an explicit interpretation and for which there is no default behavior defined. Beginning with OmniFind Enterprise Edition, Version 8.4, the default implementation for this type of query is to return documents only if they match all terms in the free style query.

Query: `computer software`

Result: This query returns documents that include the term *computer* and the term *software*, or something else depending on the semantics of the implementation.

~ (Prefix)
Precede a term with a tilde sign (~) to indicate that a match occurs anytime a document contains the word or one of its synonyms.

Query: `~fort`

Result: This query finds documents that include the term `fort` or one of its synonyms (such as `garrison` and `stronghold`).

~ (Postfix)
Follow a term with a tilde sign (~) to indicate that a match occurs anytime a document contains a term that has the same linguistic base form as the query term (also known as a lemma or stem).

Query: `apples~`

Result: This query finds documents that include the term *apples* or *apple* because apple is the base form of apples.

+ Precede a term with a plus sign (+) to indicate that a document must contain the term for a match to occur.

Query: `+computer +software`

Result: This query returns documents that include the term *computer* and the term *software*.

Beginning with OmniFind Enterprise Edition, Version 8.4, the plus sign is not needed because documents are included in the search results only if they match all terms in a free style query.

– Precede a term with a minus sign (-) to indicate that the term must be absent from a document for a match to occur.

Query: `computer -hardware`

Result: This query returns documents that include the term *computer* and not the term *hardware*.

= Precede a term with the equal sign (=) to indicate that the document must contain an exact match of the term for a match to occur. (Lemmatization is disabled.)

Query: `=apples`

Result: This query returns documents if and only if they include the plural term `apples`.

^

In versions of OmniFind Enterprise Edition that precede Version 8.4, you can use the circumflex sign (^) to indicate that a document must contain the term and at least one more term that is not preceded by a circumflex. Beginning with OmniFind Enterprise Edition, Version 8.4, the circumflex sign is not needed because documents are included in the search results only if they match all terms in a free style query.

Query: `cats dogs ^$language::en ^$doctype::html`

Result: This query returns HTML documents in English that contain the terms *cats* and *dogs*.

*

Place a wildcard character (*) anywhere in, before, or after a term or a field to indicate that the document can contain any word that matches any of the possible combinations. A term with a wildcard character is interpreted as equivalent to an OR of all its applicable expansions. Wildcard support applies the following rules:

- The set of expansions contain the maximal configured number of expansions. If there are more expansions in the index than the maximal number, those expansions are ignored. If some expansions of the wildcard term were ignored, the query result will indicate that.
- The set of expansions contains all terms in the index that can be obtained by replacing the wildcard characters with arbitrary sequences of characters.
- If wildcard character support is restricted to a set of fields, the set will contain only terms that appear in one of those fields. A term needs to appear in only one of the fields in at least one document in the index.
- If the term is a fielded term, wildcard character can appear only after the field specifier. If wildcard support is restricted to a set of fields, the field name of the wildcard term must be one of these fields. Otherwise, the term will have no expansions.
- If the wildcard term is not supported by the wildcard option that the collection is configured to use, the set of expansions is empty. However, the search runtime makes a best effort to return results for the query:
  - If wildcard characters are not supported by the collection, a wildcard term is interpreted as the same term without a wildcard character. For example, the query `to*y` returns all results for `toy`, but not for `tony`.
  - If only trailing wildcard characters are supported but the wildcard term contains a wildcard character in the middle, it is interpreted as the same term without the characters after the first wildcard character. For example, the query `to*y` returns all results that are returned by the query `to*`.
- Wildcard characters are supported only for plain text terms. Wildcard characters are not supported for XML element names, attribute names, or attribute values.
- A term that consists solely of a wildcard is not supported.
- Wildcard characters are supported within phrases.
- If the number of expansions for a wildcard term exceed the configured maximum number of expansions, the expansions that exceed that maximum are ignored by the query evaluation. In that case, the ResultSet object indicates that as follows:

- The `ResultSet` object's method `isEvaluationTruncated()` returns true. This does not uniquely identify the situation, because it will also return true if the evaluation was terminated early due to a timeout.
- A special property to the `ResultSet` object is set to true and is extractable by the `ResultSet.getProperty("WildCardTruncation")` object, which returns true. In case of truncation due to a timeout, the `isEvaluationTruncated()` object returns true, and the `ResultSet.getProperty("TimeOutTruncation")` object returns true.

Query: `app*`

Result: This query finds documents that include the terms *apple*, *apples*, *application*, and so on because these words begin with *app*.

Query: `DB2 info*`

Result: This query finds all documents that contain *DB2* followed by a word that begins with *info*.

Query: `title:tech*`

Result: This query finds the term *technology* if it appears in the field *title*.

Remember: To specify queries with wildcard characters, you must enable wildcard support when you configure search options for the collection in the enterprise search administration console.

" "

Use double quotation marks (") to indicate that a document must contain the exact phrase within the double quotation marks for a match to occur. Words inside phrases are never lemmatized.

You can also add wildcard characters (*) within phrases. The wildcard character (*) must be next to a letter or word. Standalone wildcard characters are not supported. Wildcard character support must be enabled in the enterprise search administration console.

Query: `"computer software programming"`

Result: This query finds documents that include the exact phrase `computer software programming`.

Phrases are designated as required by default. Hence the two queries `building "new york"` and `building +"new york"` are equivalent. Phrases can also be forbidden (-) and required but insufficient (^).

Query: `"app* pea*"`

Result: This query finds documents that include the terms *apples pears*, *appears peaceful*, *appreciate peas*, and so on because these words begin with *app* and *pea*. This query does not find documents with *apples and pears* or other such combinations. A standalone wildcard character (*) within a phrase is ignored, and the query `"apples * pears"` yields the same results as *apples pears*.

A query such as `"apples * pears"` does not match *apples and pears* or *apples or pears*, but it does match *apples pears*.

Restriction: Using double quotation marks for URL or e-mail address strings does not return appropriate results. To search for URL or e-mail strings such as `www.ibm.com` or `somebody@mycompany.com`, do not enclose the string in double quotation marks.

( ) Use parentheses ( ) to indicate that a document must contain one or more of the terms within the parentheses for a match to occur.

Do not use plus signs (+), minus signs (-), or circumflex (^) within the parentheses.

Use OR or a vertical bar (|) to separate the terms in parentheses.

Query: `+computer (hardware OR software)`

Query: `+computer (hardware | software)`

Result: Both of these queries find documents that include the term *computer* and at least one of the terms *hardware* or *software*.

An OR of terms can be either required (+), or required but insufficient (^), but not forbidden (-). This does not restrict the power of the query language: `-(dogs OR cats)` can be expressed by `-dogs -cats`.

An OR of terms is designated as required (+) by default. Therefore, the previous queries are equivalent to `+computer +(hardware | software)`.

Nested OR statements are not supported.

site:*text*

If you search a collection that contains Web content, use the `site` keyword to search a specific domain. For example, you can return all pages from a particular Web site.

Do not include the prefix `http://` in a site query.

Query: `+laptop site:www.ibm.com`

Result: This query finds all documents on the `www.ibm.com` domain that contain the word `laptop`.

url:*text*

If you search a collection that contains Web content, use the `url` keyword to find documents that contain specific words anywhere in the URL.

Query: `url:support`

Result: This query finds documents that have a URL that contains the value `support`, such as `http://www.ibm.com/support/fr/`.

link:*text*

If you search a collection that contains Web content, use the `link` keyword to find documents that contain at least one hypertext link to a specific Web page.

Query: `link:http://www.ibm.com/us`

Result: This query finds all documents that include one or more links to the page `http://www.ibm.com/us` .

*field***:***text*

If the documents in a collection include fields (or columns), and the collection administrator made those fields searchable by field name, you can query specific fields in the collection.

Query: `lastname:smith div:software`

Result: This query returns all documents about employees with the last name Smith (`lastname:smith`) who work for the Software division (`div:software`).

docid:*documentid*

Use the `docid` keyword to find documents that have a specific URI (or document ID). Typically, there is at most one document in a collection that matches a specific URI.

Query: `(docid:http://www.ibm.com/solutions/us/ OR docid:http://www.ibm.com/products/us/)`

Result: This query finds all documents with the URI `http://www.ibm.com/solutions/us/` or the URI `http://www.ibm.com/products/us/`.

samegroupas:*URI*

By default, enterprise search treats the URLs with the same host name as if they belong to the same group, and treats the news articles from the same thread as if they belong to the same group. For URIs from all other data sources, each URI forms its own group. However, with enterprise search, you can organize URIs that match specific prefixes into groups. For example, you can configure the following group definition:

```
http://mycompany.server1.com/hr/         hr
http://mycompany.server2.com/hr/         hr
http://mycompany.server3.com/hr/         hr
http://mycompany.server1.com/finance/    finance

file:///myfileserver1.com/db2/sales/      sale
file:///myfileserver1.com/websphere/sales/  sale
file:///myfileserver2.com/db2/sales/      sale
file:///myfileserver2.com/websphere/sales/  sale
```

In this example, all the URIs with the prefix http://mycompany.server1.com/hr/ or http://mycompany.server2.com/hr/ or http://mycompany.server3.com/hr/ belong to one group: *hr*. All URIs with the prefix http://mycompany.server1.com/finance/ belong to another group: *finance*. And all the URIs with prefix file:///myfileserver1.com/db2/sales/ or file:///myfileserver1.com/websphere/sales/ or file:///myfileserver2.com/db2/sales/ or file:///myfileserver2.com/websphere/sales/ belong to yet another group: *sale*. If file:///myfileserver2.com/websphere/sales/mypath/mydoc.txt is a URI in the collection, a query with the following search term will restrict the search to the URIs in the *sale* group:

```
samegroupas:file:///myfileserver2.com/websphere/sales/mypath/mydoc.txt
```

All results for this query will have one of the following prefixes:

```
file:///myfileserver1.com/db2/sales/
file:///myfileserver1.com/websphere/sales/
file:///myfileserver2.com/db2/sales/
file:///myfileserver2.com/websphere/sales/
```

Query: `samegroupas:http://www.ibm.com/solutions/us/`

Result: This query finds all documents with URIs, in this case URLs, that belong to the same group as `http://www.ibm.com/solutions/us/`.

*taxonomy_ID***::***category_ID*

If you search a collection that contains categories that are created for enterprise search, you can search for documents that belong to a specific category in a specific taxonomy.

To find the category ID, go to the following directory: `ES_NODE_ROOT/`*`col_xyz`*`.parserdriver/CategoryTree.xml`. The file CategoryTree.xml contains the category IDs.

Taxonomy IDs

rulebased

Use this taxonomy ID to search for documents that belong to a category that uses document content rules or document URI rules to categorize documents. For information about configuring rule-based categories, see the administration information for enterprise search.

Query: `rulebased::c1`

Result: This query returns documents that belong to a rule-based category ID named c1.

A *`taxonomy_ID`*`::`*`category_ID`* query term matches any documents that belong to the specified *`category_ID`* or any of its subcategories. This can be explicitly stated by preceding the taxonomy_ID with a tilda sign (~).

If you want the query to return documents that belong to the specified category but not return documents that belong to its subcategories, precede the *`taxonomy_ID::category_ID`* term with an equal sign (=), for example: `=rulebased::c1`.

scopes::*scope_name*

Use the scope name to search for documents that belong to a scope, which is a range of URIs in the index. For information about configuring scopes, see the administration information for enterprise search.

Query: `scopes::research ″computer science″`

Result: This query returns documents that belong to a scope named research that contain the phrase `computer science`.

$source::*source_type*

Use the `$source` keyword to find documents that come from a specific data source type. Source queries are useful in collections that contain documents from multiple sources.

To obtain a list of the available source types for a collection, call the getAvailableAttributeValues(Searchable ATTRIBUTE_SOURCE) method of that collection’s Searchable object.

Query: `$source::DB2 "computer science"`

Result: This query finds documents that were added to a collection by the DB2 crawler that contain the phrase `computer science`.

$language::*language_id*

Use the `$language` keyword to find documents that were written in a specific language.

To obtain a list of the available language IDs for a collection, call the getAvailableAttributeValues(Searchable.ATTRIBUTE_LANGUAGE) method of that collection’s Searchable object.

Query: `$language::en "computer science"`

Result: This query finds documents in English that contain the phrase `computer science`.

$doctype::*document_type*

Use the `$doctype` keyword to find documents that have a specific document format or MIME type.

To obtain a list of the available document types for a collection, call the getAvailableAttributeValues(Searchable.ATTRIBUTE_DOCTYPE) method of that collection's Searchable object.

Query: `$doctype::application/pdf "computer science"`

Result: This query finds Portable Document Format (PDF) documents that contain the phrase `computer science`.

*#field***::=***value*
Use parametric constraint syntax to find documents that have a numeric field with a value equal to the specified number.

Query: `#price::=1700 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value equal to 1700.

*#field***::>***value*
Use parametric constraint syntax to find documents that have a numeric field with a value greater than the specified number.

Query: `#price::>1700 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value greater than 1700.

*#field***::<***value*
Use parametric constraint syntax to find documents that have a numeric field with a value less than the specified number.

Query: `#price::<1700 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value less than 1700.

*#field***::>=***value*
Use parametric constraint syntax to find documents that have a numeric field with a value greater than or equal to the specified number.

Query: `#price::>=1700 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value greater than or equal to 1700.

*#field***::<=***value*
Use parametric constraint syntax to find documents that have a numeric field with a value less than or equal to the specified number.

Query: `#price::<=1700 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value less than or equal to 1700.

*#field***::>***value1***<***value2*
Use parametric constraint syntax to find documents that have a numeric field with a value that falls between a range of specified numbers.

Query: `#price::>1700<3900 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value greater than 1700 and less than 3900.

#field*::>=*value1*<=*value2
Use parametric constraint syntax to find documents that have a numeric field with a value that matches or falls between a range of specified numbers.

Query: `#price::>=1700<=3900 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value greater than or equal to 1700 and less than or equal to 3900.

#field*::>*value1*<=*value2
Use parametric constraint syntax to find documents that have a numeric field with a value that matches the criteria in the specified range of numbers.

Query: `#price::>1700<=3900 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value greater than 1700 and less than or equal to 3900.

#field*::>=*value1*<*value2
Use parametric constraint syntax to find documents that have a numeric field with a value that matches the criteria in the specified range of numbers.

Query: `#price::>=1700<3900 laptop`

Result: This query finds documents that contain the term `laptop` and a `price` field with a value greater than or equal to 1700 and less than 3900.

field_name*:=*my_content
***field_name*:=“*multiple words*”**
A complete match query for a field returns documents that exactly match only the words after the equal sign. These queries do not return documents unless *my_content* or *"multiple words"* is the only content for the field. The content that is specified in the query can be a single word or a phrase.

With complete match support enabled, you can query the entire content of a field, receive only documents that contain the exact and complete content of that field and avoid receiving documents that have extraneous content. You can enforce lemmatization or synonyms by using the ~ (tilde) sign before or after a word. You can also use a wildcard character in a complete match query.

For each field in the collection, the search administrator must enable support for complete match.

Query: `author:=Luther`

Result: This query returns documents with an author field with the exact content `Luther`. A document with an author field with the value such as `Martin Luther King` or any other word or phrase is not returned.

Query: `author:="Martin Luther"`

Result: This query returns documents with an author field with the exact content `Martin Luther`. A document with an author field with the value `Martin Luther King` is not returned.

Query: `author:="Mar* Luther"`

Result: This query with the wildcard character returns documents with the author field such as `Mark Luther`, `Martin Luther`, `Marvolo Luther`, and so on.

ACL constraints: (*security_tokens***)**
For security, you cannot specify access control constraints in the query string. Use the `setACLConstraints(String aclConstraints)` method of the `Query` interface to specify access control constraints for the query. You can specify parentheses, plus signs (+), minus signs (-),circumflexes (^), and an XML security context string in the ACL constraints string (@SecurityContext::'securityContext'). For information about the `securityContext` string syntax, see the Javadoc documentation that describes the `setACLContstraints` method. The symbols have the same meaning as described in the previous syntax descriptions.

ACL constraints string in setACLConstraints method: `(michelle_c | dev_group)`

ACL constraints string in setACLConstraints method: `michelle_c @SecurityContext::'securityContext'`

Query: `thinkpad`

Result: This query finds documents that include the term `thinkpad` and the security tokens `michelle_c` or `dev_group` in the first case, and `michelle_c` and the specified security context constraints in the second case.

Query syntax characters for opaque terms

With enterprise search, you can also create query syntax for two types of opaque terms. With opaque terms, you can allow parts of the query to be expressed in other languages, such as XML Fragment and XPath. XML Fragment and XPath are two types of XML query languages. XML Fragment can also be used to query UIMA structures. The sign for an opaque term is expressed with *@xmlf2::* (XML fragment) or *@xmlxp::* (XPath query). The XML fragment or the XPath query is enclosed in single quotation marks (' ').

The expression *xmlf2* is used for XML fragments, and *xmlxp* is used for XPath terms. An opaque term has the following syntax: @syntax_name::'value'. The expression starts with the @ sign, followed by the syntax name (xmlf2 or xmlxp), two colons (::), and a value that is enclosed in single quotation marks (' '). The value parameter is sometimes preceded by -, +, or ^. If you need to use a single quotation mark in the value section of the expression, escape the single quotation by using a backslash (\), for example, \'.

For negative terms, use a minus sign (–) before the @ symbol, for example, `-@xmlf2::'<person>michelle</person>'`. However, enterprise search does not accept negative unique query terms. The query `-@xmlf2::'<person>michelle</person>'` does not return results. To get results, use one positive term in the query, for example, `documentation -@xmlf2::'<person>michelle</person>'`.

@xmlf2::'<tag1> text1 </tag1>'
Use the @xmlf2:: prefix and enclose the query in single quotation marks to indicate a fragment query as a new search and index API opaque term.

Query: `@xmlf2::'<title>"Data Structures"</title>'`

Result: This query finds documents that contain the phrase *Data Structures* within the span of an indexed annotation called *title*.

@xmlf2::<tag1><.depth value="$number"><tag2> ... </tag2></.depth></tag1>
@xmlf2::<tag1><.depth value='$number'><tag2> ... </tag2></.depth></tag1>

The first query uses double quotation marks. The second query uses single quotation marks. However, each query returns the same results. This query syntax looks for occurrences of tag2 exactly $number levels under tag1.

$number is a positive integer. You can use single quotation marks (' ') or double quotation marks (" ") around the numerical value. This query syntax is not applicable to Unstructured Information Management Architecture (UIMA).

Query: (This query should appear on one line.)

```
@xmlf2::'<author>Albert Camus<.depth value='1'>
<publisher>Carey Press</publisher></.depth></author>'
```

Result: This query finds documents of the publisher one level under the author. A document with the following XML elements

```
<author>Albert Camus
  <ISBN>002-12345</ISBN>
  <country>USA
    <publisher>Carey Press</publisher>
  </country>
</author>
```

will not be returned with the example query because the publisher (`<publisher>`) element occurs two levels under the author (`<author>`) element.

@xmlf2::'<tag1><@tag1> ... </@tag1></tag1>'

You can distinguish between elements and attributes. Attributes are written either explicitly within the element or as subelements with a leading @ sign. The @ sign enables you to distinguish between elements and attributes that might have the same name.

