IBM DB2 Information Integrator
OmniFind Edition

Programming Guide and API Reference for Enterprise Search

Version 8.2.2

SC18-9284-01

IBM DB2 Information Integrator
OmniFind Edition

Programming Guide and API Reference for Enterprise Search

Version 8.2.2

SC18-9284-01

Before using this information and the product it supports, be sure to read the general information under ″Notices.″

You can order IBM publications online or through your local IBM representative:

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order.
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide.

Contents

Chapter 1. Enterprise search APIs

IBM® DB2® Information Integrator OmniFind Edition (DB2 II OmniFind Edition) provides Java application programming interfaces (APIs) for enterprise search. With these APIs, you can create search applications to process queries to the search collections. You can also add documents and remove documents from search collections with the data listener APIs.

IBM SIAPI

Use the IBM search and index API (SIAPI) to build custom search applications. The enterprise search implementation of SIAPI allows for remote access to the search server. The search server stores the collection data for the enterprise search system. With these APIs, you create applications that submit search requests, process search results, and browse taxonomy trees.

See Chapter 3, "Sample SIAPI applications," on page 21 for sample search applications that are provided with DB2 II OmniFind Edition.

Data listener API

The data listener is an enterprise search component that accepts requests from client applications. The requests are to add or remove data from collections. Typically, you create enterprise search collections by crawling for data, parsing and indexing that data, then making that data available for search. With the data listener client application, you can add pages to a collection, remove uniform resource identifiers (URIs) from a collection without having to wait to crawl those data sources, or instruct the Web crawler of a collection to visit or revisit uniform resource locators (URLs).

See "Sample data listener client applications" on page 34 for sample data listener client applications that are provided with DB2 II OmniFind Edition.

Search API security

The search and browse APIs communicate remotely to the ESSearchServer Enterprise Application that is installed on each WebSphere® search node.

After global security is enabled, WebSphere Application Server will require that all HTTP requests supply a valid user name and password. The user name and password entered must be valid within the active user registry that is configured through the WebSphere administration console. Any requests that do not contain valid user credentials are rejected.

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.

Store the search application names and passwords in the same repository that is used for WebSphere authentication.

Enterprise search supports HTTP BASIC authentication. There is no support for HTTPS (SSL v2 or v3).

Compiling Java source code

The enterprise search ESSearchApplication sample and the data listener samples should be compiled with IBM Software Developer's Kit 1.4.x. The SIAPI samples can be compiled with either 1.3.x or 1.4.x. IBM Software Developer's Kit 1.5 is not supported.

Before you can build your Java™ source code, you must install and configure Apache ANT, a Java-based build tool. For more information about how to install and configure Apache ANT, see http://ant.apache.org/.

The ESSearchApplication in the *ES_INSTALL_ROOT*/samples directory must be compiled with IBM

SDK Version 1.4 and must execute in a JRE Version 1.4 environment. WebSphere Application Server, Version 5.1 and WebSphere Portal, Version 5.1 both provide the JRE Version 1.4.

To compile your Java source code:

1. From the command line, change to one of the following directories:
   - For the SIAPI sample search applications: *ES_INSTALL_ROOT*/samples/siapi (for example, on Linux™ and AIX®, /opt/IBM/es/samples/siapi, on Windows®, Program Files\IBM\es\samples\siapi)
   - For the data listener client applications: *ES_INSTALL_ROOT*/samples//datalistener (for example, on Linux and AIX, /opt/IBM/es/samples//datalistener, on Windows, Program Files\IBM\es\samples\\datalistener)
   - For the ESSearchApplication: *ES_INSTALL_ROOT*/samples/ESSearchApplication (for example, on Linux and AIX, opt/IBM/es/samples/ESSearchApplication, on Windows, Program Files\IBM\es\samples\ESSearchApplication)

   Each of these directories includes a `build.xml` file that ANT uses to build the file.
2. Type `ant` and press Enter.

You will see the following message after the your Java source code compiles:

```
BUILD SUCCESSFUL
Total time: xx seconds
```

SIAPI Javadoc documentation

Use the SIAPI Javadoc documentation that is shipped with DB2 Information Integrator OmniFind Edition to help you create custom search applications that you can deploy for your enterprise search solution.

See the Javadoc documentation for a list of application programming interfaces (APIs) that you can use to create search applications. This documentation does not include APIs for the data listener client applications.

The Javadoc documentation is in the *ES_INSTALL_ROOT*/docs/api/siapi directory.

Chapter 2. Search and Index API (SIAPI)

The IBM search and index API (SIAPI) is a programming interface that enables you to search and browse collections and taxonomies.

The SIAPI provides a unified programming interface that enables you to write one program that searches different IBM back-end search products.

The SIAPI supports such tasks as:

- 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)

Structure of an SIAPI application

A SIAPI application sends queries to the search server and returns results for those queries.

An SIAPI application consists of the following tasks:

- Obtains an SIAPI implementation factory object
- Obtains a SearchService object
- Obtains a Searchable object
- Issues queries
- Processes query results

Obtaining an SIAPI implementation factory object

An SIAPI-based search application begins by obtaining an implementation factory object.

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

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

The enterprise search SIAPI implementation is provided by the com.ibm.es.api.search.RemoteSearchFactory class.

Obtaining a SearchService object

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

If the enterprise search system was installed on multiple servers, you must 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. The parameters are then passed to the getSearchService factory method that generates the SearchService object.

```
Properties configuration = new Properties();
configuration.setProperty("hostname", "es.mycompany.com");
configuration.setProperty("port", "80");
config.setProperty("username", "websphereUser");
config.setProperty("password", "websperePassword");
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.

```
ApplicationInfo appInfo = factory.createApplicationInfo("my_application_id");
appInfo.setPassword("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 you obtain a Searchable object, you can issue a query to that Searchable object. To issue a query to the Searchable object:

- Create a Query object.
- Customize the Query object.
- Submit the Query object to the Searchable object.
- Get the query results, which are specified in a ResultSet object.

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

Processing query results

The ResultSet and Result interfaces enable you to access query results.

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

The SIAPI has a variety of methods for interacting with the ResultSet interface and individual Result interface objects.

Controlling query behavior

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

See the Javadoc documentation for details about the each method.

Table 1. Query behavior methods

Method	Description
setLinguisticMode(int mode)	Sets the linguistic mode for this query. You can set one of the following modes: • LINGUISTIC_MODE_EXACT_MATCH: Unmodified terms will be matched as entered without undergoing linguistic processing. • LINGUISTIC_MODE_BASEFORM_MATCH: Unmodified terms will be matched after undergoing linguistic processing. • LINGUISTIC_MODE_ENGINE_DEFINED: Unmodified terms will be matched according to the engine's best-effort policy. This is the default mode.
setQueryLanguage(java.lang.String lang)	Specifies to use a language other than the collection default language on the search server. For example, for English, the query language parameter is en-US. For Chinese, use zh-CN for simplified Chinese and zh-TW for traditional Chinese.

Table 1. Query behavior methods (continued)

Method	Description
setProperty(String name, String value)	Sets the value of a searchable property. This method has the following modes: • 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 omits to 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 performing a non-strict search in n-gram collections. This property is boolean and its values are: – false: A strict search will be performed. This is the default if your search application omits to set the FuzzyNGramSearch property. – true: Fuzzy search will be performed. When FuzzyNGramSearch is set to true, you can set a second related property: 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. ProximityWindowSize cannot be set if FuzzyNGramSearch is set to false. • AllowStopwordRemoval: Determines whether stop words should be removed or not during query parsing. If this property is not set, the engine applies stop word removal according to its own policy. 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.
setRequestedResultRange(int fromResult, int numberOfResult)	Controls the range of the 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 100.
setReturnedAttribute(int attributeType, boolean isReturned)	Enables or disables 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).

Table 1. Query behavior methods (continued)

Method	Description
setReturnedFields(String[] fieldNames)	Controls which metadata fields are returned in the Result object. By default, enterprise search does not return any metadata fields.
setSiteCollapsingEnabled(boolean value)	Specifies if 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 will contain 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 8 for more information.
setSortKey(java.lang.String sortKey)	Specifies the sort key. The following predefined sort key values are defined in com.ibm.siapi.search.BaseQuery: • SORT_KEY_NONE • SORT_KEY_DATE • SORT_KEY_RELEVANCE You can specify the sort key to be any other valid numeric field name for the collection that is being searched. The default sort key is SORT_KEY_RELEVANCE.
setSortOrder(int sortOrder)	Specifies 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.
setSortPoolSize(int sortPoolSize)	Controls how many of the top relevant results will be sorted and returned in the result set. Values range from 1 to 500 (the default sort pool size is 500). Any other values will cause a SiapiException to be thrown by the search server. The sort pool size is ignored if the sortKey is SORT_KEY_RELEVANCE or SORT_KEY_NONE.
setPredefinedResultsEnabled (boolean value)	Specifies whether query results contain predefined links in addition to the regular results. Predefined links are enabled by default.
setSpellCorrectionEnabled(boolean enable)	Specifies whether query results contain suggestions for spelling corrections for the query. Spell correction is disabled by default.
setResultCategoriesDetailLevel (int detailLevel)	Specifies the required category detail level for query results. This method is used if the categories attribute (RETURN_RESULT_CATEGORIES) is enabled. The default value is RESULT_CATEGORIES_ALL.

Table 1. Query behavior methods (continued)

Method	Description
setSynonymExpansionMode (int mode)	Sets synonym expansion mode for a query. 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.
setACLConstraints(java.lang.String aclConstraints)	Sets the access control constraints for this query for secure searches.

Query syntax

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

Simple query syntax characters

The following table 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. The queries usually contain terms that are not preceded by a plus sign (+), a circumflex (^) or a minus sign (-), and for which there is no default behavior defined.

Query: `computer software`

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

~ (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*.

⌂ 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 circumflex (^) to indicate that a document must contain the term and at least one more term that is not preceded by a circumflex.

Query: `^computer science software`

Result: This query returns documents that contain the term `computer` and the terms *science* or *software*.

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

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

* Follow a term with a wildcard sign (*) to indicate that the document can contain any word that is prefixed by this term.

Query: `app*`

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

( ) 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)`.

" " Use double quotation marks (") to indicate that a document must contain the exact phrase within the double quotation marks for a match to occur.

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 (^).

Words inside phrases are never lemmatized.

~ (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`).

=

Precede a term with the equal sign (=) to indicate the document must contain an exact match of the term for a match to occur (disable lemmatization).

Query: `=apples`

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

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 with the value `support` anywhere in the URL, 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 belonging to the same group, and treats the news articles from the same thread as belonging to the same group. For URIs from all other data sources, each URI forms its own group. However, with enterprise search, you can group 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 (this query should be on one line)

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

as a search term will restrict the search to the URIs in the *sale* group. 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 created for enterprise search, you can search for documents that belong to a specific category in a specific taxonomy. Enterprise search supports two types of categorizers, and each categorizer has its own taxonomy ID.

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 *DB2 Information Integrator OmniFind Edition Administering Enterprise Search*.

Query:`rulebased::c1`

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

modelbased

Use this taxonomy ID to search for documents that belong to a category defined in the WebSphere Portal taxonomy. For information about migrating and using model-based categories or taxonomies, see *DB2 Information Integrator OmniFind Edition Administering Enterprise Search*.

Query:`modelbased::c3`

Result: This query returns documents that belong to a model-based category ID named c3.

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 *DB2 Information Integrator OmniFind Edition Administering 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.

ACL constraints: (*security_tokens***)**

or 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 (^), as well as an XML security context string in ACL constraints string (@SecurityContext::'securityContext'). For information on 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 sting 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::*. The XML fragment 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 enclosed in single quotation marks (' '). The 'value' is sometimes preceded with or +, ⌂, 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, \'.

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

This query syntax looks for occurrences of tag1 anywhere under tag2. Depth is measured between a word w or a tag t to the tag t' in the query that is closest to w or t from above in the query.

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

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

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

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

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

You can define other tags, words, or phrases within these tags. These words or phrases support the same features as the normal query's terms.

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 (data under elements creates a new nested query level) "+" means that the terms must appear, "-" means that the terms should not appear and others are optional and only contribute to ranking. If there are no "+" terms, then at least one of the optional terms must appear.

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

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

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

Use full boolean to express AND (<.and>) and OR (<.or>) scope within a fragment 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/.../tagn'

Use the @xmlxp:: prefix and enclose the query in single quotation marks to indicate an XPath query as an SIAPI opaque term.

If you include single quotation marks in opaque terms, they must be preceded by backslash, for example, `@xmlf2::'<T1 att1=\'value1\'></T1>`.

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.″

@xmlxp::'/tag1/@tag1'
You can distinguish between elements (XML start and end tags) and attributes. Attributes are written explicitly with a leading @ sign. This enables you to distinguish between elements and attributes that might have the same name.

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

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

@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.

@xmlf2::'text1 <tag1> ... </tag1>'
Use the @xmlf2:: prefix and enclose the query in single quotation marks to indicate a fragment query as a new SIAPI 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*.

@xmlxp::'tag1/.../tagn'
Use the @xmlxp:: prefix and enclose the query in single quotation marks to indicate an XPath query as an SIAPI 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.″

@xmlxp::'/tag1/@tag1'
You can distinguish between elements (XML start and end tags) and attributes. Attributes are written explicitly with a leading @ sign. This enables you to distinguish between elements and attributes that might have the same name.

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

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

@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.

@xmlf2::'text1 <tag1> ... </tag1>'
Use the @xmlf2:: prefix and enclose the query in single quotation marks to indicate a fragment query as an SIAPI 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><@tag1> ... </@tag1></tag1>'
You can distinguish between elements and attributes. Attributes are either written explicitly within the element or as subelements with a leading @ sign. This enables you to distinguish between elements and attributes that might have the same name.

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.

@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 (data under elements creates a new nested query level) ″+″ means that the terms must appear, ″-″ means that the terms should not appear and others are optional and only contribute to ranking. If there are no ″+″ terms, then at least one of the optional terms must appear.

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

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

@xmlf2::'<tag1><.or> ... <.and> ... </tag1>'
Use full boolean to express AND (<.and>) and OR (<.or>) scope within a fragment query.

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

Result: This query finds documents that specify a book whose author is Jane Wong 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.

SIAPI federators

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

Federators are intermediary components that exist between the requestors of service and the agents that perform that service. They are special purpose resource coordinators that are designed to manage the multitude of searches generated from a single request.

Enterprise search provides two types of SIAPI federators:

- Local federator
- Remote federator

Local and remote federators are SIAPI searchable objects. Multiple-level federation is allowed, but too many levels of federation will decrease search performance.

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.

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.

Local federator

A local federator is a client-side federator that federates over a set of searchables.

Before you can create a LocalFederator, you must create or retrieve SIAPI searchable objects. This usually involves using a SIAPI SearchFactory. The SIAPI searchable object that is passed to the LocalFederator 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 snippet shows how to create a LocalFederator 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 is a server-side federator that federates over a set of searchables.

The RemoteFederator is created by using the SIAPI 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 snippet shows how to create a RemotelFederator 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);
```

Chapter 3. Sample SIAPI applications

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

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

Simple search
The SearchExample class provides a simple example of the minimum requirements that are needed to submit a search to the search server.

Basic taxonomy browsing and navigation
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.

Advanced search
The AdvancedSearchExample class is provides an example of using advanced query settings and results processing options.

Compiling the sample SIAPI search applications

You compile the sample SIAPI search applications by running the ANT script.

To compile and run an SIAPI sample application:

1. Change to the following directory. (These are the default installation directories):

   Linux and AIX: /opt/IBM/es/samples/siapi

   Windows: C:\Program Files\IBM\es\samples\siapi
2. Run the ANT script.
3. Run the application by issuing the following command:

   On Linux and AIX:

   ```
   java -classpath ES_INSTALL_ROOT/lib/esapi.jar:ES_INSTALL_ROOT>/lib/siapi.jar:.
   SearchExample
   ```

   On Windows:

   ```
   java -classpath "ES_INSTALL_ROOT\lib\esapi.jar;ES_INSTALL_ROOT>\lib/siapi.jar;."
   SearchExample
   ```

You can run the class by executing the following command from a command line.

On Linux and AIX:

```
java -classpath ES_INSTALL_ROOT/lib/esapi.jar:ES_INSTALL_ROOT>/lib/siapi.jar
SearchExample
```

On Windows:

```
java -classpath ES_INSTALL_ROOT\lib\esapi.jar;ES_INSTALL_ROOT>\lib\siapi.jar
SearchExample
```

Replace `SearchExample` with any of the other sample search applications that you want to compile.

Simple and advanced sample search applications

The SearchExample class provides a simple example of the minimum required application 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 instead of just 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:

- Linux and AIX: /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 displaying some of the 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 directoies:

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

Retrieve all search results sample

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
- Create a new Query object
- Set the query to return unsorted results
- Run the search

Obtain a SearchFactory and a Searchable

Obtain a SearchFactory and a Searchable object as explained in "Simple and advanced sample search applications" on page 22 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 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 you interpret them for sorted query results:

- The getEstimatedNumberOfResults method will always return 0 because enterprise search does not provide a number-of-results estimate for unsorted results.
- The getAvailableNumberOfResults method will return one of two values: 0 if this is the last result page, and 1 if there are more results.
- You can use the length of the array returned by getResults mehtod 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;
}
```

Sample federated search 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 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:

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

Chapter 4. Data listener

Use the data listener application programming interfaces (APIs) to create a client application that sends data to the data listener.

