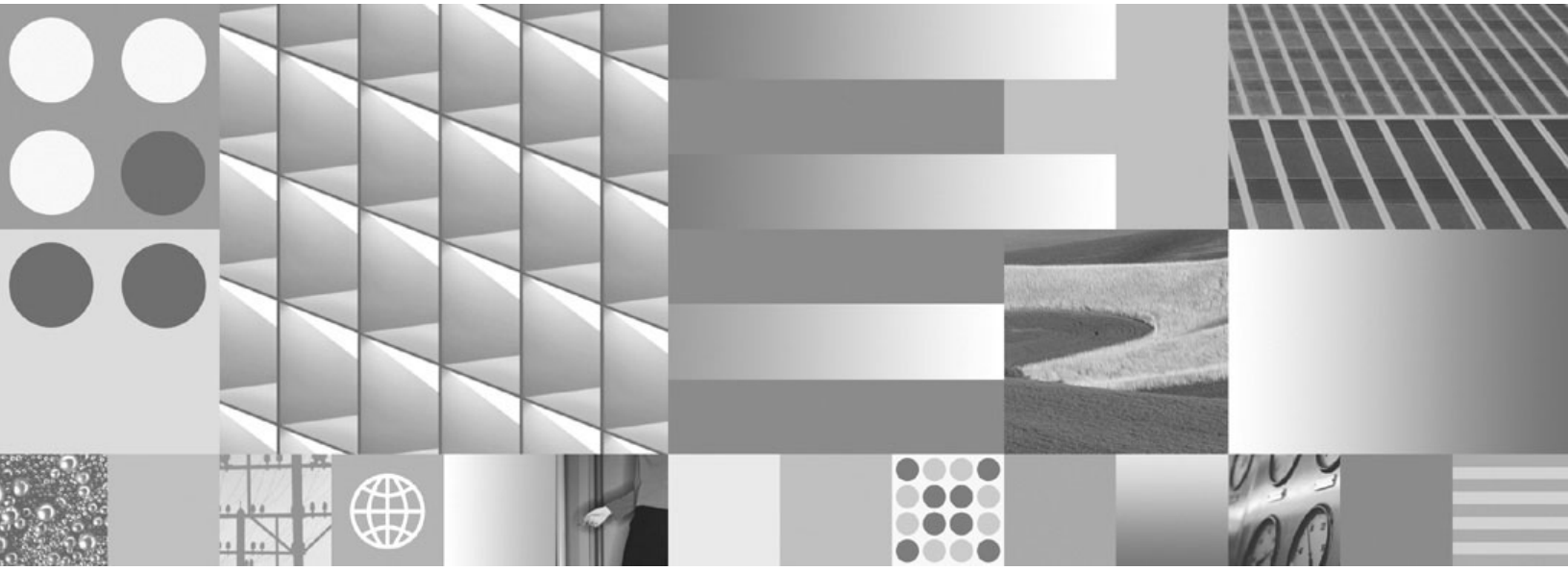
IBM OmniFind Enterprise Edition

Version 8.5

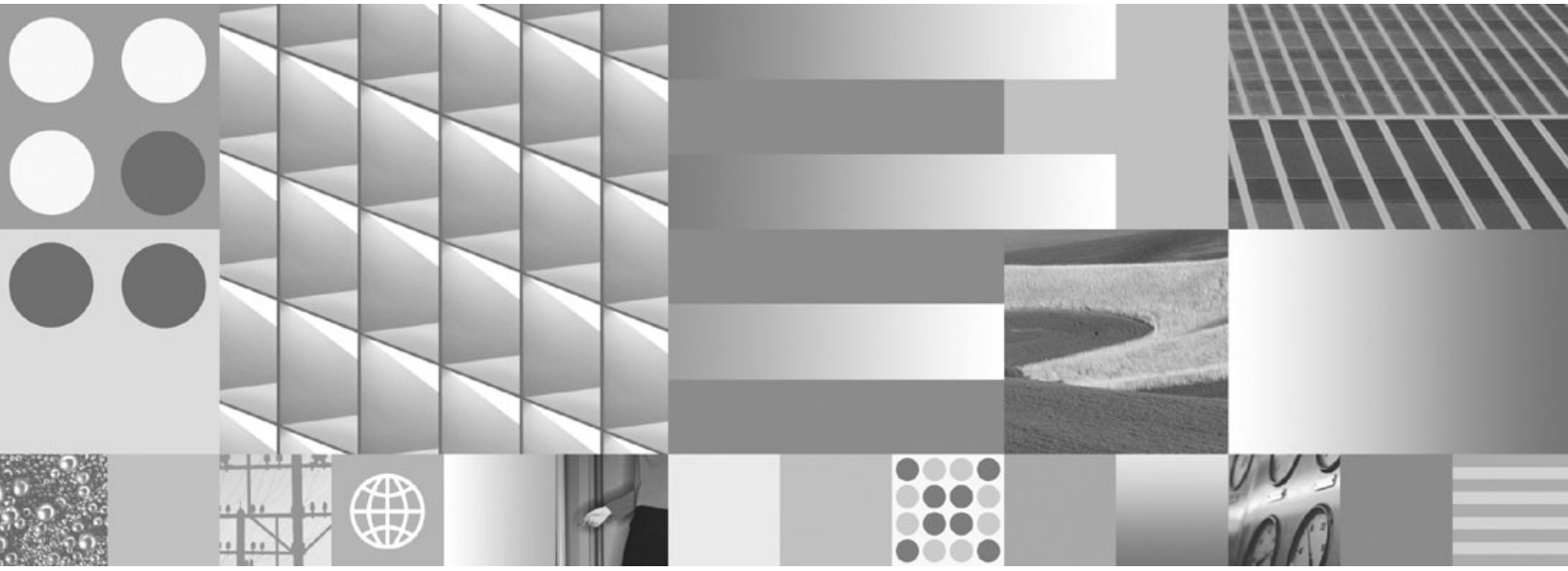


Text Analysis Integration

SC18-9674-02

IBM OmniFind Enterprise Edition

Version 8.5



Text Analysis Integration

SC18-9674-02

Note
Before using this information and the product it supports, read the information in "Notices and trademarks" on page 107.

Edition Notice

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

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

ibm.com and related resources

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

Support and assistance

Product support is available on the Web.

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

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

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

Information center

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

PDF publications

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

See the following PDF publications Web sites:

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

How to send your comments

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

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

Contacting IBM

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

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

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

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

Linguistic support for semantic search

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

You can use the linguistic support to improve the quality of search results.

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

Enterprise search includes only granular or basic linguistic functionality that is used to determine the language of an input document and to segment the document input stream into words or tokens.

If you know that your searches will be restricted primarily to basic keyword searches or native XML searches that uses the document structure, the linguistic processing that is included in enterprise search adequately covers your needs.

Most information in text documents is unstructured, which makes it difficult to use effectively because it is not easy to access the meaning of the information.

Searching for keywords is simple, but it is not always satisfactory if you want to search beyond the mere words in the document, as is illustrated in the following examples:

- In collaboration cases, information is not always explicitly marked, for example, an address or a phone number in an e-mail. In fact, the term *phone number* might not be used at all. Instead the e-mail might contain a phrase such as ″you can reach me at 555-641-1805″. The user often does not know how the information that he or she wants to search for is presented in the document, and would ideally want to enter a query like ″Barbara phone number″ when looking for the phone number of someone called Barbara. However, this query will not be successful because the word *phone number* does not occur in the document.
- In competitive intelligence, documents mention competitors and the goods that they supply or that the competitor's Web site shifted over the past three months from selling one product set to another. In this case, the user might enter a query like ″Smith & Co. goods″ or ″Smith & Co. goods Nov. 2004 till Jan. 2005″. In the first query, the term *goods* stands for a product or range of products, but the query will not return the products supplied by Smith & Co. because it is looking for the term *goods*. The same applies to the query that include a particular time period. It is almost impossible to query a time period by using keyword search.
- In customer relationship management, documents might mention automobile brake problems in repair shops in the San Francisco area. The repair shop reports describe situations such as ″shoe adjusted because of a hydraulic leak″. The user querying for more detailed information might enter a query like ″brake problem repair shops in north San Francisco″. However, this query might not return any reports that talk about ″shoe adjusted because of a hydraulic leak″ because the terms *brake problem* or *repair shops* as such do not occur in the reports. Moreover, these reports might mention only the street and district name of the repair shop, not the full address including the city name San Francisco.
- In research, documents describe a particular drug widely marketed under various trademarks and its relation to at least one disease that is mentioned in

the same paragraph. The casual user might enter a query using one of the popular terms for the drug hoping for a more detailed account of the various illnesses including symptoms. However, the query might not return satisfactory documents because the popular term might not always be used in documents and these documents often do not mention the word *illness* at all, only the name of the illness itself.

In these examples, searching for what you need in the vast collections of information sources that exist today presents new challenges that require 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 document content must be analyzed to recognize and find concepts of interest, for example, named entities like persons, organizations, locations, facilities and products, and the possible relations between these entities.

The information that you want to discover and extract in text documents is user and domain specific. To help you to design and develop your own analysis algorithms, IBM offers the IBM Unstructured Information Management Architecture (UIMA), an architecture and software framework that helps you build advanced analysis capabilities for finding information of interest in document collections in enterprise search.

Related concepts

"Custom text analysis integration" on page 3

"Basic concepts used in text analysis processing" on page 4

Custom text analysis integration

After you have built your custom analysis outside enterprise search using the Unstructured Information Management Architecture (UIMA), you can integrate the analysis logic in enterprise search using the enterprise search administration console.

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.

Advanced linguistic analysis can include a combination of many different analysis tasks. The analysis begins with language detection and segmentation, and continues with part-of-speech recognition, followed by deep grammatical parsing. The last tasks include identifying, for example, the relation between certain chemical substances and the appearance of particular symptoms. Each step in the analysis process depends on the results of the previous step.

The analysis logic for each step is contained in an *annotator*. Annotators combine to form a processing chain that iterates over each document in the collection to discover new information and store this information for downstream processing.

The annotators that are responsible for discovering and representing analysis content in text documents are contained in an *analysis engine*, a central concept in UIMA. An analysis engine might contain a single annotator or it might be a composite of many engines, each in turn containing annotators.

UIMA only 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. However, the linguistic processing that is applied in enterprise search is available as a set of annotators that you can work with in UIMA.

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/. The UIMA Software Development Kit (SDK) includes a Java™ implementation of the UIMA framework for the implementation, description, composition and deployment of UIMA components.

The UIMA SDK also provides a set of tools and utilities for working with UIMA in an Eclipse-based development environment (Eclipse plug-ins). For information about Eclipse, see `www.eclipse.org` and the UIMA documentation for instructions on how to install the UIMA Software Development Kit in the Eclipse Interactive Development Environment.

Related concepts

"Linguistic support for semantic search" on page 1

Basic concepts used in text analysis processing

Basic concepts that are used in text analysis processing include annotators, analysis results, feature structure, type, type system, annotation and common analysis structure.

Annotators contain the logic that analyzes a document and discovers and records descriptive data about the document as a whole (referred to as document meta data) and parts in the document. This descriptive data is referred to as *analysis results*. The analysis results annotate any contiguous substring (also referred to as span) of the text document. Ideally, the analysis results correspond to the information that you want to search for.

A *feature structure* is the underlying data structure that represents an analysis result. 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, such as String.

For example, the text span ″James Matthew Bloggs″ might be spanned by an annotation of type `Person` with the features `personName`, `age`, `nationality` and `profession`.

The *type system* defines the types of objects (feature structures) that may be discovered in a document. The type system defines all possible feature structures in terms of types and features (attributes), much like a class hierarchy in Java. You can define any number of different types in a type system. A type system is domain and application specific.

Most of the text analysis 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. An annotation 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 creates 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 model and store the data in feature structures.

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

The common analysis structure contains the following objects:

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

Related concepts

“Linguistic support for semantic search” on page 1

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 annotators 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 semantic search against information stored in the index.

As the UIMA Software Development Kit does not provide any pre-configured annotators, and because any custom annotators that you develop by using UIMA and then integrate in enterprise search builds upon the results of the enterprise search base annotators, you can use the base annotator package to your UIMA environment. Refer to the UIMA documentation on how to include language detection and tokenization functionality before you run the custom text analysis algorithms in your UIMA environment.

After you have developed and tested your analysis engines using the UIMA Software Development Kit, you must create a PEAR (Processing Engine ARchive) file to run your algorithms on a document collection in enterprise search. This archive file includes all of the required resources for deploying your custom analysis functionality as analysis engines in enterprise search. How to create an archive is described in the UIMA documentation provided in the Software Development Kit.

The archive that you create to upload to enterprise search must only contain your custom analysis logic. It must not contain any of the enterprise search base annotators even if your custom analysis logic builds on base annotator results because the base annotators always run before any custom analysis in enterprise search.

To learn how to configure and deploy a semantic search solution in enterprise search, run the tutorial mentioned at http://www.ibm.com/developerworks/db2/zones/db2ii/. The tutorial guides you through the steps involved in deploying custom text analysis algorithms in enterprise search and shows you how to use the analysis results in queries to improve search results.

Related tasks

"Using the enterprise search base annotators in UIMA" on page 7

Workflow for custom analysis integration

You create and test your custom text analysis algorithms using the UIMA Software Development Kit, and then deploy and run them on document collections in enterprise search.

To develop analysis algorithms and to integrate them in enterprise search:

1. Plan and design:
   a. Determine what information you want to search for. What are the documents that you want to retrieve? Which concepts and relationships are needed for each particular search task? For example, product and employee names might be needed to enhance general purpose searches on a

pharmaceutical company's internal Web site, while people in the area of research and development need to use variants of drug names and see drug-cause-cure relationships.

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 whether 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 that can be useful in a summarization or categorization annotator), create a XML elements to the common analysis structure mapping file.
- If you want to use the XML markup in your queries as it appears in the document, you must enable native XML mapping.

d. Determine which text analysis result information that is stored in the common analysis structure you want to be able to access using semantic search. Create a common analysis structure to index mapping file.

e. Determine whether you want to store analysis results in a relational database, for example, to discover trends and associations by using reporting or data mining applications. Create a common analysis structure to database mapping file.

f. Design the semantic search application. Determine the search user's use of the additional capabilities of semantic search. Design the user interface.