You can define words and phrases within attributes, which is the same as the normal terms of the query. However, you can write expressions only of words and phrases, not of tags. These words or phrases support the same features as the normal terms of the query.

Query: `@xmlf2::'<author country="USA"></author>'`

Query: `@xmlf2::'<author><@country>USA</@country></author>'`

Result: This query finds documents where the author originates from the USA.

Query:

```
@xmlf2::'<author><@country>USA</@country>
<firstName>Michelle</firstName>
<lastName>Ropelatto</lastName></author>'
```

Result: This query finds documents where the author name is Michelle Ropelatto and is from the USA.

@xmlf2::'+text1 ... +text2 -text3 ... -text4 text5'

Use a plus sign (+) or a minus sign (-) as prefixes to words or phrases (always between quotation marks (" ")). At each query level, whether for the text or the tag name, "+" means that the terms must appear; "-" means that the terms should not appear and others are optional and contribute only to ranking. If no "+" terms exist, then at least one of the optional terms must appear. The data under elements creates a new nested query level.

Query: `@xmlf2::'+"Graph Theory" -network'`

Result: This query finds documents that contain the phrase `Graph Theory`, and do not contain the term network.

Query:

`@xmlf2::'<book><author>hemingway</author> -<title>old man</title></book>`

Result: This query finds documents that contain a book by Hemingway but not the book *The Old Man and the Sea*.

@xmlf2::'<tag1> <.or> ... </.or> <.and> ... </.and> </tag1>'
Use Boolean syntax for AND (<.and>) and OR (<.or>) expressions in a query.

Query: `@xmlf2::'<book><.or><author>Sylvia Plath</author><title>XML -Microsoft</title></.or></book>'`

Result: This query finds documents that specify a book whose author is Sylvia Plath or where the title of the book includes the word XML but not Microsoft®.

@xmlf2::'<annotation1+annotation2> ... </annotation1+annotation2>'
You can express the concatenation of consecutive annotations in a fragment query by using the plus sign (+) between the start and end tags of the element. The consecutive annotations must overlap by at least one word (they must intersect). The concatenation of two or more overlapping annotations is a new virtual annotation that spans the sum of the text spanned by the annotations.

Query: `@xmlf2::'<Report+HoldsDuring> +Pakistan +March +Reuters</Report+HoldsDuring>'`

Result: This query finds documents from Reuters about events in Pakistan in March that are contained in the concatenated annotation formed by the "Report" and "HoldsDuring" annotations.

@xmlf2::'<annotation1*annotation2> ... </annotation1*annotation2>'
You can express the intersection of annotations in a fragment query using the asterisk sign (*) between the start and end tags of an element. The intersection of two or more overlapping annotations is a new virtual annotation that spans just the text that is covered by the intersection of the overlapping annotations.

Query: `@xmlf2::'<Inhibits* Activates>Aspirin</Inhibits*Activates>'`

Result: This query finds documents in which Aspirin occurs in both the 'Inhibits' and 'Activates' annotations.

@xmlxp::'/tag1/@tag1'
You can distinguish between elements (XML start and end tags) and attributes. Attributes are written explicitly with a leading @ sign. The @ sign enables you to distinguish between elements and attributes that might have the same name. Concatenations and intersections are applicable only to UIMA documents, and not to pure XML documents, where spands do not cross over by definition.

Query: `@xmlxp::'/author[@country="USA"]'`

Result: This query finds documents in which *USA* is included in the character string that is the value of the attribute *country* that is associated with *author*.

@xmlxp::'/tag1[tag2 or tag3 and tag4]'
Use full Boolean to express AND and OR scope in an XPath query.

Query: `@xmlxp::'book[author ftcontains("Jose Perez") or title ftcontains("XML -Microsoft")]'`

Result: This query finds documents that specify a book whose author is Jose Perez or where the title of the book includes the word XML, but not Microsoft.

@xmlxp::'tag1//tag2/tag3'
You can distinguish between descendent nodes (//) and child nodes (/).

Query: `@xmlxp::'/books//book/name'`

Result: This query finds documents that specify a book element as a descendant of a books element and that specify a name element as a direct child of the book.

@xmlxp::'tag1/.../tagn'
Use the @xmlxp:: prefix and enclose the query in single quotation marks to indicate an XPath query as an search and index API opaque term.

Query: `@xmlxp::'books[booktitle ftcontains("Data Structures")]'`

Result: This query finds documents that contain the phrase "Data Structures" within the span of an indexed annotation called "title."

@xmlf2::'=<*tag_name*> content </*tag_name*>'
Use the syntax `@xmlf2::'=` to issue complete match queries in native XML collections. Complete match queries return documents that have a tag named <*tag_name*> with some content. The content specified in the query can be a single word or a phrase. You can enforce lemmatization or synonyms by using the ~ (tilde) sign before or after a word. You can also use a wildcard character in a complete match query.

XPath queries are not supported for the complete match capability.

Query: `@xmlf2::'=<author>Martin Luther</author>'`

Result: This query returns documents with an author element with the exact content `Martin Luther`. A document with author element with the value `Martin Luther King` is not returned.

Related reference
"Controlling query behavior" on page 11

Query syntax structure

A search and index API query is a list of space-separated query terms that follow a specific structure.

A query has the following structure:

```
<space>*{Query_term <space>+}* Query_term
```

Query terms can be one of the following types:
- Word
- Phrase
- AttributeConstraint
- CategoryConstraint
- RangeConstraint
- OrTerm

- OpaqueTerm

Appearance modifiers

Appearance modifiers control whether a query term is:
- Mandatory: it must appear within result documents.
- Forbidden: it must not appear within result documents.
- Mandatory but insufficient: documents that contain only insufficient terms do not qualify as search results.

The semantics correspond to the BaseQueryTerm object's constants of the search and index API: `Appearance_Modifier = { + | –| ^ }`
- + denotes a mandatory term
- – denotes a forbidden term
- ^ denotes a mandatory term that is not sufficient for a document to qualify as a search result

OR terms cannot be denoted as forbidden: `OR_Appearance_Modifier = { + | ^ }`

Match type modifiers

Prematch type modifiers appear just before the word that they modify: `PreMatch_Type = { = | ~ }`
- = denotes that the word should be matched as is, that is, it should not be stemmed or lemmatized, and that the search should not be expanded to include synonyms of the word
- ~ denotes that the search should be expanded to include synonyms of the word

Postmatch modifiers appear directly after the word that they modify: `PostMatch_Type = { * | ~ }`
- * matches words having the indicated prefix
- ~ matches words that share the same base form, for example, stem or lemma with this word

By default, words that are explicitly modified by an appearance modifier but not by a match type use exact-match ("as is") semantics.

Fielded search notation

A fielded search notation, or field name (token), is immediately followed by a colon, that is, no space between the field name and the colon: `Field = field_name:`

OR terms

An OR term is comprised of a sequence of ORable terms, separated by spaces and an OR-SIGN, enclosed within parentheses. All query term types except "OrTerm" and "OpaqueTerm" qualify as ORable terms. Parentheses surround the OR expression, in which terms are separated from each other by three mandatory sequences:
- One or more spaces
- Either a vertical bar '`|`' or the upper case word 'OR' (both notations are allowed)
- One or more spaces

Semantically, at least one of the OR-ed terms must appear in documents that qualify as search results.

```
ORable_term = Query_term \ { OrTerm, OpaqueTerm }

OR-SIGN = | | OR

ORable_query = <space> * { ORable_term <space> + OR-SIGN <space> +}*
ORable_term

OrTerm = OR_Appearance_Modifier? ( ORable_query )
```

If no OR_Appearance_Modifier is given, a + is implicitly assumed. The individual terms of the ORable_query cannot have appearance modifiers of their own.

Words and phrases

```
Word = PreMatch_Type? Appearance_Modifier? Field? value  |
Appearance_Modifier? Field? value PostMatch?
```

- A word (or fielded word), possibly with a single match type indicator (at the beginning or the end), and possibly with an appearance modifier
- If neither a match type indicator nor prefix indicator are given, it is implementation dependent which form is searched
- If no appearance modifier is given, it is implementation dependent whether the results must contain the term
- The value can contain the wildcard symbol '*' anywhere; however, at least one non-wildcard character must exist in the value.

The following phrase should be expressed on one line:

```
Phrase = Appearance_Modifier? Field?"
<space>*{value<space>+}* value <space>*"
```

- A non-empty sequence of space-separated values inside quotation marks. The field and appearance modifier are both optional.
- If no appearance modifier is given, a + is implicitly assumed.
- Each value inside the phrase is searched as is, that is, with exact-match semantics.

Attribute, category, and range constraints

```
Attribute_Name: { language | source |  doctype }
AttributeConstraint = $Attribute_Name::value
```

The $ sign is followed by an attribute name, which followed by two colons and a value. If no Appearance_Modifier is given, a + is implicitly assumed.

The following category constraint should be expressed on one line:

```
CategoryConstraint =
PreMatch_Type?Appearance_Modifier?taxonomy_id::category_id
```

A taxonomy ID is followed by two colons and a category ID:

- Match_Type = restricts the documents to be members of the given category, while ~ means that documents can also belong to descendents (subcategories) of the given category.
- If no Match_Type is given, ~ is implicitly assumed.
- If no Appearance_Modifier is given, a + is implicitly assumed.

The parametric field must be greater than (or equal to in the second case) the double value:

```
Grelation = > double_value | >= double_value
```

The parametric field must be less than (or equal to, in the second case) the double value:

```
Lrelation = < double_value | <= double_value
```

The # character is followed by the field name, two colons, and at least one relation (or =). The Appearance_Modifier can be either + or ^ (- is not allowed). If no Appearance_Modifier is given, a + is implicitly assumed:

```
RangeConstraint = Appearance_Modifier?# field :: Grelation Lrelation? |
             Appearance_Modifier?# field :: Grelation? Lrelation  |
             Appearance_Modifier?# field :: =double_value
```

Opaque terms

An @ sign is followed by some syntax name, two colons, and a value enclosed in single quotation marks. The opaque term can be preceded by an appearance modifier. If a single quote is needed in the value part, it should be escaped by \, as in \':

```
OpaqueTerm = Appearance_Modifier?@ syntax_name :: ' value '
```

For the semantics of opaque terms, the search and index APIs:

- Do not attempt to parse the value inside the single quotation marks; rather, that string will be passed as-is to a parser that corresponds to the syntax_name.
- Does not define which external query languages should be supported by implementations.
- Does not define how many opaque terms can exist inside a query, and how they interact with the rest of the terms. All this is implementation defined. It is assumed that in most cases, a query either consists solely of an opaque term, or does not contain such terms at all.

Tokens, field names, and values in queries

Tokens, field names, and values have the following rules:

- Any sequence of characters without any of the special characters is a token.
- The characters = (" have only special meaning if they are preceded by a space or at the beginning of the query string. Thus, these characters can exist inside tokens, but they cannot exist at the beginning of a token.
- An exception to the previous rule is that a '(' can begin a token inside an OrTerm because those cannot be nested and so '(' has no special meaning there.
- The characters + - ^ have special meaning only if they are preceded by a space, by one of = ~ (, or at the beginning of the query string.
- The colon has meaning only as a separator between a field/constraint-type and a value. The colon is considered a regular character in all other cases.
- The character) has special meaning only inside an OrTerm, but outside of a phrase inside the OrTerm. There, it will terminate the OrTerm. In all other cases, it is considered a regular character.
- The character * has special meaning only for values; that is, wildcard characters are not applied to field names.
- The sequences <,<=,>,>= have special meaning only within a range constraint.

- All special characters except " are considered regular characters inside a phrase: they lose their special functions inside phrases. The " ends the phrase. This rule trumps all previous rules.
- Wildcard characters are allowed inside phrases

As a general, if one of the special characters has no meaning in a certain setting, it will be considered as part of the token/value.

The behavior of the query parser is undefined for nonconforming strings. In some cases, the parser implicitly overcomes problems, such as ending phrases that are not terminated, and in some cases it does not overcome such problems.

ACL expression syntax

The syntax of ACL expressions is a subset of the full query syntax. Basically, it consists of words, OR expressions over several words, and opaque terms.

```
ACL_Expression = <space>* {ACL_term<space>+}* ACL_term
```

```
ACL_term = {  ACL_Value  |
            ACL_OrTerm   |
            Security_OpaqueTerm    }
```

```
ACL_Value = Appearance_Modifier? value
```

- A value, possibly with an appearance modifier
- If no appearance modifier is given, ^ is assumed.

The ACL_OrTerm should be expressed on one line:

```
ACL_OrTerm = {+|^}? (<space>*{value <space>+
OR-SIGN<space>+}* value <space>*)
```

The sequence of values is separated by spaces and an OR-SIGN and enclosed inside parentheses. The OrTerm optionally has an appearance modifier, either + or ^ (- is not allowed) If no Appearance_Modifier is given, ^ is implicitly assumed. The individual values inside the OrTerm cannot have appearance modifiers of their own.

Security opaque terms

```
Security_OpaqueTerm = Appearance_Modifier?@ syntax_name :: ' value '
```

An @ sign is followed by some syntax name, two colons, and a value enclosed in single quotation marks. The opaque term can be preceded by an appearance modifier. If a single quotation mark is needed in the value, it should be escaped by \, as in \'.

The semantic disclaimers that were specified with respect to opaque terms in query strings apply here.

The behavior of the ACL expression parser is undefined for nonconforming strings. In some cases, the parser implicitly overcomes problems, such as ending OR-terms that are not terminated, and in some cases it does not overcome such problems.

By default, all terms are assumed to be required but insufficient, as if qualified by '^'. Although you can qualify an ACL_term by `a +`, it does not seem to match the use of ACLs as filters.

The expressiveness of this syntax is broader than simple document-level security. First, it allows forbidden tokens, for example, "do not return documents that are viewable with this ACL". Second, the ability to include several OrTerms allows this syntax to support multiple-level security:

```
(server_ACL1 | server_ACL2) (group_ACL1 | group_ACL2)
(user_ACL1 | user_ACL2)
```

Related reference

"Controlling query behavior" on page 11

Java classes for showing top results

You can create a custom Java class to show the top search results, including results from non-enterprise search sources, in any HTML format.

The product includes two sample Java classes that are available as part of the ESSearchApplication sample code after you install the product:

- The `DynamicMostRecentDocuments` class analyzes the current set of search results according to the frequency with which the analyzed values occur and shows the top search results as an unordered list that is sorted by date. Only the document titles and dates are shown.
- The `DogearSearchResults` class analyzes results and provides users with a list of bookmarks from Lotus® Connections Dogear.

You can create Java classes for many other purposes. Examples of some of the ways that you might use the capability include:

- Display results from Del.icio.us for the same query. Del.icio.us is a service that is similar to Dogear that allows users to bookmark Web pages and share bookmarks with others.
- Display results from Google for the same query.
- Display a generic company message, a message about planned system maintenance, or a welcome page.
- Display an activities list for the logged on user.

When you create a Java class for analyzing the top results, you must update the configuration file for the search application to specify options for how the results are to be displayed. How you specify these options depends on how you deploy the search application:

- If your search application runs as a stand-alone application, you can configure top result options by using the Search Application Customizer or by editing the `config.properties` file for the search application.
- If your search application runs as a portlet within WebSphere Portal, you must edit the properties and configure the Portlet instance with the WebSphere Portal administration interface. You cannot use the Search Application Customizer to configure options for top result analysis.

Related tasks

Analyzing top results

Editing the sample search application properties

Related reference

Search application properties

Search and index API federators

Use a federator to issue a federated search request across a set of heterogeneous searchable collections and get a unified document result set.

Search federators are intermediary components that exist between the requestors of service and the agents that perform that service. They are coordinate resources to manage the multitude of searches that are generated from a single request.

Enterprise search provides the following types of search and index API federators:
- Local federator
- Remote federator

Search federators are search and index API searchable objects. Multiple-level federation is allowed, but too many levels of federation will decrease search performance.

The local and remote federators can federate over collections that are created with OmniFind Enterprise Edition or collections that are created with another product. You can federate over collections that are not created with OmniFind Enterprise Edition if those collections use lightweight directory access protocol (LDAP) or Java database connectivity (JDBC).

To create an LDAP or JDBC searchable object, the application creates an `AdminService` object by passing a fully qualified LDAP or JDBC `AdminService` object class path. The `createCollection` method is used to create an LDAP or JDBC collection. The LDAP or JDBC configuration information is passed through a XML configuration file. After LDAP or JDBC collections are created, you can retrieve the searchable objects through the `Service` interface and use those searchable objects directly or through local or remote federators.

Local federator

A local federator federates from the client over a set of searchable objects.

A local federator is created by using the `createLocalFederator` method from the SIAPI `SearchFactory` class. The set of searchable collections on which the query is to be run is specified when the federator is created. A subset of searchable objects can also be specified during search calls.

Before you can create a local federator, you must create or retrieve searchable objects by using a search and index API SearchFactory. The searchable object that is passed to the local federator must be ready for search without any additional information. The local federator uses the searchable object to issue a federated search request. To complete this request, the local federator environment must have all the necessary software components for using various searchable objects.

The following code sample shows how to create a `LocalFederator` object and issue a search request:

```
Searchable[] finalSearchables;

// create searchables

// create a query and set query options
Query query = searchFactory.createQuery(queryString);
query.setRequestedResultRange(0, 100),
query.setQueryLanguage("en_US");
query.setSpellCorrectionEnabled(true);
query.setPredefinedResultsEnabled(true);

// create the local federator and call search
LocalFederator federator =
 factory.createLocalFederator(finalSearchables);
ResultSet rs = federator.search(query);
```

Remote federator

A remote federator federates from a server over a set of searchable objects.

A remote federator is run on the server and consumes server resources. A remote federator requires an extra step in which input collection IDs are mapped to the matching searchable object.

The remote federator is created by using the search and index API `AdminService` interface. During the construction of the `RemoteFederator`, the set of collection IDs must be passed. The collection IDs are mapped to SIAPI searchable objects internally by the `RemoteFederator`. The remote federator environment does not require any searchable related software components other than a small proxy that enables the remote federator to be accessible.

Each search application will have its own federator, so the federator ID is the same value as the ApplicationInfo ID value.

The following code sample shows how to create a `RemotelFederator` object and issue a search request:

```
// get collection IDs
String[] collectionIDs;

// create a query object
Query query = searchFactory.createQuery(queryString);
query.setRequestedResultRange(0, 100),
query.setQueryLanguage("en_US");
query.setSpellCorrectionEnabled(true);
query.setPredefinedResultsEnabled(true);

// create a remote federator
RemoteFederator federator = getFederator(appinfo, appinfo.getID());
// search
ResultSet rs = federator.search(query);
```

Retrieving targeted XML elements

You can specify that a returned document must be accompanied by a result field.

In the opaque term that specifies the semantic search, you can prepend a pound or hash sign (#) to one XML element (or annotation) in the xmlf2 query term. This result field enumerates all the occurrences of an Unstructured Information Management Architecture (UIMA) annotation that is designated in the XML query term. These enumerated annotation occurrences are within the returned document, and each of them makes a part of an occurrence of the whole XML query term in the document.