The data listener is an enterprise search component that accepts requests from client applications. The requests are to add or remove data from collections. Typically, you create one or more crawlers for a collection. These crawlers retrieve data from a specific data source, such as the Web or a DB2 Content Manager database. With the data listener client application, you can add pages to a collection without creating crawlers. You can also remove uniform resource identifiers (URIs) from a collection or instruct the Web crawler of a collection to visit or revisit uniform resource locators (URLs).

Figure 1 shows an overview of the search behavior of the enterprise search system without the influence of the data listener client application. Users submit queries to the search component, and the index periodically refreshes the data in the search component. (These components can be on one server or on multiple, connected servers. The descriptions in this topic assume that you are using a multiple server system).

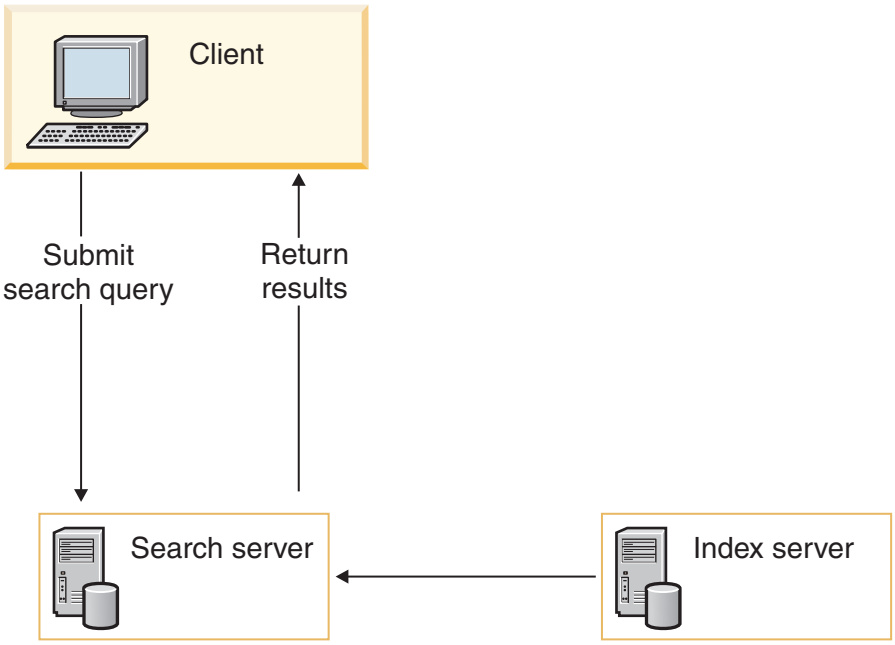

Figure 1. How queries are sent to the enterprise system

The data listener client application enables you to submit requests to the to the data listener. Depending on the type of request, the data listener will send the request to other enterprise search components for further processing.

When the client application connects to the data listener, the data listener verifies that the password that you provide is valid for the client ID and password. The data listener also verifies that the client application is authorized to add or remove data to the specified collection.

Figure 2 shows how the data listener client application sends a request to the enterprise search system.

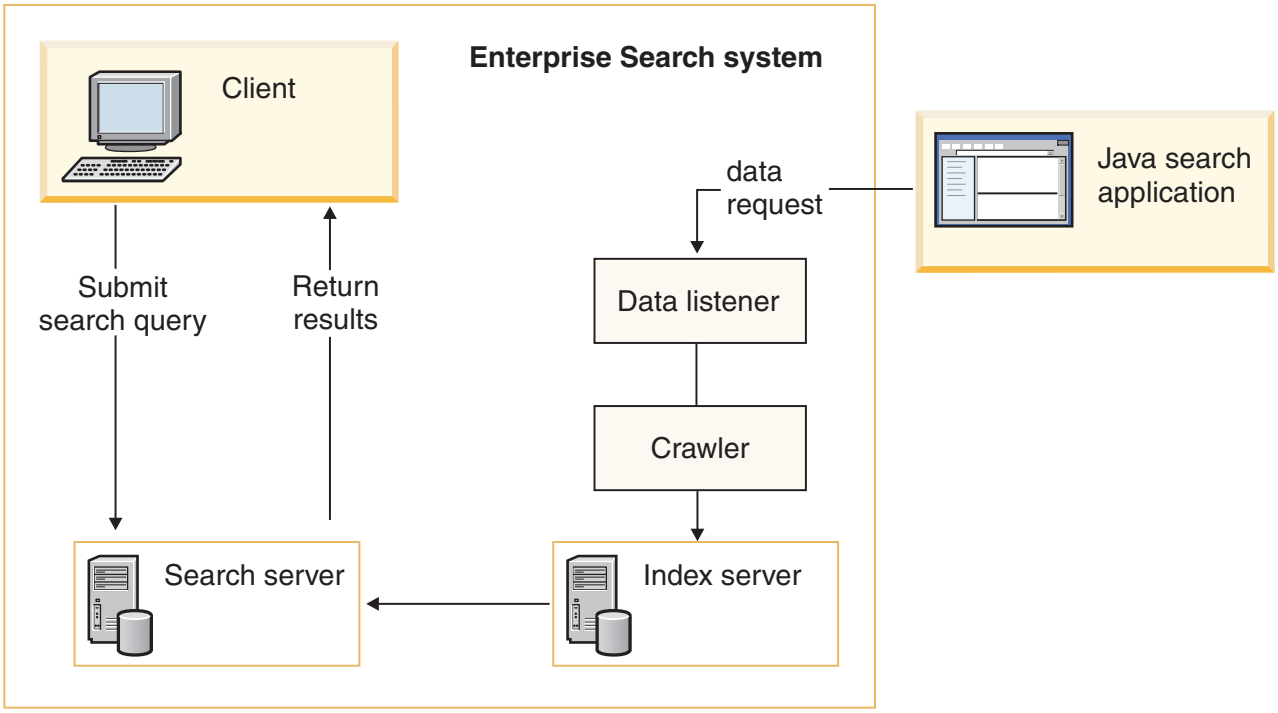

Figure 2. How the data listener API works with the enterprise search system

The data listener APIs are packaged in the following JAR files:

- es.dl.client.jar
- es.oss.jar

These JAR files are in the following directories according to your operating system:

- Linux and AIX: *INSTALL_ROOT*/IBM/es/lib
- Windows: *INSTALL_ROOT*\IBM\es\lib

If you installed the JAR files in the default installation directory, add the following value to your CLASSPATH variable:

- Linux and AIX: opt/IBM/es/lib/es.oss.jar:/opt/IBM/es/lib/es.dl.client.jar
- Windows: c:/Program Files/IBM/es/lib/es.oss.jar;c:/Program Files/IBM/es/lib/es.dl.client.jar

If the JAR files are not installed in the default directory, add the appropriate directory to your CLASSPATH.

Removing data with the data listener API

With the data listener, you can remove documents from a collection by removing specific URIs or patterns of URIs.

If you remove specific URIs, those URIs disappear from the search component immediately. If you remove URI patterns, the documents that match those patterns will be removed the next time that the index is reorganized.

Removing URIs from the search component and index

You can use the data listener API to remove a specific URI. The request is sent to the data listener, and the URI is removed from the index server and search server as shown in the figure below.

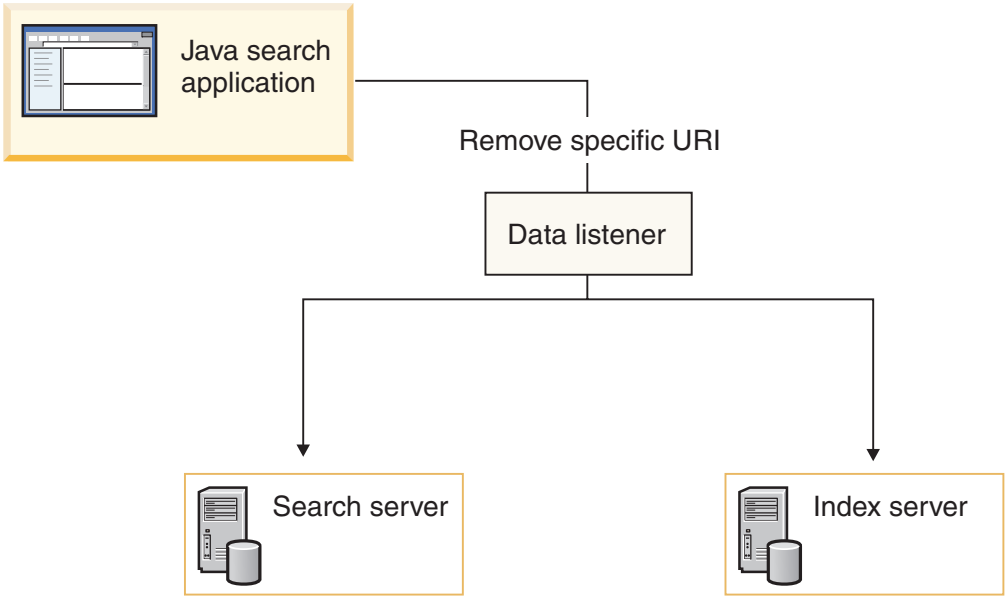

Figure 3. Removing a specific URI

Use the removeURI method to delete the data from the specified collection.

Removing URI patterns from the index

You can use the data listener APIs to remove a URI pattern. For example, if you submit a remove URI pattern request and specify http://www.ibm.com/*.html as your URI, the index server will remove the following URLs:

- http://www.ibm.com/home.html
- http://www.ibm.com/family.html
- http://www.ibm.com/pics.html

Attention: Use a remove URI pattern request with caution. The removed content cannot be recovered. The content must be added to the index again by the crawlers or by the data listener client application.

Adding data with the data listener API

With the data listener, you can add, or push, data to search collections by adding URIs with content or visiting or revisiting URLs.

Adding URIs with content

You can use the data listener API to add a URI and its content by including the content parameter in the pushData method. This method sends the specified data to the specified collection.

The added URI and its content will be available for search after the index is refreshed or is reorganized.

Visiting or revisiting URLs

You can use the data listener APIs to add specific URLs or URL patterns to the Web crawler of a collection by using the revisitURLs method. This method is valid only for Web content. The crawler will crawl the URLs and add that data (URL, metadata, and content) to the collection. The URLs can be new (visiting) to the crawler or the crawler has already discovered those URLs (revisiting). If you specify a URL pattern, you are requesting that the crawler revisit URLs that were already discovered that match the specified pattern.

Creating data listener client applications

To send requests to the data listener, you create a client application and provide specific properties about the collection that the client application must provide in the requests to the collection ID of the target collection and the authentication data. The authentication data include a client ID and the password. The client ID must have permission to update the collection.

Procedure

To create a data listener application, follow these steps:

1. Determine which collection you want to change.
2. Get the ID for that collection. A collection ID is created when you create a collection in the enterprise search administration console. You can see the collection ID from the Collections Settings page. In the enterprise search administration console from the General page of the Collections view, click **View collection settings**. You must either log in to the enterprise search administration console or ask the search administrator for this ID.
3. Provide a client ID and password for the collection. The client is the data listener application that will access the collection. You must give the data listener application permission to access a collection by creating a client ID and password for that collection. To specify a client ID and password, ask the search administrator to do this for you, or follow these steps:
   a. Log in to the enterprise search administration console.
   b. Click **System** → **Data Listener** → **Configure data listener**.
   c. Click **Add Data Listener Client ID**.
   d. Type a data listener client ID, password, and select a collection. Click **OK**.

      To add more client IDs and passwords for other collections, click **Add Data Listener Client ID** and follow the previous steps.

DLResponse class

Defines the object that the client application receives from the data listener.

When the client application sends a request to the data listener, the data listener returns the DLResponse object. The object contains information about the status of the request. This class also defines possible return codes for the object.

Type	Value
Class	public class DLResponse

The DLResponse class can return the following types of status from the data listener:

Value	Description
public final static int SUCCESS = 0;	The requested operation was successful.
public final static int COMMUNICATION_ERROR = 1;	A communication error occurred.
public final static int BAD_REQUEST = 2;	The data listener did not understand the request.
public final static int PERMISSION_DENIED = 3;	The client ID does not have permission to update the collection.
public final static int FAIL_TO_SAVE = 4;	The data listener failed to save the request.
public final static int UNSUPPORTED_REQUEST_VERSION = 5;	The client version of DB2 II OmniFind Edition does not match the server version.
public final static int UNSUPPORTED_RESPONSE_VERSION = 6;	The client version of DB2 II OmniFind Edition does not match the server version.
public final static int INVALID_PASSWD = 7;	The specified password is not valid for the specified client ID.
public final static int UNKNOWN_COLLECTION = 8;	The specified collection ID is not valid.
public final static int FAIL_TO_ADD_TO_CRAWLER = 9;	The specified URL could not be saved.
public final static int FAIL_TO_REMOVE_FROM_COLLECTION = 10;	The request to remove data failed.
public final static int UNKNOWN_URICODE = 11;	The data listener could not perform the requested operation.

getCode method

Returns the status code from the data listener.

Syntax

```
public int getCode()
```

Returns

Returns a number that corresponds to a return code from the data listener. See "DLResponse class" on page 28.

getCodeName method

Returns the message string for the return code.

Syntax

```
public String getCodeName()
```

Returns

Returns a string from one of the data listener return codes. See "DLResponse class" on page 28.

DLDataPusher class

This class provides APIs for adding data to and removing URIs from a collection. It also provides APIs for requesting that the Web crawler of a collection to visit or revisit URLs.

```
public class DLDataPusher
```

This is the main class for the data listener API.

To call these APIs, you must specify the host name and the port of the data listener and provide the client ID and password for authentication. For information about how to obtain the client ID, password, and collection ID, see "Creating data listener client applications" on page 28.

removeURIs method

Removes URIs from the specified collection.

You can specify individual URIs to be removed or specify URI patterns to be removed from a collection. When you use this method to remove individual URIs, the URIs will not be available for search immediately. However, when you use this method to remove URI patterns, the URIs that match the patterns will be removed from the index at the next time it is reorganized. Until the index is reorganized, the URIs might still be searchable.

Syntax

```
public static DLResponse removeURIs(String hostname,
                                    int port,
                                    String id,
                                    String pw,
                                    String uris,
                                    String col)
```

Parameters

hostname
The host name of the server where the data listener is running.

port
The port number of the data listener.

ID The client ID that you or the search administrator provided in the enterprise search administration console for a specific collection.

pw The password for the client ID.

uris
The specific URI, URIs, or URI patterns that you want to remove from the collection. Each URI or URI pattern must be separated by white space. Use a wildcard character (*) at the end or the middle of a URI to specify a URI pattern, for example, cm://enterprise/finance/* or cm://enterprise/*/sales.

col The ID of the collection that you want to modify. You can see the collection ID in the enterprise search administration console.

revisitURLs method

Instructs the Web crawler for the specified collection to add or revisit the URLs.

If you specify a URL pattern in this method, the Web crawler will try to revisit immediately those URLs that it discovered that match the pattern. If you specify individual URLs, the Web crawler will visit them immediately (or revisit them if the crawler already discovered them).

You can use the revisitURLs method only for Web data sources (URLs).

Syntax

```
public static DLResponse revisitURLs(String hostname,
                                     int port,
                                     String id,
                                     String pw,
                                     String urls,
                                     String col)
```

Parameters

hostname
The host name of the data listener server.

port
The port number of the data listener.

id The client ID.

pw The client password.

urls
The URL or the URL patterns, separated by white space. Use a wildcard character (*) at the end or the middle of a URI to specify a URI pattern, for example, http://www.ibm.com/* or http://www.ibm.com/*/software.

col The ID of the collection.

pushData method

Pushes data to data listener.