2. Develop: UIMA Software Development Kit activities

a. Define the individual analysis steps.

b. Describe the type system for your mappings and analysis algorithms.

c. Develop the analysis algorithms (annotators) for each analysis step and embed the annotators in analysis engines using the UIMA Software Development Kit. Build any custom analysis using the basic functionality (language identification and tokenization) in the enterprise search base annotators package.

d. After testing the analysis algorithms in UIMA, package the analysis engine(s) as a PEAR (Processing Engine Archive) file. The archive must only contain your analysis algorithms, and not the basic enterprise search linguistic functionality.

When you design a text analysis solution, it might include several analysis modules provided in more than one PEAR file. UIMA provides a means of merging two or more PEAR files into a single PEAR file that you can upload and run in enterprise search. The facility for merging PEAR files ensures that there are no naming collisions, the input and output capabilities are correctly merged, and that there is no parameter overriding if merged parameters in annotator descriptors have the same names. See the UIMA documentation for instructions on how to merge PEAR files.

3. Deploy: enterprise search activities

a. Upload the processing engine archive file (.pear) to enterprise search. Provide a name for the text analysis component by which you can refer to it in enterprise search.

b. Associate one or more document collections with your text analysis component.

c. If applicable, for each collection, upload and select the XML element to the common analysis structure mapping that you defined for your custom analysis.

d. If applicable, for each collection, upload and select the common analysis structure to database mapping that you defined for your custom analysis.
e. For each collection, upload and select the common analysis structure to index mapping that you defined for semantic search.
f. If necessary, set up your custom semantic search application, for example, deploy your browser-based search user interface into an application server.
g. Crawl, parse, and index the documents in your semantic search collection as you would for a keyword-based collection.

Related tasks

"Using the enterprise search base annotators in UIMA"

Using the enterprise search base annotators in UIMA

You can use the annotators in the enterprise search base annotator package to develop new annotators within the UIMA Software Development Kit (SDK) and to map analysis results to JDBC tables.

The set of base annotators includes:

- **Language ID annotator**

  Detects the language of a document. For the capabilities and configuration parameters, refer to the descriptor file jlangid.xml.
- **FROST dictionary lookup annotator**

  Provides tokenization and sentence detection, based on the IBM LanguageWare dictionaries. For tokens, additional linguistic information, for example the base form or lemma, is generated. For the capabilities and configuration parameters, refer to the descriptor file jfrost.xml.
- **White-space tokenizer**

  Can perform white-space based tokenization on all European language documents, or other white-space separated scripts. In addition, the annotator is able to perform n-gram tokenization on the following text scripts: Arabic, Han, Hebrew, Hiragana, Katakana, Lao, Mongolian, Thai, YI, and Hangul. This list includes all major Asian text scripts and means that the annotator supports Japanese, Chinese, and Korean.

  For the capabilities and configuration parameters, refer to the descriptor file jtok.xml.
- **Regular expression annotator**

  Detects entities or spans of information in a text document based on regular expressions. You can customize the regular expression annotator to detect the text entities that you need by defining your own rules. A sample regular expression annotator that detects telephone numbers, URLs, and e-mail addresses in text documents is included in the base annotator package.
- **Common analysis structure to database consumer**

  The common analysis structure to database consumer populates a relational database with specific text analysis results.

The enterprise search base annotator package is a zipped file that contains the base text analysis annotators, the regular expression annotator and the common analysis structure to database consumer. The Language ID annotator, the FROST dictionary lookup annotator, and the White-space tokenizer are the base text analysis annotators that always run before any custom text analysis when documents are parsed in enterprise search.

Because the base text analysis annotators always run before any custom text analysis in enterprise search, and because all custom text analysis is based on the output of the base annotators, you can use these annotators to your UIMA environment when you develop and test your custom annotators.

The regular expression annotator and the common analysis structure to database consumer are additional options that you can select on the enterprise search administration console when you configure your text processing options. You can also use them in UIMA. For advanced customization of the regular expression annotator, it is recommended that you use the supplied UIMA SDK tools to customize the annotator.

To run any of these annotators in UIMA, you must have the UIMA Software Development Kit (SDK) installed. It is available on the IBM developerWorks Web site at http://www.ibm.com/developerworks/db2/zones/db2ii/.

To install the annotator package in your UIMA SDK installation:

1. Find the annotator package OF_base_annotators.zip in your enterprise search (OmniFind Enterprise Edition) installation in the *ES_INSTALL_ROOT*/packages/uima directory.
2. Copy the zipped file to the root directory of your UIMA SDK installation.
3. Extract the zipped file to add the enterprise search base annotator files to the specified directory structure of your UIMA SDK installation. The file tt_core_typesystem.xml will be overwritten. If you want to keep your old version of this file, save it before you extract the zipped file.
4. To set the class path, open the setUIMAClasspath script in the bin directory and add a line at the end of the script that starts the OFAnnotEnv script.
5. If you want to use any custom or enterprise search specific types in UIMA, refer to the UIMA SDK documentation on how to define these.

After you install the base annotator package, you can find the annotator descriptor files in the directory *UIMA_SDK_INSTALL*/docs/examples/descriptors/analysis_engine. The file of_tokenization.xml lists the base text analysis annotators (the Language ID annotator, the FROST dictionary lookup annotator, and the White-space tokenizer) in the sequence that they are used within enterprise search.

The descriptor files contain the same configuration values that are used in enterprise search. You can change values for debugging purposes in the UIMA SDK. However, do not change these descriptor files in your enterprise search system. Making changes to these files might cause system instability or performance problems.

The enterprise search base annotator package contains only the dictionaries that are required to process English documents. If you want to process other languages in your development environment, follow these steps:

1. Find the enterprise search dictionaries in your enterprise search installation in *ES_INSTALL_ROOT*/configurations/parserservice/jediidata/frost/resources.
2. Copy the contents of the directory to your local UIMA SDK installation in *UIMA_SDK_INSTALL*/data/frost/resources .

To verify that the annotator package was successfully installed:

1. Open the Common Analysis Structure (CAS) Visual Debugger (CVD) in the following directory: *UIMA_SDK_INSTALL*/bin/cvd[.bat/.sh].
2. Click **Run → load TAE**.

3. Select the text analysis engine specifier file of_tokenization.xml in the *UIMA_SDK_INSTALL*/docs/examples/descriptors/analysis_engine directory.
4. Load a sample document and run the text analysis engine. You will see annotations of type uima.tt.TokenAnnotation in the CVD.

If you run any of the base text analysis annotators before your custom annotators in your development environment, and your custom annotators use types that are defined by the base text analysis, include a reference to the file tt_core_typesystem in the type system section of your custom annotator specifier. The tt_core_typesystem file is in the *UIMA_SDK_INSTALL*/docs/examples/descriptors/analysis_engine directory. See the file jtok.xml in the analysis_engine directory for an example of how to include references to descriptor files.

Related tasks

"Viewing base annotator and custom text analysis results" on page 11

"Enabling easy semantic search using the regular expression annotator" on page 80

Using the common analysis structure to database consumer in UIMA

Before you can use the common analysis structure to database consumer in UIMA, you must make changes to the consumer descriptor file and write the common analysis structure to database mapping file.

Before you can run the common analysis structure to database consumer in your UIMA environment, you must:

1. Open the XML descriptor file cas2jdbc.xml in *UIMA_SDK_INSTALL*/docs/examples/descriptors/cas_consumer. To avoid XML syntax errors, use an XML editor or XML authoring tool of your choice.
2. Modify the parameter **mappingFile** to include the absolute path where your the common analysis structure to database mapping file is located, for example, D:\temp\MyMapping.xml
3. Modify the parameter **docMetadata_Type** to specify the UIMA type from which all meta data for the built-in features is retrieved, for example, uima.tcas.DocumentAnnotation.
4. Modify the parameter **docId_Feature** to include the feature or feature path to the meta data type from which a document's numeric ID (of type integer) is retrieved. This is required by all built-in features that require the ID, such as, docId(), uniqueId(), objectId(), and fsId().
5. Do not set the parameter **encryptionClass** as it is used only in enterprise search to allow the common analysis structure to database consumer to work with encrypted mapping files.
6. Save the file.
7. Copy the EMF library files (common.jar, ecore.jar and ecore.xmi.jar) from the lib directory of your enterprise search installation to the lib directory of your UIMA installation. The cc_cas2jdbc.jar is already in the lib directory of your UIMA installation.
8. Create the common analysis structure to database mapping file that defines which text analysis results to store in a database. You can use the mapping file sampleMapping.xml at *UIMA_SDK_INSTALL*/docs/examples/descriptors/cas_consumer as a sample to create your own mapping file.

   Use the XML schema file called CasToJDBCMapping.xsd at *UIMA_SDK_INSTALL*/docs/examples/descriptors/cas_consumer to validate the common analysis

structure to database mapping file. For performance reasons, the common analysis structure to database consumer does not validate the mapping file, you must do this yourself.

How to run the consumer in UIMA is described in the UIMA documentation.

The following sample shows how the mandatory parameters must be defined in the descriptor:

```
 ...
 <nameValuePair>
  <name>mappingFile</name>
  <value>
   <string>D:/temp/MyMapping.xml</string>
  </value>
 </nameValuePair>
<nameValuePair>
  <name>docMetadata_Type</name>
  <value>
   <string>uima.tcas.DocumentAnnotation</string>
  </value>
 </nameValuePair>
<nameValuePair>
  <name>docId_Feature</name>
  <value>
   <string>end</string>
  </value>
 </nameValuePair>

...
```

The table shows the configuration parameters in the order in which they appear in the descriptor file and indicates which are mandatory:

Table 1. The configuration parameters in the common analysis structure to database consumer descriptor file

Parameter	Description	Mandatory
mappingFile	The absolute path to the common analysis structure to database mapping file, for example, `D:/temp/sample.xml`. On Windows® systems, use "/" as the path separator.	true
encryptionClass	Do not set this parameter, it is only used in enterprise search to allow the common analysis structure to database consumer to work with encrypted mapping files.	false
docMetadata_Type	The UIMA type from which all meta data for build-in features is retrieved.	true

Table 1. The configuration parameters in the common analysis structure to database consumer descriptor file (continued)

Parameter	Description	Mandatory
docId_Feature	The feature or feature path on the meta data type from which the document's numeric ID is retrieved. It must be of type integer and is needed for all built-in features that require the ID, such as `uniqeId()`, `objectId()`, and `fsId()`.	true
docUri_Feature	The feature or feature path on the meta data type from which the document's URI is. It must be of type string.	false
IsCompleted_Feature	The feature or feature path on the meta data type that flags whether the current document is chunked across several common analysis structures.	false
chunkNumber_Feature	The feature or feature path on the meta data type that denotes the subsequent number of the current chunk.	false

Using the regular expression annotator in UIMA

Use the regular expression annotator to detect entities or units of information in a text document. You can customize the annotator for your subject domain to meet your search needs.

To run the sample regular expression annotator that detects telephone numbers, URLs and e-mail addresses, or use the sample annotator as a basis to create your own customized version of the regular expression annotator in your UIMA environment, you need:

1. The regular expression annotator descriptor in the *UIMA_SDK_INSTALL*/docs/examples/descriptors/analysis_engine directory.
2. The sample rule set and type system description in the *UIMA_SDK_INSTALL*/docs/examples/regex directory.
3. An example text file that the sample rule set can be applied in the *UIMA_SDK_INSTALL*/docs/data directory called of_sample_regex.txt.

How to run the annotator in UIMA is described in the UIMA documentation.

Viewing base annotator and custom text analysis results

To view the analysis results produced after parsing and by any annotators in enterprise search, you must update the document collection properties to produce a readable XML version of the analysis results that are stored in the common analysis structure.

About this task

You use the XML serialization of the annotator analysis results stored in the common analysis structure to:

- View the results after parsing, before the base annotators are processed.
- View the results after parsing and tokenization (running the enterprise search base annotators). This can help you determine the input data structures to any custom analysis that you want to develop and that will always run after the base annotators.
- View and validate the results of a custom analysis run on a smaller document collection in enterprise search for test purposes before deciding to run the analysis on a complete collection.

The XML serialization produces two sets of results:

- The results after parsing. These include field mappings and document meta data.
- The results after parsing and tokenization, and if selected, custom text analysis. These include all produced tokens and annotations.

Procedure

To produce a readable XML version of the analysis results:

1. Open the file collection.properties in *ES_NODE_ROOT*/master_config/<CollectionID>.parserdriver before you begin to parse the documents in your collection.
2. To view the results after parsing, add the following line to the collection.properties file: `trevi.parser.dumpXCas=<your_dump_directory>`

   Your dump directory must already exist.

   a. Select the type of output you want. The output always includes the type system description used for the parsing results called OmniFindParserTypeSystem.xml. Add one of the following lines:

      - To view the output of the last 25 processed files, add `trevi.parser.maxXCasFileCount=25`.

        You can determine the number of files yourself although it is advisable not to set this value too high.

        Keep in mind that the file output buffer is constantly overwritten after the maximum buffer size is reached. This also implies that the document with the highest number need not be the last one processed.

        The output includes the following files: OmniFindParserXCasDump1.xml followed by OmniFindParserXCasDump2.xml, and so on until 25 files are listed.

      - To view the output of specific documents, add the document URI `trevi.parser.xCasURI.1=file://home/test/file1.txt`.

        You can add any number of documents, however, the documents must be numbered in ascending order beginning at 1 with no gaps between any numbers. For example, the second document would be `trevi.parser.xCasURI.2=file://home/test/file2.txt` and the third one `trevi.parser.xCasURI.3=file://home/test/file3.txt`

        The output includes the following files: OmniFindParserXCasDumpURI_1.xml, OmniFindParserXCasDumpURI_2.xml, and so on for as many file names as you listed
3. To view the results after tokenization, add the following line: `trevi.tokenizer.dumpXCas=<your_dump_directory>`

Again, your dump directory must exist.

a. Select the type of output you want. The created output always also includes the type system description used for the tokenization and text analysis results called OmniFindTypeSystem.xml. Add one of the following lines:
   - To view the output of the last 25 processed files, add `trevi.tokenizer.maxXCasFileCount=25`.

     You can determine the number of files yourself although it is advisable not to set this value too high.

     Keep in mind that the file output buffer is constantly overwritten after the maximum buffer size is reached. This also implies that the document with the highest number need not be the last one processed.

     The output includes the following files: OmniFindXCasDump1.xml, OmniFindXCasDump2.xml, and so on until 25 files are listed.
   - To view the output of specific documents, add the document URI `trevi.tokenizer.xCasURI.1=file://home/test/file1.txt`.