The XML element is designated as the targeted XML element whose occurrences are to be enumerated. When the semantic search is expressed by XPath, then by definition of XPath, the deepest element that is not inside the bracketed phrase [..] and not inside a predicate is the target element.

For example, the query `<book language=en> <#author> </#author> </book>`, or the equivalent query `<book language=en> <#author/> </book>`, returns documents that include at least one occurrence of the annotation `book` that has the attribute `language=en` and includes within its span an occurrence of the annotation `author`. The query also returns the enumeration of all the occurrences of the tag `<author>` that appear within the occurrence of the tag `<book>` that has the attribute `language=en`.

Each occurrence is enumerated by its unique ID. The UIMA annotators assign a unique ID to each annotation that they generate. XML elements that are part of the raw document rather than annotations that are generated by UIMA annotators do not have unique IDs, and they are not enumerated in that result field. If the summary field of the retrieved document includes text that is covered in the document by an enumerated occurrence, that text is highlighted.

The following occurrences of the tag `<author>` in the retrieved document will not be enumerated:

- An occurrence of the tag `<author>` within the span of the tag `<journal>`
- An occurrence of the tag `<author>` within the span of the tag `<book>` that has the attribute `language=ge`
- An occurrence of the tag `<author>` within the span of the tag `<book>` that does not have the attribute `language`
- An occurrence of the tag `<author>` that is part of an XML document, that is the tag `<author>` is part of the raw document rather than a generated annotation

The search application can access the enumeration of the occurrences of the target element through the `TargetElement` property of the `Result` object, for example, `Result.getProperty("TargetElement")`. The returned value of that property is a string of integers that are separated by spaces. Each integer is an ID of a single occurrence of the target element.

The actual target elements that correspond to these integer values cannot be retrieved by the API. If an application must access those elements, it must create its own mapping table during parsing. For example, you can create a common analysis for relational database mapping.

Fetching search results

The `fetch` API enables you to obtain the content of documents returned in the search results.

The `fetch` API enables users to view content by clicking documents in the search results. This API is especially useful for data sources that do not return a clickable URI, such as documents from DB2, DB2 Content Manager, and file system sources.

The `fetch` API uses client libraries that are installed when OmniFind Enterprise Edition is installed. In a multiple server installation, the libraries are installed on the crawler server. No additional application development work is required to take advantage of this API because the API is provided with the `esapi.jar` file.

To fetch certain types of documents, an enterprise search administrator must specify the "document content" option when the crawler is configured. The following discussion summarizes the requirements for the various crawler types:

DB2 crawler
: A document content field (column) must be specified when the crawler is configured to crawl a DB2 database.

Content Edition and DB2 Content Manager crawlers
: A document content field must be specified when a crawler is configured to crawl these types of data sources. Data sources that contain only metadata are not supported.

UNIX file system and Windows file system crawlers
: The `fetch` API can retrieve the content of file system documents with no special configuration by an enterprise search administrator.

All other crawlers
: For other types of crawlers, a clickable URI is returned by using the `getDocumentURI` method. The `fetch` API is not used to retrieve these types of documents.

The `fetch` API supports security at the search server level (through WebSphere global security), collection level (through application IDs), and at the document level (through indexed access controls and current user validation). The security policy relies on the security settings in the search application. If the search application returns a document in the list of search results, the `fetch` API will retrieve the document content when the user clicks the document.

A sample program, FetchSearchExample, is provided in the `ES_INSTALL_ROOT/samples/siapi` directory. Javadoc documentation is provided in the `ES_INSTALL_ROOT/docs/api/fetch` directory.

Application security

The search and index APIs communicate remotely through HTTP to the ESSearchServer enterprise application that is installed on each WebSphere Application Server search server when OmniFind Enterprise Edition installed.

You must enable WebSphere Application Server global security to secure remote communications. When the application issues remote search and index API requests that must be secure, you must set the user name and password on the Service classes with a valid user name that is stored in the enterprise user registry that is used by WebSphere for authentication. Any requests that do not contain valid user names and passwords are rejected. You can add user names and passwords in the WebSphere administrative console.

In an enterprise search application, the Properties object is passed in the call to the `getSearchService` method or `getBrowseService` method. The Properties object specifies property names called `username` and `password` for WebSphere.

Enterprise search supports HTTP basic authentication. Enterprise search also supports HTTPS secure socket layer (SSL) version 3 proxy servers and proxy servers that require a user name and password for basic authentication.

Applications and collections must have IDs. For applications that need to access specific collections, the collection ID must be associated with the application ID. You can grant access for applications to access specific collections in the enterprise search administration console.

Document-level security

To support prefiltering and post-filtering of search results, the search request must provide a user's security context by using the `setACLConstraints` method on the `Query` object.

The user's security context is provided as an XML string as part of an opaque query term, for example:

```
Query q = factory.createQuery("IBM"); q.setACLConstraints
("@SecurityContext::'<User's Security Context XML string>'"
```

You can create the user's security context XML string in two ways:

- By using the identity management API to programmatically create the XML string

  Use this method if you are building applications with OmniFind Enterprise Edition, version 8.4 or later.
- By using Java String classes to create the XML string

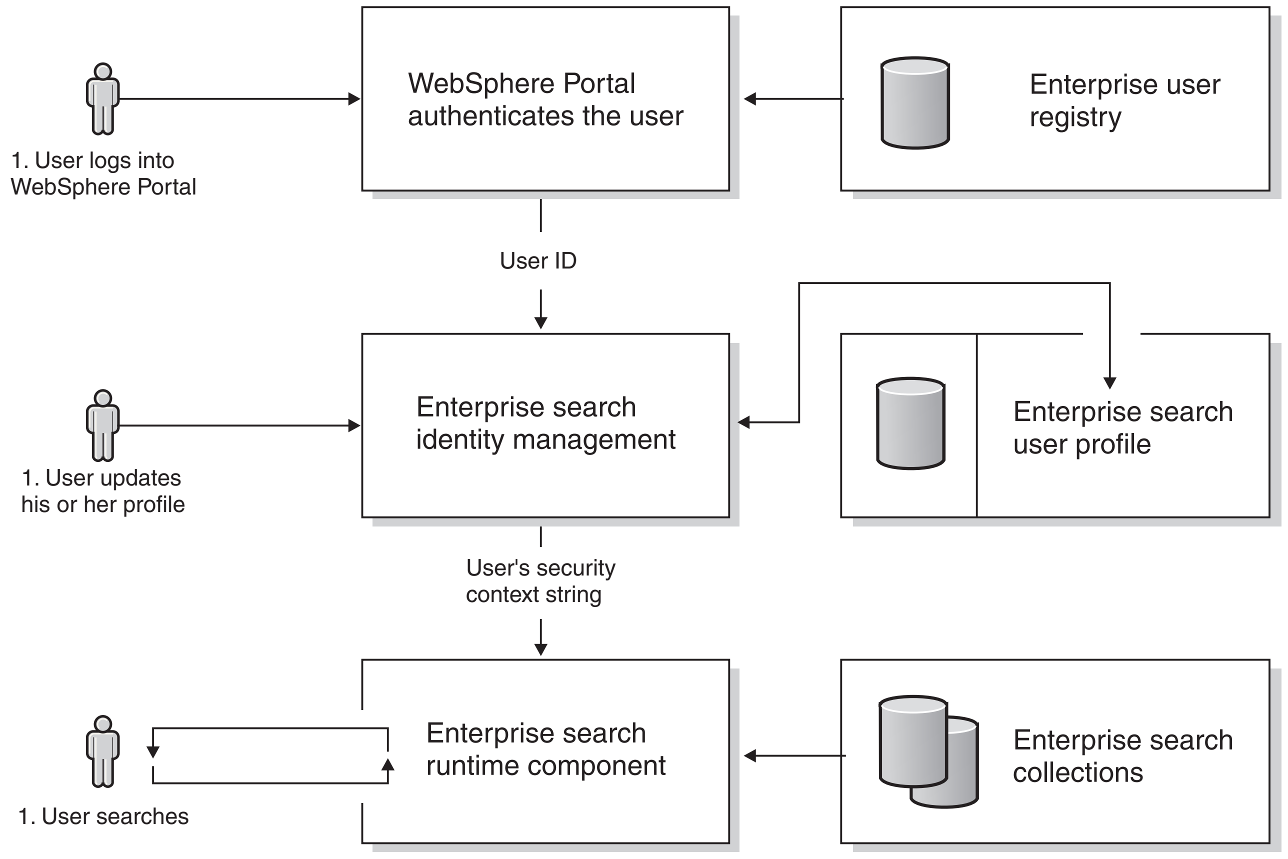
  Use this method only if you cannot build applications with the identity management API.

Identity management for single sign-on security

You can use the identity management APIs of enterprise search to create a single sign-on system that manages the multiple identities of users and to automatically generate the security context strings of users. IDs can be reused on subsequent searches without users logging on multiple times.

How the enterprise search identity management component works

With the identity management Java APIs, you can create an application to manage the security credentials of your users. The following graphic shows how users log in to a system such as WebSphere Portal and authenticate with the registry.



1. User logs into WebSphere Portal
WebSphere Portal authenticates the user
Enterprise user registry
User ID
1. User updates his or her profile
Enterprise search identity management
Enterprise search user profile
User's security context string
1. User searches
Enterprise search runtime component
Enterprise search collections

Figure 4. How users log in to WebSphere Portal or other systems

When users attempt to access enterprise search, the identity management component repeats the process of authenticating those users.

Sample code and Java APIs

You can access a sample Java program, the source code for the ESSearchApplication application, and Javadoc documentation for the identity management in the following locations:

IdentityManagementExample.java
: A standalone sample program that is available in the `ES_INSTALL_ROOT/samples/siapi` directory. You can build this code by running the ANT command.

ESSearchApplication
: The source code for the J2EE Apache Struts-based Web application that is installed by enterprise search. The source code is available in the `ES_INSTALL_ROOT/samples/ESSearchApplication` directory. You can build this code by running the ANT command.

Javadoc documentation
: Provides descriptions of the available APIs to build identity management into your search applications. The Javadoc documentation is in the `ES_INSTALL_ROOT/docs/api/imc` directory.

Running the sample application

To run the Java sample program, make sure that you have the following JAR files in your class path:

- `esapi.jar`
- `siapi.jar`
- `es.security.jar`
- `es.oss.jar`

To run the sample program, enter the following command on a single command line.

Windows

```
java –classpath $ES_INSTALL_ROOT\lib\esapi.jar;$ES_INSTALL_ROOT\lib\
siapi.jar;$ES_INSTALL_ROOT\lib\es.security.jar;.
IdentityManagementExample
```

AIX, Linux, or Solaris

```
java –classpath $ES_INSTALL_ROOT/lib/esapi.jar:$ES_INSTALL_ROOT/lib/
siapi.jar:$ES_INSTALL_ROOT/lib/es.security.jar:.
IdentityManagementExample
```

Application compatibility with previous versions of enterprise search

The identity management APIs were made available with OmniFind Enterprise Edition, Version 8.4, and they are not compatible with previous product versions. Client applications that were created with version 8.3 or earlier, including the `ESSearchApplication` and `ESSearchPortlet`, must migrate to the identity management APIs to continue using the security features of enterprise search.

Creating the user's security context XML String with the identity management API

The identity management API provides several Java classes that can be used to create the USC XML string programmatically.

To create the USC XML string for a particular user, you should first instantiate a `SecurityContext` object. The `SecurityContext` object contains a user name, an array of `Identity` objects, and optionally a Single Sign-On (SSO) token. The user name that is assigned to the `SecurityContext` is typically the value that the user specified to log in to your application.

After you create a `SecurityContext` object, you create an array of `Identity` objects. Each `Identity` object contains a user name and a password, a `String` array of group tokens, a source type, and a domain identifier. If the `SecurityContext` object contains an SSO token, then the user name is required but the password is optional. For example:

```
SecurityContext context = new SecurityContext();
context.setUserID("uid=wpsadmin,o=default organization");

Identity[] identities = new Identity[1];
identities[0] = new Identity();
identities[0].setDomain("portalserver.ibm.com:9081");
identities[0].setType("wp");
identities[0].setUsername("uid=wpsadmin,o=default organization");

String[] groups = new String[3];
groups[0] = "uid=wpsadmin,o=default organization";
groups[1] = "all authenticated portal users";
```

```
groups[2] = "wpsadmins";
identities[0].setGroups(groups);
identities[0].setProperties(new Properties());

context.setIdentities(identities);
```

After you create the context, you can easily set the ACL constraints in the query by calling the `context.serialize(true)` method. The Boolean parameter indicates that the XML string values should be Base64 encoding to ensure proper transmission to the search server. For example:

```
q.setACLConstraints("@SecurityContext::'" + context.serialize
(true) + "'");
```

Creating the user's security context XML String with the Java String classes

For applications that were written before OmniFind Enterprise Edition, version 8.4, the only way to create the user's security context XML string was to manipulate the XML as a Java String.

This method is no longer recommended, but it is still supported. However, the format has some important additions: some values must use Base64 encoding before they are transmitted as part of the query request to the search server. If you choose not to use the identity management API to create the XML string, ensure that you adhere to the following XML format:

```
<identities id="REQUIRED">
  <ssoToken>NOT REQUIRED</ssoToken>
  <nativeTokens>
   <nativeToken>REQUIRED</nativeToken>
   <nativeToken>REQUIRED</nativeToken>
  <nativeTokens>
<identity id="REQUIRED">
  <username>REQUIRED</username>
  <password encrypt="yes">NOT REQUIRED</password>
  <type>NOT REQUIRED</type>
  <groups>
   <group id="REQUIRED"/>
   <group id="REQUIRED"/>
  </groups>
  <properties>
   <property name="NOT REQUIRED">REQUIRED</property>
   <property name="NOT REQUIRED">REQUIRED</property>
  </properties>
 </identity>

 . . .
 </identities>
```

Administration applications

A search and index API administration application sends queries to the index server to administer collections.

The public search and index API can interface with internal enterprise search APIs. A search and index API administration application can do the following tasks:

- Register an application ID
- Create an instance of an application ID
- Administer collections
  - Create or destroy collections
  - Add or remove documents to or from a collection

- Refresh or reorganize indexes
- Enable or disable collections for indexing or for searching
- Unregister an application ID

The administration application retrieves an index object from the service class. The application uses the administration package of the search and index API to create or destroy collections and to enable or disable collections for indexing or searching. The administration application then uses the index package of the search and index APIs to obtain one or more index objects on which you can perform operations such as add or remove documents and build or reorganize the index with the added documents.

After you build indexes and enable them for search, search applications can search your indexes.

Use the com.ibm.siapi.admin package to create or destroy collections and to enable or disable collections for indexing and searching.

Use the com.ibm.siapi.index package to add or remove documents to or from a collection and to refresh or reorganize indexes.

Related concepts

"Search and index APIs" on page 3

"Javadoc documentation" on page 3

Related reference

"Search applications" on page 7

Registering application IDs

Register application IDs on the enterprise search server before you create an instance of the application ID.

An application instance that is created with the administrator user ID and password must be passed to the `AdminService` API, which registers an application ID on the server.

To register an application ID:

1. Create the application on the enterprise search server.
2. Create the `clientAppInfo` instance.
3. Register the application on the server. The registration API requires enterprise search administrator credentials (user name and password). If these credentials cannot be shared, then the enterprise search administrator must register the requested application IDs.

   If the credentials for the enterprise search administrator are not shareable, then the enterprise search administrator should use the SIAPI APIs to register the Application ID. Using the enterprise search administration console to create application IDs is not suitable for the implementation of the SIAPI Administration APIs.

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
```

```
* RegisterApplicationID provides sample code to create a
* unique search application ID required to create
* collection, add documents , build a
* index and search the collections.
* This sample has to be invoked to register a valid ID.
*/

public class RegsiterApplicationID {

public static void main(String[] args) {
try {
AdminFactory factory = SiapiAdminImpl.createAdminFactory(IAdminConstants.ADMIN_FACTORY_IMPL);
if(factory != null){

// name of the client application ID
String appID = "SIAPI-App";
// specify "dummy" password
ApplicationInfo appInfo = factory.createApplicationInfo(appID, "dummy");


AdminService service = factory.getAdminService(null);

if(service != null){
System.out.println("Register application ID="+ appID);
// The Omnifind adminstrator credentials
// Please contant your administrator to get the credentials
// If the credentials are not shareable, the administrator
// should implement the registration application and create
// requested IDs.
ApplicationInfo adminInfo = factory.createApplicationInfo("esadmin", "search");
// privileges are not supported
int privilege = -1;
// register application instance
service.registerApplication(adminInfo, appInfo, privilege);
}

// change password
System.out.println("Change password for appID=" + appID);
appInfo.setPassword("search");
System.out.println("New password was successfully set." );
}
} catch (SiapiException e) {
e.printStackTrace();
System.out.println(e.getLocalizedMessage());
}
}
}
```

Unregistering application IDs

To unregister an application ID, you destroy the application information on the enterprise search server.

To unregister an application ID:

Destroy the application information (`clientAppInfo` object).

```
// to unregister an application, destroy the application
// information on the server
service.unregisterApplication(clientAppInfo, clientAppInfo.getId())
```

Creating an instance of an application ID

Each client application must be identified by an application ID. You create an application ID with the `AdminFactory` factory.

You must first register application IDs on the enterprise search server. After you register the application ID, you can retrieve an instance of the application ID.

For authentication, a password must be associated with the user application ID. However, different client applications can use the same enterprise search application ID. The enterprise search application ID specifies only which collections the application has access to. You can have two different applications (for example, finance and personnel applications) that are granted access to the same set of collections as defined and controlled by the enterprise search application ID.

Contact the search administrator to get the collection ID.

To create an application ID:

Use the `AdminFactory` factory to create an application ID. The application ID can be associated with a collection ID so that associated collection can be administered.

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

AdminFactory factory = SiapiAdminImpl.createAdminFactory
( "com.ibm.es.siapi.admin.AdminFactoryImpl");

if(factory != null){
  String appID = "SIAPI-App";
  String appPw = "password";
 ApplicationInfo appInfo = factory.createApplicationInfo(appID,appPw);
 if(appInfo != null){
     System.out.println("Application ID called " + appInfo.getId() + "
      was created successfully." );
 } else {
     System.out.println("Application ID called " + appInfo.getId() + "
      was not created!!" );
 }
}
. . .
```

Remember to associate the application ID with the collection ID in the enterprise search administration console.

Creating or destroying collections

A collection is created with an initial application ID. Only this application ID is authorized to delete the specified collection. The collection can be associated with other application IDs by using the `addCollectionToApplication` method.