Syntax

```
public static DLResponse pushData(String hostname,
                                  int port,
                                  String id,
                                  String pw,
                                  String uri,
                                  String col,
                                  DataSourceMetadata metadata,
                                  byte[] content)
```

Parameters

hostname
Th host name of the data listener server.

port
The port number of the data listener.

id The client ID.

pw The client password.

uri The URI of the data.

col The ID of the collection.

metadata
An object that describes the metadata of the document.

content
The content of the document.

Creating the metadata object:

You must create a metadata object before you send the push data request. Metadata contains information about the document and can include information such as author, date modified, or date created.

Use the createDataSourceMetadata API to create the metadata object with predefined fields. Use the addMetaField API to add other fields that are specific to a data source.

createDataSourceMetadata method:

Creates a metadata object.

Syntax

```
public static
DataSourceMetadata createDataSourceMetadata(String ds,
                                             String cid,
                                             String dsName,
                                             int score,
                                             Date dt,
                                             String language,
                                             String securityACLs,
                                             String contentType,
                                             String charSet,
                                             byte[] content)
```

Parameters

ds Data source that this document is from.

cid The data listener application client ID.

dsName
The data source name, such as DB2 or Notes (string).

score
This parameter is not used.

dt A date object that indicates the currency of this document. It can be used to influence the ranking of this document.

language
The language of the document, such as "en", "en_US", or "zh.". Enterprise search will use this parameter if it fails to determine the language of the document.

securityACLs
A string of comma separated security tokens. If the value is null, the document will have no security restrictions and is accessible to anyone. For example, if a document has tokens A, B, and C, then only those users with security tokens A, B, or C can access the document.

contentType
The MIME type of this document.

charSet
The character set of the document, such as UTF-8.

content
The byte array of the document content.

addMetaField method:

Adds an element to a metadata object.

Syntax

```
public static void addMetaField(DataSourceMetadata metadata,
                                String fieldName,
                                String fieldValue,
                                boolean searchable,
                                boolean partOfResult,
                                boolean fieldSearchable,
                                boolean parametricSearchable
              boolean isContent)
```

Parameters

metadata
The metadata object to which a new field is to be added.

fieldName
The name of the field.

fieldValue
The value of the field.

searchable
Whether this field is searchable.

partOfResult
Whether this field is part of search result.

fieldSearchable
Whether field search is supported for this field.

parametricSearchable
Whether parametric search is supported for this field.

isContent
Whether this field is considered as a part of content.

setKnownLanguageFor method:

Specifies the language of the document.

If you do not specify a language with this method, enterprise search attempts to determine the language of the document.

Syntax

```
static public void setKnownLanguageFor(DataSourceMetadata dsmd,
String knownLanguage);
```

Parameters

dsmd
The metadata object.

knownLanguage
The name of the language ID, such as ″en″, ″en_US″, or ″zh.″

Sample data listener client applications

DB2 II OmniFind Edition provides sample client applications that show you how to use the data listener APIs to add or remove data to or from enterprise search collections.

The following sample client applications show you how to do the following tasks:

DLRemoveURIs.java
Remove URIs from a collection.

DLPushData.java
Add URIs and content to a collection.

DLRevisitURLs.java
Instruct the Web crawler to revisit URLs.

DLSampleClient.java
Combines all tasks in one application that removes URIs, adds URIs, and revisits URLs.

Sample data listener client application: removing URIs from a collection

The DLRemoveURIs class includes an example of client application APIs that remove URIs or URI patterns for a collection.

The DLRemoveURIs class defines two static methods: main and removeURIsExample.

The main method first prepares the host name, port, client ID, password, and collection ID. Then, the main method calls the removeURIsExample method with those parameters.

The removeURIsExample method takes the host name, port, client ID, password, and collection ID as parameters, and it prepares the URIs or URI pattern strings.

After preparing the URIs or URI patterns, the removeURIsExample calls the removeURIs method of the DLDataPusher class. Then, the removeURIsExample method checks and prints the response from the data lister.

```
public class DLRemoveURIs {

    /**
     *  To remove URIs or URI patterns from a collection, you
     *  need to specify the host name and the port of the
     *  data listener server. You also need to provide
     *  the client ID and password for authentication.
     *  You need to specify which collection the data is
     *  applied to by providing the collection ID.
     *
     *  This function shows how to use the data listener
     *  APIs to remove URIs from a collection. In enterprise
     *  search, you can remove specific URIs and URIs that
     *  match patterns. If you remove specific URIs, those
     *  URIs are removed from the index. If you remove URI patterns,
     *  those URIs that match those patterns are not removed until
     *  the next index reorganization, and those URIs are still
     *  searchable. After the URIs are removed from a collection, those
```

```
     *  URIs will not be searchable until the they are re-crawled by
     *  the crawler or added through the data listener later.
     */
    static void removeURIsExample(String hostname, int port,
    String clientID, String passwd, String collectionID) {

        // In this example, you will remove some URIs from
        // one collection.
        // URIs can be either specific URIs or URIs that match patterns.
        StringBuffer sb = new StringBuffer();
        sb.append("uri1");
        sb.append("\n");
        sb.append("uri*pattern1");
        sb.append("\n");

        sb.append("uri2");
        sb.append("\n");
        sb.append("uri*pattern2");

        String uris = sb.toString();

        // Now, you call the data listener API to remove URIs
        // from the collection.
        // Note that you can specify multiple URIs and URI patterns
        // in a single call. The URIs or URI patterns are seperated
        // by a newline character. In your application, you need
        // to replace the URIs and URI patterns from the sample
        // application with the real URIs that make sense for a
        // real collection.
        DLResponse dlRes = DLDataPusher.removeURIs(hostname, port,
        clientID, passwd, uris, collectionID);

        // Check the result.
        // The DLResponse object contains a result code and
        // a message that indicates the result of the operation.
        // See the documentation for the DLResponse class for the
        // code values and their meanings.
        if (dlRes != null) System.out.println(dlRes.toString());
    }

    public static void main (String [] argv) {
        if (argv.length > 0 && argv[0].equalsIgnoreCase("-h")) {
            System.out.println("usage: java DLRemoveURIs [hostname [port]]");
            return;
        }

        // First, you get the host name and the port number
        // for the data listener.
        // You need to ensure that the data listener is running on
        // the host server and listening to the port.
        // Otherwise, you might get a connection refused exception.
        String hostname = "localhost";
        if (argv.length > 0) hostname = argv[0];
        int port = 6668;
        if (argv.length > 1) {
            try {
                port = Integer.parseInt(argv[1]);
            }
            catch (Exception e) {
               port = 6668;
            }
        }

        // Assume that client_id_1 is a valid client id with password_1.
        // If not, you will get INVALID_PASSWD as the result code

        // Assume that client_id_1 has the authorization to update
```

```
            // collection with collection ID collection_id_1.
            // If not, you will get PERMISSION_DENIED as the result code.
            //
            // Assume that the collection  with collection ID collection_id_1
            // is a valid collection. Otherwise, you will get
            // UNKNOWN_COLLECTION as the result code.
            //
            // You need to contact the enterprise search administrator
            // to find out what client ID, password, and collection ID
            // are valid and use those valid values here.
            //
            removeURIsExample(hostname, port, "client_id_1", "password_1",
            "collection_id_1");
        }
    }
```

Sample data listener client application: adding URIs and content to a collection

The DLPushData class includes an example of client application APIs that add, or push, URIs and each URI's content to a collection.

The DLPushData class defines two static methods: main and dataPushExample.

The main method first prepares the host name, port, client ID, password, and collection ID. Then main method calls the dataPushExample method with those parameters.

To add URIs and their content to a collection, you prepare the document URI, the content, and the metadata. To prepare the URI and content, you add simple text strings. To prepare the metadata of the document, you create the DataSourceMetadata object from the DLDataPusher class. Then, add more fields to the DataSourceMetadata object.

Call the pushData method and check the response from the data listener.

```
public class DLPushData {

    /**
     *  To push data to a data listener, you need to specify the host name
     *  and the port of the data listener server.
     *  You also need to provide the client ID and password for authentication.
     *  You need to specify which collection the data is applied to by providing the
     *  collection ID.
     *
     *  This function shows how to use the data listener APIs to push a document
     *  to an enterprise search collection for indexing. A document consists of
     *  three parts: the URI, the metadata and the content.
     *  The content is the raw data of the document.
     *  The metadata contains the attributes values of
     *  the document.
     */
    static void dataPushExample(String hostname, int port, String clientID,
    String passwd, String collectionID) {

        // Prepare the data.
        // Suppose this is a very simple document with "almost empty" content
        String content = "almost empty";
        // Suppose the URI for the document is "myURI"
        String uri = "myURI";

        // Use data listener client APIs to prepare metadata.
        //
        // First, create a DataSourceMetadata object.
```

```
                //
                // The first argument indicates the data source type
                // of this document. The second argument specifies the client ID.
                // The third argument is a string that specifies the data source name.
                // The fourth argument is a number that indicates the quality of this
                // this document. This number is not currently used.
                // The fifth argument is a date that indicates
                // the currency (how up to date it is) of this document. It can be
                // used to influence the ranking of this document in the search results.
                // The sixth argument indicates language of the document.
                // Normally, enterprise search will detect the language ID of a
                // given document.
                // However, if the detection fails, the enterprise search system
                // will assume the document is in this specified language.
                // The seventh argument specifies security tokens.
                // If it is null, then this document is assumed to be available
                // to anyone. Otherwise, the document is accessible only by
                // a user with a security token that is specified here.
                // The eigthth argument specifies the MIME type of the document.
                // The ninth argument specifies the character set of the content.
                // Finally, the tenth argument is the content.
                DataSourceMetadata md =
            DLDataPusher.createDataSourceMetadata("CustomerDataSource",
             clientID,
                                "CustomerDataSourceName",
                                90,
                                new Date(),
                                "en",
                                "securityToken",
                                "text/plain",
                                "iso-8859-1",
                                content.getBytes());

                // Second, you add more fields to the metadata.
                // Each field is a name/value pair.
                // The fourth argument specifies whether the field value is searchable.
                // The fifth argument specifies whether the field value will
                // be part of the search result.
                // The sixth argument specifies whether to support field search.
                // The seventh argument specifies whether the field is
                // parametric searchable.
                // The eight argument specifies whether the field is part of the content.
                DLDataPusher.addMetaField(md,
                                          "fieldName1", "fieldValue1",
                                          true, false, true, false, true);
                DLDataPusher.addMetaField(md,
                                          "fieldName2", "fieldValue2",
                                          true, false, true, false, true);
                System.out.println("metadata:\n" +  md.generateXML().toString());

                // Call the pushData method.
                DLResponse dlRes = DLDataPusher.pushData(hostname, port, clientID,
                passwd, uri, collectionID, md, content.getBytes());

                // Check the result from the data listener.
                if (dlRes != null) System.out.println(dlRes.toString());
              }

              public static void main (String [] argv) {
                if (argv.length > 0 && argv[0].equalsIgnoreCase("-h")) {
                    System.out.println("usage: java DLPushData [hostname [port]]");
                    return;
                }

                // First, you get the host name and the port number for the data listener.
                // You need to ensure that the data listener is running on the host server
                // and listening to the port.
```

```
            // Otherwise, you might get a connection refused exception.
            String hostname = "localhost";
            if (argv.length > 0) hostname = argv[0];
            int port = 6668;
            if (argv.length > 1) {
                try {
                    port = Integer.parseInt(argv[1]);
                }
                catch (Exception e) {
                   port = 6668;
                }
            }

            // Assume that client_id_1 is a valid client ID with password_1.
            // Assume that client_id_1 has the authorization to update the
            // collection with collection ID collection_id_1.
            //
            // You need to contact the enterprise search administrator to find out what
            // client ID, password, and collection ID are valid and
            // use those valid values here.
            //
            dataPushExample(hostname, port, "client_id_1", "password_1",
                            "collection_id_1");
        }
    }
```

Sample data listener client application: revisiting URLs

The DLRevisitURLs class includes an example of client application APIs that instruct the Web crawlers to revisit specific URLs or URL patterns.

The DLRevisitURLs class defines two static methods: main and revisitURLsExample.

The main method first prepares the host name, port, client ID, password, and collection ID. Then main method calls the revisitURLsExample method with those parameters.

The revisitURLsExample method takes the host name, port, client ID, password, and collection ID as parameters, and it prepares the URLs or URL pattern strings.

After preparing the URLs or URL patterns, the revisitURLsExample method calls the revisitURLs method of the DLDataPusher class. Then, the revisitURLsExample method checks and prints the response from the data lister.

```
public class DLRevisitURLs {

    /**
     *  To revisit URLs, you need to specify the host name
     *  and the port of the data listener server.
     *  You also need to provide the client ID and password for
     *  authentication.
     *  You need to specify which collection the data is applied
     *  to by providing the collection ID.
     *
     *  This function shows you how to use the data listener APIs
     *  to instruct the Web crawler of a collection to revisit
     *  URLs or URLs that match patterns.
     */
    static void revisitURLsExample(String hostname, int port,
    String clientID, String passwd, String collectionID) {

        // You will revisit some URLs from one collection.
        // URLs can be either individual URLs or URLs that
        // match patterns.
```

```
                StringBuffer sb = new StringBuffer();
                sb.append("uri1");
                sb.append("\n");
                sb.append("uri*pattern1");
                sb.append("\n");

                sb.append("uri2");
                sb.append("\n");
                sb.append("uri*pattern2");

                String uris = sb.toString();

                // Now, you call the data listener API to revisit URLs.
                // Note that you can specify multiple URLs and URL
                // patterns in a single call. The URLs (URL patterns)
                // are seperated by a newline character.
                DLResponse dlRes = DLDataPusher.revisitURLs(hostname, port,
                clientID, passwd, uris, collectionID);

                // Check the result from the data listener.
                // The DLResponse object contains a result code and a
                // message that indicates the result of the operation.
                // See the documentation for the DLResponse class for
                // the code values and their meanings.
                if (dlRes != null) System.out.println(dlRes.toString());
            }

            public static void main (String [] argv) {
                if (argv.length > 0 && argv[0].equalsIgnoreCase("-h")) {
                    System.out.println("usage: java DLRevisitURLs [hostname [port]]");
                    return;
                }

                // First, you get the host name and the port number for
                // the data listener. You need to ensure that the data
                // listener is running on the host server and listening
                // to the port.
                // Otherwise, you might get a connection refused exception.
                String hostname = "localhost";
                if (argv.length > 0) hostname = argv[0];
                int port = 6668;
                if (argv.length > 1) {
                    try {
                        port = Integer.parseInt(argv[1]);
                    }
                    catch (Exception e) {
                       port = 6668;
                    }
                }

                // Assume that client_id_1 is a valid client ID with password_1.
                // If not, you will get INVALID_PASSWD as the result code.

                // Assume that client_id_1 has the authorization to update
                // collection with collection ID collection_id_1.
                // If not, you will get PERMISSION_DENIED as the result code
                //
                // Assume that the collection  with collection id collection_id_1
                // is a valid collection. Otherwise, you will get
                // UNKNOWN_COLLECTION as the result code.
                //
                // You need to contact the enterprise search administrator
                // to find out what  client ID, password, and collection ID
                // are valid and use those valid values here.
                //
```

```
            revisitURLsExample(hostname, port, "client_id_1", "password_1",
            "collection_id_1");
        }
    }
```

Sample data listener client application: adding, removing, and revisiting data in a collection

The DLSampleClient class provides sample code that demonstrates how to add, remove, and revisit data for one or more collections.