     You can add any number of documents, however, the documents must be numbered in ascending order beginning at 1 with no gaps between any numbers. For example, the second document would be `trevi.tokenizer.xCasURI.2=file://home/test/file2.txt` and the third one `trevi.tokenizer.xCasURI.3=file://home/test/file3.txt`

     The output includes the following files: OmniFindXCasDumpURI_1.xml, OmniFindXCasDumpURI_2.xml, and so on for as many file names as you listed.

In enterprise search, you can use the XCAS Annotation Viewer to view the content of the XML files. Start the XCAS Annotation Viewer by running the `xcasAnnotationViewer` script file located in the `ES_INSTALL_ROOT/bin` directory. You are prompted for:

- Your dump directory where the results are placed after parsing or tokenization
- The descriptor file, either OmniFindParserTypeSystem.xml (for parser results) or OmniFindTypeSystem.xml (for tokenization and analysis results), likewise in your dump directory.

Selecting a document from the list displays the analysis results for the document. Clicking on a highlighted annotation in the document displays the details of the annotation.

Type system description

The type system defines the types of objects and their properties (or features) that may be instantiated in a common analysis structure.

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

The type systems include the definitions of types, their properties, and single-inheritance hierarchy of types. A common analysis structure must conform to a particular type system.

The types and features that are defined in the type system description must also be used in all mapping files that are associated with the document analysis, including

the XML elements to the common analysis structure mapping file, the common analysis structure to index mapping file, and the common analysis structure to database mapping file.

The type system description of an annotator can be part of the annotator's descriptor, or it can be contained in a separate type system descriptor file. Sometimes it is part of the descriptor of another annotator contained in the same analysis engine.

When you have completed developing and testing your analysis engine in your UIMA environment, the archive file (.pear file) that you create and upload to enterprise search contains your analysis logic files as well as your type system description.

The enterprise search base annotators use three type system descriptions; a core type system description that is always included, and two others that you can activate optionally to change document collection base analysis processing to advanced analysis mode. Whether you need to include either one or both of the extended type system descriptions depends on which additional text analysis processing results you want to include during base analysis processing.

You can enable advanced analysis mode by including one or both of the extension type systems. In advanced analysis mode, additional analysis features are made available during base analysis processing and are saved to the common analysis structure. For example, if you require more information about a token (more feature information), such as all possible lemmas for the token, or if the lemma is a stop word, or the lemma's part of speech, or special features for morphological processing, also for Japanese, you need to activate advanced analysis mode.

Changing from base analysis mode to advanced analysis mode

To change the document collection processing that is carried out by the enterprise search base annotators from base analysis mode to advanced analysis mode, you must include the type system descriptions for advanced analysis mode.

Restrictions

There are two type system descriptions that you can select to activate advanced analysis mode:

- The `tt_extension_typesystem` description, which includes more detailed lexical typed feature information on lemmas.
- The `dlt_extension_typesystem` description, which includes additional morphological features and special lexical types.

Procedure

To change the base collection processing to advanced analysis mode:

1. Open the file tt_core_typesystem.xml in the *ES_NODE_ROOT*`/master_config/`*CollectionID*`.parserdriver/specifiers` directory. To avoid XML syntax errors, use an XML editor or XML authoring tool of your choice.
2. Remove the comment tags surrounding the `<import>` element in the `<imports>` section to include either one or both of the extension type system description files.

```
<imports>
<!-- imports the tt_extension_typsystem for advanced analysis -->
<!-- <import location="tt_extension_typesystem.xml"/>-->
<!-- imports the dlt extension typesystem -->
<!-- <import location="dlt_extension_typesystem.xml"/> -->
</imports>
```

3. Open the two descriptor files jfrost.xml and jfrost_ngram.xml, and modify the content of the `<outputs>` element to include the types (in a `<type>` element) and features (in a `<feature>` element) listed in the `<description>` element in the `<capabilities>` section that you want to include during analysis. Save your changes.
4. Open the descriptor file jtok.xml and modify the content of the `<outputs>` element to include the features (in a `<feature>` element) listed in the `<description>` element in the `<capabilities>` section that you want to include during analysis. Save your changes.
5. Open the descriptor file es_tok_no_stw.xml and here too, modify the content of the `<outputs>` element to include the features (in a `<feature>` element) listed in the `<description>` element in the `<capabilities>` section that you want to include during analysis. Save your changes.
6. When you change to advanced analysis mode, you must parse your document collection again.

Related reference

"Types and features defined for enterprise search"

Types and features defined for enterprise search

The type system defined for enterprise search covers document metadata handling and basic linguistic analysis.

The types used in enterprise search are defined in three separate type system description files, beginning with the type system description file that contains the core types always required for all basic linguistic analysis and continuing with type system descriptions that define advanced linguistic features that are normally only required in advanced analysis mode.

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 `tt_core_typesystem` description is used and the following information is added to the common analysis structure that you can use in subsequent custom analysis:

- Document metadata of type `com.ibm.es.tt.DocumentMetaData`.
- Document structure information such as sentence, and paragraph annotations of type `uima.tt.SentenceAnnotation`, and `uima.tt.ParagraphAnnotation`.
- Lexical annotations such as tokens and compounds of type `uima.tt.TokenAnnotation`.

The `tt_core_typesystem` description is adequate for most text analysis processing.

If you want to change collection processing to the advanced analysis mode, you can include the following two type systems. The type systems primarily include additional features that are not created during basic linguistic processing.

- `tt_extension_typesystem` that includes more token, lemma, paragraph and sentence feature information

- `dlt_core_typesystem` that contains some of the IBM LanguageWare extended annotation types, for example, URLs and addresses. It also include morphological features that are not utilized frequently.

tt_core_typesystem

The following types and features are defined in the `tt_core_typesystem` description:

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

- `categories` with document categories added by a text categorizer. Each added category is of the type `com.tt.CategoryConfidencePair`
- `languageCandidates` with the document languages automatically detected during parsing. The languages are added to a list of type `com.tt.LanguageConfidencePair`, with the most likely language listed first
- `id` with the document ID, such as the URL

uima.tt.TTAnnotation
This is the root type for annotations defined in `tt_core_typesystem`. It supertype is `uima.tcase.Annotation`. It has the following types:

uima.tt.DocStructureAnnotation
Annotations about the document structure. It has the following subtypes:

uima.tt.SentenceAnnotation
Sentences

uima.tt.ParagraphAnnotation
Document paragraph

uima.tt.LexicalAnnotation
Lexical annotations such as tokens or multi-word expressions. It has the following subtypes:

uima.tt.TokenLikeAnnotation
Single token annotations that can have the following features:

- `tokenProperties` with the token properties
- `lemma` with the lemma or stem of the term
- `normalizedCoveredText` with the normalized representation of the covered text

This annotation type has the following subtypes:

uima.tt.TokenAnnotation
Actual tokens to be distinguished from compound parts.

uima.tt.CompPartAnnotation
The compound parts of a term.

uima.tt.CompoundAnnotation
The annotation of a compound token. The compound token usually spans more than one token annotation.

uima.tt.MultiTokenAnnotation
Lexical annotation consisting of more than one token. This annotation type has the following subtypes:

uima.tt.StopwordAnnotation
Annotations of stop words. The stop words can also be multi-term words.

uima.tt.SynonymAnnotation
The annotation of a term for which there are synonyms. It has the feature `synonyms` that lists the found synonyms for the term.

uima.tt.SpellCorrectionAnnotation
The annotation of a term for which there are spell corrections. It has the feature `correctionTerms` that lists likely corrections in a sorted order beginning with the most probable corrections.

uima.tt.MultiWordAnnotation
The annotation of a multi-word term.

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

uima.tt.KeyStringEntry
The abstract type for String data structures. It includes the feature `key` that contains the string key and the following subtype:

uima.tt.Lemma
Dictionary lemma entries.

uima.tt.CategoryConfidencePair
The confidence value for the found category. It has the following features:

- `categoryString` with the name of the category
- `categoryConfidence` with the confidence value for the category
- `mostSpecific` with a flag indicating if this category is the most specific for the document
- `taxonomy` with the name of the taxonomy the category is derived from

uima.tt.LanguageConfidencePair
The confidence value for the found category. This type includes the features `languageConfidence`, `language`, and `languageID`.

tt_extension_typesystem

The `tt_extension_typesystem` includes additional text analysis features for more advanced processing.

uima.tt.TokenLikeAnnotation
This annotation type in the `tt_extension_typesystem` has the following features:

- `lemmaEntries` lists all of the possible lemmas for the token. The list items are of type `uima.tt.Lemma`
- `tokenNumber`
- `stopwordToken`

uima.tt.Lemma

This annotation of type `uima.tt.KeyStringEntry` has the following features:

- `isStopword` is true if the lemma is a stop word
- `isDeterminer` is true if the lemma is a determiner
- `partOfSpeech`. The following part-of-speech number description codes exist:
  - 0: unknown
  - 1: pronoun
  - 2: verb
  - 3: noun
  - 4: adjective
  - 5: adverb
  - 6: adposition
  - 7: interjection
  - 8: conjunction

uima.tt.DocStructureAnnotation

Annotations about the document structure. This has the following subtypes:

uima.tt.SentenceAnnotation

Document sentence. It has the feature `sentenceNumber`.

uima.tt.ParagraphAnnotation

Document paragraph. It has the feature `paragraphNumber`.

dlt_extension_typesystem

The `dlt_extension_typesystem` includes additional features used by IBM LanguageWare.

uima.tt.LexicalAnnotation

This annotation has the following subtypes:

uima.tt.TokenLikeAnnotation

In the `dlt_extension_typesystem`, this annotation has the following features:

- `synonymEntries`
- `frost_TokenType`
- `inflectedForms`
- `spellAid`
- `decomposition`

com.ibm.dlt.uimatypes.FilePath

com.ibm.dlt.uimatypes.Email

com.ibm.dlt.uimatypes.Number

com.ibm.dlt.uimatypes.URL

com.ibm.dlt.uimatypes.Date

com.ibm.dlt.uimatypes.Time

com.ibm.dlt.uimatypes.Tel

com.ibm.dlt.uimatypes.Currency

com.ibm.dlt.uimatypes.Acronym

uima.tt.TokenLikeAnnotation
This annotation type in the `dlt_extension_typesystem` has the following type:

com.ibm.dlt.uimatypes.MWU
This type is used by IBM LanguageWare to annotate multi-word expressions.

uima.tt.KeyStringEntry
String annotations. This has the following subtypes:

uima.tt.Lemma
It has the following features:

- `frost_Constraints` with constraint flags
- `frost_MorphBitMasks` containing a morphological bit mask array
- `frost_ExtendedPOS` with extended part of speech information, such as JPOS for Japanese and CPOS for Chinese
- `frost_JKom` containing Japanese morphological data
- `frost_JPStart` containing Japanese start analysis data
- `morphID` containing lemma properties

uima.tcas.Annotation
This has the following subtype:

com.ibm.dlt.uimatypes.Decomp_Analysis
Full structural analysis of a compound. This has the following features:

- `headComponentIndex` with the head component of the compound
- `route` containing a list of tokens that comprise a single decomposition route

Related reference

"Type system description sample" on page 22

Specific types and features for enterprise search

The types and features defined in the of_typesystem description cover specific types for OmniFind Enterprise Edition. These types are used for document specific metadata. They also describe the representation of fields and XML markup information or HTML anchors.

The of_typesystem description is not defined 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, you must define the types again in the type system description of your analysis engine. For example, you might want to access document security information or access the crawler type or document type.

The following types and features are defined in the of_typesystem description:

uima.tcas.DocumentAnnotation
The standard UIMA document annotation is extended by the following feature:

esDocumentMetaData
Contains document metadata of the type `com.ibm.es.tt.DocumentMetaData`.

com.ibm.es.tt.DocumentMetaData

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

- CM, for documents that are crawled by the DB2 Content Manager crawler
- Database, for documents that are crawled by the JDBC database crawler
- DB2, for documents that are crawled by the DB2 crawler
- DominoDoc, for documents that are crawled by the Domino Document Manager crawler
- Exchange, for documents that are crawled by the Exchange Server crawler
- NNTP, for documents that are crawled by the NNTP crawler
- Notes, for documents that are crawled by the Notes crawler
- QuickPlace, for documents that are crawled by the QuickPlace crawler
- Seedlist, for documents that are crawled by the Seed list crawler
- UnixFS, for documents that are crawled by the UNIX file system crawler
- VBR, for documents that are crawled by the Content Edition crawler
- WCM, for documents that are crawled by the Web Content Management crawler
- Web, for documents that are crawled by the Web crawler
- WinFS, for documents that are crawled by the Windows file system crawler
- WP, for documents that are crawled by the WebSphere Portal crawler

dataSourceName

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

docType

One of the following document types. The feature value is of type `uima.cas.String`.

- 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)

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`.

mimeType
The MIME type or document type, for example, XML. 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 that the field is free-text searchable.

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

parametric
A flag indicating that the field is searchable with a parametric query.

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

resolveConflict
A flag for resolving metadata 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 by using the field name. The feature value is of type `uima.cas.String`.

sortable
A flag indicating that the field is string sortable.

exactMatch
A flag indicating that the search must be an exact match of the query terms.

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.MetaDataField
The metadata field data is not part of the document content but is stored in the ″text″ feature:

parameters
Metadata field parameters of type `com.ibm.es.tt.CommonFieldParameters`.

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

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`.

Type system description sample

The type system description describes the feature structures (the underlying data structures that represent the analysis results) that are used in custom analysis.

The type system description must be part of the analysis engine archive (.pear file) that is imported from your UIMA environment to enterprise search.