To create or destroy a collection, you must first call the `AdminService` class as shown in the following example. This example shows the `AdminService` class creating a collection:

```
class AdminService {
       void createCollection(ApplicationInfo appInfo,
      java.lang.String collectionID,
      java.lang.String collectionLabel,
      int optimizationMode,
      java.lang.String defaultLanguage,
      java.util.Properties config)
```

A set of optional properties can also be passed to the collection with the `java.util.Properties` instance. To create a collection with the ApplicationInfo object, you must create the AdminService object first. The following sample shows all the optional properties:

```
 AdminFactory factory = SiapiAdminImpl.createAdminFactory
  (IAdminConstants.ADMIN_FACTORY_IMPL);
   if(factory != null){
    AdminService service = factory.getAdminService(null);
    if(service != null){
     ApplicationInfoImpl appInfo = new ApplicationInfoImpl();
     appInfo.setId("SIAPI-App");
     appInfo.setPassword("search");

     Properties config = new Properties();
     config.setProperty(IAdminConstants.KEY_INDEX_LOCATION,
           "/home/esadmin/siapidata");
     config.setProperty(IAdminConstants.KEY_ENABLE_COLLECTION_NGRAM,
           "true");
     config.setProperty(IAdminConstants.KEY_ENABLE_COLLECTION_SECURITY,
           "true");
     config.setProperty(IAdminConstants.KEY_MAX_DOCS_IN_INDEX, "10000");

     // if collection ID can be set to "", system generates a
     // unique id
     String colID = "col_123";
     String colLabel = "SIAPI Client";
     // if language is "", a default language of "en" is associated
     String colLangauge = "en";
     // identified the ranking model for the collection.
           // DATE_BASED, LINK_BASED and NO_STATIC_RANK are the supported models
     int optimizationMode = IAdminConstants.DATE_BASED;
     adminService.createCollection(appInfo,
       colID,
       colLabel,
       optimizationMode,
       colLangauge,
       config);
    }
   }
....
```

For the `config.setProperty` method, specify the location of the index.

After you create the collection and enable it for indexing, you can add documents. After you add the documents, you can refresh or reorganize the index and make the documents ready for search. The indexRef.build API refreshes the index and the indexRef.reorganize API reorganizes, or merges, available refreshed indexes into a main index. The index can later be enabled for search.

The following sample shows you how to destroy a collection. By destroying the collection, you also delete all the indexed documents on the disk.

```
class AdminService {
        void destroyCollection(ApplicationInfo appInfo,
 java.lang.String collectionID);

...
AdminService service = factory.getAdminService(null);
if(service != null){
 ApplicationInfoImpl appInfo = new ApplicationInfoImpl();
 appInfo.setId("SIAPI-App");
 appInfo.setPassword("search");
 String colID = "col_123";
 adminService.destroyCollection(appInfo, colID);
}
    }
. . .
```

Adding documents to a collection

Use the `IndexFactory` object to add documents to a collection.

To add documents to a collection:

1. Enable the collection for indexing by using the `adminService.isEnabledForIndexing` method as in the following example:

   ```
   . . .
   boolean enabled = adminService.isEnabledForIndexing(appInfo, colID);
           if(!enabled){
                   // enable for indexing
                   System.out.println("Enabling collection for indexing");
                   adminService.enableCollectionForIndexing(appInfo, colID, null);
           }
   . . .
   ```

   To add documents to a collection, you must enable the collection for indexing. This function starts the parser driver that is associated with the collection.
2. Create documents from a data source. With the following APIs, you can add documents as:
   - A byte array of the content
   - A character array of the content
   - A String of the content

   The following example shows how to create documents:

   ```
   class IndexFactory{
           // Create a raw document by its byte array content.
     Document createDocument(java.lang.String documentID, byte[] content,
     java.lang.String documentType, java.lang.String documentSource);

           // Create a raw document by its char array content.
        Document createDocument(java.lang.String documentID, char[] content,
     java.lang.String documentType, java.lang.String documentSource) ;

           // Create a raw document by its string content.
        Document createDocument(java.lang.String documentID,
     java.lang.String content, java.lang.String documentType,
     java.lang.String documentSource) ;


   ...
   IndexFactory indexFactory = SiapiIndexImpl.createIndexFactory
   ("com.ibm.es.siapi.index.IndexFactoryImpl");
   . . .

   String documentContent = readDocumentFromDB2
   ("docid:db2://sample//EmployeePicture");
   Document document = indexFactory.createDocument
   ("docid:db2://sample//EmployeePicture",
   documentContent, "DB2 type", "Sample");
   ```
3. Add the document to the collection by using one of the following APIs:

   ```
   class Index{
           // Add raw document to index
           void addDocument(Document doc);
           // Add raw document with meta data to the index
           void addDocument(Document doc, java.util.HashMap fieldMapping);
   ```

```
IndexService indexService = indexFactory.getIndexService(null);
Index indexRef = indexService.getIndex(appInfo, collectionID);
// add raw document
indexRef.addDocument(document);
```

4. Optional: Add the metadata of a document to the index. Metadata is a list of fields in a document that can be configured in the index as part of the query term. You can use any of the following APIs to create fields with different constructors:

```
class IndexFactory{
        Field createField(java.lang.String fieldName, boolean value);
        Field createField(java.lang.String fieldName, byte[] value);
        Field createField(java.lang.String fieldName, java.util.Date value);
        Field createField(java.lang.String fieldName, double value);
        Field createField(java.lang.String fieldName, int value);
        Field createField(java.lang.String fieldName, java.lang.String text);

// create a searchable/parametric/returnable field
Field myField = indexFactory.createField("empid", "00140");
myField.setFieldSearchable(true);
myField.setParametric(true);
myField.setReturnable (true);

// create a searchable and returnable field
Field myField2 = indexFactory.createField("Format", "image/gif");
myField2.setFieldSearchable(true);
myField2.setParametric(false);
myField2.setReturnable (true);


HashMap fieldMapping= new HashMap();
fieldMapping.put (myField.getID(), myField);
fieldMapping.put (myField2.getID(),myField2);
// add raw document with meta data
indexRef.addDocument(document, fieldMapping);
```

Building indexes

After you add or remove documents from an index, you need to rebuild the index.

When you build an index for the first time, you must build a main index. After you build the first index, you update that index regularly by building delta indexes that contain information about new, changed, and deleted documents. To improve performance and space allocation of the index, you must regularly rebuild the main index. Building the main index means that you merge the smaller delta indexes with the larger main index. Building a main index requires more time and resources than building a delta index.

Use the following APIs build the index:

```
Class Index {
        // Causes the index to reflect all pending index operations
       // in the index data structures or storage, so that
       // added documents can be searched, and deleted
       // documents cannot be searched.
        void build();
        // Instructs the index to reorganize its persistent storage,
        // usually to remove unused space, and so on.
        void reorganize();


indexRef.build(); or indexRef.reorganize()
```

You can set a fragmentation count property that tells the search and index API code when indexes should be updated.

The value 2 in the following property setting means the main index will be built after two delta indexes are built:

```
indexes.index.setProperty
(IAdminConstants.BUILD_FRAGMENTATION_COUNT, 2)
```

See the index build sample for more examples.

After the collection is built, it can be made available for search by using the administration package.

Enabling indexes for search

After an index is populated with documents, it can be enabled for search so that search applications can search the indexed documents.

Use the following APIs to enable the index for search:

```
class AdminService {
        // Make a collection available for search.
        void enableCollectionForSearch(ApplicationInfo appInfo,
 java.lang.String collectionID,
         java.util.Properties config) ;
        // Make a collection unavailable for search.
        void disableCollectionForSearch(ApplicationInfo appInfo,
 java.lang.String collectionID,
         java.util.Properties config) ;

adminService. enableCollectionForSearch(appInfo, collectionID, null);
```

Web services for enterprise search

Web services are available to support federated search capability so that you can search enterprise search collections.

Restrictions

- The Web services client proxy supports WebSphere Application Server, version 6.0 and version 6.1 only. If your enterprise search system uses an older version of WebSphere Application Server, you cannot use the Web services.
- The Web services interface does not support single sign-on security capabilities for enterprise search.
- For URI sources such as DB2 databases or file systems, the ability to documents from search results is not supported.

Toolkit for application development

A compiled Java client proxy is bundled in an application development toolkit so that you can develop customized applications.

The Web services client is integrated in the enterprise search ESSearchServer.ear application. The Web services and the ESSearchServer application share common copies of search factories.

The JAR file `es.siapi.toolkit.jar` contains the necessary Java packages, samples, and Javadoc documentation to help you develop custom applications for administration and search. The JAR package includes the following items that you need to create a Web services client proxy:

- Compiled Java client proxy (also included in the JAR package)
- Default configuration file to specify endpoints
- Javadoc documentation

A compiled version of the proxy is provided and most of the common search options, such as the number of results to return, language, collections, application ID, have default values that you can overwrite. This WSDL code can be provided to any compliant IDE to generate a copy of client proxy.

Enterprise search provides a Java version of the client proxy to invoke the Web service. However, you can access the WSDL to generate client proxies in other languages, such as PHP, C++, or C#.

Use the following URL to access the WSDL file:

```
http://your_search_server/ESSearchServer/wsdl/com/
ibm/es/ws6/server/search/ofsearch.wsdl
```

Use the following URL as the end point for the Web service:

```
http://your_search_server/ESSearchServer/services/ofsearchBinding
```

For a multiple server installation, the Web services are hosted on all of the configured search servers. This is done to scale the availability of search services.

Crawler plug-ins

Crawler plug-ins are Java application programming interfaces (APIs) that you can use to change content or metadata in crawled documents. There are two types of crawler plug-ins: one for non-Web sources and one for Web sources.

Crawler plug-ins for non-Web sources

You can apply business and security rules to enforce document-level security and add, update, or delete the crawled metadata and document content that is associated with documents in an enterprise search index. The crawler plug-in APIs can be used for any of the crawlers other than Web sources.

Crawler plug-ins for Web sources

You can add fields to the HTTP request header that is sent to the origin server to request a document. You can also view the content, security tokens, and metadata of a document after the document is downloaded. You can add to, delete from, or replace any of these fields, or stop the document from being parsed.

Web crawler plug-ins support two kinds of filtering: prefetch and postparse. You can specify only a single Java class to be the Web crawler plug-in, but because the prefetch and postparse plug-in behaviors are defined in two separate Java interfaces and because Java classes can implement any number of interfaces, the Web crawler plug-in class can implement either or both behaviors.

The Web crawler plug-in has two specific plug-in types:

Prefetch plug-in
A prefetch plug-in is called before the crawler downloads a document.

Your plug-in is given the document URL, the fetch method, the HTTP version, and the HTTP request header. Your plug-in can use these elements to decide whether to modify the request header (for example, to add cookies) or even to cancel the download.

Postparse plug-in
The postparse plug-in is called after any download attempt. The download does not need to produce content or parses the content. The plug-in is given the document URL, the metadata that is extracted by the crawler from various sources, and the document's content. The plug-in can determine whether to alter any of these items in the document and whether to save the content of the document before it is parsed.

Javadoc documentation for crawler plug-ins

For detailed information about each plug-in API, see the Javadoc documentation in the following directory: `ES_INSTALL_ROOT/docs/api/`.

Related tasks

"Creating a prefetch plug-in for the Web crawler" on page 55

"Creating a postparse plug-in for the Web crawler" on page 58

Related reference

"Sample plug-in application for non-Web crawlers" on page 83

Crawler plug-ins for non-Web sources

Data source crawler plug-ins are Java applications that can change the content or metadata of crawled documents.

You can configure a data source crawler plug-in for the following enterprise search crawlers:

- Content Edition
- DB2 Content Manager
- DB2
- Domino Document Manager
- Exchange Server
- NNTP
- Notes
- QuickPlace
- UNIX file system
- Web Content Management
- WebSphere Portal
- Windows file system

To modify Web documents, use the Web crawler plug-in.

With the crawler plug-in for data source crawlers, you can add, change, or delete crawled content or metadata.

When you specify the Java class as the new crawler plug-in, the crawler calls the class for each document that it crawls.

For each document, the crawler passes to your Java classes the document identifier, the security tokens, the metadata, and the content that was specified by an administrator. Your Java class can return a new or modified set of security, metadata, and content.

If you created a crawler plug-in for a crawler with an earlier version of enterprise search, you can use the crawler plug-in as is. However, you cannot use version 8.4 of the crawler plug-in with older versions of a crawler plug-in. The APIs are different. If you want to add, change, or delete crawled content, use version 8.4 of the crawler plug-in.

Creating a crawler plug-in for non-Web data sources

You can create a Java class to programmatically update the value of security tokens, metadata, and the document content of data sources other than Web.

About this task

When the crawler is started, the plug-in process is forked. An `AbstractCrawlerPlugin` object is instantiated with the default constructor and the `init`, `isMetadataUsed`, and `isContentUsed` methods are called once. When the crawler is stopped, the `term` method is called and the object is destroyed.

Procedure

To create a Java class for use as a crawler plug-in with content-related functions:

1. Inherit `com.ibm.es.crawler.plugin.AbstractCrawlerPlugin` and implement the following methods:
   ```
   init()
   isMetadataUsed()
   isContentUsed()
   term()
   updateDocument()
   ```
   The `AbstractCrawlerPlugin` class is an abstract class. The `init` method and the `term` method are implemented to do nothing. The `isMetadataUsed` method and `isContentUsed` method are implemented to return false by default. The `updateDocument` method is an abstract method, so you must implement it.

   For name resolution, use one of the following JAR files:
   - AIX, Linux, or Solaris: `ES_INSTALL_ROOT/lib/dscrawler.jar`
   - Windows: `ES_INSTALL_ROOT\lib\dscrawler.jar`
2. Compile the implemented code and create a JAR file for it. Add the file `dscrawler.jar` to the class path when you compile.
3. In the enterprise search administration console, edit the appropriate collection. Select the Crawl page and edit the crawler properties for the crawler that will use your plug-in. Specify the fully qualified class name of the implemented Java class. Also specify the fully qualified class path for the JAR file and the directory where all files that are required by the Java class are located.

   Restriction: In the administration console, the value of the plug-in class path file is ignored for the security token plug-in. If you use the security token plug-in that was provided in an earlier version of enterprise search, the plug-in needs to copy all libraries (JAR files) that it requires.
4. On the Crawl page, click **Monitor**. Then, click **Stop** and **Start** to restart the session for the crawler that you edited. Click **Details** and start a full crawl.

If the crawler stops when it is loading the plug-in, view the log file and verify that:

- The class name and class path that you specified in the crawler properties page are correct.
- All necessary libraries are specified for the plug-in class path.
- The crawler plug-in does not throw a `CrawlerPluginException` error.

Web crawler plug-ins

The Web crawler plug-in provides two types of plug-ins: a prefetch plug-in and a postparse plug-in.

With the prefetch plug-in, you can use Java APIs to add fields to the HTTP request header that is sent to the origin server to request a document.

With the postparse plug-in, you can use Java APIs to view the content, security tokens, and metadata of a document before the document is parsed and tokenized. You can add to, delete from, or replace any of these fields, or stop the document from being sent to the parser.

If your plug-in requires Java classes or non-Java libraries or other files besides the plug-in, you must write the plug-in to handle that requirement. For example, your plug-in can invoke a class loader to bring in more Java classes and can also load libraries, make network connections, make database connections, or do anything else that it needs.

Plug-ins run as part of the crawler JVM process. Exceptions and errors will be caught, but crawler performance is affected by plug-in execution. You should write plug-ins to do the minimum amount of processing and catch all anticipated exceptions. Plug-in code must be multithread-safe. If you have 200 concurrent downloads, you might have 200 concurrent calls to your plug-in.

Using a plug-in to crawl secure WebSphere Portal sites

If application security is enabled in WebSphere Application Server and you want to crawl secure WebSphere Portal sites with the Web crawler, you must create a crawler plug-in to handle the form-based authentication requests. For a discussion about form-based authentication and a sample program that you can adapt for your custom Web crawler plug-in, see http://www.ibm.com/developerworks/db2/library/techarticle/dm-0707nishitani.

The plug-in is required if you use the Web crawler to crawl any sites through WebSphere Portal, including Workplace Web Content Management™ sites and Lotus Quickr™ sites.

Creating a prefetch plug-in for the Web crawler

To create a prefetch plug-in, you write a Java class that implements the interface `com.ibm.es.wc.pi.PrefetchPlugin`.

To create a prefetch plug-in:

1. Inherit the `com.ibm.es.wc.pi.PrefetchPlugin` interface and implement the following methods:

```
public class MyPrefetchPlugin implements com.ibm.es.wc.pi.PrefetchPlugin {
    public MyPrefetchPlugin() { ... }
    public boolean init() { ... }
    public boolean processDocument(PrefetchPluginArg[] args) { ... }
    public boolean release() { ... }
 }
```

The `init` method is called once when the plug-in is instantiated. If you specify that you have a plug-in class, the crawler loads that class when the crawler is stared and creates a single instance of the class. Your plug-in class must have a no-argument constructor. The crawler creates only one instance of the class. After creating the instance of the class, the crawler calls the `init` method before the first use. This method does the required setup tasks that cannot be done until an instance of the class is in memory.

If the plug-in is not supposed to be used or other errors occur, the `init` method can return false, and the crawler removes this instance from the list of prefetch plug-ins. If the `init` method returns true, the plug-in is ready to use. The `init` method cannot throw an exception.

The `processDocument` method is called on the single plug-in instance for every document that will be downloaded. The crawler uses from one to several hundred download threads, which run asynchronously, so this method can be called from multiple threads concurrently.

The `release` method is called once when the crawler stops to allow the plug-in object to release any system resources or flush any queued objects. This method cannot throw exceptions. A true result means success. A false result is logged.

For name resolution, use one of the following files:

- AIX, Linux, or Solaris: *`ES_INSTALL_ROOT`*`/lib/URLFetcher.jar`
- Windows: *`ES_INSTALL_ROOT`*`\lib\URLFetcher.jar`

2. Compile the implemented code and make a JAR file for it. Add the `URLFetcher.jar` file to the classpath when you compile.
3. In the enterprise search administration console, follow these steps:
   a. Edit the appropriate collection.
   b. Select the Crawl page and edit the crawler properties for the crawler that will use the custom Java class.
   c. In the enterprise search administration console, specify the following items:
      - The fully qualified class name of the implemented Java class
      - The classpath for the plug-in, including all needed JAR files.
   d. Stop and restart the session for the crawler that you edited. Then, start a full crawl.

If an error occurs and the Web crawler stops while it is loading the plug-in, view the log file and verify that:

- The class name and classpath that you specified on the crawler properties page is correct.
- All necessary JAR files were specified for plug-in classpath.
- The crawler plug-in does not throw `CrawlerPluginException` or any other unexpected exception, and no fatal errors occur in the plug-in.

You must write this method to be multithread-safe, which you can do by wrapping its entire contents in a synchronized block, but that permits only one thread to execute the method at a time, which causes the crawler to become single-threaded during plug-in operation, creating a performance bottleneck.

A better way to make the method multithread-safe is by using local (stack) variables for all states, which minimizes the amount of global data and synchronizes only during access to objects that are shared between threads. This method cannot throw an exception. It can return true to indicate successful processing of a document or false to indicate a problem. A false return value is logged with the URL by the crawler.