The DLSampleClient combines the sample client applications for adding URIs, removing URLs, and revisiting URLs.

```
import java.io.*;
import java.net.*;
import java.util.Date;

import com.ibm.es.datalistener.client.*;
import com.ibm.es.datalistener.common.*;
import com.ibm.es.util.DataSourceMetadata;

/**
 * This class is sample code that shows you how to use the data listener APIs
 * to update collections.
 */
public class DLSampleClient {

    /**
     *  This example shows how to use data listener APIs to revisit URLs of a
     *  collection. You need to specify the host name
     *  and the port of the data listener server.
     *  You also need to provide the client ID and password for authentication.
     *  You need to specify which collection this operation is applied to
     *  by providing the collection ID.
     */
    static void dataPushExample_1(String hostname, int port, String clientID,
    String passwd, String collectionID) {

        // You will visit (add) and revisit several URLs of one collection.
        // You can revisit URLs or URLs that match patterns. URLs and
        // URL patterns are separated by a newline character. A URL pattern
        // is a string with a wildcard character (*).
        //
        StringBuffer sb = new StringBuffer();
        sb.append("url1");
        sb.append("\n");
        sb.append("url*pattern1");
        sb.append("\n");

        sb.append("url2");
        sb.append("\n");
        sb.append("url*pattern2");

        String urls = sb.toString();

        // Now, you call the revisitURLs method of DLDataPusher class.
        DLResponse dlRes = DLDataPusher.revisitURLs(hostname, port, clientID,
        passwd, urls, collectionID);

        // Check the response from the data listener.
        if (dlRes != null) System.out.println(dlRes.toString());
    }

    /**
     *  This example shows how to use data listener APIs to remove URIs of a
```

```
 *  collection. You need to specify the host name
 *  and the port of the data listener server.
 *  You also need to provide the client ID and password for authentication.
 *  You need to specify which collection this operation is applied to
 *  by providing the collection ID.
 */
static void dataPushExample_2(String hostname, int port, String clientID,
String passwd, String collectionID) {

    // You now remove some URIs from a collection.
    // URIs are separated by a  newline characters.
    // Those URIs will be removed immediately so that they do
    // not appear in the search result.
    StringBuffer sb = new StringBuffer();
    sb.append("url1");
    sb.append("\n");
    sb.append("url2");
    String urls = sb.toString();

    // Now, you call the removeURIs method of DLDataPusher class.
    DLResponse dlRes = DLDataPusher.removeURIs(hostname, port, clientID,
    passwd, urls, collectionID);
    // Check the response from the data listener.
    if (dlRes != null) System.out.println(dlRes.toString());



    // You can also remove URIs that match patterns.
    // URI patterns are separated by newline characters.
    // Those URIs that match these patterns will be removed
    // during the next index reorganization. Note that
    // these results might still appear in the search result until
    // the next index reorganization.
    sb = new StringBuffer();
    sb.append("url*pattern1");
    sb.append("\n");
    sb.append("url*pattern2");
    String url_patterns = sb.toString();

    // Now, you call the removeURIs method of DLDataPusher class.
    dlRes = DLDataPusher.removeURIs(hostname, port, clientID, passwd,
    url_patterns, collectionID);
    // Check the response from the data listener.
    if (dlRes != null) System.out.println(dlRes.toString());

    // You can even remove both individual URLs and URLs that match
    // patterns in the same request.
    sb = new StringBuffer();
    sb.append("url3");
    sb.append("\n");
    sb.append("url*pattern4");
    sb.append("\n");
    sb.append("url5");

    // Now, you call the removeURIs method of DLDataPusher class.
    dlRes = DLDataPusher.removeURIs(hostname, port, clientID, passwd,
    sb.toString(), collectionID);
    // Check the response from the data listener.
    if (dlRes != null) System.out.println(dlRes.toString());

}

/**
 *  This example shows how to use data listener APIs to push
 *  documents to a collection. You need to specify the host name
 *  and the port of the the data listener server.
 *  You also need to provide the client ID and password for authentication
```

```
     *  You need to specify which collection the data is applied to
     *  by providing the collection ID.
     */
    static void dataPushExample_3(String hostname, int port, String clientID,
    String passwd, String collectionID) {

        // Prepare the content.
        String content = "Almost empty";

        // Prepare the URI.
        String uri = "myURI2";

        // Prepare the metadata.
        //
        // First, create a DataSourceMetadata object
        DataSourceMetadata md =
   DLDataPusher.createDataSourceMetadata("CustomerDataSource",
    clientID,
                          "CustomerDataSourceName",
                          90,
                          new Date(),
                          "en",
                          "securityToken",
                          "text/plain",
                          "iso-8859-1",
                          content.getBytes());

        // Second, add more fields to the metadata.
        // Each field is a name/value pair.
        // The fourth argument specifies whether the field value is
        // searchable.
        // The fifth argument specifies whether the field value will be part
        // of the search result.
        // The sixth argument specifies whether to support field search.
        // The seventh argument specifies whether the field is parametric
        // searchable.
        // The eight argument specifies whether the field is part of the
        // content.
        DLDataPusher.addMetaField(md,
                                  "fieldName1", "fieldValue1",
                                  true, false, true, false, false);
        DLDataPusher.addMetaField(md,
                                  "fieldName2", "fieldValue2",
                                  true, false, true, false, false);
        System.out.println("metadata:\n" +  md.generateXML().toString());

        // Call the pushData method
        DLResponse dlRes = DLDataPusher.pushData(hostname, port, clientID,
        passwd, uri, collectionID, md, content.getBytes());

        // Check the response from the data listener.
        if (dlRes != null) System.out.println(dlRes.toString());

        // Push the same result again. This one will overwrite the previous one.
        dlRes =  DLDataPusher.pushData(hostname, port, clientID, passwd, uri,
        collectionID, md, content.getBytes());
        if (dlRes != null) System.out.println(dlRes.toString());
    }

    public static void main (String [] argv) {
        if (argv.length > 0 && argv[0].equalsIgnoreCase("-h")) {
            System.out.println("usage: java DLSampleClient [hostname [port]]");
            return;
        }

        // First, you obtain the host name and the port number for the data
        // listener.
```

```
                        // You need to ensure that the data listener is running on the
                        // host server and listening to the port.
                        // Otherwise, you might get a connection refused exception.
                        String hostname = "localhost";
                        if (argv.length > 0) hostname = argv[0];
                        int port = 6668;
                        if (argv.length > 1) {
                            try {
                                port = Integer.parseInt(argv[1]);
                            }
                            catch (Exception e) {
                               port = 6668;
                            }
                        }

                        // Assume that client_id_1 is a valid client ID with password_1.
                        // Assume that client_id_1 has the authorization to update the
                        // collection with collection ID collection_id_1.
                        //
                        // You need to contact the enterprise search administrator to find
                        // out what client ID, password, and collection ID are valid and
                        // use those valid values here.
                        //
                        dataPushExample_1(hostname, port, "client_id_1", "password_1",
                        "collection_id_1");

                        dataPushExample_2(hostname, port, "client_id_1", "password_1",
                        "collection_id_1");


                        // Assume that client_id_2 is a valid client ID with password_2.
                        // Assume that client_id_2 has the authorization to update the
                        // collection with collection ID collection_id_2.
                        //
                        dataPushExample_2(hostname, port, "client_id_2", "password_2",
                        "collection_id_2");

                        dataPushExample_3(hostname, port, "client_id_1", "password_1",
                        "collection_id_1");
                    }
                }
```

Chapter 5. Linguistic support

Enterprise search offers linguistic search support for text documents in most Indo-European languages and Asian languages such as Japanese.

The purpose of linguistic support during search is to improve document search results, that is, to arrive at the best possible collection of documents that match the query.

Linguistic processing is performed during two stages: when a document is processed to be added into the index, and when a user inputs a query during search.

Enterprise search includes only granular linguistic functionality that is required for determining the language of an input document and segmenting the document input stream into words or tokens.

If you know that your search will be restricted primarily to basic keyword search or native XML search using the document structure, the linguistic processing included in enterprise search will adequately cover what you require for search.

However, this type of processing alone is not always satisfactory if you want to search more specifically beyond the mere words in the document, for example:

- In collaboration cases, involving searching for information that is not always explicitly marked, for example, searching for a particular address or phone number
- In competitive intelligence, searching for documents that mention competitors and the goods that they supply, or in recognizing that your competitor's Web site has shifted over the past three months from one product set to another
- In customer relationship management, searching for documents that mention particular car brake problems in repair shops in the San Francisco area
- In research, searching for documents that talk about a particular protein and its relation to at least one disease appearing in the same paragraph

In these scenarios, searching for what you need in the vast collections of information sources that exist today presents new challenges requiring sophisticated analysis that surpasses the segmentation level and dictionary-based analysis that is offered in enterprise search. Most of the information that is of interest is not explicitly tagged or marked in any way in the original document. Instead, the information must be analyzed to recognize and locate concepts of interest, for example, named entities like persons, organizations, locations, facilities and products, and the possible relations between these concepts.

The IBM Unstructured Information Management Architecture (UIMA) is an architecture and software framework that helps you build the advanced analysis capabilities in enterprise search you require to detect and locate information of interest in document collections.

Related concepts

Chapter 6, "Custom text analysis integration," on page 47
Unstructured information management architecture (UIMA) is a software

architecture that supports creating, discovering, composing and deploying analysis capabilities. Using UIMA, you can build your own custom text analysis.

"Unstructured information management architecture (UIMA)" on page 48
Unstructured information management architecture (UIMA) is an architecture and software framework that helps you build the advanced analysis capabilities that can find specific information in document collections.

Chapter 6. Custom text analysis integration

Unstructured information management architecture (UIMA) is a software architecture that supports creating, discovering, composing and deploying analysis capabilities. Using UIMA, you can build your own custom text analysis.

UIMA is an open platform that identifies components for each conceptually distinct analysis function, and it ensures that these components can be easily reused and combined with one another.

A central concept of UIMA is the *analysis engine*, which is responsible for discovering and representing analysis content in text documents. The analysis logic component is called the *annotator*. An annotator focuses only on an analysis task and is not concerned with any other processing. An analysis engine may contain a single annotator or it may be a composition of many engines, each in turn containing annotators.

The inferred information that is produced by an analysis engine is referred to as the *analysis results*. Ideally, the analysis results correspond to the information you want to search for.

Advanced linguistic analysis includes a combination of many differing analysis tasks. The analysis begins with, for example, language detection and segmentation, and continues with part-of-speech recognition, followed by deep grammatical parsing. The last step includes the identification of, for example, the relation between certain chemical substances and the appearance of particular symptoms. Each step in the analysis process is required by the subsequent step.

UIMA provides the basic building blocks for you to create, test, and deploy your own analysis engines; it does not provide you with any linguistic analysis functionality in the form of pre-configured analysis engines that you can deploy in your UIMA environment.

The UIMA Software Development Kit (SDK) includes a Java implementation of the UIMA framework for the implementation, description, composition and deployment of UIMA components. It also provides an Eclipse-based development environment that includes a set of tools and utilities for using UIMA. For information on Eclipse, see www.eclipse.org.

To work with UIMA, you must install the UIMA Software Development Kit. The development kit is available on IBM developerWorks®. Visit the WebSphere Information Integrator zone for information at http://www.ibm.com/developerworks/db2/zones/db2ii/. Refer to the UIMA documentation for instructions on how to install the UIMA Software Development Kit in the Eclipse Interactive Development Environment.

Related concepts

Chapter 5, “Linguistic support,” on page 45
Enterprise search offers linguistic search support for text documents in most Indo-European languages and Asian languages such as Japanese.

“Unstructured information management architecture (UIMA)” on page 48
Unstructured information management architecture (UIMA) is an architecture

and software framework that helps you build the advanced analysis capabilities that can find specific information in document collections.

Unstructured information management architecture (UIMA)

Unstructured information management architecture (UIMA) is an architecture and software framework that helps you build the advanced analysis capabilities that can find specific information in document collections.

A *feature structure* is the underlying data structure that represents the result of an analysis. A feature structure is an attribute-value structure. Each feature structure is of a type and every type has a specified set of valid features or attributes (properties), much like a Java class. Features have a range type that indicates the type of value that the feature must have, for example, String.

Most analysis programs, also referred to as annotators, produce their analysis results in the form of annotations. Annotations are a special kind of feature structure that is designated for linguistic analysis processing. A feature structure spans or covers a piece of input text and is defined in terms of its beginning and end positions in the input text.

For example, an annotator that recognizes monetary expressions would create, for the text "100.55 US Dollars", an annotation of type `monetaryExpression` that covers the text, with the feature `currencySymbol` set to "$".

All annotators in UIMA use feature structures to store or read information, in other words, all data is modeled as feature structures.

The type system defines all possible feature structures in terms of types and features, much like a class hierarchy in Java.

All feature structures are represented in a central data structure called the *common analysis structure*. All data exchange is handled using the common analysis structure.

The common analysis structure contains the following objects:

- The text document
- The type system description indicating the types, subtypes, and their features
- Analysis results describing the document or a region of the document
- An index repository that supports access to and iteration over the analysis results

Related concepts

Chapter 5, "Linguistic support," on page 45
Enterprise search offers linguistic search support for text documents in most Indo-European languages and Asian languages such as Japanese.

Chapter 6, "Custom text analysis integration," on page 47
Unstructured information management architecture (UIMA) is a software architecture that supports creating, discovering, composing and deploying analysis capabilities. Using UIMA, you can build your own custom text analysis.

Workflow for custom analysis integration

Custom text analysis algorithms are created and tested using the UIMA Software Development Kit, and then deployed and run on document collections in enterprise search.

The main steps involved when developing your own analysis algorithms and incorporating this functionality in enterprise search are the following:

1. Planning and design
   a. Determine what information you want to search for. What are the documents you want to retrieve?
   b. Specify the kind of text analysis that you need to retrieve the information in the documents that you want to search.
   c. If your collection contains XML documents, decide if you want to exploit the XML markup in your solution. In enterprise search, you can use XML markup in one of two ways:
      - If you can use the XML markup in your custom analysis (for example, your documents contain `<summary>` or `<topic>` elements which can be useful in a summarization or categorization annotator), define XML to common analysis structure mappings.
      - If you want to use the XML markup as it appears in the document in your queries, enable native XML mapping.
   d. Determine which text analysis result information stored in the common analysis structure you want to be able to access using semantic search. Define the common analysis structure to index mapping.
2. Development: UIMA Software Development Kit activities
   a. Define the individual analysis steps.
   b. Describe the type system for your mappings and analysis logic.
   c. Develop the analysis logic (annotator) for each analysis step and embed the annotators in analysis engines using the UIMA Software Development Kit. You should build any custom analysis on top of the basic functionality (language identification and tokenization) in the enterprise search text analysis package when you develop your annotators.
   d. After testing the analysis algorithms in UIMA, package the analysis engine as a PEAR (Processing Engine Archive) file. The archive must only contain your analysis algorithms, and not the basic enterprise search linguistic functionality.
3. Deployment: Enterprise search activities
   a. Upload the analysis engine archive (.pear) to enterprise search, giving the analysis component a name by which you to refer to it in enterprise search.
   b. Associate one or more document collections with your analysis engine.
   c. For each collection, upload and select the XML to common analysis structure mappings that you defined for your custom analysis.
   d. For each collection, upload and select the common analysis structure to index mapping that you defined for semantic search.

Text analysis algorithms

The UIMA Software Development Kit includes APIs and tools with which you can create annotators (analysis algorithms including the type system description) and embed these in analysis engines.

The UIMA documentation includes a tutorial-style guide that helps you build these components. The Software Development Kit includes utilities for testing and viewing your results, and a small scale semantic search engine for indexing your analysis results. You can also perform more advanced search against information stored in the index.

The UIMA Software Development Kit does not provide any pre-configured analysis engines. However, you can use the basic linguistic support that is offered in enterprise search in a UIMA environment. Refer to the UIMA documentation on how to include language detection and tokenization functionality before your text analysis algorithms when developing these in your UIMA environment.

After you finish developing and testing your analysis engines using the UIMA Software Development Kit, and want to run these algorithms on a document collection in enterprise search, you must create a PEAR (Processing Engine ARchive) file. This archive file includes all of the required resources for deploying your custom analysis functionality as analysis engines in enterprise search. All of the processing steps that are required for creating an archive are described in the UIMA documentation provided in the Software Development Kit.

The archive must only contain your custom analysis, even if it is founded on the basic linguistic functionality offered in enterprise search. The basic enterprise search analysis steps always run before any custom analysis.

Approaches for mapping XML document structures to a common analysis structure

You can map information in XML structures in a document directly to a common analysis structure without you having to write a UIMA annotator.

Using a configuration file, you can determine which XML elements in your input documents contain either content that forms the basis on which further analysis can be performed, or which elements contain document metadata, for example, the author or the date.

Sometimes, knowledge about the content and semantics of certain elements is precise and you can map information directly to specific feature structures without any further analysis. For example, the element `<addressee>` in documents on billings usually contains customer names, and these names in combination with price information automatically infer a person-price relation or transaction.

You can select to use more than one XML configuration for a document collection. Which configuration is used for which XML document is determined by the `<identifier>` element. The `<identifier>` element in the configuration file must match the root element in the XML document. For example, if the root element of your document is `doc`, the value of the `<identifier>` element in the configuration file must also be "doc".