The following type system description sample describes police reports that contain information on suspects, crime location, crime time, and date:

The same sample type system description is used in all of the text analysis topics that discuss the different types of mappings that you can select with custom analysis.

```
<?xml version="1.0" encoding="UTF-8"?>
<typeSystemDescription>
   <name>Police Reports Type System</name>
   <description>Type system description for
        police reports</description>
   <version>1.0</version>
   <types>
     <typeDescription>
       <name>com.ibm.omnifind.types.PoliceReport</name>
       <description>Annotates a police report</description>
       <supertypeName>uima.tcas.Annotation</supertypeName>
```

```
  <features>
    <featureDescription>
      <name>time</name>
      <description>Time the crime was reported to have happened
         </description>
      <rangeTypeName>com.ibm.omnifind.types.Time</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>date</name>
      <description>When the crime happened</description>
      <rangeTypeName>com.ibm.omnifind.types.Date</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>location</name>
      <description>Where the crime took place</description>
      <rangeTypeName>com.ibm.omnifind.types.City</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>knownSuspects</name>
      <description>Contains annotations of type Suspect</description>
      <rangeTypeName>uima.cas.FSArray</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>crimeDescription</name>
      <description>Short description of the crime</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
  </features>
</typeDescription>
<typeDescription>
  <name>com.ibm.omnifind.types.City</name>
  <description>The name of a city</description>
  <supertypeName>uima.tcas.Annotation</supertypeName>
  <features>
    <featureDescription>
      <name>cityName</name>
      <description>The name of the city</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>cityDistrict</name>
      <description>The name of the district</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
  </features>
</typeDescription>
<typeDescription>
  <name>com.ibm.omnifind.types.Person</name>
  <description>A person annotation</description>
  <supertypeName>uima.tcas.Annotation</supertypeName>
  <features>
    <featureDescription>
      <name>role</name>
      <description>For example, suspect or witness</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>firstName</name>
      <description>The first name of the person</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>surName</name>
      <description>The surname of the person</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
```

```
      <name>title</name>
      <description>For example, Mr. or Ms.</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>gender</name>
      <description>Male or female</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
  </features>
</typeDescription>
<typeDescription>
  <name>com.ibm.omnifind.types.Suspect</name>
  <description>A found suspect</description>
  <supertypeName>com.ibm.omnifind.types.Person</supertypeName>
  <features>
    <featureDescription>
      <name>description</name>
      <description>Suspect description,
    for example, bearded with dark glasses</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
  </features>
</typeDescription>
<typeDescription>
  <name>com.ibm.omnifind.types.Date</name>
  <description>A date</description>
  <supertypeName>uima.tcas.Annotation</supertypeName>
  <features>
    <featureDescription>
      <name>year</name>
      <description>The year, for example, 2005</description>
      <rangeTypeName>uima.cas.Integer</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>month</name>
      <description>The month in digits, for example, 7</description>
      <rangeTypeName>uima.cas.Integer</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>day</name>
      <description>The day in digits</description>
      <rangeTypeName>uima.cas.Integer</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>dayOfWeek</name>
      <description>The day of the week, for example, Monday</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>quarter</name>
      <description>The quarter, for example, Q1-2005</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>englDate</name>
      <description>Date as mm/dd/yyyy</description>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
  </features>
</typeDescription>
<typeDescription>
  <name>com.ibm.omnifind.types.Time</name>
  <description>A time</description>
  <supertypeName>uima.tcas.Annotation</supertypeName>
  <features>
    <featureDescription>
```

```
        <name>hours</name>
        <description>Hours from 00-23</description>
        <rangeTypeName>uima.cas.Integer</rangeTypeName>
      </featureDescription>
      <featureDescription>
        <name>minutes</name>
        <description>Minutes in the hour</description>
        <rangeTypeName>uima.cas.Integer</rangeTypeName>
      </featureDescription>
      <featureDescription>
        <name>timeOfDay</name>
        <description>Time periods, such as morning, noon</description>
        <rangeTypeName>uima.cas.String</rangeTypeName>
      </featureDescription>
    </features>
  </typeDescription>
 </types>
</typeSystemDescription>
```

XML markup in analysis and search

You can map information in XML structures that are in a document directly to a common analysis structure without writing a UIMA annotator.

If the documents in your collection are in XML, and you want to exploit the XML markup during text analysis or semantic search, you have the following options:

Native XML search

Use this option if you want to use all of the XML tags and attributes as they appear in the document during semantic search. For example, if you have billing documents that contain an `<addressee>` element, enabling native XML search allows you to use this tag in a semantic search query to search for a certain customer name within this element.

With this option, the XML structure of the document is represented in the common analysis structure using the `com.ibm.es.tt.MarkupTag` type. For each XML tag, an annotation of this type is created. This annotation contains the name of the tag, its attributes and the attribute content. This information is always indexed and is accessible to semantic search.

Native XML search does not require a mapping configuration file. You can enable native XML search from the administration console for enterprise search.

XML elements to the common analysis structure mapping

Use this option in the following cases:

- The semantics of certain XML elements are precise and can be used in further text analysis steps. These analysis steps can operate directly on the annotations and features created from the XML structures, and are shielded from the potentially different formats of the original documents. For example, the element `<addressee>` in documents on billings usually contains customer names. Using the XML elements to the common analysis structure mapping, the content of this element can be mapped directly to annotations of type `Customer`. An annotator can then infer a Customer-located-at relationship, using the information surrounding the `Customer` annotation.

- You want to limit the processing scope of a custom annotator to specified areas in the XML input. For example, you might want to limit the analysis to the content of the <technicianComment> tags only in an annotator that detects car problems.
- You want to restrict both text analysis processing and subsequent search to certain parts of the XML document, and filter out irrelevant or non-textual content.
- You want to map XML tags that have different names to a common span that is to be used in semantic search. For example, mapping <mainHeading> or <doc> to title.

In these cases, you must create an XML elements to the common analysis structure mapping file that defines which XML elements map which feature structures. The feature structures that you define in the mapping file are created when the documents are parsed, and are accessed by the custom annotators.

You can use more than one XML elements to the common analysis structure mapping file for a document collection. Which mapping file is used for which XML document is determined by the <identifier> element. The <identifier> element in the mapping 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 mapping file must also be ″doc″.

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

If you want to extract information that is only contained 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 quantities of content that are not subject to analysis, for example, binary attachments. Using content extraction reduces the document size significantly, speeding up processing and avoiding analysis errors that start 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.

Using native XML search and the content extraction options in the XML elements to the common analysis structure mapping are mutually exclusive options, because either all content or only specified content can be considered. If you specify content extraction, native XML mapping is ignored. Without content extraction, you can have both XML elements to common analysis structure mapping and native XML search.

All the types and features that you use in your 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 by using the Component Descriptor Editor Eclipse plug-in. This plug-in allows you to create a descriptor file without needing to know about the necessary XML syntax.

After you have built and tested the custom analysis, use the UIMA PEAR (Processing Engine ARchive) generation wizard to create an archive that contains the custom analysis files including the type system description. Then, you can upload the custom analysis archive and your XML elements to the common analysis structure mapping files into enterprise search by using the administration console for enterprise search.

Creating an XML elements to the common analysis structure mapping file

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

About this task

The XML to the common analysis structure mapping file is shown in the following example.

The sample police report has XML tags for the crime type, crime date, crime location, reporting officer, the police precinct where the officer is employed, suspect description, and abstract. This is followed by a body section. For example:

```
<report>
 <doc>
  <crimeType>Car theft</crimeType>
  <crimeDate>04/23/05 09:23 pm</crimeDate>
  <crimeLocation>27 Main Street, Brynston, Springfield, New Jersey</crimeLocation>
  <reportingOfficer rank="Lt">Jakob
   <lastName>Collins</lastName>
  </reportingOfficer>
  <policePrecinct>14th Precinct</policePrecinct>
  <suspectDescription>Male, dark haired, dark glasses,
    blue jeans with dark, probably black,
    jacket</suspectDescription>
  <abstract>A Mercedes CLK was stolen on 04/23/2005 from a parking
   lot in front of the Blue Lagoon restaurant on
   27 Main Street, Brynston.(serial number: 32 2761 50871)</abstract>
  <body>A Mercedes CLK was stolen on 04/23/2004 from a parking
   lot in front of the Blue Lagoon restaurant on 27 Main Street,
   Brynston.(serial number: 32 2761 50871)

It has a black color and wide Michelin tires.

Eyewitnesses in front of the restaurant saw two darkly dressed
males drive away in the car at high speed. The car was
found abandoned on Aliway Ave in Brooklyn. The fuel tank was empty.
The seats were badly stained and the back seat was vandalized.
Nothing was stolen out of the car....</body>
 </doc>
 <image>
  <--! image of the crime scene as a base64-encoded string -->
 </image>
</report>
```

Based on the sample report, an XML to the common analysis structure mapping file might have the following structure. The sample uses the type system that is defined for the police report scenario.

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

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

 <contentElements>
   <element>/report/doc</element>
 </contentElements>

 <elementToTypeMappings>
   <elementToTypeMapping>
     <element>//doc//reportingOfficer</element>
     <type>com.ibm.omnifind.types.Person</type>
     <featureValueAssignment>
       <feature>role</feature>
       <basicValue default="Reporting officer">
           </basicValue>
     </featureValueAssignment>
     <featureValueAssignment>
       <feature>gender</feature>
       <basicValue default="male"
          useAttributeValue="sex"/>
     </featureValueAssignment>
     <featureValueAssignment>
       <feature>surName</feature>
       <values concatenate="true" delimiter=" ">
         <basicValue useAttributeValue="rank"
                 default="Lt"/>
         <basicValue useElementContent="lastName"/>
       </values>
     </featureValueAssignment>
   </elementToTypeMapping>
   <elementToTypeMapping>
     <element>//doc</element>
     <type>com.ibm.omnifind.types.PoliceReport</type>
     <featureValueAssignment>
       <feature>crimeDescription</feature>
       <basicValue useElementContent="abstract"
                trim="true">
           </basicValue>
     </featureValueAssignment>
   </elementToTypeMapping>
 </elementToTypeMappings>

</xmlCasInitializerConfiguration>
```

Restrictions

The mapping file is split into two sections:

<contentElements> element
: Use this element if you want specific content extraction. The sample mapping file extracts the content in the <doc> section of a document and ignores other sections in the document. In the XML police report, the image might be large and not very useful for text processing. By specifying <doc> as a content element and not <image>, the image is filtered out before any text processing begins.

<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, the XML elements that are specified in the <elementToTypeMappings> section must be contained within the XML elements that are specified in the <contentElements> section.

Procedure

To create an XML to the common analysis structure mapping file:

1. Create an XML file. To avoid XML syntax errors, use an XML editor or XML authoring tool to validate the XML. The XSD schema for the mapping file is called XMLCasInitSchema.xsd and is contained in your enterprise search installation at *ES_INSTALL_ROOT*/packages/uima/configuration_xsd/.
2. Include your mappings in an <xmlCasInitializerConfiguration xmlns="http://www.ibm.com/2005/uima/jedii_ci_xml"> element. The namespace (specified in the xmlns attribute) must be exactly as shown.
3. Add a <contentElements> element if you want to extract specific content from sections in the document and a <elementToTypeMappings> element that specifies which individual XML elements in the document you want to map to which feature structures in the common analysis area.
4. Add an <identifier> element and a <description> element. The identifier determines which mapping to use for which XML document. The identifier must contain the root element of the document, such as doc. If the identifier is set to Default, the root element of the document is irrelevant and the mapping is applied to any XML document.
5. Add a <contentElements> element if you want to extract information that is contained only in relevant parts of a document. It has the following component element:
   - One or more <element> elements that contain the path of an XML element in the document and follows XPath syntax, for example <element>/doc/crimeType</element>.
6. Add an <elementToTypeMappings> element if you want to specify which XML elements in the document to map to which feature structures in the common analysis structure. It has the following component elements:
   - One or more <elementToTypeMapping> elements. This element must have the following nested elements:
     - An <element> element that is used to specify the path of an XML element and follows XPath syntax: A leading forward slash (/) means that a full path is given. For example, abstract under the root element doc. Two forward slashes (//) means any path subset. For example, birthDate must occur within reportingOfficer, although other elements can occur between these two.
     - A <type> element, which specifies a type that is defined in the type system description. It must be of type Annotation.
     - Zero or more <featureValueAssignment> elements.
7. In a <featureValueAssignment> element, name a feature of type String in the <feature> element and assign a value in the <basicValue> element. Multiple <basicValue> elements can be added between a <values> element.

   The <basicValue> element can have attributes. These include useAttributeValue, useElementContent, default, and trim.

Use useAttributeValue if you want to use the value of an attribute as the value for a feature. The following example

```
<elementToTypeMapping>
  <element>/doc//reportingOfficer</element>
  <type>com.ibm.omnifind.types.Person</type>
  <featureValueAssignment>
    <feature>role</feature>
    <basicValue default="Reporting officer"/>
  </featureValueAssignment>
  <featureValueAssignment>
    <feature>gender</feature>
    <basicValue default="male" useAttributeValue="sex"/>
  </featureValueAssignment>
</elementToTypeMapping>
```

results in the following output:

- For each <reportingOfficer> XML tag that occurs somewhere within a <doc> XML tag in the document, a feature structure of type com.ibm.omnifind.types.Person is created.
- If the <reportingOfficer> tag contains an attribute sex, the feature gender of the newly created feature structure is set to the value of the attribute.

Use the attribute useElementContent to add content as the value of a feature. For example, in the following mapping snippet:

```
<elementToTypeMapping>
  <element>//doc</element>
  <type>com.ibm.omnifind.types.PoliceReport</type>
  <featureValueAssignment>
    <feature>crimeDescription</feature>
    <basicValue useElementContent="abstract" trim="true"/>
  </featureValueAssignment>
</elementToTypeMapping>
```

the text covered by the element <abstract> in <doc> becomes the value of the feature structure crimeDescription. All leading and trailing blanks are removed.

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

- The feature to be set is of type StringArray.
- Many strings are concatenated to one string by using the delimiter attribute 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>//doc//reportingOfficer</element>
  <type>com.ibm.omnifind.types.Person</type>
  <featureValueAssignment>
    <feature>surName</feature>
    <values concatenate="true" delimiter=" ">
      <basicValue default="Mr."/>
      <basicValue useAttributeValue="rank"
         default="Lt."/>
      <basicValue useElementContent="lastName"/>
    </values>
  </featureValueAssignment>
</elementToTypeMapping>
```

String feature values are extracted from the mapping file as is. The values retain any leading or trailing blanks. However, names of types and features are trimmed of any blanks. For example, <type> **com.ibm.omnifind.types.Person** </type> becomes <type>**com.ibm.omnifind.types.Person**</type>.

Set conditions on attributes by using the <condition> element. For example, the feature structure of type com.ibm.omnifind.types.Person is created only if <suspectDescription> occurs in the document with attribute armed set to yes:

```
<elementToTypeMapping>
  <element>//suspectDescription</element>
  <type>com.ibm.omnifind.types.Person</type>
  <condition attribute="armed"  value="yes"/>
</elementToTypeMapping>
```

Based on the sample police report and the defined mapping file, the following feature structures are created:

com.ibm.omnifind.types.PoliceReport

- covered text: ″Car theft 04/23/05 09:23 pm 27 Main Street, Brynston, Springfield, New Jersey Jakob Collins 14th Precinct Male, dark haired, dark glasses, blue jeans with dark, probably black, jacket A Mercedes CLK was ... Nothing was stolen out of the car.
- begin = 2
- end = 904
- knownSuspects = null
- crimeDescription = ″A Mercedes CLK was stolen on 04/23/2005 from a parking lot in front of the Blue Lagoon restaurant on 27 Main Street, Brynston.(serial number: 32 2761 50871)″

com.ibm.omnifind.types.Person

- covered text = ″Jakob Collins″
- begin = 112
- end = 127
- role = ″Reporting officer″
- firstName = null
- surName = ″Lt Collins″
- gender = ″male″

After you create the mapping file, you must upload it to enterprise search and select the XML to the common analysis structure mapping file with your other custom analysis selections by using the enterprise search administration console.