Prefetch plug-in example

You can use a prefetch plug-in to add a cookie to the HTTP request header before the document is downloaded.

```
package com.mycompany.ofpi;
// OmniFind plug-ins

 public class MyPrefetchPlugin implements com.ibm.es.wc.pi.PrefetchPlugin {
    public MyPrefetchPlugin() { }
    public boolean init() { return true; }
    public boolean release() { return true; }
    public boolean processDocument(PrefetchPluginArg[] args) {
       PrefetchPluginArg1 arg = (PrefetchPluginArg1)args[0];
              if (arg.getURL().startsWith("http://special.mysite.com/")) {
              String httpHeader = arg.getHTTPHeader();
              httpHeader = httpHeader.substring
               (0, orgHeaders.lastIndexOf(CRLF));
              httpHeader = httpHeader + "Cookie: Special=values" + CRLF;
              httpHeader = httpHeader + CRLF;
         }
        return true;
    }
    static final String CRLF = "\r\n";
 }
```

This example shows:

- The first element ([0]) in the argument array that is passed to your plug-in is an object of type `PrefetchPluginArg1`, which is an interface that extends the interface `PrefetchPluginArg`. This is the only argument and the only argument type that is passed to the prefetch plug-in. You can safely cast to it. To be completely safe, you can enclose the cast in a try/catch block and look for a `ClassCastException` object or do an ″instanceof″ test first.
- After you have the argument, you can call any method in the PrefetchPluginArg1 interface. The `getURL` method returns the URL (in String form) of a document that the crawler downloads. You can use this URL to decide if the document requires additional information in the request header, such as a cookie.
- The `getHTTPHeader` method returns a String that contains the all of the content of the HTTP request header that the crawler sends so that the crawler can download the document. The plug-in can inspect and modify this header if necessary. For example, a single cookie can be added to the header or any other information if it is valid for an HTTP request header. You can also remove any of this information. If you modify the header, you must conform to HTTP protocol requirements. For example, every line must end with a CRLF sequence, and the header must use ISO-8859-1 encoding.
- The `processDocument` method is called once for every document that the crawler downloads. If the `processDocument` method returns false, its results are ignored. If it returns true, the crawler checks what it did. To stop the download, the Prefetch plug-in calls the `setFetch(false)` method.

Related concepts

"Crawler plug-ins" on page 52

Deploying a prefetch plug-in

To identify your plug-in class to the crawler, put the class in a JAR file and enter the name of the plug-in class and the location of the JAR file in the crawler window in the enterprise search administration console. You must enter the fully qualified name of the plug-in class, and the absolute path name of the JAR file.

To deploy a prefetch plug-in in enterprise search:

1. Compile the JAVA file and create a JAR file for it. The JAR can also contain supporting classes and resources, so you might name it `ofplugins.jar`.
2. Copy this JAR file to the computer that runs the Web crawler in your enterprise search installation. Enter the absolute path for the JAR file in the administration console on the crawler window when you enable plug-in.
3. In the enterprise search administration console, specify the following items:
   - The fully qualified class name of the implemented Java class, for example, `com.mycompany.ofpi.MyPrefetchPlugin`
   - The qualified class path for the JAR file

   Ensure that the information that you enter is correct. Enterprise search does not check that the JAR file exists.

   When the crawler is started and finds a plug-in JAR file and class name, the crawler loads the JAR and instantiates the class by using the no-argument constructor. The crawler then initializes the instance by calling the init method. If that method returns true, the plug-in is added to the list of prefetch plug-ins.

After you run the crawler, the return value is logged in the collection log file as informational message. To see information messages, choose **All messages** as **Type of information to log**.

Creating a postparse plug-in for the Web crawler

With the postparse plug-in, you view the content, security tokens, and metadata of a document after the document is downloaded so that you can add to, delete from, or replace any of these fields, or stop the document from being sent to the parser.

To create a postparse plug-in, you write a Java class that implements the interface `com.ibm.es.wc.pi.PostparsePlugin`, for example:

```
public class MyPostparsePlugin implements
com.ibm.es.wc.pi.PostparsePlugin {
   public MyPostparsePlugin () { ... }
   public boolean init() { ... }
   public boolean processDocument(PostparsePluginArg[] args) { ... }
   public boolean release() { ... }
}
```

The plug-in class can implement both interfaces, but it needs only one `init` method and one `release` method. If the class does both prefetch and postparse processing, you need to initialize and release resources for both tasks. Both the `init` method and the `release` method are called once.

The `processDocument` method is called on the single plug-in instance for every URL for which a download was attempted. Not all downloads return content. The HTTP return codes, such as 200, 302, or 404, can be used by your plug-in to determine what to do when called. If content was obtained and if the content was suitable for HTML parsing, the content is put through the parser, and the results of parsing are available when your plug-in is called.

Postparse plug-in examples

The following example shows how to add security ACLs to the metadata that the crawler sends with documents that are downloaded from a particular site. You can use a postparse plug-in to add those ACLs just before the crawler writes the document to the parser's input buffer:

```
package com.mycompany.ofpi;  // OmniFind plug-ins

import com.ibm.es.wc.pi.*;

 public class MyPostparsePlugin implements PostparsePlugin {
    public MyPostparsePlugin() { }
    public boolean init() { return true; }
    public boolean release() { return true; }
    public boolean processDocument(PostparsePluginArg[] args) {
              try {
          PostparsePluginArg1 arg = (PostparsePluginArg1)args[0];
                 if (arg.getURL().startsWith("http://mysite.com/users/")) {
                    // Extract user name from URL; look up appropriate tokens.
             String acls = // Create a comma-separated list of the
                           // additional ones.
             arg.addSecurityACLs(acls);
          }
          return true;
              } catch (Exception e) {
                 return false;  // disregard returned results
              }
    }
 }
```

You can also use a postparse plug-in to add a new metadata field to your crawled documents. For example, if some of your documents contain a keyword, you might want to add a metadata field called "MyUserSpecificMetadata" to the search record that contains a string that you need to look up at when the crawler is running with various "searchability" attributes. The following example shows how to add a metadata field:

```
package com.mycompany.ofpi;  // OmniFind plug-ins

import com.ibm.es.wc.pi.*;   // Plug-in API

public class MyPostparsePlugin implements PostparsePlugin {

   public MyPostparsePlugin() { }
   public boolean init() { return true; }
   public boolean release() { return true; }
   public boolean processDocument(PostparsePluginArg[] args) {
      try {
         PostparsePluginArg1 arg = (PostparsePluginArg1)args[0];
         if (arg.getContent() != null && arg.getContent().length > 0) {
            String content = new String( arg.getContent(), arg.getEncoding() );
            if (content != null && content.indexOf(keyword) > 0) {
               final String userdata = null; // look up string by keyword.
               FieldMetadata mf = new FieldMetadata(
                  "MyUserSpecificMetadata" // field name
                  userdata,                // field value
                  false,                   // searchable?
                  true,                    // field-searchable?
                  false,                   // parametric-searchable?
                  true,                    // can be extracted by search?
                  "MetadataPreferred",     // metadata value rather
                                           // than content
                  false);                  // show in summary?
               addMetadataField(mf);    // Add it to the list.
               return true;             // Use results.
            }
         }
         return false;  // ignore results
      } catch (Exception e) {
            return false;  // disregard returned results
        }
    }
}
```

The document content is available from the plug-in argument (`arg.getContent`). The encoding that the crawler found is available. With the content and encoding, you can create a String. You can then look for some keyword (content.indexOf(...)), associate new data with it (userdata = ...), and insert that new data as the content of the new field.

To define a new metadata field, create an instance of the `FieldMetadata` object and set its field values.

Related concepts

"Crawler plug-ins" on page 52

Sample code

The enterprise search sample application

The sample search application, ESSearchApplication, is a working sample of a browser-based user interface built around SIAPI that fully demonstrates the advanced search capabilities provided by OmniFind Enterprise Edition.

You can run the sample search application in two environments: as an Enterprise Application Resource (EAR) file in a standard WebSphere Application Server environment and as a JSR-168 compliant portlet in a WebSphere Portal Server environment.

Important: If you customize the sample search application, you must rename it to ensure that your changes are not overwritten when you install a fix pack or upgrade to a new version of OmniFind Enterprise Edition.

Overview

The sample application comprises two separate packages that contain two individual applications. The search servlet application EAR file name is `ESSearchApplication.ear` and the search portlet application file name is `ESSearchPortlet.war`. The `ESSearchApplication.ear` file contains a Web Application Resource (WAR) file named `ESSearchApplication.war`, which contains the stand-alone search application.

The sample code for the search applications is provided in the `ES_INSTALL_ROOT/samples/ESSearchApplication` directory.

The search application is built by using a number of modern Web technologies and frameworks that include but are not limited to the ones that are listed here. You must be familiar with these technologies before you customize the source code for the applications.
- Apache Struts 1.1
- WebSphere Portal Struts Portlet Framework
- Java Servlet Specification Level 2.3
- Java Server Pages (JSP)
- JavaScript™
- Asynchronous JavaScript and XML (AJAX)
- Cascading Style Sheets (CSS)

Struts and the Struts Portlet Framework

Apache Struts is a popular open source project for implementing Web applications that use a Model-View-Controller (MVC) design pattern. The Model consists of the business logic or database layer, the View represents the page design, and the Controller represents the code that is used to control the navigation through the application. By separating these three aspects of the design of a Web application, the Struts framework allows for extensibility and maintainability. The reasons for the popularity of the Struts framework in a standard servlet environment also apply to developing portlet applications.

The Struts Portlet Framework allows developers to build portlet applications for use in the WebSphere Portal environment by using the Struts framework. With the exception of a few special considerations, the packaging and structure of a Struts portlet application is very similar to that of a standard Struts servlet application. Due to these similarities, a single code base can be used to create both a sample search servlet application and a sample search portlet application.

Java classes

The Java classes that make up the sample search application are shared across all three of the Web application samples that are provided for enterprise search. The source code for these classes is included with the product in the `ES_INSTALL_ROOT/samples/ESSearchApplication/JavaSource` directory. The classes are divided into the following packages under the `com.ibm.es.searchui` base package:

actions
: This package contains all of the Struts Action implementation classes that support the actions that the user can take within the search application view. The Action class names map to the pages that are available to the user within the application. Additionally, there are a few specialized classes:

- `AuthAction`: This class implements the javax.servlet.http.HttpServlet interface. This class is used only within the sample search servlet application when WebSphere security is enabled on the search server. Its specific purpose is to intercept the user name and password that is entered on the login form by the user. This user name and password pair is then stored in the HttpSession as a Base64-encoded value that is used by the SIAPI for communicating to the search server.
- `InitAction`: This class implements the org.apache.struts.action.PlugIn interface. This class is used only within the sample search servlet application to perform some initialization tasks before loading the actual Struts ActionServlet class.
- `BaseAction`: This class extends the org.apache.struts.action.Action class. This BaseAction class is a superclass to all of the other Struts action classes in the com.ibm.es.searchui.actions package. This class provides many common methods that are shared across the action classes, such as those required to process a user's query request, common logging, and other such common tasks.

charts and charts.servlet
: This package provides the valueObjects and servlet classes for the Top Results analysis charts.

fetch
: This package provides clickable URL support for documents that are crawled by crawlers that are not directly URL-addressable, such as the Windows file system or Content Edition crawlers. When a user clicks such a URL, the request is forwarded to this servlet, which in turn writes the file contents directly to the servlet's output stream.

filters
: This package contains classes that implement the Java Servlet Filter interface. The package consists of a single class that is used to set the request character encoding to UTF-8 encoding. This encoding allows the search application pages to display multiple languages within the same view.

helpers
: This package contains classes that encapsulate several functions, including the SIAPI methods, identity management API methods, and the objects that are used by the category tree.

resources
: This package contains the translated resource bundles for the application. All of the predefined strings displayed on the application screens are contained in these bundles.

tags
: This package contains all of the custom Java Tag Library classes used by the application's JSP pages. These tag libraries provide custom functions such as formatting clickable document URIs, manipulating document titles, converting SIAPI date field values to readable string representations, and many other such functions.

valueObjects
: This package contains value objects (beans) that are used by the application to pass information between the Action classes and the View (JSP pages).

Web content

The directory structure for the Web applications is very similar. Many of the JSP pages and other Web application support files are shared. The source code for these files is included with the product in the `ES_INSTALL_ROOT/samples/ESSearchApplication/WebContent` directory. The following information describes the content and layout of the directory structure, starting from the root directory:

root
: The root directory is the directory named after the Web application. This directory contains all of the JSP pages for the given application and other subdirectories. For example, the root directory for the search portlet application is named ESSearchPortlet.war. When working with the source code, the root directory is the WebContent folder.

css
: This directory contains all of the cascading style sheet (CSS) files that are used by the various Web applications. The majority of the style sheets used within the search servlet application are provided by WebSphere Portal so that the appearance and behavior of the servlet application matches the portlet application.

images
: This directory contains all of the GIF files that are used by the Web applications.

layouts
: This directory contains all of the Struts Tiles layout JSP pages. These JSP files determine how the view of the application is structured in different instances. For example, different layouts are defined for the servlet application and the portlet application.

META-INF
: This directory contains the files that are included in the META-INF directory of the `ESSearchApplication.ear` file. These files include the enterprise application deployment descriptor and the WebSphere policy file.

scripts
: This directory contains various JavaScript files that are used to provide some of the advanced interactivity within the search applications. Some of these functions include the calendar pop-ups, the search result tooltip

preview, the AJAX functionality for the top results analysis charts, the AJAX functionality for the search application customizer, the AJAX functionality for the search result paging, and some helper functions.

WEB-INF
This directory is a required directory in any Web application. This directory contains the `struts-config.xml` definition file, the Web application deployment descriptor, the `config.properties` file that is used to control the search servlet application settings, and the portlet deployment descriptor that contains the definitions of the portlet application.

WEB-INF/conf
This directory contains the Struts Tiles definition file (`tiles-def.xml`).

WEB-INF/portal
This directory contains the Web deployment descriptor (`web.xml`), Struts configuration definition (`struts-config.xml`), and portlet deployment descriptor (`portlet.xml`) for the portlet application.

WEB-INF/tld
This directory contains all of the tag library definition (`tld`) files for the applications. The tag library definition files include the standard Struts `tld` files and the custom enterprise search `tld` file.

WEB-INF/was
This directory contains the Web deployment descriptor (`web.xml`) and Struts configuration definition (`struts-config.xml`) for the servlet application.

Search portlet application

Despite the fact that the majority of the JSP and Java source is shared between the servlet application and the portlet application, several differences are worth noting:

Layout
The portlet application has a unique layout that does not include the banner, the search tabs, the topmost toolbar, or integrated links to the product information center. These changes are required to allow the portlet application to integrate more naturally with the overall portal theme and layout.

CSS The portlet application does not include the `WEB-INF/css` directory. Instead, it relies on the WebSphere Portal Server theme to define the current CSS files in use.

Configuration parameters
The portlet application does not use the `config.properties` file and it does not support the search application customizer. The portlet's configuration is controlled through the WebSphere Portal Administration Portlet configuration interface.

Namespace
Several key JavaScript functions require direct access to the current HTML form by name. In these cases, the JSR-168 `portlet:namespace` tag library must be used to ensure that the form name is properly scoped.

Action and Render phases
The portlet application has multiple Action class implementation classes for the Action phase and Render phases of the portlet life cycle. This distinction between phases in the WebSphere Portal Server framework

requires the IStrutsPrepareRender interface. See the following developerWorks® article for more information: http://www-128.ibm.com/developerworks/websphere/techjournal/0504_pixley/0504_pixley.html

Logging and debugging

The search applications all share a common way of logging trace and error information. The logging and tracing infrastructure for the applications is built by using the Java Logger classes provided in the core J2SE 1.4 Java Runtime Environment. The BaseAction Java class defines a root logger for the com.ibm.es.searchui package. This root logger is initialized with the logging.level property when the search application is initialized.

The default logging level is set to SEVERE, which means that the applications, as provided, log only Java Exceptions. You can enable more detailed information to be logged by changing the logging.level property to INFO, FINE, FINER, FINEST, or ALL. For the servlet application, you can change the logging.level property by using the search application customizer or by changing the `ES_INSTALL_ROOT/ESSearchApplication.ear/ESSearchApplication.war/WEB-INF/config.properties` file. For the portlet application, you can change the logging.level property by using the WebSphere Portal Administration and changing the Portlet parameter value.

The various levels of logging add information and add overhead to the applications. Some examples of the types of information logged include:
- The query string and any security constraints
- The processing of the identity management component, including such things as the user name that is entered by the user, the source type, and any data returned from the identity management APIs
- The values of the HTTP headers and request parameters and any cookies that are included with the request
- The user name of the current user, if WebSphere security is enabled

In the case of the servlet application, the logging information is written to the `ES_NODE_ROOT/logs/ESSearchApplication.0.log` file. For the portlet application, the logging information is written to the WebSphere Portal Server `SystemErr.log` file.

Compiling the applications

The sample code for all three of the sample applications can be compiled from the command line by using a single build process. To build the applications, you must install and configure the following tools:

Apache ANT
ANT is a standard, open source tool for building source code by using an XML descriptor file. For more information about how to install and configure Apache ANT, see http://ant.apache.org.

IBM Java Development Kit 1.4.2
The IBM Java Development Kit 1.4.2 is the IBM implementation of the Java Development Kit. For more information about how to install and configure the IBM JDK, see http://www.ibm.com/developerworks/java.

The ANT XML descriptor file (`build.xml`) is in the `ES_INSTALL_ROOT/samples/ESSearchApplication` directory. This single ANT build definition file contains instructions on how to build all of the sample applications in a single build process. When the build is done, the message BUILD SUCCESSFUL is displayed on the console. The output of the build is located in the `ES_INSTALL_ROOT/samples/ESSearchApplication/bin` subdirectory and consists of the following files:

ESSeachApplication.ear
: Contains both the ESSearchApplication.war servlet application and the GDSSearchApplication.war servlet application.

ESSearchPortlet.war
: Contains the search portlet application. This file can be deployed onto a WebSphere Portal Server by using the WebSphere Portal administration interface.

To compile the sample search applications:

1. From the command line, change to the `ES_INSTALL_ROOT/samples/ESSearchApplication` directory.
2. Enter the command `ant`. This command automatically invokes the ANT build process and uses the `build.xml` file.

Rational Application Developer

Typically, many development organizations work with the source code in a graphical user interface such as the Eclipse framework. Enterprise search includes support for working with the servlet application within the Rational® Application Developer version 7.0 product.

The product includes two Project Interchange ZIP files that you can import into a Rational Application Developer environment. These files are located in the `ES_INSTALL_ROOT/samples/ESSearchApplication/rad` subdirectory:

- `ESSearchApplication.zip`, which is for a stand-alone search application
- `ESSearchPortlet.zip`, which is for a search application that runs as a portlet in WebSphere Portal

To import the servlet application Project Interchange file in to your Rational Application Developer environment:

1. Open the Rational Application Developer tool.
2. Click **File** → **File**.
3. Select **Other** → **Project Interchange** and click **Next**.
4. Click **Browse** next to the **From ZIP file** list and browse to the `ES_INSTALL_ROOT/samples/ESSearchApplication/rad` subdirectory.
5. Select the ZIP file that you want to use and click **Open**.
6. Click **Select All** to select both the EAR and WAR projects and then click **Finish**.

Sample search applications

The search and index API includes several sample applications that show you how to create simple or advanced search applications.

The sample search applications are available after you install OmniFind Enterprise Edition:

- The sample code is available in the `ES_INSTALL_ROOT/samples/siapi` directory.
- The Javadoc documentation is available in the `ES_INSTALL_ROOT/docs/api/siapi` directory.