If no match is found, the program will search for a configuration file with the `<identifier>` element set to "Default". If no default configuration is found, the textual sections of the document (with no tag information) are mapped to the document annotation in the common analysis structure.

There are basically two types of mapping:

Element-to-data type mapping

If you know that the documents in your collection share the same XML structure (the same elements), you can map these XML elements directly to UIMA feature structures. For example, the element <addressee> can be mapped to feature structure of type customer, itself derived from the feature structure of type person.

You can also set the attributes of features structures. For example, if you know that all of the documents in your collection mention customers who have acquired a certain version of a product, you can set the attribute product version to this value without having to do this explicitly during an analysis process.

Instead of setting values only at configuration time, you can set values dependent on the content of a mapped element, for example, that the content of an element must be a string value.

You can also use the text covered by other elements within the element you are mapping (referred to as nested elements) and set a feature value to the text in the a nested element. For example, if the elements <firstName>, <middleName>, and <lastName> are all contained within the element <author>, you can set the value of a feature author to the whole text covered by the nested elements.

Additionally, you can specify conditions for mappings, for example that an element occurs at a certain place in the document structure (extract the content of all <addressee> elements in <body> regardless of whether they appear within a <shipping> element), or that an element has a certain attribute, possibly even with a certain value.

If you want to extract information that is only contain in relevant parts of a document, while ignoring irrelevant parts, simply specify which XML elements in the documents contain relevant information. This is referred to as content extraction. For example, you can extract the input specified in the title and body elements, while ignoring the input in author, date, ID, and publisher.

Content extraction can improve analysis processing for the following types of XML documents:

- Documents that contain large amounts of content not subject to analysis, for example, binary attachments. Using content extraction reduces the document size significantly, speeding up processing and avoiding analysis errors that originate from unsuitable data.
- Documents in which document text is interspersed with irrelevant text, for example, documents that contain editorial information within <note> tags. Ignoring this information leads to better results when analyzing the document content.

Native XML mapping

This process involves removing all of the XML tagging information from a document and using the remaining document content for possible further analysis.

To keep the XML structuring information for the document, the name of the elements, their attributes and values are saved in enterprise search using the com.ibm.es.tt.MarkupTag type.

This way, XML information is accessible for native XML search. Native XML mapping does not require a mapping configuration file; you can enable native XML mapping using the administration console for enterprise search.

The native XML mapping and content extraction options of the element-to-type mapping contradict each other, as either all content, or only specified content can be considered. If you specify the content extraction option, native XML mapping is ignored. Without content extraction, it is possible to have both element-to-type mapping and native XML mapping.

All of the types and features that you use in your XML mapping configuration file must be described in the type system description of your custom analysis steps. You can create a type system descriptor in your UIMA environment using the Component Descriptor Editor Eclipse plug-in. This plug-in allows you to create a descriptor file without having to worry about the required XML syntax.

After you have completed building and testing your custom analysis, use the UIMA PEAR (Processing Engine ARchive) generation wizard to create an archive containing your custom analysis files including your type system description.

Thereafter, you can upload the custom analysis archive and your XML mapping configuration files into enterprise search using the administration console for enterprise search.

Related tasks

"XML mapping configuration file"
In an XML mapping configuration file, you can employ the full range of configuration options for mapping XML to a common analysis structure.

Related reference

"XML mapping sample and the output results" on page 55
Mapping XML document structures to the custom analysis structure is illustrated based on a short sample document.

XML mapping configuration file

In an XML mapping configuration file, you can employ the full range of configuration options for mapping XML to a common analysis structure.

The configuration file must be an XML file and must comply with a specific schema. To avoid XML syntax errors, write the configuration file using an XML editor.

The XML mapping schema is split up into two parametric sections:

- **<contentElements>**: Use this element if you want specific content extraction, for example, to extract the content in the <Abstract> and <Body> sections of a document. All other sections in the document will be ignored.
- **<elementToTypeMappings>**: Use this element to specify which individual XML elements (specified in an <elementToTypeMapping> element) in the document to map to which feature structures in the common analysis structure.

  If you use the content extraction option, note that the XML elements specified in the <elementToTypeMappings> section must be contained within the XML elements specified in the <contentElements> section.

A sample configuration file is as follows:

```
<?xmlversion="1.0"?>
<xmlCasInitializerConfiguration
 xmlns="http://www.ibm.com/2005/uima/jedii_ci_xml">

 <identifier>Default</identifier>
 <description>Sample configuration</description>
```

```
 <contentElements>
   <element>/doc/Title</element>
   <element>/doc/Abstract</element>
   <element>/doc/Body</element>
 </contentElement>

 <elementToTypeMappings>
   <elementToTypeMapping>
     <element>//Employees//IBMer</element>
     <type>example.Person</type>
     <featureValueAssignment>
       <feature>employer</feature>
       <basicValue>default="IBM"</basicValue>
     </featureValueAssignment>
     <featureValueAssignment>
       <feature>age</feature>
       <basicValue default="34"/>
     </featureValueAssignment>
   </elementToTypeMapping>
 </elementToTypeMappings>

</xmlCasInitializerConfiguration>
```

Each `<elementToTypeMapping>` must contain the following elements:

- An `<element>` element that is used to specify the path of an XML element and follows XPath syntax: A leading `"/"` means that a full path is given, for example, `Title` under the root element `doc`, `"//"` means any path subset, for example, `IBMer` must occur within `Employees`, although other elements can occur between these two.
- A `<type>` element, which specifies a type defined in the type system description.
- Zero or more `<featureValueAssignment>` elements.

In a `<featureValueAssignment>` element, a feature must be named in the `<feature>` element and a value assigned in the `<basicValue>` element. The `<basicValue>` element can have the following attributes. Note that the attributes `useAttributeValue` and `useContentValue` are mutually exclusive:

- `<basicValue useAttributeValue="date" default="UNKNOWN">` takes the value of the attribute `date` for the defined feature. If the attribute does not exist, use the default value `UNKNOWN`.
- `<basicValue useContentValue="date" trim="yes" default="UNKNOWN">` takes the content in the XML element `date`, trimmed of any blanks, as the value for the defined feature. If there is no content in `date`, use the default value `UNKNOWN`.
- `<basicValue default="UNKNOWN">` always takes the default value, in this case, `UNKNOWN`.

Use `useAttributeValue` if you want to use the value of an attribute as the value for a feature. For example, the following configuration snippet:

```
<elementToTypeMapping>
  <element>/Doc//IBMer</element>
  <type>example.Person</type>
  <featureValueAssignment>
    <feature>age</feature>
    <basicValue useAttributeValue="age"/>
  </featureValueAssignment>
</elementToTypeMapping>
```

results in the following:

- For each `IBMer` tag that occurs within a `Doc` tag, a feature structure of type `example.Person` is created.

- If the IBMer tag contains an attribute age, the feature age of the newly created feature structure is set to the value of the attribute.

To add content as the value of a feature, use the attribute useElementContent. For example, the text covered by the element author in book becomes the value of the feature author and all leading and trailing blanks are removed:

```
<elementToTypeMapping>
  <element>//book</element>
  <type>example.Book</type>
  <featureValueAssignment>
    <feature>author</feature>
    <basicValue useElementContent="author" trim="true"/>
  </featureValueAssignment>
</elementToTypeMapping>
```

More than one value is specified between the <values> element for the following cases:

- The feature is of type array. For example, 7 and 12 are the values in the feature kidsAges of type integer array:

```
<elementToTypeMapping>
  <element>/Doc/Body/Father</element>
  <type>example.Father</type>
  <featureValueAssignment>
    <feature>kidsAges</feature>
    <values>
      <basicValue default="7"/>
      <basicValue default="12"/>
    </values>
  </featureValueAssignment>
</elementToTypeMapping>
```

- Many strings are concatenated to one string and therefore map to a feature of type String. For example, the title Mr. is a constant, the first name is the value of an attribute and the last name is covered by an XML element:

```
<elementToTypeMapping>
  <element>//IBMer</element>
  <type>example.Person</type>
  <featureValueAssignment>
    <feature>fullName</feature>
    <values concatenate="true" delimiter=" ">
      <basicValue default="Mr."/>
      <basicValue useAttributeValue="firstName"/>
      <basicValue useElementContent="IBMer"/>
    </values>
  </featureValueAssignment>
</elementToTypeMapping>
```

String feature values are extracted from the configuration file as is. The values retain any leading or trailing blanks. However, names of types and features are trimmed of any blanks.

You can also set conditions on attributes using the <condition> element. For example, the feature structure of type example.IBMFather is created only if Employee occurs in the document with attribute employer set to ibm, and has a second attribute sonsName.

```
<elementToTypeMapping>
  <element>//Employee</element>
  <type>example.IBMFather</type>
  <condition attribute="employee" value="ibm"/>
  <condition attribute="sonsName"/>
</elementToTypeMapping>
```

Related concepts

"Approaches for mapping XML document structures to a common analysis structure" on page 50
You can map information in XML structures in a document directly to a common analysis structure without you having to write a UIMA annotator.

Related reference

"XML mapping sample and the output results"
Mapping XML document structures to the custom analysis structure is illustrated based on a short sample document.

XML mapping sample and the output results

Mapping XML document structures to the custom analysis structure is illustrated based on a short sample document.

The sample XML input document has the following structure:

```
<Doc>
By
  <Head>
  <Author><FirstName>Nina</FirstName><LastName>Eisenberg</LastName></Author>
and
  <Reviewer>Tanja Stahlhugel</Reviewer>
  <Date type="text">26 November 2004</Date>
  <GMT type="gmt">26 November 2004</GMT>
A publication of
  <Publisher>The San Diego Journal</Publisher>
  <Title>Chip, heal themself</Title>
  </Head>
  <Abstract>I.B.M. invents chips that reconfigure themselves.</Abstract>
  <Body>At I.B.M., researchers have designed chips with built-in fuses
that can do some self-repair jobs, said
    <IBMer>Subramanian S. Iyer</IBMer>, an inventor of the technology. Miss
    <OtherEmployee  company="Siemens" firstName="Gundula"
      middleInitial="O." age="56">Baumgarten</OtherEmployee>
from Siemens congratulated. Another employee, Mister
    <OtherEmployee firstName="Titus">Jones</OtherEmployee> said ...

Mister
<Father lastName="Clark">Clark is father of <Child>Clara</Child></Father>
</Body>
</Doc>
```

The corresponding XML to common analysis structure sample configuration mapping is as follows:

```
<?xmlversion="1.0"?>
<xmlCasInitializerConfiguration>
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="XMLCasInitSchema.xsd">

 <identifier>Doc</identifier>
 <description>Sample configuration</description>

 // content elements from which to extract the text. Abstract is ignored
 <contentElements>
   <element>/Doc/Title</element>
   <element>/Doc/Body</element>
   <element>/Doc/Head</element>
 </contentElement>

 // single constant feature values
 <elementToTypeMappings>
   <elementToTypeMapping>
```

```
  <element>/Doc//IBMer</element>
  <type>example.Person</type>
  <featureValueAssignment>
    <feature>employer</feature>
    <basicValue useAttributeValue="company" trim="true"
      default="IBM"/>
  </featureValueAssignment>
  <featureValueAssignment>
    <feature>age</feature>
    <basicValue useAttributeValue="age" default="-1"/>
  </featureValueAssignment>
 </elementToTypeMapping>
 <elementToTypeMapping>
  <element>/Doc//OtherEmployee</element>
  <type>example.Person</type>
  <featureValueAssignment>
    <feature>employer</feature>
    <basicValue useAttributeValue="company" trim="true"
       default="IBM"/>
         </featureValueAssignment>
  <featureValueAssignment>
    <feature>age</feature>
    <basicValue useAttributeValue="age" default="-1" />
  </featureValueAssignment>
  <featureValueAssignment>
    <feature>fullName</feature>
    <values concatenate="true" delimiter=",">
      <basicValue useAttributeValue="firstName" />
      <basicValue useElementContent="OtherEmployees"
         default="UNKNOWN"/>
    </values>
  </featureValueAssignment>
 </elementToTypeMapping>
 // annotate nested elements
 <elementToTypeMapping>
  <element>//Head</element>
  <type>example.Book</type>
  <featureValueAssignment>
    // the feature author is set to the contents of the element Author
    <feature>author</feature>
    <basicValue useElementContent="Author"/>
  </featureValueAssignment>
  <featureValueAssignment>
    <feature>publisher</feature>
    <basicValue useElementContent="Publisher" trim="true"
       default="UNKNOWN"/>
  </featureValueAssignment>
  <featureValueAssignment>
    <feature>title</feature>
    <basicValue useElementContent="Title" />
  </featureValueAssignment>
</elementToTypeMapping>
<elementToTypeMapping>
  <element>/Doc//Father</element>
  <type>example.Father</type>
  <featureValueAssignment>
    <feature>kidsNames</feature>
    <values>
      <basicValue useElementContent="Child"/>
      <basicValue default="Sarah"/>
    </values>
  </featureValueAssignment>
  <featureValueAssignment>
    <feature>kidsAges</feature>
      <values>
        <basicValue default="5" />
      </values>
```

```
    </featureValueAssignment>
  </elementToTypeMapping>
  // annotate the date if the attribute type=text is true
  <elementToTypeMapping>
     <element>/Doc//Date</element>
     <type>example.Date</type>
     <condition attribute="type" value="text" />
     <featureValueAssignment>
       <feature>date</feature>
       <basicValue useElementContent="Date"/>
     </featureValueAssignment>
     <featureValueAssignment>
       <feature>gmt</feature>
       <basicValue default="-1" />
     </featureValueAssignment>
  </elementToTypeMapping>
  // annotate the date if the attribute type=long is true.
  // In input text this condition is not true, so no annotation is
  // created.
  <elementToTypeMapping>
     <element>/Doc//GMT</element>
     <type>example.Date</type>
     <condition attribute="type" value="long" />
     <featureValueAssignment>
       <feature>gmt</feature>
       <basicValue useElementContent="GMT"/>
     </featureValueAssignment>
     <featureValueAssignment>
       <feature>date</feature>
       <basicValue default="UNKNOWN" />
     </featureValueAssignment>
  </elementToTypeMapping>

 </elementToTypeMappings>

</xmlCasInitializerConfiguration>
```

Based on the configuration file, the following annotations are produced in the common analysis structure:

```
uima.tcas.DocumentAnnotation:
 begin=0
 end=479
 covered text:
"  Nina Eisenberg
and
Tanja Stahlhugel
26 November 2004
26 November 2004
A publication of The San Diego Journal
Chip, heal themself

At I.B.M., researchers have designed chips with
built-in fuses that can do some self-repair jobs,
said Subramanian S. Iyer,
an inventor of the technology.

Miss Baumgarten from Siemens congratulated.

Another employee,
Mister Jones
said ......

Mister  Clark is the father of Clara."

example.Book:
 begin=0
```

```
 end=166
 author= "Nina Eisenberg"
 publisher="The San Diego Journal"
 title="Chip, heal themself"
 covered text:
"  Nina Eisenberg
and
Tanja Stahlhugel
26 November 2004
26 November 2004

A publication of The San Diego Journal
Chip, heal themself"

example.Date:
 begin=55
 end=71
 date="26 November 2004"
 gmt=-1
 covered text: "26 Novmber 2004"

example.Person:
 begin=277
 end=296
  employer = "IBM"
 fullName = null
 age = -1
 covered text: "Subramanian S. Iyer"

example.Person:
 begin=342
 end=352
  employer = "Siemens"
 fullName = "Gundula,Baumgarten"
 age = 56
 covered text: "Baumgarten"

example.Person:
 begin=414
 end=419
  employer = "IBM"
 fullName = "Titus,Jones"
 age = -1
 covered text: "Jones"

example.Father:
 begin=446
 end=477
 kidsNames = [Clara,Sarah]
 kidsAges = [5]
 covered text: " Clark is father of Clara. "
```

Related concepts

"Approaches for mapping XML document structures to a common analysis structure" on page 50
You can map information in XML structures in a document directly to a common analysis structure without you having to write a UIMA annotator.

Related tasks

"XML mapping configuration file" on page 52
In an XML mapping configuration file, you can employ the full range of configuration options for mapping XML to a common analysis structure.

Approaches for indexing custom analysis results

After you have run your custom analysis on a collection of documents, you can use the search engine in enterprise search to build an index from the information stored in the common analysis structure produced by the custom analysis algorithms.

Mapping your analysis results to fields, spans of text and attributes in the enterprise search index enables you to use this information in queries. Combining custom analysis with enterprise search that is capable of indexing both words and spans of text, enables semantic search.