Related reference

“Type system description sample” on page 22

The text analysis results

All text analysis results are stored in the common analysis structure.

Annotators typically read from and write to the common analysis structure. Common analysis structure consumers (*CAS consumers*) only read from the common analysis structure. CAS consumers do the final processing on the analysis results that are stored in the common analysis structure. Enterprise search contains two CAS consumers:

- The consumer that indexes the contents of the common analysis structure in a search engine. This consumer requires a common analysis structure to index mapping file that you select with the custom text analysis on the enterprise search administration console.

- The consumer that populates a relational database with specific analysis results. This consumer also requires a common analysis structure to database mapping file that you select with the custom text analysis options on the enterprise search administration console.

If required, you can deploy custom CAS consumers in enterprise search. See the UIMA documentation about how to write a consumer. To learn how to upload and use your consumer in enterprise search, see the IBM UIMA developerWorks Web site at http://www.ibm.com/developerworks/db2/zones/db2ii/.

Related concepts

"Index mapping for custom analysis results" on page 36

"Database mapping for selected analysis results" on page 43

Feature paths

A feature path provides a way to access feature values in the common analysis structures, similar to XPath statements that are used to access XML elements in an XML document.

Feature paths are useful if you want to access a feature structure that combines complex features, for example features that are array valued or point to another feature structure. Using a feature path, you can associate the value of a feature directly with a feature structure, and store this value in the semantic search index or in a database.

For example, consider an annotator that identifies cars and their makes. 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″, by using ′/car[@make=″Chevrolet″]′.

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

- The value of a feature path can be String, Integer, Float, or an array of one of those types.
- All features within the path from f1 to fn-1 must have a complex type, that is, of type `uima.cas.TOP`, `uima.cas.FSArray`, `uima.cas.FSList`, or of one of their subtypes.
- The last feature fn in the path can include a complex type. Additionally, it can include a (sub-)type of `uima.cas.Float`, `uima.cas.Integer`, `uima.cas.String`, `uima.cas.FloatArray`, `uima.cas.IntegerArray`, `uima.cas.StringArray`, `uima.cas.FloatList`, `uima.cas.IntegerList`, or `uima.cas.StringList`.
- 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` .

You can 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 of type `EmployeeInfo` which has the feature `salary`, you can access this feature using `additionalInfo/EmployeeInfo:salary`.

Note that in this example, the feature path `additionalInfo/salary` would result in an error, as `salary` has not been defined for the type `uima.cas.TOP`.

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

- Use brackets ([<number>]) to select a certain element in the array or list. An array starts at zero (0). For example, to select the first element in the companies array, use companies[0]. The special marker [last] can be used to select the last entry in an array, irrespective of its size, for example, companies[last].
- Use empty brackets ([]) to denote all elements. Only one empty bracket ([]) is allowed in a feature path. For example, if there is an array of suspects, the feature path `knownSuspects[]/com.ibm.omnifind.types.Suspect:surName` collects all the last names of suspects into a String array.
- When a feature path that returns an array is used during indexing, the array elements are concatenated (separated by white-spaces) and written to the index as a single, multi-term attribute or field.
- 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[2]/Company:profit`.

Built-in features

Built-in features are predefined feature names with special semantics. They can be used to access information that is not contained in the feature structure itself, for example, the type of the feature structure or the covered text of an annotation. They can be used in a feature path as the last, or sole element.

The following built-in features can be used in both mapping configuration files:

- `fsId()` returns the ID of the feature structure. The returned ID is an integer (32 bit). Use this built-in feature to access parts of a document that match the query exactly.
- `typeName()` returns the common analysis structure object type as a string. The type is the fully qualified type name including any namespace prefixes, for example `uima.tcas.Annotation`. In a database context, `typeName()` is especially useful if you store types and subtypes in the same column and want to know a the real type of an annotation or feature structure. The following example stores the person type, such as *suspect* or *witness*, in the role column.

```
<explicitMappingRule  applyToSubTypes="false">
   <type>com.ibm.omnifind.types.Person</type>
   <table>sample.person</table>
   <featureMappings>
     <featureMapping>
       <feature>typeName()</feature>
       <column>role</column>
     </featureMapping>
   </featureMappings>
 </explicitMappingRule>
```

- `coveredText()` returns the text that is spanned by the common analysis object. `coveredText()` is available only for annotations and their subtypes. Do not use this built-in feature on feature structures that are not subsumed by the annotation type. The following example stores the name of a suspect in the suspectName column.

```
<implicitMappingRule applyToSubTypes="false">
   <type>com.ibm.omnifind.types.Suspect</type>
   <relation>sample.person</relation>
   <featureMappings>
     <featureMapping>
       <feature>coveredText()</feature>
       <column>suspectName</column>
       <length>128</length>
     </featureMapping>
   </featureMappings>
 </implicitMappingRule>
```

- [] returns a handle to the current container entry (array or list). The feature implies an iteration, which means that an entry is made in the database table or index for each element in the array or list. The following example is taken from a common analysis structure to database mapping file in which the built-in function [:index] is also permitted.

```
<implicitMappingRule applyToSubTypes="false">
   <type>uima.cas.FSArray</type>
   <table>sample.knownSuspects</table>
   <featureMappings>
     <featureMapping>
       <feature>uniqueId()</feature>
       <column>arrayId</column>
     </featureMapping>
     <featureMapping>
       <feature>[:index]</feature>
       <column>arrayIndex</column>
     </featureMapping>
     <featureMapping>
       <feature>[]/com.ibm.omnifind.types.Suspect:uniqueId()</feature>
       <column>suspectId</column>
     </featureMapping>
   </featureMappings>
 </implicitMappingRule>
```

The following built-in features can be used only in the common analysis structure to database mapping file:

- `uniqueId()` returns the global unique ID of the feature structure. The returned unique ID is a string of fixed length (27 characters) and is a concatenation of the result of `fsId()`, `docId()`, `docTimestamp()`, and the number of the current chunk because documents can be chunked into multiple common analysis structures in enterprise search.

  The returned string can include any characters between ″a-z″ and ″A-Z″, the numbers ″0-9″, semicolon (″;″), and colon (″:″).

  The result of `uniqueId()` can be used as the primary key for tables.
- `objectId()` returns the ID of the annotation or feature structure. `objectId()` is similar to `uniqueId()`, only it does not contain the result of `docTimestamp()`. The returned ID is unique only in a collection in which documents are parsed once. If you require uniqueness across all documents and document versions, you must use `uniqueId()`.

  The returned string of the built-in feature `objectId()` has a fixed length of 16 characters and can include any characters between ″a-z″ and ″A-Z″, the numbers ″0-9″, semicolon (″;″), and colon (″:″).

  If `uniqueId()` or `objectId()` reference feature structures that are empty, the default value defined in the database table definition is taken, no empty objects of a referenced type are stored.
- `docId()` returns the document ID. The returned value is of type integer (32-bit).

  The following example shows these built-in features:

```
<explicitMappingRule applyToSubTypes="true">
   <type>com.ibm.omnifind.types.PoliceReport</type>
   <table>sample.PoliceReport</table>
   <featureMappings>
     <featureMapping>
       <feature>uniqueId()</feature>
       <column>policeReportId</column>
     </featureMapping>
     <featureMapping>
       <feature>docId()</feature>
       <column>policeReportDocId</column>
     </featureMapping>
   </featureMappings>
 </explicitMappingRule>
```

- `docUri()` returns the document URI.
- `docTimestamp()` returns the time (in milliseconds) when the document was processed. This built-in feature is useful for tracking document versions, for example, if you want to know whether the document version that you are using is the latest passed by the crawler.

```
<explicitMappingRule applyToSubTypes="false">
   <type>com.ibm.omnifind.types.PoliceReport</type>
   <relation>sample.PoliceReport</relation>
   <featureMappings>
     <featureMapping>
       <feature>uniqueId()</feature>
       <column>policeReportId</column>
     </featureMapping>
     <featureMapping>
       <feature>docTimestamp()</feature>
       <column>reportVersion</column>
     </featureMapping>
   </featureMappings>
 </explicitMappingRule>
```

- `parentId()` returns the `fsId()` of the feature structure that encloses a container mapping. `parentId()` is valid only within the context of a container mapping.
- `uniqueParentId()` returns the `uniqueId()` of the annotation or feature structure that is enclosed in a container mapping. This built-in feature is also valid only within the context of a container mapping.
- `[:index]` returns the index of the current container entry (array or list).

Related tasks

"Retrieving parts of a document that match a semantic search query" on page 53

Filters

Filters are used to restrict mapping rules in the common analysis structure to index mapping files and the common analysis structure to database mapping files. Only if the filter is true are analysis results added to the index or to a JDBC table.

The `<filter>` element is optional and is used to restrict mappings only to features that have a certain attribute value. This is useful if you want an attribute to act as a switch for what to index or add to the database. 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 to the mapping rule:

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

Each filter expression takes the form:

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

where:

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

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

The following examples are valid filters:

- `<filter syntax="FeatureValue"> foo = "hello world" </filter>`

  The feature foo contains the string hello world.
- `<filter syntax="FeatureValue"> foo < 42 </filter>`

  The feature foo has an integer value smaller than 42.
- `<filter syntax="FeatureValue"> make/company = "Chevrolet" </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"> bar7 >= 0.5 </filter>`

  The feature bar7 has a float value larger or equal to 0.5.

Index mapping for custom analysis results

After you 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 that is stored in the common analysis structure that is created 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 the common analysis structure to index mapping file, you can determine which analysis results in the common analysis structure that you want to index.

You can use different styles 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 by using the annotation style, all annotations of the specified types are stored in the index as searchable spans.

For example, if a feature structure that spans a certain area of text is of type `person` and is indexed by 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/>

Table 2. Sample queries (continued)

Required information	Possible query
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 by 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 use the following query: ″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.

Analysis results can be used also to affect the document ranking in enterprise search, even for simple keyword queries. This is done in two steps:

1. Map feature structures to searchable spans or fields, using the Annotation or Field mapping style.
2. Define a boost class by using the enterprise search administration console and map the span or field name to this boost class.

If the user enters a search term that is contained within the feature structure, the document is ranked higher. For example, consider an annotator that detects person and company names. By mapping these feature structures to spans (such as ″person″ and ″company″) and then mapping these spans to boost classes, the search result for ″gap″ will rank documents higher that talk about the company ″Gap″ than those that merely contain the term ″gap″.

After you write the common analysis structure to index mapping file, you can upload it to enterprise search by using the administration console.

Creating the common analysis structure to index mapping file

Using the common analysis structure to index mapping file, you can determine which analysis results in the common analysis structure you want to index to enable search.

About this task

The common analysis structure to index mapping file is in XML. The sample common analysis structure to index mapping file is based on the type system defined for the police report scenario.

```
<?xml version="1.0" encoding="UTF-8"?>
<indexBuildSpecification
xmlns="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>
   <name>com.ibm.omnifind.types.Person</name>
   <indexRule>
     <style name="Annotation">
       <attributemappings>
         <mapping>
           <feature>role</feature>
           <indexName>role</indexName>
         </mapping>
         <mapping>
           <feature>title</feature>
           <indexName>title</indexName>
         </mapping>
         <mapping>
           <feature>gender</feature>
           <indexName>gender</indexName>
         </mapping>
       </attributemappings>
     </style>
   </indexRule>
 </indexBuildItem>
 <indexBuildItem>
   <name>com.ibm.omnifind.types.Suspect</name>
   <indexRule>
     <style name="Annotation"/>
     <style name="Field">
       <attribute name="parametric" value="false"/>
       <attribute name="fieldSearchable"
         value="true"/>
       <attribute name="returnable" value="true"/>
     </style>
   </indexRule>
 </indexBuildItem>
 <indexBuildItem>
   <name>com.ibm.omnifind.types.City</name>
   <indexRule>
       <style name="Annotation">
       <attributemappings>
         <mapping>
           <feature>cityDistrict</feature>
           <indexName>district</indexName>
         </mapping>
       </attributemappings>
     </style>
   </indexRule>
 </indexBuildItem>
 <indexBuildItem>
   <name>com.ibm.omnifind.types.Date</name>
   <indexRule>
     <style name="Field">
       <attribute name="fixedName" value="Date"/>
       <attribute name="fieldSearchable"
           value="true"/>
       <attribute name="returnable" value="true"/>
     </style>
```

```
    <style name="Field">
      <attribute name="fixedName" value="hour"/>
      <attribute name="valueFeature"  value="hour"/>
      <attribute name="parametric" value="true"/>
    </style>
  </indexRule>
  <filter syntax="FeatureValue">year="2005"</filter>
</indexBuildItem>
<indexBuildItem>
  <name>com.ibm.omnifind.types.PoliceReport</name>
  <indexRule>
    <style name="Annotation">
      <attribute name="fixedName"
         value="PoliceReport"/>
      <attributemappings>
        <mapping>
          <feature>crimeDescription</feature>
          <indexName>crimeDescription</indexName>
        </mapping>
        <mapping>
          <feature>time/coveredText()</feature>
          <indexName>time</indexName>
        </mapping>
        <mapping>
          <feature>date/englDate</feature>
          <indexName>date</indexName>
        </mapping>
        <mapping>
          <feature>location/coveredText()</feature>
          <indexName>location</indexName>
        </mapping>
        <mapping>
          <feature>knownSuspects[]/com.ibm.omnifind.types.Suspect:surName</feature>
          <indexName>suspectsLastNames</indexName>
        </mapping>
      </attributemappings>
    </style>
  </indexRule>
</indexBuildItem>
</indexBuildSpecification>
```

Restrictions

The common analysis structure to index mapping file must contain all of the analysis results that you want to be able to search in queries.