The following sample applications demonstrate how to do various search tasks:

Simple and advanced search
: The `SearchExample` class provides a simple example of the minimum requirements that are needed to submit a search to the search server. The `AdvancedSearchExample` class is an example that demonstrates some of the advanced query settings and result processing options.

Streaming queries
: The `StreamingSearchExample` class gives a simple example of how to submit and process a streaming query against the search server. Streaming in this case is used to return all results from a particular collection. The results are returned unsorted and only the document ID and the score are provided.

Browse and navigate
: The `BrowseExample` class provides an example of accessing a collection's taxonomy tree and displaying some of the basic navigation properties.

Retrieve all search results
: This sample application (code snippet) shows how to set a query to return unsorted results and loop over the query results.

Federated search
: The `FederatedSearchExample` class provides a simple example of the minimum tasks that are required to submit a federated search to the search server.

Fetch search result documents
: The `FetchSearchExample` class provides an example of how to submit a fetch request to retrieve the content search result documents.

Secured search
: The `SecuredSearchExample` class gives a simple example of how to submit a search to the search server when document level security is enabled for the collection. This example takes a user name and looks up the user's credentials in the identity management credential store, then it passes that information on the SIAPI `Query.setACLConstraints` method.

Identity management
: The `IdentityManagementExample` class provides a working sample program that demonstrates how to use the Identity Management API. For a more complete, user interface-based example, see the ESSearchApplication sample code in the `ES_INSTALL_ROOT/samples/ESSearchApplication` directory.

Related tasks

"Installing the client toolkit" on page 2

Compiling the sample search applications

The ESSearchApplication sample code and search and index API code must be compiled with the IBM Software Development Kit (SDK) for Java 1.4.2, Service Release 5 (SR5).

Restrictions

Before you can build Java applications for enterprise search, you must install and configure Apache ANT, a Java-based build tool. For information about how to install and configure Apache ANT, see http://ant.apache.org/.

The ESSearchApplication application in the `ES_INSTALL_ROOT/samples` directory must run in a JRE Version 1.4 environment. WebSphere Application Server and WebSphere Portal both provide the JRE Version 1.4.

Procedure

To compile and run a sample search application:

1. From the command line, change to one of the following directories:

   For the sample search applications

   `ES_INSTALL_ROOT/samples/siapi`. The default installation paths are:
   - AIX: `/usr/IBM/es/samples/siapi`
   - Linux and Solaris: `/opt/IBM/es/samples/siapi`
   - Windows: `C:\Program Files\IBM\es\samples\siapi`

   For the ESSearchApplication application

   `ES_INSTALL_ROOT/samples/ESSearchApplication`. The default installation paths are:
   - AIX: `/usr/IBM/es/samples/ESSearchApplication`
   - Linux and Solaris: `/opt/IBM/es/samples/ESSearchApplication`
   - Windows: `C:\Program Files\IBM\es\samples\ESSearchApplication`

   Each of these directories includes a `build.xml` file that ANT uses to build the file.
2. Run the ANT script:
   ```
   ant
   ```
   You see the following message after the Java source code compiles:
   ```
   BUILD SUCCESSFUL
   Total time: xx seconds
   ```
3. Run the application by specifying the following commands, where *`search_app`* is the search application that you want to compile:

   AIX, Linux, and Solaris
   ```
   java -classpath ES_INSTALL_ROOT/lib/esapi.jar:ES_INSTALL_ROOT/lib/siapi.jar:.
   search_app
   ```
   Windows
   ```
   java -classpath "ES_INSTALL_ROOT\lib\esapi.jar;ES_INSTALL_ROOT\lib\siapi.jar;."
   search_app
   ```
   Related tasks

   "Compiling the sample administration applications" on page 82

Simple and advanced sample search applications

The `SearchExample` class provides a simple example of the minimum number of required tasks that the application does to submit a search query to the search server. The `AdvancedSearchExample` class shows the same tasks as the simple example, but it prints the full `ResultSet` object instead of only a few values

The simple sample application demonstrates how to:
- Access the service
- Specify a collection
- Specify an application
- Submit a query

- Process the returned results

The advanced sample application does the same tasks as the simple sample except that it processes the returned results differently than the simple sample.

The simple sample application (SearchExample.java) and the advanced sample application (AdvancedSearchExample.java) are in the following default directories:
- AIX, Linux, and Solaris: `/opt/IBM/es/samples/siapi`
- Windows: `C:\Program Files\IBM\es\samples\siapi`

Browse and navigation sample application

The `BrowseExample` class provides a sample application that accesses a collection's taxonomy tree and displays some basic navigation properties.

This sample demonstrates how to:
- Obtain the browse factory
- Obtain a browse service
- Obtain a browser reference
- Get and display the root category
- Get the root's first child category
- Display the child category and its path from root

The sample BrowseExample.java application is in the following directories:
- AIX, Linux, and Solaris: `/opt/IBM/es/samples/siapi`
- Windows: `C:\Program Files\IBM\es\samples\siapi`

Retrieve all search results sample application

This sample code shows how to set a query to return unsorted results and loop over the query results. You can obtain only a maximum of 500 sorted results for your queries. However, you can obtain all unsorted results.

The following sample code shows you how to:
- Obtain a SearchFactory and a Searchable object
- Create a new Query object
- Set the query to return unsorted results
- Run the search

Obtain a SearchFactory and a Searchable object

Obtain a SearchFactory and a Searchable object as explained in "Simple and advanced sample search applications" on page 68 sample.

```
SearchFactory factory;
Searchable searchable;

... // obtain a SearchFactory and Searchable object
```

Create a new Query object

```
Query q = factory.createQuery("big apple");
```

Set the query to return unsorted results

```
q.setSortKey(Query.SORT_KEY_NONE);
```

Run the search

Run the query in a loop to obtain one page of results at a time. The maximum result page size that is allowed in enterprise search is 100.

When you receive the results pages, you need to interpret the `getAvailableNumberOfResults` method and `getEstimatedNumberOfResults` method differently from the way that you interpret them for sorted query results:

- The `getEstimatedNumberOfResults` method always returns 0 because enterprise search does not provide a number-of-results estimate for unsorted results.
- The `getAvailableNumberOfResults` method returns one of two values: 0 if this is the last result page, and 1 if more results exist.
- You can use the length of the array that is returned by the `getResults` method to find out how many results are within this result page.

```
int fromResult = 0;
int pageSize = 100;
boolean moreResults = true;

// loop over query results, pageSize results at a time
while (moreResults) {

 // set the result range for the next page of results
 q.setRequestedResultRange(fromResult, pageSize);

 // execute the search
 ResultSet resultPage = s.search(q);

 // loop over the results from the ResultSet
 Result[] results = resultPage.getResults();
 for (int i=0;i<results.length;i++) {
... // process result
 }

 // check if there are more available results
 moreResults = (resultPage.getAvailableNumberOfResults() == 1);

 // modify the range for getting the next page of results
 fromResult += pageSize;
}
```

Fetch document content sample application

This sample code shows how to fetch the content of documents that cannot be viewed by clicking a clickable URI in the search results.

For a complete example, see the sample program, FetchSearchExample, in the `ES_INSTALL_ROOT/samples/siapi` directory.

The `fetch` API provides the `com.ibm.es.fetch` package in the `esapi.jar` file and the following interfaces:

- `com.ibm.es.fetch.Document`
- `com.ibm.es.fetch.Fetcher`
- `com.ibm.es.fetch.FetchRequest`
- `com.ibm.es.fetch.FetchService`

- `com.ibm.es.fetch.FetchServiceFactory`

You can use these classes the same way that you use other SIAPI classes.

Fetching a document

First, create the factory object. Using this factory class, create the `FetchService` object and `FetchRequest` object. The Fetcher class can be created through the `FetchService` object. You can then get the `Document` object by calling the `fetch` method of the `Fetcher` object. Finally, you can get the binary data by calling the `getBytes` method of the `Document` object.

```
// obtain the Fetch Service Factory factory implementation
Class cls = Class.forName("com.ibm.es.api.fetch.RemoteFetchFactory");
FetchServiceFactory factory = (FetchServiceFactory) cls.newInstance();

// create a valid Application ID that will be used
// by the Search Node to authorize this access to the collection
ApplicationInfo applicationInfo = factory.createApplicationInfo(applicationName);

// obtain the Fetch Service implementation
FetchService fetchService = factory.getFetchService(config);
// create a new Fetch Request object using the specified uri string
FetchRequest fetchRequest = factory.createFetchRequest(uri, null);
// obtain a Fetcher object to the specified collection ID
Fetcher fetcher = fetchService.getFetcher(applicationInfo, collectionId);

// execute the search by calling the Fetcher's fetch method.
// A Document object will be returned
Document doc = fetcher.fetch(fetchRequest);

// dump the binary content of the document
byte[] buf = doc.getBytes();
```

Enforcing document security

You can set ACL constraints to the `FetchRequest` object. If its value is set, ACL constraints will be delivered to the search server, and the search server will verify the user's authority to access the document by checking the ACL constraints.

```
String aclConstraints = (String) parameters.get("SecurityContext");
aclConstraints = "@SecurityContext::'" + aclConstraints + "'";
FetchRequest fetchRequest = factory.createFetchRequest(uri, aclConstraints);
```

The ACL constraints value is a String value that must conform to the SIAPI format.

Federated search sample application

The `FederatedSearchExample` class provides a simple example of the minimum tasks that are required to submit a federated search to the search server.

The FederatedSearchExample application shows how to:

- Obtain a `RemoteFederator` object with a federator ID. This ID is the same as the ApplicationInfo object ID.
- Create a new query object.
- Set the result range.
- Run the search by calling the `RemoteFederator` object's default search method.

The `FederatedSearchExample.java` file is in the following directories:

- AIX, Linux, and Solaris: `/opt/IBM/es/samples/siapi`
- Windows: `C:\Program Files\IBM\es\samples\siapi`

Sample administration applications

With the sample administration applications, you can create an application ID, create or destroy a collection, enable or disable a collection for indexing, enable or disable a collection for searching, add or remove documents, or build an index.

The sample administration applications are available after you install the client toolkit (the `es.siapi.toolkit.jar` archive file):

- The sample code is available in the `ES_INSTALL_ROOT/samples/siapiAdmin` directory.
- The Javadoc documentation is available in the `ES_INSTALL_ROOT/docs/api/siapi` directory.

Sample: create an application ID

You must register an application ID before you create an application ID. See "Registering application IDs" on page 45 for a sample application.

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Creates a application ID on the omnifind server.
 * An authenticated application ID is required to
 * administer Omnifind server.
 *
 */
public class CreateApplicationID {
public static void main(String[] args) {
try {
 // instiantiate the admin factory
AdminFactory factory = SiapiAdminImpl.createAdminFactory
("com.ibm.es.siapi.admin.AdminFactoryImpl");
if(factory != null){
String appID = "SIAPI-App";
String appPw = "password";
ApplicationInfo appInfo = factory.createApplicationInfo(appID,appPw);
if(appInfo != null){
System.out.println("Application ID called " +
appInfo.getId() + " was created successfully." );
} else {
System.out.println("Application ID called " +
appInfo.getId() + " was not created!!" );
}

// change the password
appInfo.setPassword("search");
System.out.println("New password was successfully set." );
}
} catch (SiapiException e) {
e.printStackTrace();
System.out.println(e.getMessage());
}

}
}
```

Sample: create a collection

```
import java.util.Properties;

import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.es.siapi.common.ApplicationInfoImpl;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Creates a sample collection on the omnifind server
 * and associates the collection
 * to a already created application ID.
 *
 * @see CreateApplicationID
  */
public class CreateCollection {

public static void main(String[] args) {
  try {
    AdminFactory factory =
SiapiAdminImpl.createAdminFactory
("com.ibm.es.siapi.admin.AdminFactoryImpl");
      if (factory != null) {
        AdminService service = factory.getAdminService(null);
        if (service != null) {
          // get handle to a created application ID
          ApplicationInfo appInfo = factory.createApplicationInfo
          ("SIAPI-App","password");
           Properties config = new Properties();
           // specify custom data directory
          config.setProperty(IAdminConstants.KEY_INDEX_LOCATION,
          "/home/esadmin/siapidata");
           // specify optional n-gram option
          config.setProperty(IAdminConstants.KEY_ENABLE_COLLECTION_NGRAM,
          "true");
           // specify optional security option
          config.setProperty
          (IAdminConstants.KEY_ENABLE_COLLECTION_SECURITY, "true");
           // specify optional max index in the collection
          config.setProperty(IAdminConstants.KEY_MAX_DOCS_IN_INDEX, "10000");

          String colID = "col_123";
          String colLabel = "SIAPI Collection";
          String colLangauge = "en";

        service.createCollection(appInfo,
  colID,
  colLabel,
  0,
  colLangauge,
  config);
       }
    }
  } catch (SiapiException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
}
}
```

Sample: destroy a collection

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
  * Deletes the sample collection created by CreateCollection.
  *
  * @see CreateCollection
  */
public class DestroyCollection {

public static void main(String[] args) {
  try {
    AdminFactory factory =
SiapiAdminImpl.createAdminFactory("com.ibm.es.siapi.admin.AdminFactoryImpl");
if (factory != null) {
  ApplicationInfo appInfo = factory.createApplicationInfo
  ("SIAPI-App","password");
      AdminService service = factory.getAdminService(null);
      if (service != null) {
        // collection ID is required to destroy the collection
        String colID = "col_123";
        service.destroyCollection(appInfo, colID);
      }
    }
  } catch (SiapiException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
}
}
```

Sample: enable a collection for indexing

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Enables a collection for indexing.
 * The parser driver and the data listener sessions
 * are checked and started.
 *
*/
public class EnableCollectionForIndexing {
public static void listIndexableCollectionIDs
(AdminService service, ApplicationInfo appInfo)
throws SiapiException{

  String [] indexables = service.getIndexableCollectionIDs(appInfo);
  if (indexables != null && indexables.length > 0) {
    System.out.println("Following are indexable collection ids:");
    for (int i=0; i<indexables.length; ++i) {
      System.out.println(indexables[i]);
    }
  }
}

  public static void main(String[] args) {
    try {
      AdminFactory factory = SiapiAdminImpl.createAdminFactory
     (IAdminConstants.ADMIN_FACTORY_IMPL);
    if (factory != null) {
      ApplicationInfo appInfo = factory.createApplicationInfo
      ("SIAPI-App","password");
      AdminService service = factory.getAdminService(null);
      if (service != null) {
        String colID = "col_123";
        listIndexableCollectionIDs(service, appInfo);
        boolean enabled = service.isEnabledForIndexing(appInfo, colID);
        if (!enabled) {
          // enable for indexing
          System.out.println("Enabling collection for indexing");
          service.enableCollectionForIndexing(appInfo, colID, null);
        }
        //check if enabled for indexing
        enabled = service.isEnabledForIndexing(appInfo, colID);
        if (enabled) {
          System.out.println("Collection is enabled for indexing");
        } else {
          System.out.println("Collection is not enabled for indexing");
        }
    listIndexableCollectionIDs(service, appInfo);
    }
   }
  } catch (SiapiException e) {
    e.printStackTrace();
  }
 }
}
```

Sample: disable a collection for indexing

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Disables a collection for indexing.
 *
  */
public class DisableCollectionForindexing {

  public static void main(String[] args) {
    try {
      AdminFactory factory = SiapiAdminImpl.createAdminFactory
(IAdminConstants.ADMIN_FACTORY_IMPL);
      if (factory != null) {
        ApplicationInfo appInfo = factory.createApplicationInfo
        ("SIAPI-App","password");
        AdminService service = factory.getAdminService(null);
        if (service != null) {

          String colID = "col_123";

          boolean enabled = service.isEnabledForIndexing(appInfo, colID);
          if (enabled) {
            // enable for indexing
            System.out.println("Disabling collection for indexing");
            service.disableCollectionForIndexing(appInfo, colID, null);
          }
          //check if collection is disabled for indexing
          enabled = service.isEnabledForIndexing(appInfo, colID);
          if (enabled) {
            System.out.println("Collection is enabled for indexing");
          } else {
            System.out.println("Collection is not enabled for indexing");
          }
         }
      }
    } catch (SiapiException e) {
      e.printStackTrace();
    }
  }
}
```

Sample: enable a collection for search

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Enables a collection for searching.
 * The search runtimes for the specified collection are started
 *
 */
public class EnableCollectionForSearch {

  public static void listSearchableCollectionIDs
(AdminService service, ApplicationInfo appInfo) throws SiapiException{
    String [] searchables = service.getSearchableCollectionIDs(appInfo);
    if (searchables != null && searchables.length > 0) {
      System.out.println("Following are searchable collection ids:");
      for (int i=0; i<searchables.length; ++i) {
        System.out.println(searchables[i]);
      }
    }
  }


  public static void main(String[] args) {
    try {
      AdminFactory factory = SiapiAdminImpl.createAdminFactory
(IAdminConstants.ADMIN_FACTORY_IMPL);
      if (factory != null) {
        ApplicationInfo appInfo = factory.createApplicationInfo
        ("SIAPI-App","password");
        AdminService service = factory.getAdminService(null);
        if (service != null) {
          String colID = "col_123";
          listSearchableCollectionIDs(service, appInfo);
          boolean enabled = service.isEnabledForSearch(appInfo, colID, null);
          if (!enabled) {
            // enable for searching
            System.out.println("Enabling collection for searching");
            service.enableCollectionForSearch(appInfo, colID, null);
          }
          //check if enabled for searching
          enabled = service.isEnabledForSearch(appInfo, colID, null);
          if (enabled) {
            System.out.println("Collection is enabled for searching");
          } else {
            System.out.println("Collection is not enabled for searching");
          }

          listSearchableCollectionIDs(service, appInfo);
        }
      }
    } catch (SiapiException e) {
      e.printStackTrace();
    }
  }
}
```

Sample: disable a collection for search

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 *
 * Disables a collection for search.
 *  The search runtimes for the specified collection
 *  are stopped.
 */
public class DisableCollectionForSearch {

  public static void main(String[] args) {
    try {
        AdminFactory factory = SiapiAdminImpl.createAdminFactory
        (IAdminConstants.ADMIN_FACTORY_IMPL);
        if (factory != null) {
          ApplicationInfo appInfo = factory.createApplicationInfo
          ("SIAPI-App","password");
          AdminService service = factory.getAdminService(null);
          if (service != null) {

            String colID = "col_123";

            boolean enabled = service.isEnabledForSearch
            (appInfo, colID, null);
            if (enabled) {
              // disable for searching
              System.out.println("Disabling collection for indexing");
              service.disableCollectionForSearch(appInfo, colID, null);
            }
            //check if enabled for searching
            enabled = service.isEnabledForSearch(appInfo, colID, null);
            if (enabled) {
              System.out.println("Collection is enabled for searching");
            } else {
              System.out.println("Collection is not enabled for searching");
            }
          }
        }
    } catch (SiapiException e) {
      e.printStackTrace();
    }
  }
}
```