Using an index build configuration file, you can determine which analysis results in the common analysis structure you want to index.

There are different styles that you can use to map feature structures in the common analysis structure to the enterprise search index.

Annotation

If you index feature structures in the common analysis structure using the annotation style, all annotations of the specified types will be stored in the index as searchable spans.

For example, if a feature structure spanning a certain area of text is of type `person` and is indexed using the annotation style, the following queries are possible:

Table 2. Sample queries

Required information	Possible query
Give me all documents that contain at least one person name	<person/>
Give me all documents where boss is contained within a person annotation	<person>boss</person>
Give me all documents where Lang is mentioned in the same sentence as one of my competitors	<sentence><person>Lang</person> <competitor/></sentence>

Attributes of feature structures are also indexed as part of the span. For example, consider an annotator that detects cars and stores the car make as a feature `make` of the `car` annotation. This enables the following type of query: "Give me documents that mention cars of the make Chevrolet".

Field Use this style if you want to make the content of feature structures accessible during search using the field search capabilities in enterprise search. In this way, the content of a feature structure can be displayed in search results, or used in parametric search.

For example, if you map drug dosages to a parametric field, you can query as follows: "Give me all documents that talk about a certain drug taken at a dosage above 100 milligrams".

Breaking

Use this style if you want a particular feature structure to be interpreted as a clear delimiter, for example, sections or paragraphs. Enterprise search detects sentences and paragraphs by default. Use this style only if your

custom analysis detects additional structural elements in a document that you want to have interpreted differently.

After you have written the index build configuration file, you can upload the index build configuration file into enterprise search using the administration console for enterprise search.

Definition of a feature path

The index build specification uses feature paths to specify feature structures in the common analysis structures, similar to XPath statements used to access XML elements in an XML document.

A feature path coded in the `<feature>` element in the index build specification defines the path to a simple-valued feature in the common analysis structure, that is, a feature of the type `uima.cas.Float`, `uima.cas.Integer` or `uima.cas.String`.

In its simplest form, a feature path is the name of a feature of a type mentioned in the mapping element. If a type defines complex features, that is, features that have feature structures as their value, the feature path can be used to access a simple feature value in the common analysis structure.

For example, consider an annotator that identifies cars and their make. It creates annotations of type `car` that have an attribute `make`. However, `make` does not contain the actual company (for example, Chevrolet), but contains a feature structure of type `Company`, which itself has a string-valued attribute `companyname`. To enable a semantic query that combines car names and company names, a feature path `make/companyname` is used to attach the value of `companyname` to the car span that is generated for the car annotation. This enables the query, ″Give me documents that contain cars made by Chevrolet″, using ′/car[@make=″Chevrolet″]′.

A feature path is a sequence of feature names (f1/.../fn) with the following properties:

- The last feature fn in the path must be simple-valued, that is, of type `uima.cas.Float`, `uima.cas.Integer`, or `uima.cas.String`
- All features within the path from f1 to fn-1 are not allowed to be simple-valued.
- Optionally, a feature can be typed. The fully qualified type name must be prepended to the feature name, and be separated by a colon. For example, `f1/com.ibm.es.SomeType:f2/.../fn`

  This can be used to narrow the type scope of a particular feature. For example, consider a feature `additionalInfo` of type `uima.cas.TOP`. If you know that the value of your feature `additionalInfo` is actually `EmployeeInfo`, you could write `EmployeeInfo/additionalInfo:fn`

Features that are array-valued have the following additional properties:

- The next element in the feature path must be typed. The type name is the type of the elements within the array. For example, consider a feature structure of type `Info`. This type has a feature named `companies`, whose range is an FSArray. The elements of the array are of type `Company`. `Company`, in turn, has a feature named `profit`. To obtain the profit of the third company, write (using fully qualified type names) `companies[3]/Company:profit`
- One element of the array must be selected, using brackets ([ ]). An array starts at zero (0). For example, to select the first element in the companies array, use `Company:companies[0]` The special marker [last] can be used to select the last entry in an array, irrespective of its size, for example, `Company:companies[last]`

- If the last feature fn of the path is array-valued, only the array types `uima.cas.FloatArray`, `uima.cas.IntegerArray`, and `uima.cas.StringArray` are allowed.
- Features in the path from f1 to fn-1 must not be of type `uima.cas.FloatArray`, `uima.cas.IntegerArray`, or `uima.cas.StringArray`
- If the last feature fn of the path is array-valued, use empty brackets ([]) to denote “all elements”. These elements will be concatenated, and written to the index as a single, multi-term attribute. For example, if there is an array of `nameCandidates`, each one specifying a likely name, all of the candidate names are mapped to a certain attribute (or field).

Writing the index build configuration file

The index build configuration file must be an XML file and must comply with a specific schema.

A sample index build configuration file is as follows:

```
<?xmlversion="1.0" encoding="UTF-8"?>
<indexBuildSpecification
xmlns:namespace="http://www.ibm.com/of/822/consumer/index/xml">
 <skipCondition>
   <type>com.ibm.uima.tt.DocumentAnnotation</type>
   <filter syntax="FeatureValue">toBeprocessed = 0</filter>
 </skipCondition>

 <indexBuildItem>
   <typeName>com.ibm.uima.tt.PersonAnnotation</typeName>
   <indexRule>
     <style name="Annotation">
       <attribute name="fixedName" value="Person"/>
       <attributeMappings>
         <mapping>
           <feature>title</feature>
           <indexName>title</indexName>
         </mapping>
         <mapping>
           <feature>firstName</feature>
           <indexName>name</indexName>
         </mapping>
       </attributeMappings>
     </style>
     <style name="Field">
       <attribute name="fixedName" value="People"/>
       <attribute name="parametric" value="false"/>
       <attribute name="fieldSearchable" value="true"/>
       <attribute name="returnable" value="true"/>
     </style>
   </indexRule>
   <filter syntax="FeatureValue">confidence = 0.7</filter>
 </indexBuildItem>
  <indexBuildItem>
   <name>com.ibm.uima.tt.GeneralEntity</name>
   <indexRule>
     <style name="Annotation">
       <attribute name="nameFeature" value="categoryName" />
     </style>
     <style name="Field">
       <attribute name="nameFeature" value="categoryName" />
       <attribute name="parametric" value="false" />
       <attribute name="fieldSearchable" value="true" />
       <attribute name="returnable" value="true" />
     </style>
   </indexRule>
```

```
 </indexBuildItem>
 <indexBuildItem>
   <name>com.ibm.uima.tt.DrugDosage</name>
   <indexRule>
     <style name="Field">
       <attribute name="fixedName" value="dosage" />
       <attribute name="parametric" value="true" />
       <attribute name="fieldSearchable" value="true" />
       <attribute name="returnable" value="false" />
      </style>
     <style name="Field">
       <attribute name="fixedName" value="make" />
       <attribute name="valueFeature" value="make/companyname" />
       <attribute name="parametric" value="false" />
       <attribute name="fieldSearchable" value="true" />
       <attribute name="returnable" value="false" />
     </style>
   </indexRule>
   <filter syntax="FeatureValue">confidence >= 0.7</filter>
 </indexBuildItem>
</indexBuildSpecification>
```

The <skipCondition> element

The <skipCondition> element is optional and is used to prohibit certain documents from being indexed, based on a certain feature value. In the example, documents will not be indexed that contain a data structure of type com.ibm.uima.tt.DocumentAnnotation with a feature named toBeProcessed set to zero.

The <indexBuildItem> element

The index build specification configuration file contains one or more <indexBuildItem> elements. Each element describes the mapping of one particular feature structure in the common analysis structure to a structure in the index (a span or field).

The <name> element contains the feature structure type. There are two ways to specify a type:

- Using the full type name, for example, com.ibm.uima.tt.DrugDosage
- Using a wildcard, for example, com.ibm.uima.tt.* The wildcard character can only be added at the end of the type specification.

If type A is a subtype of type B (“Person” as a subtype to “Entity”), and there are <indexBuildItem> elements Ia and Ib defined for both types, processing is as follows:

- Feature structures of type B will be processed according to Ib
- Feature structures of type A will be processed using a combination of Ia and Ib
- If Ia and Ib both define the same <attributemappings> element, an error will occur. Ia must be defined as a “true” extension of Ib, dealing only with the additional attributes introduced in A.

Only use subtypes of uima.tcas.Annotation as index build items. If a subtype of uima.tcas.Annotation is a feature structure of subtype uima.cas.TOP (and not of uima.tcas.Annotation), you can still access this feature structure using a feature path.

The <filter> element is optional and is used to restrict the <indexBuildItem> mapping only to feature structures that have a certain attribute value. This is useful if you want an attribute to act as a switch for what to index. For example, persons and organizations might be recorded in an annotation of type EntityAnnotation. Its feature called type is set to either person or organization. To extract only the persons, and not the organizations, you can add the following filter:

```
<filter syntax="FeatureValue">type = "person"</filter>
```

Moreover, you could choose to index persons and organizations under different span names, for example, person and organization. To do this, define two <indexBuildItem> elements of type EntityAnnotation and use two filters on the type feature to trigger either the persons or the organizations.

Each filter expression takes the form:

```
<FeaturePath> <Operator> <Literal>
```

where:

- FeaturePath is a the name of a feature structure in the common analysis structure
- Operator is =, !=, <, <=, > or >=. Note that < (and only <) has to be expressed as <
- Literal is an integer, floating point number (no exponent syntax is supported) or string literal enclosed in double quotes, with embedded quotes and backslashes escaped by a backslash.

Note that <FeaturePath>, <Operator> and <Literal> must be separated by a blank space.

These are examples of valid filters:

```
The feature foo contains the string hello world
<filter syntax="FeatureValue"> foo = "hello world" </filter>

The feature foo contains the integer value 42
<filter syntax="FeatureValue"> foo &lt; 42 </filter>

The feature path make/company where the feature make contains
a feature structure which has a feature company with the
value Chevrolet
<filter syntax="FeatureValue"> make/company = "Chevrolet" </filter>

The feature bar7 contains the float value 0.5
<filter syntax="FeatureValue"> bar7 >= 0.5 </filter>
```

The <indexRule> element

Each <indexBuildItem> element contains one <indexRule> element. Each <indexRule> element contains all the information needed to map a feature structure in the common analysis structure to the index as a field, an annotation, or a breaking style. The annotation and field styles support a number of attributes. You cannot use the term style, which is supported in UIMA in enterprise search (Term style is skipped).

For the annotation and field styles, the following alternatives exist when you specify the annotation or field name in the index:

- Use fixedName if each feature structure is accessible in the index under the same name. In the following example, each feature structure of type PersonAnnotation will be mapped to a span named ″Person″ in the index.

```
<indexBuildItem>
   <name>com.ibm.tt.PersonAnotation</name>
   <indexRule>
     <style name="Annotation">
       <attribute name="fixedName" value="Person" />
     </style>
   </indexRule>
<indexBuildItem>
```

  This enables queries like ″Give me documents where Boss is contained as a person name″. The query is expressed as follows using XML fragments: `@xmlf2::'<person>Boss</person>'<attribute name="fixedName" value="Person" />`

- Use nameFeature if the annotation stores different entities that you want to be able to access using different spans depending on the value of a certain feature of the annotation. In the following example, EntityAnnotation is indexed as a person or organization span, depending on the value of the feature named type. The feature can also be a feature path.

```
<indexBuildItem>
   <name>com.ibm.tt.EntityAnotation</name>
   <indexRule>
     <style name="Annotation">
       <attribute name="nameFeature" value="type" />
     </style>
   </indexRule>
<indexBuildItem>
```

  This enables queries like ″Give me documents about the WHO″ (as opposed to the English term ″who″). The query is expressed as follows in limited XPath syntax: `@xmlp::'/organization[ftcontains="WHO"]'`

- If none of the above attributes is used, the short name of the annotation type in the <indexBuildItem> element is used. This is the default. For example:

```
<indexBuildItem>
   <name>com.ibm.uima.tutorial.RoomNumber</name>
   <indexRule>
     <style name="Annotation" />
     <style name="Field" />
   </indexRule>
 </indexBuildItem>
```

  This <indexBuildItem> element results in annotations and fields named RoomNumber populated with the text covered by com.ibm.uima.tutorial.RoomNumber.

The <style name="Annotation" /> element

Annotation in the <style> element specifies how you can access span information in enterprise search. Besides allowing the use of the fixedName and nameFeature attributes, this style also supports the <attributemappings> element. Within this element, it is possible to map the value of a feature structure to an attribute of the resulting span in the index, which you can subsequently use in a search expression.

Each mapping is done within a separate <mapping> element. The <feature> element contains a feature path, and the <indexName> element contains the name of the attribute that is used in the index to store the value of <feature>. For example,

```
<mapping>
  <feature>make/companyname</feature>
  <indexName>company</indexName>
</mapping>
```

This <mapping> element stores the value of the feature in the path make/companyname directly in the index attribute company.

Mapping feature values to index attributes is especially useful if the type system used during text analysis is complex, including many nested feature structures. Using the <mapping> element, relevant attributes can be exposed, allowing you to use them in queries without detailed knowledge of the original type system structure.

The <style name="Field" /> element

Field in the <style> element specifies how you can access field information in enterprise search. Besides the fixedName and nameFeature attributes, you can set the following attributes. The default is false if the following attributes are not set.

parametric
: If set to true, the field value can be searched using parametric search, for example, #dosage:>100

fieldSearchable
: If set to true, the field value can be used in search, for example, make:Bayer

returnable
: If set to true, the field and its values are returned in the search result

Field information is always content searchable, that is, field information is accessible in normal keyword searches.

The optional attribute valueFeature defines which feature value to take as the field value. If the feature structure is an annotation, and the attribute is not set, the covered text of the annotation is used as the field value. In the example,

```
<indexBuildItem>
  <name>com.ibm.uima.tt.DrugDosage</name>
  <indexRule>
    <style name="Field">
      <attribute name="fixedName" value="dosage" />
      <attribute name="parametric" value="true" />
      <attribute name="fieldSearchable" value="true" />
      <attribute name="returnable" value="false" />
    </style>
    <style name="Field">
      <attribute name="fixedName" value="make" />
      <attribute name="valueFeature" value="make/companyname" />
      <attribute name="parametric" value="false" />
      <attribute name="fieldSearchable" value="true" />
      <attribute name="returnable" value="false" />
    </style>
  </indexRule>
</indexBuildItem>
```

two fields are generated for DrugDosage. One field named dosage contains the covered text, for example, 100. In this case, you can query using '#dosage::>100'. Another field contains the value of the attribute companyname in the feature path make/companyname. Here you can query using 'make:Bayer'.

The <style name="Breaking" /> element

The value Breaking in the <style> element does not include any further elements.

Types and features defined in enterprise search

The type system defined in enterprise search covers document meta data handling and basic linguistic analysis.

Basic linguistic analysis in the form of document language recognition and segmentation always takes place when a document is indexed, irrespective of whether custom analysis is selected or not. During basic document analysis, the following information is added to the common analysis structure that you can use in your custom analysis:

- Document meta data (of type com.ibm.es.tt.DocumentMetaData)
- Token, sentence, and paragraph annotations (of type uima.tt.TokenAnnotation, uima.tt.SentenceAnnotation and uima.tt.ParagraphAnnotation). The token annotation includes the feature lemma.

The type system defined in enterprise search does not include any sophisticated types and features specific to text analysis. These are included in the UIMA type system that you can use (and extend) when creating your custom analysis in your UIMA environment.

Unlike the UIMA type system that you are likely to extend to include the new types that are required by your custom analysis, you will probably not need to extend the enterprise search type system.

The enterprise search type system is not defined in the UIMA Software Development Kit (SDK). If you want to use any of those types when you write an annotator in UIMA, for example, if you want to access document security information, or access the crawler type or document type, you must define the types again in the type system description of your analysis engine.