Procedure

To create the common analysis structure to index mapping file:

1. Create an XML file. To avoid XML syntax errors, use an XML editor or XML authoring tool of your choice. The XSD schema for the mapping file is called `CasToIndexMapping.xsd` and is contained in your enterprise search installation at *ES_INSTALL_ROOT*`/packages/uima/configuration_xsd/`.
2. Include your mappings in a `<indexBuildSpecification xmlns="http://www.ibm.com/of/822/consumer/index/xml">` element. The namespace (specified in the xmlns attribute) must be exactly as shown.
3. Add a `<skipCondition>` element to prohibit certain documents from being indexed, based on a certain feature value. This element is optional. 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.

4. Add one or more <indexBuildItem> elements that contains the mapping of one particular feature structure in the common analysis structure to a structure in the index.
5. Save and validate the XML file.

The <indexBuildItem> element

The common analysis structure to index mapping 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:

- The full type name. For example, com.ibm.omnifind.types.Suspect
- A wildcard. For example, com.ibm.omnifind.types.*. The wildcard character can be added only at the end of the type specification.

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

If type A is a subtype of type B (in the sample, com.ibm.omnifind.types.Suspect as a subtype to com.ibm.omnifind.types.Person), and there are <indexBuildItem> elements Ia and Ib defined for both types, processing is as follows:

- Each index rule that is defined in Ib is applied to feature structures of type B and feature structures of type A
- Each index rule that is defined in Ia is applied to feature structures of type A only

In the example, the <indexBuildItem> element that is defined for com.ibm.omnifind.types.Person annotations also applies to com.ibm.omnifind.types.Suspect annotations. Two spans are created for a suspect annotation: one named Person and the other Suspect.

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.

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, and a breaking style. The annotation and field styles support a number of attributes. You

cannot use the term style, which is supported in the UIMA Software Development Kit 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 you want each feature structure to be accessible in the index under the same name. In the following example, each feature structure of type `com.ibm.omnifind.types.Person` will be mapped to a span named ″Person″ in the index.

  ```
  <indexBuildItem>
     <name>com.ibm.omnifind.types.Person</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 by using XML fragments: `@xmlf2::'<Person>Boss</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, `com.ibm.tt.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 organization 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 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.

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.omnifind.types.Date</name>
   <indexRule>
     <style name="Field">
       <attribute name="fixedName" value="date"/>
       <attribute name="fieldSearchable"
           value="true"/>
       <attribute name="returnable" value="true"/>
     </style>
     <style name="Field">
       <attribute name="fixedName" value="hour"/>
       <attribute name="valueFeature"  value="hour"/>
       <attribute name="parametric" value="true"/>
     </style>
   </indexRule>
   <filter syntax="FeatureValue">year="2005"</filter>
</indexBuildItem>
```

two fields are generated for `com.ibm.omnifind.types.Date`. One field named `date` contains the covered text, for example, 5:15pm. Another field contains the value of the attribute `hour`. Here you can query using 'hour::<17'.

The `<style name="Breaking" />` element

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

After you create the XML file, you must upload it to enterprise search and select the common analysis structure to index mapping file with your other custom analysis selections using the enterprise search administration console.

Related concepts

"Feature paths" on page 32

Related reference

"Filters" on page 35

"Type system description sample" on page 22

Database mapping for selected analysis results

After you have analyzed your documents in enterprise search, you can store selected text analysis results in a JDBC-capable database.

This version supports DB2 Universal Database™, Version 8.2.2 (com.ibm.db2.jcc.DB2Driver Version 2.3) or higher and Oracle 10g (oracle.jdbc.driver.OracleDriver Version 1.0).

For DB2 Universal Database and Oracle, you can choose to insert the analysis results directly into the database or to generate the equivalent database-specific load files and the corresponding script that runs the load commands.

Mapping your analysis results to tables in a database enables you to use this information in subsequent business intelligence processing steps or to directly access the relevant parts of a document that match a semantic search query.

The common analysis structure to database mapping file contains database connection configuration information and describes which custom analysis results are to be stored in which tables and columns. The table and column names in your mapping file must correspond to the tables and columns that are created in the database.

After you have written the common analysis structure to database mapping file, you can upload the file to enterprise search by using the administration console.

Storing analysis results in a database

To store selected analysis results in a JDBC capable database, you must write the common analysis structure to database mapping file that defines which analysis results to store in a database, and the necessary JDBC driver libraries must be in the path that you defined in the mapping file.

To store analysis results in a JDBC capable database:

1. Decide which analysis results that you want to store in the database. Create a database that contains the tables with all the necessary columns of the appropriate data types.

2. In an XML editor, write the common analysis structure to database mapping file with the database configuration data and the analysis results that you want to store. To determine which analysis results to include in the mapping file, you must know the underlying type system that is used when the documents are processed.
3. Put the JDBC driver libraries in a directory on the indexer node where they can be accessed by the enterprise search system.
4. Upload and select the mapping file by using the enterprise search administration console.

Using load file sets

You can either store analysis results directly in a JDBC-capable database, or you can configure processing to use load file sets and load the data into a database at a later stage.

Using load file sets has the following advantages:

- In total, a set of load files can never be larger than the maximum file size supported by the operating system
- You can start loading data into a database as soon as a load file set is full, and do not have to stop and restart the document parser to avoid file access conflictions

Switching from one load file set to the next one is done on a document level even if the document is chunked across multiple common analysis structures. After a document has been processed, and if a load file in the current load file set exceeds the defined limit, a new load file set is used. This guarantees load file set consistency. After the content of one load file set is loaded into the database, the data model remains consistent because all entries in the master table contain the matching entries in the database table.

The load files and script files are identified by the file extension .cur. When a load file set is closed, the files are renamed to have the extension .dat. This indicates that the files can be copied or moved to a database server while the document parser is still running.

You can specify the size of a load file. When the load file size limit is reached, a new load file set is started. You specify the load file size in the common analysis structure to database mapping file in the `<loadFile>` XML element section. The parameter loadFileSize is defined using the `<loadFileSize>` element and is specified in megabytes with 10 <= loadFileSize <= 10240 (10MB <= loadFileSize <= 10GB). The `<loadFileSize>` element is optional. If no value is set, the default value is 1024MB (1GB).

The single load files in a set are numbered using a ten digit number that identifies which file belongs to which load file set. A load file set is closed when:

- A load file in the set exceeds the defined size limit
- Processing stopped because the parser stopped or an error occurred

If the parser is restarted, processing continues from where it stopped using a new load file set.

Important: If you use Cas2Jdbc to generate load files, ensure that only one parser thread is configured. Using multiple parser threads for a collection that is configured to generate Cas2Jdbc load files can result in invalid load files. To

specify how many parser threads will be used, use the enterprise search administration console to edit a collection. Select the Parse page, select the option to configure parsing options, and then specify 1 for the number of parser threads.

Creating the common analysis structure to database mapping file

To add analysis results to a database, you must create the common analysis structure to database mapping file that contains the database connection configuration information and a description of which custom text analysis results are to be stored in which database tables and columns.

About this task

The common analysis structure to database mapping file is in XML. The following sample is based on the type system that is defined for the police report scenario.

In the example, only the police reports and the cities that appear in these police crime reports are added to the database. The example shows the use of built-in features and the <constant> element mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
<cas2JdbcConfiguration xmlns="http://www.ibm.com/uima/consumer/jdbc/100/xml">
   <databaseConnection>
     <connectionUrl>db2://myMachine:myPort/myDatabase</connectionUrl>
     <driver type="jdbc">com.ibm.db2.jcc.DB2Driver</driver>

     <driverLibraries>
       <driverLibrary>C:\db2\db2jcc.jar</driverLibrary>
       <driverLibrary>C:\db2\db2jcc_license_cu.jar</driverLibrary>
       <driverLibrary>C:\db2\db2jcc_license_cisuz.jar</driverLibrary>
     </driverLibraries>

     <authentication>
       <username>myUser</username>
       <password>myPassword</password>
     </authentication>

     <loadFile>
       <loadFileDirectory>/home/cas2jdbc/load/</loadFileDirectory>
       <loadFileSize>1048</loadFileSize>
       <loadScript>/home/cas2jdbc/load/load.sh</loadScript>
     </loadFile>

   </databaseConnection>

   <cas2JdbcMappingSpec>
     <skipCondition>
       <name>com.ibm.uima.tt.DocumentAnnotation</name>
       <filter syntax="FeatureValue">toBeProcessed=0</filter>
     </skipCondition>

     <cas2JdbcMappings>
      <explicitMappings>
       <explicitMappingRule applyToSubtypes="false">
        <type>com.ibm.omnifind.types.PoliceReport</type>
        <table>sample.policeReport</table>
        <featureMappings>
         <featureMapping>
          <feature>uniqueId()</feature>
          <column>policeReportId</column>
         </featureMapping>
         <featureMapping>
          <feature>location/uniqueId()</feature>
```

```
          <column>crimeLocationId</column>
         </featureMapping>
        </featureMappings>
        <filter syntax="FeatureValue">location/coveredText()="Los Angeles"</filter>
       </explicitMappingRule>
      </explicitMappings>

      <implicitMappings>
       <implicitMappingRule applyToSubtypes="false">
        <type>com.ibm.omnifind.types.City</type>
         <table>sample.City</table>
        <featureMappings>
         <featureMapping>
          <feature>uniqueId()</feature>
          <column>crimeLocationId</column>
         </featureMapping>
         <featureMapping>
          <feature>coveredText()</feature>
          <column>cityName</column>
          <length>150</length>
         </featureMapping>
         <featureMapping>
          <constant>USA</constant>
          <column>country</column>
         </featureMapping>
        </featureMappings>
       </implicitMappingRule>
      </implicitMappings>

     </cas2JdbcMappings>
    </cas2JdbcMappingSpec>
</cas2JdbcConfiguration>
```

Procedure

To create the common analysis structure to database mapping file:

1. Create an XML file. To avoid XML syntax errors, use an XML editor or XML authoring tool of your choice. The XSD schema for the mapping file is called `CasToJDBCMapping.xsd` and is contained in your enterprise search installation at *ES_INSTALL_ROOT*`/packages/uima/configuration_xsd/`.
2. Include your mappings in a `<cas2JdbcConfiguration xmlns="http://www.ibm.com/uima/consumer/jdbc/100/xml">` element. The namespace (specified in the xmlns attribute) must be exactly as shown.
3. Add a `<databaseConnection>` element that contains all database connection configuration information and a `<cas2JdbcMappingSpec>` element that describes the mapping rules for the analysis results that are stored in the database or load files.
4. Add the following component elements to the `<databaseConnection>` element:
   - Mandatory: A `<connectionUrl>` element. This element contains the database connection URL. Depending on the JDBC driver implementation, you can use local or remote access to the database.
   - Mandatory: A `<driver>` element. This element contains the name of the JDBC driver class, for example `com.ibm.db2.jcc.DB2Driver` for DB2®, or `oracle.jdbc.driver.OracleDriver` for Oracle.
   - Mandatory: A `<driverLibraries>` element. This element lists the driver libraries. Each library is listed in a `<driverLibrary>` element. The libraries are in your DB2 or Oracle installation directory. For DB2, the libraries are `c:\your_db2_dir\db2jcc.jar`, `c:\your_db2_dir\db2jcc_license_cu.jar` and

c:\your_db2_dir\db2jcc_license_cisuz.jar. For Oracle, the library to include is c:\your_oracle_dir\classes12.zip.

Ensure that the driver libraries are always at the same maintenance level as the DB2 applet server.

- Mandatory: An <authentication> element. This element contains the user name and password for the database.
- Optional: A <loadFile> element. This element contains the following component elements:
  - The load file directory in a <loadFileDirectory> element.
  - Optional: The load file size in a <loadFileSize> element. The load file size limits are 10 <= loadFileSize <= 10240 (10MB <= loadFileSize <= 10GB). If no value is defined, the default is 1024 MB (1GB).
  - The load script name in a <loadScript> element.

  If you do not specify a <loadFile> element, all data is stored directly in the database by using JDBC.

  You must also add all of the database configuration parameters when you use database-specific load files and scripts.

5. Add the following component elements to the <jdbcMappingSpec> element:
   - Optional: A <skipCondition> element. If no skip condition is defined, all of the documents are processed.

     ```
     <skipCondition>
       <name>com.ibm.uima.tt.DocumentAnnotation</name>
       <filter syntax="FeatureValue">toBeProcessed=0</filter>
     </skipCondition>
     ```

     In the example, documents that contain an annotation of type com.ibm.uima.tt.DocumentAnnotation with a feature named toBeProcessed set to zero will not be considered.
   - A <cas2JdbcMappings> element that shows which types and features are mapped to which database tables and columns. The element contains an explicit and an implicit mappings section.
6. Add an <explicitMappings> element. This element is mandatory. It must have one or more <explicitMappingRule> elements that define the explicit mappings and can only be defined for annotation types and their subtypes. If a mapping is defined in the explicit mappings section, all of the annotations that match the mapping definition will be stored in the database.
7. Optional: Add an <implicitMappings> element. This element supports all feature structure types. If this element is present, it must contain at least one <implicitMappingRule> element. Mappings that are defined in the implicit mappings section are added to the database only if the matching annotation types are referenced by another annotation that can match either an explicit or an implicit mapping rule.

   The purpose of implicit mapping is to enable you to store only analysis results which appear in a particular context. For example, if the mapping for an annotation of type com.ibm.omnifind.types.City is implicit, only cities that are referenced by the com.ibm.omnifind.types.PoliceReport mapping definition in the explicit mappings section are stored in the database. This means that only cities mentioned in police reports are added to the database.

   If there is an explicit mapping rule for the City annotation, all cities are added to the database. In both cases, if a city is referenced by multiple police reports, it is added to the database only once.

8. The <explicitMappingRule> and <implicitMappingRule> elements must contain the attribute applyToSubtypes, which, if set to true, stores not only the feature structure that is listed in the <type> element, but also all of the feature structures derived from it. Add the following component elements to <explicitMappingRule> and <implicitMappingRule> elements:
   - A <type> element that contains the feature structure type.
   - A <table> element that contains the database schema and table name. The syntax follows the rule schema.table_name, or only table_name if no schema is defined.
   - A <featureMappings> element with one or more <featureMapping> elements or one <containerMapping> element.
   - Optional: An <filter> element that contains a condition that is evaluated each time the mapping rule matches. If the condition evaluates to true, the annotation or feature structure is stored in the database. In the example, only police reports that deal with crimes committed in Los Angeles are stored in the database.
9. The <featureMapping> element component structure varies depending on whether you are mapping a feature or a constant.

   If you are mapping a feature or feature path, the component elements include:
   - A <feature> element with the name of the feature. The feature must be defined for the feature structure in the type element. You can also use a feature path construct or any of the system defined built-in features.
   - Optional: A <length> element with the length a string can have in the specified database column. Longer strings are truncated.
   - A <column> element with the name of the column in which the feature value is to be stored. Database columns that are not used in any feature mappings use a default value (usually null) that is configured in the database.

     Make sure that the value of the feature element is stored in a column of the appropriate type. The following table shows you which UIMA types match which database types.

Table 3. Mapping between UIMA types and corresponding database types

UIMA type or built-in feature	Recommended DB2 data type	Recommended Oracle data type
Float	REAL	FLOAT
String	VARCHAR	VARCHAR2
Integer	INTEGER	INTEGER
uniqueId(), uniqueParentId()	CHAR(27)	CHAR(27)
objectId(), parentId()	CHAR(16)	CHAR(16)
docTimestamp()	BIGINT	LONG
fsId()	INTEGER	INTEGER

   For a constant, the component feature mapping elements are as follows:
   - A <constant> element that contains the value of a constant.
   - A <column> element with the name of the column to which the constant value is added.
10. The <containerMapping> element contains the mapping for a container type feature (array or list). This element must be used only for container types. It has the following component elements:

- A <feature> element with the name of the feature. You can also use a feature path construct or any of the system defined built-in features.
- A <table> element that contains the database schema and table name. The syntax follows the rule schema.table_name or only table_name if no schema is defined.
- One or more <featureMapping> elements that contain the names of the feature structures and the column names to which the features are added.

11. Save and validate the XML file by using the provided schema.

After you create the XML file, you must upload it to enterprise search and select the common analysis structure to database mapping file with your other custom analysis selections by using the enterprise search administration console.

Related concepts

"Feature paths" on page 32

Related reference

"Filters" on page 35

"Built-in features" on page 33

"Type system description sample" on page 22

Container type mapping

A container type is one of the built-in array or list types in the common analysis structure. Container type mapping is a way of mapping array or list values to a relational database.

There are two approaches for handling container types in the common analysis structure to database mapping file. One method uses the defined built-in feature constructs and a generic link table which contains arrays or lists that are values of a feature mapping rule. As different arrays or lists are stored in the same link table, the table says nothing about the relation of the stored information.

In the second method, the link table definition which is defined using a <containerMapping> element explicitly denotes the relation between the specified information you want to have.

An example of what a generic link table mapping could look like is as follows. There is a n:m relation between police reports and suspect persons, meaning one suspect person can be mentioned in more than one police report, and one police report can mention more than one suspect.

The generic sample.fsarray table in the example is the link table between police reports and suspect persons. If another mapping type exists besides com.ibm.omnifind.types.PoliceReport that has a feature of type com.ibm.omnifind.types.FSArray, it is also mapped to this table. You can still query the table for the relation between a police report and a suspect correctly, however you cannot conclude, by simply looking at the table, that it contains the relation or link between police reports and possible suspects.