Sample: add documents to an index

```
import java.util.Date;

import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;
import com.ibm.siapi.index.Document;
import com.ibm.siapi.index.Field;
import com.ibm.siapi.index.Index;
import com.ibm.siapi.index.IndexFactory;
import com.ibm.siapi.index.IndexService;
import com.ibm.siapi.index.IndexStats;
import com.ibm.siapi.index.SiapiIndexImpl;

/**
 * Add documents to a index.
 *
 * Type comment
 */
public class AddDocumentsAndFields {

  public static void main(String[] args) {
    try {
      IndexFactory iFactory = SiapiIndexImpl.createIndexFactory
(IAdminConstants.INDEX_FACTORY_IMPL);
      AdminFactory aFactory = SiapiAdminImpl.createAdminFactory
(IAdminConstants.ADMIN_FACTORY_IMPL);

      if (iFactory != null) {
        ApplicationInfo appInfo = iFactory.createApplicationInfo
        ("SIAPI-App","password");
        if (aFactory != null) {
        // enable collection for indexing
        AdminService aService = aFactory.getAdminService(null);
        if (aService != null) {
          aService.enableCollectionForIndexing(appInfo, "col_123", null);
        }
        if (!aService.isEnabledForIndexing(appInfo, "col_123")) {
          System.out.println("Sorry, can't add document to the index.
                              The parser driver can not be started");
          System.exit(0);
        }
      }

      IndexService service = iFactory.getIndexService(null);
      if (service != null) {
        Index index = service.getIndex(appInfo, "col_123");
        if (index == null) {
          System.out.println("Index instance could not be
                              instantiated.. return now!!");
          System.exit(0);
        }

        for (int i=0;i<3; ++i) {
          Document doc = iFactory.createDocument( "docid" + i,
                                        "My document id="+i,
                                        "custom" + i,
                                        "custom" + i);


          doc.setDate(new Date());
          doc.setLanguage("en");
          doc.setRawContentFormat("text/plain");

          // set security tokens
          /*
          String[] nativeACL = new String [1];

          if(i == 0){
            nativeACL[0] = "<NativeACL><Groups><Group>Admins
            </Group><Group>Staff</Group></Groups></NativeACL";
          } else if (i== 1){
            nativeACL[0] = "Admin,staff";
          } else {
            nativeACL[0] = "";
          }
          doc.setACL(nativeACL);
          */

          // create fields
          Field field1 = iFactory.createField("Title", "Mr. Srinivas");
          field1.setContentSearchable(true);
          field1.setFieldSearchable(true);
          field1.setParametric(false);
          field1.setReturnable(true);
          field1.setConflictResolutionPolicy
          (Field.CONFLICT_FIELD_OVERRIDES);


          Field field2 = iFactory.createField("Address",
                        "200 Gibralter Drive, USA");
          field2.setContentSearchable(true);
          field2.setFieldSearchable(true);
          field2.setParametric(false);
          field2.setReturnable(true);
          field2.setConflictResolutionPolicy
          (Field.CONFLICT_CONTENT_OVERRIDES);

          Field field3 = iFactory.createField("Street number", 200);
          field3.setContentSearchable(true);
          field3.setFieldSearchable(true);
          field3.setParametric(true);
          field3.setReturnable(true);
          field3.setConflictResolutionPolicy(Field.CONFLICT_COEXIST);

          doc.addField(field1);
          doc.addField(field2);
          doc.addField(field3);

          // add document to the index
          index.addDocument(doc);
        }
        // get statistics of documents in the store
        IndexStats stats = index.getStatistics();
        if (stats != null && stats.getNumPendingUpdates() > 0) {
          // build main index
          index.reorganize();
        }
       }

      }
```

Sample: remove documents from an index

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.common.ApplicationInfo;
import com.ibm.siapi.index.Index;
import com.ibm.siapi.index.IndexFactory;
import com.ibm.siapi.index.IndexService;
import com.ibm.siapi.index.IndexStats;
import com.ibm.siapi.index.SiapiIndexImpl;

/**
 * Remove specified documents from the index.
 *
 * Type comment
 */
public class RemoveDocument {
  public static void main(String[] args) {
    try {
      IndexFactory factory =
SiapiIndexImpl.createIndexFactory(IAdminConstants.INDEX_FACTORY_IMPL);
      if (factory != null) {
        ApplicationInfo appInfo =
        factory.createApplicationInfo("SIAPI-App","password");
        IndexService service = factory.getIndexService(null);
        if (service != null) {
          Index index = service.getIndex(appInfo, "col_123");
          if (index == null) {
            System.out.println("Index instance could not be
                                instantiated.. return now!!");
            System.exit(0);
          }

          index.removeDocument("docid0");

        }
      }
    } catch (SiapiException e) {
      e.printStackTrace();
      System.out.println(e.getLocalizedMessage());
    }
  }
}
```

Sample: build an index

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.common.ApplicationInfo;
import com.ibm.siapi.index.Index;
import com.ibm.siapi.index.IndexFactory;
import com.ibm.siapi.index.IndexService;
import com.ibm.siapi.index.IndexStats;
import com.ibm.siapi.index.SiapiIndexImpl;

/**
 * Build index with documents.
 * The
 * Type comment
 */
public class BuildIndex {

  public static void main(String[] args) {
    try {
      IndexFactory factory =
SiapiIndexImpl.createIndexFactory(IAdminConstants.INDEX_FACTORY_IMPL);
      if (factory != null) {
        ApplicationInfo appInfo =
  factory.createApplicationInfo("SIAPI-App","password");

        IndexService service = factory.getIndexService(null);
        if (service != null) {
          Index index = service.getIndex(appInfo, "col_123");
          if (index == null) {
            System.out.println("Index instance could not be
                                instantiated.. return now!!");
            System.exit(0);
          }

          // set fragmentation count to 2
          // after every 2 delta build, the index
          // will be reorganized
          index.setProperty
          (IAdminConstants.BUILD_FRAGMENTATION_COUNT, "2");
          // get statistics of documents in the store
          IndexStats stats = index.getStatistics();
          if (stats != null && stats.getNumPendingUpdates() > 0) {
            // build index
            index.build();
          }
        }
      }
    } catch (SiapiException e) {
      e.printStackTrace();
      System.out.println(e.getLocalizedMessage());
    }
  }
}
```

Sample: revisiting URLs

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
* PerformAdminCommand provides sample code to revisit a specified URL.
* Before runningthis sample, a Web crawler has to be associated with the
* specified collection. Use Admin GUI to create the web crawler. Also
* contact your administrator to learn about the collection ID that
* is associated to the web crawler.
*/



public class PerformAdminCommand {
public static void main(String[] args) {
String appName ;
String appPwd ;
String colID

if(args.length < 3){
System.out.println("Missing arguments");
System.out.println("Usage: java PerformAdminCommand <appName>
<appPassword> <collection ID>>");
System.exit(1);
}
try {
appName = args[0];
appPwd = args[1];
colID = args[2];

AdminFactory factory = SiapiAdminImpl.createAdminFactory
(IAdminConstants.ADMIN_F
ACTORY_IMPL);
if(factory != null){
ApplicationInfo appInfo = factory.createApplicationInfo
(appName, appPwd);

AdminService service = factory.getAdminService(null);
if(service != null){
// Isolate multiple URLs with a space
String command =
"revisitURLs http://www.ibm.com/us http://www.ibm.com/products/
*";
String status = service.performAdminCommand(appInfo, colID, command, null);
System.out.println ("status on the command = " + status);
}
}
} catch (SiapiException e) {
e.printStackTrace();
System.out.println(e.getLocalizedMessage());
}
}
}
```

Related tasks

"Installing the client toolkit" on page 2

Compiling the sample administration applications

The sample administration applications must be compiled with the IBM Software Development Kit (SDK) for Java 1.4.2, Service Release 5 (SR5).

Restrictions

Before you can build Java applications for enterprise search, you must install and configure Apache ANT, a Java-based build tool. For information about how to install and configure Apache ANT, see http://ant.apache.org/.

About this task

The following steps assume that you installed the client toolkit (the `es.siapi.toolkit.jar` archive file) and that you have the required JDK and ANT compiler.

Procedure

To compile and run a sample administration application:

1. From the command line, change to the `ES_INSTALL_ROOT/samples/siapiAdmin` directory. The default installation paths are:
   - AIX: `/usr/IBM/es/samples/siapiAdmin`
   - Linux and Solaris: `/opt/IBM/es/samples/siapiAdmin`
   - Windows: `C:\Program Files\IBM\es\samples\siapiAdmin`

   The directory includes a `build.xml` file that ANT uses to build the file.
2. Run the ANT script:
   ```
   ant
   ```
   You see the following message after the Java source code compiles:
   ```
   BUILD SUCCESSFUL
   Total time: xx seconds
   ```
3. Edit the `runSample` script (`runSample.sh` or `runSample.bat`) and ensure that the class path and the path to the enterprise search `es.cfg` file are correct.
4. Run the application by specifying the following command on one line, where *`admin_app`* is the administration application that you want to compile:

   AIX, Linux, and Solaris
   ```
   runSample.sh admin_app
   ```
   Windows
   ```
   runSample.bat admin_app
   ```

Related tasks

"Installing the client toolkit" on page 2

"Compiling the sample search applications" on page 67

Sample plug-in application for non-Web crawlers

The sample crawler plug-in application shows how you can change security token values, metadata, and the content of crawled documents.

```
package sample;

import java.io.BufferedWriter;
import java.io.OutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.List;

import com.ibm.es.crawler.plugin.AbstractCrawlerPlugin;
import com.ibm.es.crawler.plugin.Content;
import com.ibm.es.crawler.plugin.CrawledData;
import com.ibm.es.crawler.plugin.CrawlerPluginException;
import com.ibm.es.crawler.plugin.FieldMetadata;

/**
 * The <code>MyCrawlerPlugin</code> is a sample crawler plugin module.
 */
public class MyCrawlerPlugin extends AbstractCrawlerPlugin {


   /**
    * Default constructor.
    */
   public MyCrawlerPlugin() {
      super();
   }

   /**
    * Initialize this object.
    *
    * This sample program has nothing in this method.
    *
    * @see com.ibm.es.crawler.plugin.AbstractCrawlerPlugin#init()
    */
   public void init() throws CrawlerPluginException {

      /*
       * [Tips]
       * If your crawler plugin module requires something to do for
       * initialization, add the code here.
       * [Example]
       * Get JDBC connection for your local system.
       * connection = DriverManager.getConnection("jdbc::db2::xxxx);
       */

   }

   /**
    * Returns the boolean value for metadata usage.
    *
    * This sample program returns <code>true</code>.
    *
    * @see com.ibm.es.crawler.plugin.AbstractCrawlerPlugin#isMetadataUsed()
    */
   public boolean isMetadataUsed() {

      /*
       * [Tips]
       * If your crawler plugin module updates both metadata and security
       * tokens, returns ture.
       * If your cralwer plugin module updates security tokens only,
     * returns false.
     * [Example]
     * Close JDBC connection for your local system.
     * connection.close();
     */
    return true;
}

/**
 * Terminate this object.
 *
 * This sample program has nothing in this method.
 *
 * @see com.ibm.es.crawler.plugin.AbstractCrawlerPlugin#term()
 */
public void term() throws CrawlerPluginException {

    /*
     * [Tips]
     * If your crawler plugin module requires something to do
     * for termination, add the code here.
     */

    return;

}

/**
 * Update crawled data.
 *
 * This sample program updates the security tokens.
 *
 * @see com.ibm.es.crawler.plugin.AbstractCrawlerPlugin#updateDocument
    (com.ibm.es.crawler.plugin.CrawledData)
 */
public CrawledData updateDocument(CrawledData crawledData)
throws CrawlerPluginException {

    // Get uri string, security tokens, and field metadata
    String url = crawledData.getURI();
    String securityTokens = crawledData.getSecurityTokens();
    List metadataList = crawledData.getMetadataList();
    if (metadataList == null) {
       metadataList = new ArrayList();
    }

    /*
     * [Tips]
     * If your crawler plugin module rejects some crawled data,
     * add the check code here and returns null.
     */
    // This sample always returns updated document.
    if (false) {
       return null;
    }

    /*
     * [Tips]
     * If your crawler plugin module updates the security tokens,
     * add the code here.
     */
    // update security token (for sample)
    String newToken = "SampleToken";
    String newSecurityTokens = securityTokens + "," + newToken;
    crawledData.setSecurityTokes(newSecurityTokens);

    /*
```

```
         * [Tips]
         * If your crawler plugin module updates metadata,
         * add the code here.
         */
        // update metadata (for sample)
        FieldMetadata newFieldMetaData = new FieldMetadata("copyright", "IBM");
        metadataList.add(newFieldMetaData);
        crawledData.setMetadataList(metadataList);


        /*
         * Set language.
         */
        crawledData.setLanguage("en");
        crawledData.setLanguageAutoDetection(true);

        /*
         * Update Content. since 8.3
         */
        Content content = crawledData.getOriginalContent();

        java.io.InputStream in = null;

        try{
           // if the original crawled content is null, create the new content.
           if(content == null){
              crawledData.createNewContent();
              content = crawledData.createNewContent();
           } else {
              // if the original crawled content exists, get InputStream
              // object to access it.
              in = content.getInputStream();

              // read the content

              in.close();
           }
        }catch(IOException ioe){
           throw new CrawlerPluginException(ioe);
        }

        // set information against the content.
        content.setCodepage("UTF-8");
        content.setCodepageAutoDetection(true);
        content.setMimeType("text/plain");

  // Overwrite the content.
        try{

           OutputStream outputStream = content.getOutputStream();

           // write content to OutputStream
           String newText = "The new content of plain text ";
           BufferedWriter br = new BufferedWriter(new OutputStreamWriter
           (outputStream, "UTF-8"));
           br.write(newText);
           br.flush();
           br.close();

        }catch(IOException ioe){
           throw new CrawlerPluginException(ioe);
        }

        // Submit change for the content.
        crawledData.submitContent(content);

        return crawledData;
```

```
    }

    /* (non-Javadoc)
     * @see com.ibm.es.crawler.plugin.AbstractCrawlerPlugin#isContentUsed()
     */
    public boolean isContentUsed() {
       return true;
    }