The following types and features are defined in enterprise search:

uima.tcas.Annotation
An annotation comprises the following types:

uima.tcas.DocumentAnnotation
The document annotation has the following feature:

esDocumentMetaData
Contains document meta data of the type com.ibm.es.tt.DocumentMetaData

com.ibm.es.tt.ContentField
The content field annotation has the following feature:

parameters
The content field parameters are of type com.ibm.es.tt.CommonFieldParameters

com.ibm.es.tt.Anchor
The anchor annotation for anchor text in HTML documents. It has the following feature:

uri The target URI of the anchor text. The feature value is of type uima.cas.String.

com.ibm.es.tt.MarkupTag

The markup information annotations, for example, of an XML tag. The markup information is stored in the following features:

name The name for the markup tag. The feature value is of type `uima.cas.String`.

depth The nesting depth. The feature value is of type `uima.cas.Integer`.

attributeName

The name for the feature attribute. The feature value is of type `uima.cas.StringArray`.

attributeValues

A string of values for the attribute. The feature value is of type `uima.cas.StringArray`.

uima.CAS.TOP

The root of the type system. It has the following types:

com.ibm.es.tt.DocumentMetaData

Document meta data has the following features. The features are connected to the document annotation feature `esDocumentMetaData`.

crawlerId

The crawler name. The feature value is of type `uima.cas.String`.

dataSource

One of the following data source types:

- Web (for documents that originate from the Web Crawler)
- NNTP (for documents that originate from the News Group Crawler)
- DB2 (for documents that originate from the DB2 Crawler)
- Notes® (for documents that originate from the Notes Crawler)
- CM (for documents that originate from the Content Management Crawler)
- FS (for documents that originate from the UNIX® File System Crawler)
- WinFS (for documents that originate from the Windows File System Crawler)
- Exchange (for documents that originate from the Exchange Crawler)
- VBR (for documents that originate from the VeniceBridge Crawler)

The feature value is of type `uima.cas.String`.

dataSourceName

The name of the crawler (data source). The feature value is of type `uima.cas.String`.

charset

Document code page. The feature value is of type `uima.cas.String`.

docType
One of the following document types:

- text/html
- application/postscript
- application/pdf
- application/x-mspowerpoint
- application/msword
- application/x-msexcel
- application/rtf
- application/vnd.lotus-wordpro
- application/x-lotus-123
- application/vnd.lotus-freelance
- text/xml
- text/plain
- application/x-js-taro (Ichitaro)

The feature value is of type `uima.cas.String`.

securityTokens
The document security tokens. The feature value is of type `uima.cas.StringArray`.

date The document date. The feature value is of type `uima.cas.String`.

baseUri
The base URI of the page. The feature value is of type `uima.cas.String`.

metaDataFields
The feature value is of type `uima.cas.FSArray`. Each element in this array is of type `com.ibm.es.tt.MetaDataField`.

redirectUrl
The redirected URL. The feature value is of type `uima.cas.String`.

contentLanguage
The language for content defined by the user using meta data settings. The feature value is of type `uima.cas.String`.

hasSeparateContent
A flag indicating if the document has content and meta data.

mimeType
Mime type, or document type, for example, XML. The feature value is of type `uima.cas.String`.

metaLanguage
The language of the meta data. The feature value is of type `uima.cas.String`.

url The document URL. The feature value is of type `uima.cas.String`.

com.ibm.es.tt.CommonFieldParameters
Common field parameters include:

searchable
A flag indicating if the field is free-text searchable.

fieldSearchable
A flag indicating if the field is searchable as a field.

parametric
A flag indicating parametric search.

showInSearchResult
A flag indicating if annotated data is included in the search result details.

resolveConflict
A flag for resolving meta data conflicts between `MetadataPreferred`, `ContentPreferred`, and `Coexist`. The feature value is of type `uima.cas.String`.

name The name of the field. You can search for this field using the field name. The feature value is of type `uima.cas.String`.

com.ibm.es.tt.MetaDataField
Meta data field data is not part of the document content but is stored in the ″text″ feature:

parameters
Meta data field parameters of type `com.ibm.es.tt.CommonFieldParameters`.

text The meta data text is stored in this feature of type `uima.cas.String`.

Types and features defined in UIMA

UIMA Software Development Kit defines a few basic linguistic types and features that might be discovered in a document during text analysis.

Each analysis engine has its own type system descriptor that describes the input requirements and output types for the annotators contained in the analysis engine. Type system descriptions are domain and application specific.

You can extend the UIMA type system to include your own types and features. In the UIMA environment, there is an Eclipse plug-in that helps you edit type system descriptors for annotators. Refer to the UIMA documentation for details on installing and using the Component Descriptor Editor plug-in.

When you have completed developing and testing your analysis engine in the UIMA environment, the archive file (.pear) that you create containing your analysis engine files will also include your type system description.

The following sections list the types and features defined in UIMA:

uima.tcas.Annotation
An Annotation comprises the following types:

uima.tcas.DocumentAnnotation

uima.tt.TTAnnotation

uima.tcas.DocumentAnnotation
A document annotation includes the following features:

categories
A list of category names or labels for the document. The feature value is of type `uima.cas.FSList`.

languageCandidates
A list of document language references. The feature value is of type `uima.cas.FSList`.

id A form of document identification, for example, a URL. The feature value is of type `uima.cas.String`.

uima.tt.TTAnnotation
A TT annotation includes the following types:

uima.tt.DocStructureAnnotation
Structural information about the document. The document structure annotation includes the following types:

uima.tt.SentenceAnnotation
A sentence including opening and closing punctuation. Its features include:

sentenceNumber
The sequence number of the sentence in the paragraph. Reset to 1 at the beginning of each paragraph. The feature value is of type `uima.cas.Integer`.

uima.tt.ParagraphAnnotation
A Paragraph. Its features include:

paragraphNumber
The sequence number of the paragraph. The feature value is of type `uima.cas.Integer`.

uima.tt.LexicalAnnotation
Content information about the document. A lexical annotation comprises the following types:

uima.tt.CompPartAnnotation
A part of a compound word. Compound words in many Germanic languages are words written together without any separating blanks. For example, the German word ″Abteilungsleiter″ (department manager) consists of the parts ″Abteilung″ (department) and ″Leiter″ (manager).

uima.tt.TokenAnnotation
A token without surrounding white space. Its features include:

lemma
A list of references to TCAS entries of type `uima.tt.Lemma`. Each entry is a possible dictionary entry for the token.

lemmaEntries
A list of references to TCAS entries of type `uima.tt.Lemma`. Each entry is a possible dictionary entry for the token.

tokenNumber
The sequence number of the token in the sentence.

Reset to 1 at the beginning of each sentence. The feature value is of type `uima.cas.Integer`.

tokenProperties
A token property, for example, uppercase, numerics. The feature value is of type `uima.cas.Integer`.

stopwordToken
A token that is marked as being a stop word. The feature value is of type `uima.cas.Integer`.

synonymEntries
A list of references to entries of type `uima.tt.Synonym`. Each entry is a possible synonym entry for the token.

normalizedCoveredText
The normalized representation of the text covered by the annotation. The feature value is of type `uima.cas.String`.

uima.CAS.TOP
The root of the type system. It has the following types:

uima.tt.KeyStringEntry
A string with the following feature:

key The actual string.

uima.tt.Lemma
A dictionary entry with the following morphological information:

partOfSpeech
An integral encoding of the part of speech of the lemma.

morphID
An integral encoding of morphological information.

uima.tt.Synonym
A synonym entry for a given word of type `uima.tt.keyStringEntry`.

uima.tt.LanguageConfidencePair
A type with the following features that describes the document language selection.

uima.tt.LanguageConfidencePair

languageConfidence
An indication (a float value between 0 and 1) of how well the chosen language actually fits the language of the document.

language
The language of the document (ISO value). The value is of type `uima.cas.String`.

languageID
The language ID. The value is of type `uima.cas.Integer`.

uima.tt.CategoryConfidencePair
A type with the following features that describes the category selection for the document.

uima.tt.CategoryConfidencePair
A category has the following features:

categoryString
The name of the category. The value is of type `uima.cas.String`.

categoryConfidence
An indication of how well the category fits the document. The value is of type float.

mostSpecific
A flag (of type `uima.cas.Integer`) indicating whether the category is the most specific for the document.

taxonomy
The name of the taxonomy to which the category belongs. Documents can have categories from differing taxonomies. The value is of type `uima.cas.String`.

Semantic search applications

The are four types of document information stored in the enterprise search index that you can query in search applications using the Search and Index API (SIAPI) interface.

The four different types of information include:

- Text words found in a document, for example, computer software.
- Span names, for example, an XML document that includes <author>James</author>, yields the span <author>.
- Attribute names, for example, an XML document that includes <author countryOfBirth=USA>James</author>, yields the attribute "countryOfBirth".
- Attribute values, for example, USA is the value of the attribute "countryOfBirth".

The SIAPI query language has been extended to include the semantic search query term. The term specifies a twig pattern. A twig is a little tree the leaves of which are words of the four types mentioned above, and the internal nodes of the tree specify how their occurrence in a document relate to one another. There are five types of internal nodes specifying relationships:

- and
- or
- not
- in_the_span_of
- attribute_in_the_span_of

A document is said to satisfy a given semantic search term if it includes occurrences of the leaves and the constraints specified by the internal nodes (the defined relationships) are respected.

The semantic search query term helps retrieve better quality documents. You can now not only search using boolean combinations of word and annotations, but also retrieve documents where, for example, James appears in the span named author, or where the terms ″ibm″ and ″search″ appear in the same sentence.

Related concepts

“Semantic search query”
The semantic search query term is communicated as an opaque term.

Semantic search query

The semantic search query term is communicated as an opaque term.

There are two forms of syntax to express an opaque term in the Search and Index API (SIAPI):

- XML fragments
- Limited XPath

The XML fragment query term looks like a well-balanced fragment of an XML document. An XML fragment query term is prefixed by the opaque term sign `@xmlf2::` followed by the XML fragment expression enclosed between single quotes (’...’).

Limited XPath query terms, however, are prefixed by `@xmlxp::` followed by the XPath query enclosed between single quotation marks (’...’).

As with general query terms in the Search and Index API (SIAPI) interface, each term can have an appearance modifier:

Plus sign (+)
Term must appear.

Prefix =
Term must be an exact match.

Prefix tilde (~)
Consider synonyms of the query term.

Postfix tilde (~)
Consider words that have the same lemma as the query term.

The following examples show XML fragment queries.

@xmlf2::’<title>″Data Structures″</title>’
Finds documents that include the span (annotation) title containing the phrase ″Data Structures″

@xmlf2::’<author country=″USA″></author>’
Finds documents where the author originates from the USA.

@xmlf2::’<book><.or><author>John Smith</author><title>XML -Microsoft</title></.or></book>’
Finds documents that specify a book whose author is John Smith or where the title of the book includes the word XML but not Microsoft®.

The corresponding XPath queries have the following structure:

@xmlxp::’/booktitle[ftcontains(″Data Structures″)]’
Finds documents that include the span (annotation) booktitle containing the phrase ″Data Structures″

@xmlxp::'//author[@country="USA"]'
Finds documents where the author originates from the USA.

@xmlxp::'/book[author[ftcontains("Jane Smith")] or title[ftcontains("XML -Microsoft")]]'
Finds documents that specify a book whose author is Jane Smith or where the title of the book includes the word XML but not Microsoft.

Related concepts

"Semantic search applications" on page 72
The are four types of document information stored in the enterprise search index that you can query in search applications using the Search and Index API (SIAPI) interface.

Chapter 7. Text analysis included in enterprise search

The text analysis included in enterprise search comprises document language detection and segmentation.

At document processing time, enterprise search determines the language of a document and breaks up the stream of input text into distinct units or tokens.

At search time, the user, or an application default, must select the query language manually. The query string is segmented, analyzed, and searched in the index.

Both document and query string analysis can be split up into:

- Basic nondictionary-based support. This includes white space and n-gram segmentation.
- Dictionary-based linguistic support. This includes word and sentence segmentation, and lemmatization.

  Linguistic processing involves lexical analysis which is the process of creating alternative representations of the input text that associates all available dictionary data to the tokens recognized in the input text. Search quality is greatly enhanced using advanced language processing.

Related concepts

"Language identification"
Before word and sentence segmentation, character normalization, or lemmatization can occur, enterprise search must determine the language of the source document.

"Linguistic support for nondictionary-based segmentation" on page 76
For documents in languages that are not supported by language detection and lexical analysis technology, enterprise search provides basic support in the form of Unicode-based white space and n-gram segmentation.

Language identification

Before word and sentence segmentation, character normalization, or lemmatization can occur, enterprise search must determine the language of the source document.

Enterprise search can automatically detect the following languages:

Arabic	French	Korean
Chinese (Traditional and Simplified)	German	Polish
Czech	Greek	Portuguese
Danish	Hebrew	Russian
Dutch	Hungarian	Spanish
English	Italian	Swedish
Finnish	Japanese	Turkish

The linguistic processes for enterprise search detect the language of a source document during indexing, not during query processing.

Documents for which the language cannot be detected automatically are processed with basic language-independent technology.

The enterprise search language detection technology is best suited for monolingual documents. If a document is multilingual, an attempt is made to determine the most dominant language used in the document. However, the analysis results are not always satisfactory.

The language detection technology in enterprise search can be used to restrict your search results to only documents that are in a particular language. For example, if you search for documents about Jacques Chirac, you can specify that only documents that are written in French are to be included in the search results.

Related concepts

Chapter 7, "Text analysis included in enterprise search," on page 75
The text analysis included in enterprise search comprises document language detection and segmentation.

"Linguistic support for nondictionary-based segmentation"
For documents in languages that are not supported by language detection and lexical analysis technology, enterprise search provides basic support in the form of Unicode-based white space and n-gram segmentation.

Linguistic support for nondictionary-based segmentation

For documents in languages that are not supported by language detection and lexical analysis technology, enterprise search provides basic support in the form of Unicode-based white space and n-gram segmentation.

Unicode-based white space segmentation
This method of linguistic processing uses the white space (or blank space) between words as a word delimiter.

N-gram segmentation
This method of linguistic processing treats overlapping sequences of *n* characters as a single word. This simple method of segmentation is sufficient for many retrieval tasks.

These methods are independent of any language dictionary and do not include sophisticated linguistic processing technology, such as base-form reduction.

N-gram segmentation is used for languages such as Thai that have no blank spaces to use as delimiters. The same method applies to Hebrew and Arabic. Although these two languages use white space delimiters, n-gram segmentation returns better results than the basic form of Unicode-based white space segmentation does.

Related concepts

Chapter 7, "Text analysis included in enterprise search," on page 75
The text analysis included in enterprise search comprises document language detection and segmentation.

"Language identification" on page 75
Before word and sentence segmentation, character normalization, or lemmatization can occur, enterprise search must determine the language of the source document.

Linguistic support for dictionary-based segmentation

If the language of a document is correctly detected, and language-specific dictionaries are available, then appropriate linguistic processing is applied.

Segmentation is the process by which input text is broken down into distinct lexical units. This process comprises some of the following linguistic processing activities:

Word segmentation
Word segmentation is used for languages that do not use white spaces (or delimiters) between words, such as Japanese and Chinese.

Lemmatization
Lemmatization is a form of linguistic processing that determines the lemma for each word form that occurs in text. The *lemma* of a word encompasses its base form plus inflected forms that share the same part of speech. For example, the lemma for go encompasses go, goes, went, gone, and going. Lemmas for nouns group singular and plural forms (such as calf and calves). Lemmas for adjectives group comparative and superlative forms (such as good, better, and best). Lemmas for pronouns group different cases of the same pronoun (such as I, me, my, and mine).

Lemmatization requires a dictionary for both indexing and searching.

Enterprise search indexes the lemmas and the inflected words, and lemmatizes all inflected words in a query. Lemmatization enhances search quality by finding documents that contain variants of an inflected word in the query. For example, documents that contain the word *mice* are found when a query includes the word *mouse*.

Contraction splitting
Search quality is improved by identifying contractions and splitting them up into their component parts. For example:

wouldn't is split into *would* + *not*
Horse's is split into *Horse* + *is* or *'s* (to account for query ambiguity)

Clitic identification
Clitics are a special form of contractions, and search quality is improved by determining their component parts. A *clitic* is an element that behaves like an affix and a word. However, clitics are difficult to identify because they are also part of word formation. Unlike other morphological (word structure) phenomena, clitics occur in a syntactic structure and their attachment to words is not part of the word formation rules. For example:

reparti-lo-emos has the components *repartir* + *lo* + *emos*
l'avenue has the components *le* + *avenue*
dell'arte has the components *dello* + *arte*.

Nonalphabetic character recognition
The linguistic processes recognize nonalphabetic characters. Depending on the internal language-dependent logic, some nonalphabetic characters are returned as separate lexical units of different types, and some are grouped.

For example, apostrophes or hyphens in the case of clitics are considered word parts, and they are considered full stops (or periods) in the case of unknown abbreviations. The linguistic processing can also recognize some special sequences of characters as tokens, for example, URLs, e-mail addresses, and dates.