}
```

Related concepts

"Crawler plug-ins for non-Web sources" on page 53

Related tasks

"Creating a crawler plug-in for non-Web data sources" on page 54

Sample code for Web services

Sample code is provided for a Web Services Definition Language (WSDL) file, an XML schema (XSD) file, a sample client proxy application for Web services, and a detailed sample client application that demonstrates how to control query behavior.

A sample Web service application is available after you install the client toolkit (the `es.siapi.toolkit.jar` archive file). See the `ES_INSTALL_ROOT/samples/siapiAdmin/webservices` directory.

Related concepts

"Web services for enterprise search" on page 51

WSDL for Web services

The Web Services Definition Language (WSDL) file exposes several Web service search functions.

Search functions

SearchResponse search(SearchRequest *request***)**
This function performs a federated search. The search terms, application ID, search options, and so on, can be specified in the search request instance. The federated search can be overridden by specifying a particular collection ID.

CollectionInfo[] getAvailableCollections(ApplicationInfo *appInfo***)**
This function returns an array of searchable collections. Each instance of collection information consists of an ID and a label. The ID can be specified in the search request instance to do collection-specific searches.

FieldInfo[] getAvailableFields(ApplicationInfo *appInfo***)**
This function returns a list of federated fields. The FieldInfo instance contains the names and properties of fields defined across all of the collections. The field name and properties can be used for performing fielded queries.

String[] getAvailableDocumentSources(ApplicationInfo *appInfo***)**
This function returns an array of document types configured for crawling. The document type can be used to invoke searches per document type.

String[] getAvailableAttributeValues(int *attributeType***)**
This function returns an array of attributes that match the specified attribute type. The accepted integer values are:

- Document language type is -1

- Document source type is -2
- Document type is -3

SpellCorrection[] getSpellCorrections(String *queryTerm*)
: This function returns an array of spelling corrections for a specified search term. The spelling corrections are federated across all of the configured collections.

SynonymExpansion[] getSynonymExpansions(String *queryTerm*)
: This function returns an array of synonyms for a specified search term. The synonym expansions are federated across all of the configured collections.

WSDL file for enterprise search

The following WSDL file and the associated XML schema define the preceding search functions. You can access the WSDL file for enterprise search at the following URL:

http://*your_search_server*/ESSearchServer/wsdl/com/ibm/es/ws6/server/search/ofsearch.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:ns="http://www.ibm.com/of/types"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/of/search"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  name="ofsearch" targetNamespace="http://www.ibm.com/of/search">
 <wsdl:import location="ofTypes.xsd" namespace="http://www.ibm.com/of/types">
 </wsdl:import>
 <wsdl:types>
 </wsdl:types>
 <wsdl:message name="searchResponse">
  <wsdl:part element="ns:SearchResponse" name="searchResponse"/>
 </wsdl:message>
 <wsdl:message name="searchRequest">
  <wsdl:part element="ns:SearchRequest" name="searchRequest"/>
 </wsdl:message>
 <wsdl:message name="siapiException">
  <wsdl:part element="ns:SiapiException" name="siapiException">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="collectionInfo">
  <wsdl:part element="ns:Collections" name="collectionInfo">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="availableFields">
  <wsdl:part element="ns:AvailableFields" name="availableFields">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="applicationInfo1">
  <wsdl:part name="applicationInfo1" type="ns:ApplicationInfo">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="documentSources">
  <wsdl:part name="documentSources" type="ns:DocumentSourceArray">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="applicationInfo2">
  <wsdl:part name="applicationInfo2" type="ns:ApplicationInfo">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="applicationInfo3">
  <wsdl:part name="applicationInfo3" type="ns:ApplicationInfo"/>
 </wsdl:message>
```

```
<wsdl:message name="synonymExpansions">
 <wsdl:part name="synonymExpansions" type="ns:SynonymExpansionArray">
 </wsdl:part>
</wsdl:message>
<wsdl:message name="synonymExpansionTerm">
 <wsdl:part name="synonymExpansionTerm" type="xsd:string">
 </wsdl:part>
</wsdl:message>
<wsdl:message name="spellCorrections">
 <wsdl:part name="spellCorrections" type="ns:SpellCorrectionArray">
 </wsdl:part>
</wsdl:message>
<wsdl:message name="spellCorrectionTerm">
 <wsdl:part name="spellCorrectionTerm" type="xsd:string">
 </wsdl:part>
</wsdl:message>
<wsdl:message name="attributeValues">
 <wsdl:part name="attributeValues" type="ns:DocumentAttributeArray">
 </wsdl:part>
</wsdl:message>
<wsdl:message name="attributeType">
 <wsdl:part name="attributeType" type="xsd:int"/>
</wsdl:message>
<wsdl:portType name="ofsearch">
 <wsdl:operation name="search">
  <wsdl:documentation>
   This function should be used to execute federated
   searches. Searches could also be limited to particular
   collections.
  </wsdl:documentation>
  <wsdl:input message="tns:searchRequest"/>
  <wsdl:output message="tns:searchResponse"/>
  <wsdl:fault message="tns:siapiException" name="siapiException">
  </wsdl:fault>
 </wsdl:operation>
 <wsdl:operation name="getAvailableCollections">
  <wsdl:documentation>
   This function returns an array of collection information
   instances for a specified application ID. The collection
   information consists of the collection title and
   internal id. The internal ID could be used to execute
   searches for specific collections.
  </wsdl:documentation>
  <wsdl:input message="tns:applicationInfo2"/>
  <wsdl:output message="tns:collectionInfo"/>
  <wsdl:fault message="tns:siapiException" name="siapiException">
  </wsdl:fault>
 </wsdl:operation>
 <wsdl:operation name="getAvailableFields">
  <wsdl:documentation>
   This function returns an array of fields configured
   across all the collections for a specified application
   ID. The field information could be used to execute
   fielded queries.
  </wsdl:documentation>
  <wsdl:input message="tns:applicationInfo1"/>
  <wsdl:output message="tns:availableFields"/>
  <wsdl:fault message="tns:siapiException" name="siapiException">
  </wsdl:fault>
 </wsdl:operation>
 <wsdl:operation name="getAvailableDocumentSources">
  <wsdl:documentation>
   This function returns an array of source types
   configured across all collections for a specified
   application ID. The source types could be used to
   understand the unstructured content in the collection.
   This information could also used to execute searches
```

```
     based on source types.
    </wsdl:documentation>
    <wsdl:input message="tns:applicationInfo3"/>
    <wsdl:output message="tns:documentSources"/>
    <wsdl:fault message="tns:siapiException" name="siapiException">
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getSynonymExpansions">
    <wsdl:documentation>
     This function returns an array of synonyms for a
     specified search term. The synonym expansions will be
     federated across all the configured collections.
    </wsdl:documentation>
    <wsdl:input message="tns:synonymExpansionTerm"/>
    <wsdl:output message="tns:synonymExpansions"/>
    <wsdl:fault message="tns:siapiException" name="siapiException">
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getSpellCorrections">
    <wsdl:documentation>
     This function returns an array of spell corrections for
     a specified search term. The spell corrections will be
     federated across all the configured collections.
    </wsdl:documentation>
    <wsdl:input message="tns:spellCorrectionTerm"/>
    <wsdl:output message="tns:spellCorrections"/>
    <wsdl:fault message="tns:siapiException" name="siapiException">
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getAvailableAttributeValues">
    <wsdl:documentation>
     This function returns a array of attributes that matches
     the specifed type. Following are the list of accepted
     integer values: Document language type is -1 Document
     source type is -2 Document type is -3
    </wsdl:documentation>
    <wsdl:input message="tns:attributeType"/>
    <wsdl:output message="tns:attributeValues"/>
    <wsdl:fault message="tns:siapiException" name="siapiException">
    </wsdl:fault>
   </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ofsearchBinding" type="tns:ofsearch">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="search">
    <soap:operation soapAction="http://www.ibm.com/of/search/search"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="siapiException">
     <soap:fault name="siapiException" use="literal"/>
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getAvailableCollections">
    <soap:operation soapAction="http://www.ibm.com/of/search/getAvailableCollections"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="siapiException">
     <soap:fault name="siapiException" use="literal"/>
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getAvailableFields">
    <soap:operation soapAction="http://www.ibm.com/of/search/getAvailableFields"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="siapiException">
     <soap:fault name="siapiException" use="literal"/>
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getAvailableDocumentSources">
    <soap:operation soapAction="http://www.ibm.com/of/search/getAvailableDocumentSources"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="siapiException">
     <soap:fault name="siapiException" use="literal"/>
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getSynonymExpansions">
    <soap:operation soapAction="http://www.ibm.com/of/search/getSynonymExpansions"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="siapiException">
     <soap:fault name="siapiException" use="literal"/>
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getSpellCorrections">
    <soap:operation soapAction="http://www.ibm.com/of/search/getSpellCorrections"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="siapiException">
     <soap:fault name="siapiException" use="literal"/>
    </wsdl:fault>
   </wsdl:operation>
   <wsdl:operation name="getAvailableAttributeValues">
    <soap:operation soapAction="http://www.ibm.com/of/search/getAvailableAttributeValues"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="siapiException">
     <soap:fault name="siapiException" use="literal"/>
    </wsdl:fault>
   </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="ofsearch">
   <wsdl:port binding="tns:ofsearchBinding" name="ofsearchBinding">
```

```
    <soap:address location="http://localhost:9081/ESSearchServer/services/ofsearchBinding"/>
   </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

XML schema associated with the WSDL file

The XML schema (XSD file) identifies the required data types.

XSD file for enterprise search

The following XSD file is associated with the WSDL file for enterprise search. You can access the XML schema at the following URL:

http://*your_search_server*/ESSearchServer/wsdl/
com/ibm/es/ws6/server/search/ofTypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:ns="http://www.ibm.com/of/types"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="http://www.ibm.com/of/types">

 <xsd:complexType name="ApplicationInfo">
  <xsd:sequence>
   <xsd:element name="ID" type="xsd:string"/>
   <xsd:element name="password" nillable="true" type="xsd:string"/>
   <xsd:element name="token" nillable="true" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="Property">
  <xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="value" nillable="true" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="ReturnedField">
  <xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="NameValuePair">
  <xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="value" nillable="true" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="Result">
  <xsd:sequence>
   <xsd:element name="date" nillable="true" type="xsd:date"/>
   <xsd:element name="collectionID" nillable="true" type="xsd:string"/>
   <xsd:element name="description" nillable="true" type="xsd:string"/>
   <xsd:element name="documentID" nillable="true" type="xsd:string"/>
   <xsd:element name="documentSource" nillable="true" type="xsd:string"/>
   <xsd:element name="documentURI" nillable="true" type="xsd:string"/>
   <xsd:element name="language" nillable="true" type="xsd:string"/>
   <xsd:element name="title" nillable="true" type="xsd:string"/>
   <xsd:element name="firstOfASite" type="xsd:boolean"/>
   <xsd:element name="score" type="xsd:double"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="categories"
       type="ns:ResultCategory"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="fields"
```

```
      type="ns:NameValuePair"/>
  <xsd:element name="documentType" type="xsd:string"/>
  <xsd:element name="author" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CategoryInfo">
 <xsd:sequence>
  <xsd:element name="ID" type="xsd:string"/>
  <xsd:element name="label" nillable="true" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ResultCategory">
 <xsd:sequence>
  <xsd:element name="confidence" type="xsd:double"/>
  <xsd:element name="taxonomyID" nillable="true" type="xsd:string"/>
  <xsd:element name="info" nillable="true" type="ns:CategoryInfo"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="pathFromRoot"
      type="ns:CategoryInfo"/>
 </xsd:sequence>
</xsd:complexType>

<!--  Global elements -->
<xsd:element name="SearchRequest">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="applicationInfo" type="ns:ApplicationInfo"/>
   <xsd:element name="collectionIDs">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="item"
              type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="queryLanguage" nillable="true" type="xsd:string"/>
   <xsd:element name="queryText" type="xsd:string"/>
   <xsd:element name="aclConstraints" nillable="true" type="xsd:string"/>
   <xsd:element name="linguisticMode" type="xsd:int"/>
   <xsd:element name="predefinedResultsEnabled" type="xsd:boolean"/>
   <xsd:element name="queryID" nillable="true" type="xsd:string"/>
   <xsd:element name="firstRequestedResult" type="xsd:int"/>
   <xsd:element name="numRequestedResults" type="xsd:int"/>
   <xsd:element name="resultCategoriesDetailLevel" type="xsd:int"/>
   <xsd:element name="sortKey" nillable="true" type="xsd:string"/>
   <xsd:element name="sortOrder" type="xsd:int"/>
   <xsd:element name="sortPoolSize" type="xsd:int"/>
   <xsd:element name="synonymExpansionMode" type="xsd:int"/>
   <xsd:element name="spellCorrectionEnabled" type="xsd:boolean"/>
   <xsd:element name="siteCollapsingEnabled" type="xsd:boolean"/>
   <xsd:element name="returnedFields">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="item"
              type="ns:ReturnedField"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="properties">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="item"
              type="ns:Property"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="sortKeyByLocale">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="key" type="xsd:string">
      </xsd:element>
      <xsd:element name="locale" type="ns:Locale">
      </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="returnAttribute">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="attributes"
              type="ns:PredefinedAttribute">
      </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="clientLocale" type="ns:Locale"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name="SearchResponse">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="availableNumberOfResults" type="xsd:int"/>
   <xsd:element name="estimatedNumberOfResults" type="xsd:int"/>
   <xsd:element name="queryEvaluationTime" type="xsd:long"/>
   <xsd:element name="hasUnconstrainedResults" type="xsd:int"/>
   <xsd:element name="evaluationTruncated" type="xsd:boolean"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="predefinedResults"
        type="ns:Result"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="results"
        type="ns:Result"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="properties"
        type="ns:Property"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="spellCorrections"
        type="ns:SpellCorrection"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="synonymExpansions"
        type="ns:SynonymExpansion"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="messages"
        type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>


<xsd:element name="SiapiException">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="errorCode" type="xsd:string"/>
   <xsd:element name="message" type="xsd:string"/>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="messages"
        type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:complexType name="SpellCorrection">
 <xsd:sequence>
  <xsd:element name="querySubstring" type="xsd:string"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="suggestions"
      type="xsd:string">
  </xsd:element>
 </xsd:sequence>
```

```
</xsd:complexType>

<xsd:complexType name="SynonymExpansion">
 <xsd:sequence>
  <xsd:element name="querySubstring" type="xsd:string"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="suggestions"
      type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Locale">
 <xsd:sequence>
  <xsd:element name="language" type="xsd:string"/>
  <xsd:element name="country" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PredefinedAttribute">
 <xsd:sequence>
  <xsd:element name="attribute" type="xsd:int"/>
  <xsd:element name="returned" type="xsd:boolean"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FieldInfo">
 <xsd:sequence>
  <xsd:element name="iD" type="xsd:string"/>
  <xsd:element name="type" type="xsd:int"/>
  <xsd:element name="contentSearchable" type="xsd:boolean"/>
  <xsd:element name="exactMatchSupported" type="xsd:boolean"/>
  <xsd:element name="fieldSearchable" type="xsd:boolean"/>
  <xsd:element name="parametric" type="xsd:boolean"/>
  <xsd:element name="returnable" type="xsd:boolean"/>
  <xsd:element name="sortable" type="xsd:boolean"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CollectionInfo">
 <xsd:sequence>
  <xsd:element name="iD" type="xsd:string"/>
  <xsd:element name="label" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="Collections" type="ns:CollectionArray">
 </xsd:element>

<xsd:complexType name="CollectionArray">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="collectionInfos"
      type="ns:CollectionInfo"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="AvailableFields">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="fields"
       type="ns:FieldInfo"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name="DocumentSources" type="ns:DocumentSourceArray"/>

<xsd:complexType name="Documentsource">
 <xsd:sequence>
```

```
   <xsd:element name="source" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="DocumentSourceArray">
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="documentSources"
       type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="DocumentAttributeArray">
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="documentAttributes"
       type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="SpellCorrectionArray">
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="spCorrection"
       type="ns:SpellCorrection"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="SynonymExpansionArray">
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="synExpansion"
       type="ns:SynonymExpansion"/>
  </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

Sample client proxy application for Web services

With the sample Web services application, you can search enterprise search collections remotely.

The sample client proxy application is available after you install the client toolkit (the `es.siapi.toolkit.jar` archive file).

Sample endpoints.properties file

Before you invoke the sample code that is provided in the client toolkit (the `es.siapi.toolkit.jar` archive file), you must edit a configuration file, `endpoints.properties`, to specify information about the search server to use for enterprise search. Shown below is the portion of the `endpoints.properties` file where you identify the search server. This file is bundled with the `es.siapi.toolkit.jar` file.

```
# Configuration file that contains the host names
# and URIs to resolve the Web service endpoint

# omnifind search nodes
SearchServer=yoursearchserver\:80

# Accessible WSDL
# Modify the localhost below to one of the search servers
# One can use this WSDL to develop custom client proxies.
WSDL=http\://your_search_server/ESSearchServer/wsdl/com/ibm/es/ws6/server/search/ofsearch.wsdl

# URI of the webservice end point
# The search servers defined above will be pre-pended to
# the URI below to create a fully qualified path to the
# web service end point.
EndPointURI=/ESSearchServer/services/ofsearchBinding
```

Sample client proxy application

The following sample code is based on the client proxy that is provided in the toolkit. Pass the `endpoints.properties` file as the argument to this sample.

```
import com.ibm.es.ws6.client.OmniFindSearchable;
import com.ibm.es.ws6.client.types.NameValuePair;
import com.ibm.es.ws6.client.types.Result;
import com.ibm.es.ws6.client.types.SearchRequest;
import com.ibm.es.ws6.client.types.SearchResponse;

public class WebServiceClient {

   public static void main(String[] args) {
      try {
         if(args.length==0){
            System.out.println
            ("Usage: java WebServiceClient<absolute path for
              endpoints.properties>");
            System.out.println(" - Specify absolute path for
              endpoints.properties.");
            System.exit(1);
         }
         WebServiceClient mySample = new WebServiceClient();
         mySample.search(args[0]);
      } catch (Exception e) {
         e.printStackTrace();
         System.err.println
         ("Search failed!! with following error ***
         " + e.getLocalizedMessage() + " *** ");
      }
   }

   /* (non-Java-doc)
    * @see java.lang.Object#Object()
    */
   public WebServiceClient() {
      super();
   }


   public void search(String endPointsProperties) throws Exception{

      OmniFindSearchable searchble = new OmniFindSearchable
      (endPointsProperties);
      // the default values are set in the constructor
      SearchRequest request = new SearchRequest();

      // if the collection IDs are not specified, then all collections
      // for the specified application will be searched.
      request.setCollectionIds(null);
      // You can also specify collection IDs  that might interest you
      // request.setCollectionIds(new String[] {"col_34567"});

      // search query
      request.setQueryText("IBM");
      request.setFromResult(0);
      request.setNumberOfResult(10);
      // invoke the search
      SearchResponse response = searchble.search(request);

      if (response != null){
         System.out.println
```

```
                ("Number of results found = " + response.getAvailableNumberOfResults());
                System.out.println
                ("Query evaluation time=" + response.getGetQueryEvaluationTime() + "ms");
                System.out.println(" *** ");
                Result[] results = response.getResults();
                if(results != null && results.length>0){
                   for(int i=0;i<results.length; ++i){
                      System.out.println
                      ("results[" + i +"].getDocumentId=" + results[i].getDocumentId());
                      System.out.println
                      ("results[" + i +"].getDescription=" + results[i].getDescription());
                      System.out.println
                      ("results[" + i +"].getDocumentSource=" +
                       results[i].getDocumentSource());
                      System.out.println
                      ("results[" + i +"].getScore=" + results[i].getScore());
                      System.out.println
                      ("results[" + i +"].getTitle=" + results[i].getTitle());
                      // iterate through fields
                      NameValuePair pairs[] = results[i].getFields();
                      if(pairs != null && pairs.length>0){
                         for(int j=0; j<pairs.length; ++j){
                            System.out.println
                            ("results[" +i+"].field["+j+"].name=" + pairs[j].getName());
                            System.out.println
                            ("results[" +i+"].field["+j+"].value=" + pairs[j].getValue());
                         }
                      }
                      System.out.println(" *** ");
                   }
                }
             }
          }

}
```

Related tasks

"Installing the client toolkit" on page 2

Detailed Web service client application

The `WebServiceDetailClient.java` application is a detailed sample Web service client application for enterprise search.

The detailed sample Web service application is available in the `ES_INSTALL_ROOT/samples/siapiAdmin/webservices` directory after you install the client toolkit (the `es.siapi.toolkit.jar` archive file). The sample application includes usage comments that tell you how you can use Web services to control query behavior.

This sample application illustrates the following query properties:
- Specifying query languages
- Setting linguistic modes
- Returning metadata fields
- Enabling predefined result attribute values
- Specifying result ranges
- Setting result details
- Enabling site collapse
- Sorting results by relevance, date, numeric fields, or text fields
- Sorting the order of results (such as ascending or descending)

- Sorting the number of relevant results
- Enabling spelling correction
- Enabling query expansion

Related tasks

"Installing the client toolkit" on page 2

Enterprise search documentation

You can read the OmniFind Enterprise Edition documentation in PDF or HTML format.

The OmniFind Enterprise Edition installation program automatically installs the information center, which includes HTML versions of the documentation for enterprise search. For a multiple server installation, the information center is installed on both search servers. If you do not install the information center, when you click help, the information center opens on an IBM Web site.

To see installed versions of the PDF documents, go to `ES_INSTALL_ROOT/docs/`*`locale`*`/pdf`. For example, to find documents in English, go to `ES_INSTALL_ROOT/docs/en_US/pdf`.

To access the PDF versions of the documentation in all available languages, see the OmniFind Enterprise Edition, Version 8.5 documentation site.

You can also access product downloads, fix packs, technotes, and the information center from the OmniFind Enterprise Edition Support site.

The following table shows the available documentation, file names, and locations.

Table 2. Documentation for enterprise search

Title	File name	Location
Information center		http://publib.boulder.ibm.com/infocenter/discover/v8r5/
Installation Guide for Enterprise Search	`iiysi.pdf`	`ES_INSTALL_ROOT/docs/`*`locale`*`/pdf/`
Quick Start Guide (This document is also available in hardcopy for English, French, and Japanese.)	`OmniFindEE850_qsg_`*`two-letter locale`*`.pdf`	`ES_INSTALL_ROOT/docs/`*`locale`*`/pdf/`
Administering Enterprise Search	`iiysa.pdf`	`ES_INSTALL_ROOT/docs/`*`locale`*`/pdf/`
Programming Guide and API Reference for Enterprise Search	`iiysp.pdf`	`ES_INSTALL_ROOT/docs/en_US/pdf/`
Troubleshooting Guide and Messages Reference	`iiysm.pdf`	`ES_INSTALL_ROOT/docs/`*`locale`*`/pdf/`
Text Analysis Integration	`iiyst.pdf`	`ES_INSTALL_ROOT/docs/`*`locale`*`/pdf/`

Accessibility features

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features

The following list includes the major accessibility features in OmniFind Enterprise Edition:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers

The OmniFind Enterprise Edition Information Center, and its related publications, are accessibility-enabled. The accessibility features of the information center are described at http://publib.boulder.ibm.com/infocenter/discover/v8r5m0/topic/com.ibm.classify.nav.doc/dochome/accessibility_info.htm.

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

You can also use the following keyboard shortcuts to navigate and advance through the OmniFind Enterprise Edition installation program.

Table 3. Keyboard shortcuts for the installation program

Action	Shortcut
Highlight a radio button	Arrow key
Select a radio button	Tab key
Highlight a push button	Tab key
Select a push button	Enter key
Go to the next or previous window or cancel	Highlight a push button by pressing the Tab key and press Enter
Make the active window inactive	Ctrl + Alt + Esc

Interface information

The user interfaces for the administration console, sample search application, and search application customizer are browser-based interfaces that you can view in Microsoft Internet Explorer or Mozilla FireFox. See the online help for Internet Explorer or FireFox for a list of keyboard shortcuts and other accessibility features for your browser.

Related accessibility information

You can view the publications for OmniFind Enterprise Edition in Adobe Portable Document Format (PDF) using the Adobe Acrobat Reader. The PDFs are provided on a CD that is packaged with the product, or you can access them at http://www.ibm.com/support/docview.wss?rs=63&uid=swg27010938.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

Notices and trademarks

Notices

Trademarks

Index

SC18-9284-04