Abbreviation recognition
The linguistic processes recognize abbreviations that are in the dictionary as one lexical unit. If the abbreviation is not in the dictionary, then the

abbreviation is recognized as a lexical item, but the abbreviation will not have any associated dictionary information.

Recognizing abbreviations correctly is vital for sentence recognition. For example, the period at the end of an abbreviation is not necessarily the end of a sentence.

End-of-sentence marker recognition
The linguistic processes correctly identify end-of-sentence markers for sentence segmentation.

Dictionary-based linguistic support is available for the following languages:

```
Chinese (Simplified and Traditional)   Italian
Czech                                  Japanese
Danish                                 Korean
Dutch                                  Norwegian (Bokmal and Nynorsk)
English                                Polish
Finnish                                Portuguese (National and Brazilian)
French (National and Canadian)         Russian
German (National and Swiss)            Spanish
Greek                                  Swedish
```

Related concepts

"Word segmentation in Japanese"
If the text document or the query string is recognized as being in Japanese, enterprise search performs relevant word segmentation by using morphological analysis technology that is optimized for the Japanese language.

"Orthographic variants in Japanese"
Japanese uses many orthographic variants. Katakana variants are the most important because Katakana is often used to spell and pronounce foreign words. Many Katakana variants are commonly used in Japanese.

Word segmentation in Japanese

If the text document or the query string is recognized as being in Japanese, enterprise search performs relevant word segmentation by using morphological analysis technology that is optimized for the Japanese language.

An example of this optimization is word decomposition. Japanese uses a large number of compound words. These words are decomposed into tokens of optimal size to achieve better search results. Inflected words and prepositions are also decomposed to improve search performance.

Related concepts

"Linguistic support for dictionary-based segmentation" on page 76
If the language of a document is correctly detected, and language-specific dictionaries are available, then appropriate linguistic processing is applied.

"Orthographic variants in Japanese"
Japanese uses many orthographic variants. Katakana variants are the most important because Katakana is often used to spell and pronounce foreign words. Many Katakana variants are commonly used in Japanese.

Orthographic variants in Japanese

Japanese uses many orthographic variants. Katakana variants are the most important because Katakana is often used to spell and pronounce foreign words. Many Katakana variants are commonly used in Japanese.

Enterprise search uses a variant dictionary to map typical Katakana variants to their base forms (similar to a lemma) so that all documents, including those with orthographic variants of the Katakana word in the query string, are found.

Enterprise search also supports typical Okurigana variants, which are Kanji word endings that are written in Hiragana.

Related concepts

"Linguistic support for dictionary-based segmentation" on page 76
If the language of a document is correctly detected, and language-specific dictionaries are available, then appropriate linguistic processing is applied.

"Word segmentation in Japanese" on page 78
If the text document or the query string is recognized as being in Japanese, enterprise search performs relevant word segmentation by using morphological analysis technology that is optimized for the Japanese language.

Stop word removal

In enterprise search, all stop words, for example, common words like *a* and *the*, are removed from multiple word queries to increase search performance.

Stop word recognition in Japanese is based on grammatical information. For example, enterprise search recognizes whether the word is a noun or a verb, whereas the other languages work with special lists.

Related concepts

"Character normalization"
Character normalization is a process that can improve recall. Improving recall by character normalization means that more documents are retrieved even if the documents do not exactly match the query.

Character normalization

Character normalization is a process that can improve recall. Improving recall by character normalization means that more documents are retrieved even if the documents do not exactly match the query.

Enterprise search uses Unicode compatibility normalization that includes the normalization of Asian full-width and half-width characters.

For example, in Japanese, a full-width alphanumeric character is normalized to the half-width character, a half-width Katakana character to the full-width character, and so on. Enterprise search also removes Katakana middle dots, which are used as compound word delimiters in Japanese.

Other forms of character normalization include:

Case normalization
For example, finding documents with *USA* when searching for *usa*.

Umlaut expansion
For example, finding documents containing *schoen* when searching for *schön*.

Accent removal
For example, finding documents containing *é* when searching for *e*.

Other diacritics removal
For example, finding documents containing *ç* when searching for *c*.

Ligature expansion
For example, finding documents containing *Æ* when searching for *ae*.

All normalizations work both ways. You can find documents that contain *usa* when you search for *USA*, documents that contain words with *e* when you search for *é*, and so on. These normalizations can also be combined. For example, you can find documents that contain *météo* when you search for *METEO*.

The normalizations are based on Unicode character properties and are not language-dependent. For example, enterprise search supports diacritic removal for Hebrew and ligature expansion for Arabic.

Related concepts

"Stop word removal" on page 79
In enterprise search, all stop words, for example, common words like *a* and *the*, are removed from multiple word queries to increase search performance.

DB2 Information Integrator documentation

This topic provides information about the documentation that is available for DB2 Information Integrator.

The tables in the following topics provide the official document title, form number, and location of each PDF book. To order a printed book, you must know either the official book title or the document form number. Titles, file names, and the locations of the DB2 Information Integrator release notes and installation requirements are also provided in the following topics.

Documentation about event publishing function for DB2 Universal Database on z/OS

Documentation about event publishing function for DB2 Universal Database on z/OS

Purpose

Documentation about event publishing function for DB2 Universal Database on z/OS.

Table 3. DB2 Information Integrator documentation about event publishing function for DB2 Universal Database on z/OS

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about event publishing function for IMS and VSAM on z/OS

Documentation about event publishing function for IMS and VSAM on z/OS

Purpose

Documentation about event publishing function for IMS and VSAM on z/OS.

Table 4. DB2 Information Integrator documentation about event publishing function for IMS and VSAM on z/OS

Name	Form number	Location
Client Guide for Classic Federation and Event Publisher for z/OS	SC18-9160	DB2 Information Integrator Support Web site
Data Mapper Guide for Classic Federation and Event Publisher for z/OS	SC18-9163	DB2 Information Integrator Support Web site
Getting Started with Event Publisher for z/OS	GC18-9186	DB2 Information Integrator Support Web site
Installation Guide for Classic Federation and Event Publisher for z/OS	GC18-9301	DB2 Information Integrator Support Web site
Operations Guide for Event Publisher for z/OS	SC18-9157	DB2 Information Integrator Support Web site
Planning Guide for Event Publisher for z/OS	SC18-9158	DB2 Information Integrator Support Web site
Reference for Classic Federation and Event Publisher for z/OS	SC18-9156	DB2 Information Integrator Support Web site
System Messages for Classic Federation and Event Publisher for z/OS	SC18-9162	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Event Publisher for IMS for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Event Publisher for VSAM for z/OS	N/A	DB2 Information Integrator Support Web site

Documentation about event publishing and replication function on Linux, UNIX, and Windows

Documentation about event publishing and replication function on Linux, UNIX, and Windows

Purpose

Documentation about event publishing and replication function on Linux, UNIX, and Windows.

Table 5. DB2 Information Integrator documentation about event publishing and replication function on Linux, UNIX, and Windows

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site

Table 5. DB2 Information Integrator documentation about event publishing and replication function on Linux, UNIX, and Windows (continued)

Name	Form number	Location
Installation Guide for Linux, UNIX, and Windows	GC18-7036	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Migrating to SQL Replication	N/A	DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
SQL Replication Guide and Reference	SC27-1121	DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site
Tuning for SQL Replication Performance	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews > Information Integration > DB2 Information Integrator overview > Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about federated function on Linux, UNIX, and Windows

Documentation about federated function on Linux, UNIX, and Windows

Purpose

Documentation about federated function on Linux, UNIX, and Windows.

Table 6. DB2 Information Integrator documentation about federated function on Linux, UNIX, and Windows

Name	Form number	Location
Application Developer's Guide	SC18-7359	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site

Table 6. DB2 Information Integrator documentation about federated function on Linux, UNIX, and Windows (continued)

Name	Form number	Location
C++ API Reference for Developing Wrappers	SC18-9172	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Data Source Configuration Guide	N/A	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Federated Systems Guide	SC18-7364	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Guide to Configuring the Content Connector for VeniceBridge	N/A	DB2 Information Integrator Support Web site
Installation Guide for Linux, UNIX, and Windows	GC18-7036	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Java API Reference for Developing Wrappers	SC18-9173	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Migration Guide	SC18-7360	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Wrapper Developer's Guide	SC18-9174	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about federated function on z/OS

Documentation about federated function on z/OS

Purpose

Documentation about federated function on z/OS.

Table 7. DB2 Information Integrator documentation about federated function on z/OS

Name	Form number	Location
Client Guide for Classic Federation and Event Publisher for z/OS	SC18-9160	DB2 Information Integrator Support Web site
Data Mapper Guide for Classic Federation and Event Publisher for z/OS	SC18-9163	DB2 Information Integrator Support Web site
Getting Started with Classic Federation for z/OS	GC18-9155	DB2 Information Integrator Support Web site
Installation Guide for Classic Federation and Event Publisher for z/OS	GC18-9301	DB2 Information Integrator Support Web site
Reference for Classic Federation and Event Publisher for z/OS	SC18-9156	DB2 Information Integrator Support Web site
System Messages for Classic Federation and Event Publisher for z/OS	SC18-9162	DB2 Information Integrator Support Web site
Transaction Services Guide for Classic Federation for z/OS	SC18-9161	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Classic Federation for z/OS	N/A	DB2 Information Integrator Support Web site

Documentation about replication function on z/OS

Documentation about replication function on z/OS

Purpose

Documentation about replication function on z/OS.

Table 8. DB2 Information Integrator documentation about replication function on z/OS

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	DB2 Information Integrator Support Web site
Migrating to SQL Replication	N/A	DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Replication Installation and Customization Guide for z/OS	SC18-9127	DB2 Information Integrator Support Web site
SQL Replication Guide and Reference	SC27-1121	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site

Table 8. DB2 Information Integrator documentation about replication function on z/OS (continued)

Name	Form number	Location
Tuning for SQL Replication Performance	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about enterprise search function on Linux, UNIX, and Windows

Documentation about enterprise search function on Linux, UNIX, and Windows

Purpose

Documentation about enterprise search function on Linux, UNIX, and Windows.

Table 9. DB2 Information Integrator documentation about enterprise search function on Linux, UNIX, and Windows

Name	Form number	Location
Administering Enterprise Search	SC18-9283	DB2 Information Integrator Support Web site
Installation Guide for Enterprise Search	GC18-9282	DB2 Information Integrator Support Web site
Programming Guide and API Reference for Enterprise Search	SC18-9284	DB2 Information Integrator Support Web site
Release Notes for Enterprise Search	N/A	DB2 Information Integrator Support Web site

Release notes and installation requirements

Release notes provide information that is specific to the release and fix pack level for your product and include the latest corrections to the documentation for each release. Installation requirements provide information that is specific to the release of your product.

Table 10. DB2 Information Integrator Release Notes and Installation Requirements

Name	File name	Location
Installation Requirements for IBM DB2 Information Integrator Event Publishing Edition, Replication Edition, Standard Edition, Advanced Edition, Advanced Edition Unlimited, Developer Edition, and Replication for z/OS	Prereqs	• The *DB2 Information Integrator* product CD • DB2 Information Integrator Installation Launchpad
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	ReleaseNotes	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD
Release Notes for IBM DB2 Information Integrator Event Publisher for IMS for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Event Publisher for VSAM for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Classic Federation for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for Enterprise Search	N/A	DB2 Information Integrator Support Web site

Viewing release notes and installation requirements

Viewing release notes and installation requirements

Purpose

To view release notes and installation requirements from the CD on Windows operating systems, enter:

x\doc\%*L*

Parameters

x The Windows CD drive letter

%*L*
The locale of the documentation that you want to use, for example, en_US.

Purpose

To view release notes and installation requirements from the CD on UNIX operating systems, enter:

/cdrom/doc/*%L*

Parameters

cdrom
The UNIX mount point of the CD

%L
The locale of the documentation that you want to use, for example, en_US.

Viewing and printing PDF documentation

Viewing and printing PDF documentation

To view and print the DB2 Information Integrator PDF books from the *DB2 PDF Documentation* CD

1. From the root directory of the *DB2 PDF Documentation* CD, open the index.htm file.
2. Click the language that you want to use.
3. Click the link for the document that you want to view.

Accessing DB2 Information Integrator documentation

Accessing DB2 Information Integrator documentation

All DB2 Information Integrator books and release notes are available in PDF files from the DB2 Information Integrator Support Web site at www.ibm.com/software/data/integration/db2ii/support.html.

To access the latest DB2 Information Integrator product documentation, from the DB2 Information Integrator Support Web site, click on the Product Information link, as shown in Figure 4 on page 89.

Figure 4. Product information link on the DB2 Information Integrator Support Web site

You can access the latest DB2 Information Integrator documentation, in all supported languages, from the Product Information link:

- DB2 Information Integrator product documentation in PDF files
- Fix pack product documentation, including release notes
- Instructions for downloading and installing the DB2 Information Center for Linux, UNIX, and Windows
- Links to the DB2 Information Center online

The DB2 Information Integrator Support Web site also provides support documentation, IBM Redbooks, white papers, product downloads, links to user groups, and news about DB2 Information Integrator.

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. The following list specifies the major accessibility features in DB2® Version 8 products:

- All DB2 functionality is available using the keyboard for navigation instead of the mouse. For more information, see "Keyboard input and navigation."
- You can customize the size and color of the fonts on DB2 user interfaces. For more information, see "Accessible display."
- DB2 products support accessibility applications that use the Java™ Accessibility API. For more information, see "Compatibility with assistive technologies" on page 92.
- DB2 documentation is provided in an accessible format. For more information, see "Accessible documentation" on page 92.

Keyboard input and navigation

Keyboard focus

Keyboard focus

In UNIX® operating systems, the area of the active window where your keystrokes will have an effect is highlighted.

Keyboard input

Keyboard input

You can operate the DB2 tools using only the keyboard. You can use keys or key combinations to perform operations that can also be done using a mouse. Standard operating system keystrokes are used for standard operating system operations.

For more information about using keys or key combinations to perform operations, see Keyboard shortcuts and accelerators: Common GUI help.

Keyboard navigation

Keyboard navigation

You can navigate the DB2 tools user interface using keys or key combinations.

For more information about using keys or key combinations to navigate the DB2 Tools, see Keyboard shortcuts and accelerators: Common GUI help.

Accessible display

Accessible display

Purpose

Accessible display

Font settings

Font settings

You can select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

For more information about specifying font settings, see Changing the fonts for menus and text: Common GUI help.

Non-dependence on color

Non-dependence on color

You do not need to distinguish between colors to use any of the functions in this product.

Compatibility with assistive technologies

Compatibility with assistive technologies

The DB2 tools interfaces support the Java Accessibility API, which enables you to use screen readers and other assistive technologies with DB2 products.

Accessible documentation

Accessible documentation

Documentation for DB2 is provided in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

Syntax diagrams are provided in dotted decimal format. This format is available only if you are accessing the online documentation using a screen-reader.

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

To locate an IBM office in your country or region, see the IBM Directory of Worldwide Contacts on the Web at www.ibm.com/planetwide.

Obtaining product information

Information about DB2 Information Integrator products is available by telephone or on the Web.

Information about DB2 Information Integrator products is available by telephone or on the Web. The phone numbers provided here are valid in the United States.

1. To order products or to obtain general information: 1-800-IBM-CALL (1-800-426-2255)
2. To order publications: 1-800-879-2755
3. Visit the Web at www.ibm.com/software/data/integration/db2ii/support.html.

This site contains the latest information about:

- The technical library
- Ordering books
- Client downloads
- Newsgroups
- Fix packs
- News
- Links to Web resources

Providing comments on the documentation

Please send any comments that you have about this book or other DB2 Information Integrator documentation.

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 Information Integrator documentation.You can use any of the following methods to provide comments:

1. Send your comments using the online readers' comment form at www.ibm.com/software/data/rcf.
2. Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to: IBM Director of Licensing IBM Corporation North Castle Drive Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:IBM World Trade Asia Corporation Licensing 2-31 Roppongi 3-chome, Minato-ku Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Corporation J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

Outside In (®) Viewer Technology, ©1992-2004 Stellent, Chicago, IL., Inc. All Rights Reserved.

Trademarks

This topic lists IBM trademarks and certain non-IBM trademarks.

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
AIX
AIX 5L
DB2
DB2 Universal Database
Domino
Domino.doc
Informix
Lotus
Lotus Notes
Notes
OmniFind
POWER4
POWER5
Tivoli
WebSphere
Workplace
xSeries
z/OS

The following terms are trademarks or registered trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Index

U

W

Printed in USA

SC18-9284-01