```
<cas2JdbcMappings>
  <explicitMappings>
    <explicitMappingRule applyToSubtypes="false">
      <type>com.ibm.omnifind.types.PoliceReport</type>
      <table>sample.policeReport</table>
      <featureMappings>
        <featureMapping>
          <feature>uniqueId()</feature>
```

```
        <column>policeReportId</column>
      </featureMapping>
      <featureMapping>
        <feature>knownSuspects/uniqueId()</feature>
        <column>suspectArrayId</column>
      </featureMapping>
      <featureMapping>
        <feature>location/cityName</feature>
        <column>city</column>
      </featureMapping>
    </featureMappings>
  </explicitMappingRule>
</explicitMappings>

<implicitMappings>
  <implicitMappingRule applyToSubtypes="false">
    <type>com.ibm.omnifind.types.Suspect</type>
    <table>sample.suspect</table>
    <featureMappings>
      <featureMapping>
        <feature>uniqueId()</feature>
        <column>suspectID</column>
      </featureMapping>
      <featureMapping>
        <feature>surName</feature>
        <column>lastName</column>
      </featureMapping>
      <featureMapping>
        <feature>description</feature>
        <column>description</column>
      </featureMapping>
    </featureMappings>
  </implicitMappingRule>

  <implicitMappingRule applyToSubtypes="false">
    <type>uima.cas.FSArray</type>
    <table>sample.fsarray</table>
    <featureMappings>
      <featureMapping>
        <feature>uniqueId()</feature>
        <column>arrayId</column>
      </featureMapping>
      <featureMapping>
        <feature>[:index]</feature>
        <column>arrayIndex</column>
      </featureMapping>
      <featureMapping>
        <feature>[]/uniqueId()</feature>
        <column>suspectId</column>
      </featureMapping>
    </featureMappings>
  </implicitMappingRule>
</implicitMappings>

</cas2JdbcMappings>
```

The following shows the database tables based on the generic mapping rules above.

Table 4. The sample.policeReport table

policeReportId	**suspectArrayId**	**city**
aaa...1	**bbb...1**	Springfield
aaa...2	**bbb...2**	Ladysmith

Table 5. The sample.fsarray table

arrayId	arrayIndex	suspectId
bbb...1	1	*ccc...1*
bbb...1	2	*ccc...2*
bbb...2	1	*ccc...3*

Table 6. The sample.suspect table

suspectID	lastname	description
ccc...1	Brown	Dark complexion
ccc...2	Smith	Wears glasses
...	...	...

The example shows the mapping for feature structure arrays. You can apply this type of mapping to StringArray, IntegerArray, and FloatArray too. If you include mapping rules for these simple valued arrays, replace `[]/uniqueId()` with `[]`.

The same generic table approach can be used for feature structures lists as well as simple typed lists (StringList, IntegerList and FloatList).

An easier way to handle relations is to use an explicit container mapping element which defines the iteration over the elements contained in the arrays or lists.

An example of a mapping that denotes an explicit link table is as follows. Again, there is a n:m relation between police reports and suspect persons. However, this time, the `sample.reports_suspects` table is the link table between police reports and suspect persons.

In this approach, you do not have to consider dealing with array IDs, or head and tail entry mapping for list types. The link table contains one explicit relation.

```
<cas2JdbcMappings>
  <explicitMappings>
    <explicitMappingRule applyToSubtypes="false">
      <type>com.ibm.omnifind.types.PoliceReport</type>
      <table>sample.policeReport</table>
      <featureMappings>
        <featureMapping>
          <feature>uniqueId()</feature>
          <column>policeReportID</column>
        </featureMapping>
        <featureMapping>
          <feature>location/cityName</feature>
          <column>city</column>
        </featureMapping>
        <featureMapping>
          <feature>knownSuspects</feature>
          <containerMapping>
            <table>sample.reports_suspects</table>
            <featureMapping>
              <feature>com.ibm.omnifind.types.PoliceReport
                /objectId()</feature>
              <column>policeReportId</column>
            </featureMapping>
            <featureMapping>
              <feature>knownSuspects/[]/objectId()</feature>
              <column>suspectId</column>
            </featureMapping>
```

```
        </containerMapping>
      </featureMapping>
    </featureMappings>
  </explicitMappingRule>
</explicitMappings>

<implicitMappings>
  <implicitMappingRule applyToSubtypes="false">
    <type>com.ibm.omnifind.types.Suspect</type>
    <table>sample.suspect</table>
    <featureMappings>
      <featureMapping>
        <feature>objectId()</feature>
        <column>suspectID</column>
      </featureMapping>
      <featureMapping>
        <feature>surName</feature>
        <column>lastName</column>
      </featureMapping>
      <featureMapping>
        <feature>description</feature>
        <column>description</column>
      </featureMapping>
    </featureMappings>
  </implicitMappingRule>
</implicitMappings>

</cas2JdbcMappings>
```

A `<containerMapping>` element is used to define the iteration over elements contained in the array. In the example, the `sample.reports_suspects` link table contains a link to the `policeReportId` and the `suspectId` columns. Do not nest `<containerMapping>` elements.

The following shows the database tables based on explicit link table mapping rules.

Table 7. The sample.policeReport table

policeReportId	**city**
aaa...1	Springfield
aaa...2	Ladysmith

Table 8. The sample.reports_suspect table

policeReportId	**suspectId**
bbb...1	*ccc...1*
bbb...2	*ccc...2*
...	...

Table 9. The sample.suspect table

suspectID	**lastname**	**description**
ccc...1	Brown	Dark complexion
ccc...2	Smith	Wears glasses
...	...	...

Related reference

"Built-in features" on page 33

Retrieving parts of a document that match a semantic search query

You can retrieve just the parts of a document that match the query exactly by mapping the relevant feature structures to both the index and database, and specifying the span in the semantic search query.

To access all instances of a specific annotation type in the search result, for example, to obtain all persons, include a field style mapping for the annotation type, and mark it as returnable in the common analysis structure to index mapping file. For example:

```
<indexBuildItem>
 <name>com.ibm.omnifind.types.Person</name>
 <indexRule>
  <style name="Annotation"/>
  <style name="Field">
   <attribute name="returnable" value="true"/>
  </style>
 </indexRule>
</indexBuildItem>
```

In this example, annotations of type `com.ibm.omnifind.types.Person` are mapped to a span named Person in the enterprise search index where they can be accessed during semantic search. Moreover, the covered text of the annotations, for example, the full person names, is stored as a returnable field. To retrieve these annotation values, call getFields(″Person″) on each result object that is returned from the search query (keyword or semantic). This method returns a String array with the annotation values, in this case, the person names.

However, this approach returns all instances of a given annotation type and is not suitable if you want to limit your result processing to documents that matched the query exactly. For example, a document might mention five persons. However, in the semantic search query '`<sentence><person/>IBM</sentence>` the user is interested only in the person who is mentioned in the same sentence that the term IBM appears in. The user is not interested in the other persons.

To access and process feature structures that match the query exactly:

1. Map the relevant feature structure types to the enterprise search index by using the annotation mapping style. For example:

   ```
   <indexBuildItem>
    <name>com.ibm.omnifind.types.Person</name>
    <indexRule>
     <style name="Annotation"/>
    </indexRule>
   </indexBuildItem>
   ```

2. Map the relevant feature structure types to JDBC tables. As part of the mapping, you must include two columns for the document URI and for the feature structure ID. Although you can map all feature structure types to the same database table, you should map each type to a different table. For example:

   ```
   <explicitMappingRule applyToSubtypes="false">
    <type>com.ibm.omnifind.types.Person</type>
    <table>sample.person</table>
    <featureMappings>
     <featureMapping>
      <feature>objectId()</feature>
      <column>primaryId</column>
     </featureMapping>
     <!-- Contains the covered text of the annotation-->
     <featureMapping>
   ```

```
      <feature>coveredText()</feature>
      <column>personName</column>
     </featureMapping>
     <!-- Other mapping go in here-->
     <!-- To access the relevant person annotations in the query result-->
     <featureMapping>
      <feature>docUri()</feature>
      <column>docUri</column>
     </featureMapping>
     <featureMapping>
      <feature>fsId()</feature>
      <column>annotationId</column>
     </featureMapping>
    </featureMappings>
   </explicitMappingRule>
```

3. Crawl, parse, and index the documents.
4. Retrieve the IDs of instances that match the query. In the search and index API (SIAPI), these instances are referred to as target elements. A target element specifies the input span to be returned. It is defined as follows:
   - In XML fragments, the target element is identified by a prepending number sign (#). The number sign is only allowed once and can appear anywhere in the XML fragment query. For example: `$xmlf2::'<sentence><#person/>IBM</sentence>'`
   - In XPath by default, the target element is the last field in the XPath expression.
   - Access these instances by using the method `Result.getProperty("TargetElement")`. The returned property is a string concatenation of all occurrence IDs that are separated by spaces. Each occurrence in the property can be translated into an integer value.
5. SIAPI does not return the feature structures themselves, only their occurrence IDs. These IDs correspond to the fsId() value that is stored in the database table. To retrieve these instances and their associated information, your application must:
   a. Select the right database table, depending on the span name of the target element. In the example, the application contains a mapping from person to the sample.Person table. This information is deduced from the common analysis structure to index mapping file, which yields the span name, and the common analysis structure to database mapping file, which yields the table name.
   b. For each result object in the search result:
      1) Parse the string that is returned by `Result.getProperty ("TargetElement")` to find the occurrence IDs.
      2) Issue a SELECT statement for the table by using the result URI (accessible by using `Result.getDocumentId()`) as the value in the docUri column and the occurrence IDs as the value in the annotationId column. The column names depend on your mapping file. The column names are taken from the previous example.

      The returned rows contain the information that is stored for the feature structure, for example, the covered text, or specific attributes of the feature structure, such as "last name" or "city of birth."

Ensure that updates to your database are synchronized with the index updates in enterprise search. If the database contains outdated information (for example, because you used database load files and you did not update the database, but you refreshed or reorganized the index), some occurrence IDs might not be found

in the database. Enterprise search keeps a record only of the last document version in its index. Hence, the occurrence IDs are valid only for the last document.

If you store multiple versions of the same document in the same database table, there might be multiple rows that match the same occurrence IDs, each for different versions of the document. In this case, you must define a document version column and populate it by using application logic or built-in features like docTimestamp(). This way, you can filter the result to obtain only the latest document version.

Related concepts

"Semantic search query term"

Related tasks

"Creating the common analysis structure to index mapping file" on page 37

"Creating the common analysis structure to database mapping file" on page 45

Semantic search applications

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

The four different types of information include:

- Text words that are found in a document, for example, a phrase such as *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 includes the semantic search query term. The term specifies a twig pattern. A twig is a small tree with leaves. Each leaf represents the four types of information (text words, span names, and so on). The internal nodes of the tree specify how their occurrence in a document relates to one another. There are five types of internal nodes that specify 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 by 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.

Semantic search query term

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 quotation marks ('...').

However, limited XPath query terms are prefixed by `@xmlxp::` followed by the XPath query that are 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.

Number sign (#)
Term is highlighted.

The following examples show XML fragment queries.

@xmlf2::'<City>Springfield</City>'
Finds documents that include the span (annotation) City containing the string Springfield.

@xmlf2::'<Person gender="female"/>'
Finds documents where a female person is annotated.

@xmlf2::'<Person><.or><@gender>female</@gender> <@title>Mrs</@title><@title>Ms</@title></.or></Person>'
Finds documents that specify a person as a women either by gender or title.

@xmlf2::'<Person gender="male" role="suspect"/> <PoliceReport><@crimeDescription><.or>robbery theft</.or>-accident </@crimeDescription></PoliceReport> <City>Springfield<.or> <@district>Brynston</@district><@district>Brooklyn</@district></.or></City>'
Finds documents that specify male persons who are considered as suspects and a PoliceReport annotation that is attributed by the string *robbery* or *theft* in attribute crimeDescription, but not the string *accident*. The documents must also contain a city annotation that covers the text word *Springfield*, an annotation that is attributed with the district *Brynston* or *Brooklyn*.

The corresponding XPath queries have the following structure:

@xmlxp::'//City ftcontains ("Springfield")'
Finds documents that include the span (annotation) City containing the string *Springfield*.

@xmlxp::'//PoliceReport[City ftcontains("Springfield")]'
Finds documents that include the span (annotation) City in the span PoliceReport containing the string *Springfield*.

@xmlxp::'//Person[@gender="female" or @title ftcontains("Ms") or @title ftcontains("Mrs")]'
Finds documents where a female person is annotated. In the gender attribute, the value must match exactly, whereas for the title attribute, *Ms* and *Mrs* need not be an exact match of the attribute value.

Synonym support in search applications

You can expand the search results by searching for documents that contain synonyms of the query terms.

Synonyms typically include multi word terms such as product names like *OmniFind Enterprise Edition*. Multi word terms that are contained in the synonym dictionary are correctly identified in user queries and do not have to appear within quotes.

The Search and Index API (SIAPI) for enterprise search supports several ways for users to search for synonyms of query terms:

- The SIAPI query syntax supports the tilde (~) operator for synonym expansion. If the user prepends this operator to a query term, synonym expansion is performed for the word. For example, the query `~WAS` returns documents that discuss WebSphere Application Server and any other synonyms that exist for this abbreviation.
- Synonym expansion can be enabled using the SIAPI synonym expansion interface from within a search application. Query terms can be automatically expanded to include synonyms, or the search application might include options that enable the user to specify whether synonyms of the query terms are to be returned in the search results.

  During automatic synonym expansion, synonym lookup is performed on all query words. The search results include documents that contain either the query terms or synonyms of the query terms. The SIAPI also supports the generation of a list of synonym expansions for the submitted query.
- Synonym expansion in n-gram collections allows for phrase segmentation of the query text. If an entire phrase appears in the synonym dictionary, then the search succeeds. A phrase is extracted according to these delimiters:

  Punctuation

  The following characters are delimiters: `- ( ) + . ,`

  Quotation marks are ignored and do not delimit phrases.

  Change in alphabet

  For example, for an n-gram collection, the query will be expanded to include the synonyms of `ABC` in the following sample queries if `ABC` is in the synonym dictionary:

  `ABC run DEF stand` (where `ABC` and `DEF` are Japanese text)
  `ABC+DCF+GHI`

Creating an XML file for synonyms

To expand queries in enterprise search to include synonyms of the query terms, you must specify which words qualify as synonyms of each other in an XML file. This XML file is used to build a binary dictionary file that you upload to enterprise search and assign to appropriate collections.

About this task

The XML file that lists the synonyms must comply with a specific schema. This is an example XML file for synonyms:

```
<?xml version="1.0" encoding="UTF-8"?>
<synonymgroups xmlns="http://www.ibm.com/of/822/synonym/xml">
  <synonymgroup>
    <synonym>Think Pad</synonym>
    <synonym>Notebook</synonym>
    <synonym>Notebooks</synonym>
  </synonymgroup>
  <synonymgroup>
    <synonym>WebSphere Application Server</synonym>
    <synonym>WAS</synonym>
  </synonymgroup>
</synonymgroups>
```

Restrictions

You must group words that are synonyms of each other (the <synonym> elements) in a <synonymgroup> element. A synonym can include white-space characters, but it cannot include punctuation characters, such as a comma (,) or vertical bar (|), because these characters might interfere with the enterprise search query syntax.

You must enumerate all possible inflections of the terms that you add as synonyms, such as the singular and plural forms of a word. You do not need to enumerate normalizations of the term, such as the removal of accents or umlauts (enterprise search handles normalization automatically), nor include upper and lower case variants of the term. For example, if you want to include the term météo as a synonym, you do not need to include the term METEO, too.

Procedure

To create a list of synonyms for enterprise search:

1. Create an XML file. To avoid XML syntax errors, use an XML editor or XML authoring tool of your choice. The XSD schema for the XML file is called synonyms.xsd and is contained in your enterprise search installation at *ES_INSTALL_ROOT*/packages/uima/configuration_xsd/.
2. Add a <synonymgroup> element, then insert a <synonym> element for each word that is to be treated as a synonym of other words in the synonym group.

   Be sure to include your mappings in a <synonymgroups xmlns="http://www.ibm.com/of/822/synonym/xml"> element. The namespace (specified in the xmlns attribute) needs to be exactly as shown.
3. Repeat the preceding step until you have specified all of the synonyms that you want to use for searching documents in an enterprise search collection.
4. Save and exit the XML file.

After you create the XML file, you must convert it to a synonym dictionary so that it can be added to the enterprise search system.

Creating a synonym dictionary

After you create or update a list of synonyms in an XML file, you must convert the XML file to a binary synonym dictionary.

About this task

To create a synonym dictionary, use the command line tool called `essyndictbuilder`, which is provided with OmniFind Enterprise Edition. The tool is in the *ES_INSTALL_ROOT*`/bin` directory.

The input to the tool is the XML file that lists your synonyms, and the output from the tool is a synonym dictionary. The dictionary must have the suffix `.dic`. For example, `c:\mydictionaries\products.dic`.

The default location for both files is the directory where the script is invoked. If a dictionary with the same name exists, the script produces an error.

The maximum size of a `.dic` in enterprise search is 8 MB.

Procedure

To create a synonym dictionary for enterprise search:

1. On the index server, log in as the enterprise search administrator. This user ID was specified when OmniFind Enterprise Edition was installed.
2. Enter the following command, where *XML_file* is the fully qualified path to the XML file that contains the list of synonyms and *DIC_file* is the fully qualified path to the synonym dictionary.

   AIX®, Linux®, or Solaris: `essyndictbuilder.sh` *XML_file DIC_file*
   Windows: `essyndictbuilder.bat` *XML_file DIC_file*

After you create a synonym dictionary, use the enterprise search administration console to add the dictionary to the enterprise search system and associate it with one or more collections.

Only the generated `.dic` file is uploaded to the enterprise search system. Ensure that the source XML file is kept in an access-controlled environment, and ensure that you back up the file regularly. You need this XML file to update your synonym dictionary.

Custom stop word dictionaries

You can define enterprise-specific vocabulary that is removed from a query to increase search relevance.

There are two kinds of stop word support in enterprise search:

- Language-specific stop word recognition that removes all frequently used common words like *a* and *the* from a multiple word query. The stop word dictionary that exists for each language cannot be modified by users. This stop word recognition is carried out automatically on all queries to improve search relevance.
- User-defined or custom stop word recognition that removes enterprise-specific vocabulary from queries. This stop word dictionary, which is defined by the administrator, can contain only special vocabulary. The user-defined stop word dictionary does not replace the enterprise search language-specific stop word dictionaries that contain common words. User-defined stop word dictionaries are language independent.

User-defined stop words typically include multiple word terms such as product names like *OmniFind Enterprise Edition*. Multiple word terms that are contained in the stop word dictionary are correctly identified in user queries and do not have to appear between quotation marks.

Compound terms in Germanic languages are also correctly identified in queries. A compound term is the combination of two or more words that is used as a single word. Lexicalized compounds like *Reisebüro* (travel agency) are not considered to be compounds.

Compound terms in a query are broken up into the individual terms that make up the compound. If any of the individual terms that make up the compound are in the stop word dictionary, the compound term is not removed from the query.

For example, the query term `Versicherungspolice` (insurance policy) returns documents that contain the compound terms *Lebensversicherungspolice* (life insurance policy) and *Haftpflichtversicherungspolice* (third party insurance policy). Even if the word *Police* is listed in the stop word dictionary, the compound query term `Versicherungspolice` is not removed from the query.

You must list the enterprise-specific vocabulary in an XML file that you must then convert to a stop word dictionary so that it can be added to the enterprise search system.

You can select which stop word dictionary to use in the enterprise search administration console. You can select one stop word dictionary for each collection. A stop word dictionary can be shared by several collections.

Creating an XML file for stop words

To remove enterprise-specific vocabulary from queries, you must specify which words qualify as stop words in an XML file.

About this task

The XML file that lists the stop words must comply with a specific schema specified in the XML document. This is an example of an XML file for stop words:

```
<?xml version="1.0" encoding="UTF-8"?>
<stopWords xmlns="http://www.ibm.com/of/83/stopwordbuilder/xml">
   <stopWord>OmniFind Edition</stopWord>
   <stopWord>WAS</stopWord>
   <stopWord>...</stopWord>
</stopWords>
```

Restrictions

A stop word can include white-space characters, but it cannot include punctuation characters, such as a comma (,) or vertical bar (|), because these characters might interfere with the enterprise search query syntax.

You do not need to enumerate normalizations of the term, such as the removal of accents or umlauts (enterprise search handles normalization automatically). For example, if you want to include the term météo as a stop word, you do not need to include the term METEO, too.

Procedure

To create a list of stop words for enterprise search:

1. Create an XML file. To avoid XML syntax errors, use an XML editor or XML authoring tool that can validate the XML. The XSD schema for the XML file is called `stopWords.xsd` and is contained in your enterprise search installation at *ES_INSTALL_ROOT*`/packages/uima/configuration_xsd/`.
2. Add a `<stopWord>` element for each word that is to be treated as a stop word.

   Be sure to include your mappings in a `<stopWords xmlns="http://www.ibm.com/of/83/stopwordbuilder/xml">` element. The namespace (specified in the xmlns attribute) needs to be exactly as shown.
3. Repeat the preceding step until you have specified all of the stop words that you want to be removed from queries when users search enterprise search collections.
4. Save and exit the XML file.

After you create the XML file, you must convert it to a stop word dictionary so that it can be added to the enterprise search system.

Creating a stop word dictionary

After you create or update a list of user-defined stop words in an XML file, you must convert the XML file to a stop word dictionary.

About this task

To create a stop word dictionary, use the command line tool called `esstopworddictbuilder`, which is provided with OmniFind Enterprise Edition. The tool is in the *ES_INSTALL_ROOT*`/bin` directory.

The input to the tool is the XML file that lists the stop words, and the output from the tool is a stop word dictionary. The dictionary must have the suffix `.dic`. For example, `c:\mydictionaries\productstopwords.dic`.

The default location for both files is the directory where the script is invoked. If a dictionary with the same name exists, the script produces an error.

The maximum size of a `.dic` in enterprise search is 8 MB.

Procedure

To create a stop word dictionary for enterprise search:

1. On the Index server, log in as the enterprise search administrator. This user ID was specified when OmniFind Enterprise Edition was installed.
2. Enter the following command, where *XML_file* is the fully qualified path to the XML file that contains the list of stop words and *DIC_file* is the fully qualified path to the stop word dictionary.

   AIX, Linux, or Solaris: `esstopworddictbuilder.sh` *XML_file DIC_file*
   Windows: `esstopworddictbuilder.bat` *XML_file DIC_file*

After you create a stop word dictionary, use the enterprise search administration console to add the dictionary to the enterprise search system and associate it with one or more collections.

Only the generated `.dic` file is uploaded to the enterprise search system. Ensure that the source XML file is kept in an access-controlled environment, and ensure that you back up the file regularly. You need this XML file to update your stop word dictionary.

Custom boost word dictionaries

You can define specific terms or multi-word terms that raise or lower the rank value of the document in which the term appears.

Each term in the boost dictionary is associated with a boost factor that can range from -10 to +10. The terms that you particularly want to see in your result documents are allocated a higher boost factor, while those that you do not want to have appear at all, or in combination with higher boosted terms, are given a lower value. The values -1, 0, and 1 have no boost effect.

If a query term which is listed in the boost dictionary with a particular boost factor appears in a retrieved document, the document rank value is either raised or lowered depending on the boost value. The boost value assigned to a term is relative as it also is affected by other factors. Thus if the term X is boosted by B1 and the term Y by B2, and B1 > B2, then boost(X) >= boost(Y).

A boost word typically includes multi-word terms such as product names like *OmniFind Enterprise Edition*. Multi-word terms contained in the boost word dictionary are correctly identified in user queries and do not have to appear within quotes.

Boost word dictionaries are language independent.

Compound terms in Germanic languages are also correctly identified in queries. A compound term is the combination of two or more words that is used as a single word. Lexicalized compounds like *Reisebüro* (travel agency) are not considered to be compounds.

Compound terms in a query are broken up into the individual terms that make up the compound. If boost values exist of the individual terms of a compound, the retrieved documents are ranked, although the value assigned is lower than it is if the term appears on it own in the document (and not as part of a compound). This broadens the search scope which is useful in cases in which only a few documents are found that contain the full compound.

For example, the query term `Versicherungspolice` (insurance policy) will return documents that contain the compound terms *Lebensversicherungspolice* (life insurance policy) and *Haftpflichtversicherungspolice* (third party insurance policy). If the word *Police* (policy) exists in the boost word dictionary, the document containing the compound query term `Versicherungspolice` is assigned a boost value.

You must list the terms with their boost value in an XML file which you must then convert to a boost word dictionary so that it can be added to the enterprise search system.

You can select which boost word dictionary to use on the enterprise search administration console. One boost word dictionary can be selected for each collection. A boost word dictionary can be shared by several collections.

Creating an XML file for boost words

To raise or lower the importance of certain result documents, you must specify which words can influence document ranking in an XML file.

About this task

The XML file that lists the boost words must comply with a specific schema specified in the XML file. This is an example of an XML file for boost words:

```
<?xml version="1.0" encoding="UTF-8"?>
<boostTerms xmlns="http://www.ibm.com/of/83/boostbuilder/xml">
    <!-- group boost terms by boost value-->
   <boostTermList boost="5">
    <!-- each term can specify the synonym expansion separately-->
     <term useVariants="true">OmniFind Edition</term>
     <term useVariants="false">Edition</term>
    <term>OmniFind</term>
   </boostTermList>
   <boostTermList boost="8">
        <term useVariants="true">WAS</term>
        <term>term9</term>
    </boostTermList>
</boostTerms>
```

Restrictions

You can group terms that share the same boost value in a `<boostTermList>` element, but a boost value can occur multiple times, for example, if you want to sort boost words alphabetically in the XML file.

A boost word can include white-space characters, but it cannot include punctuation characters, such as a comma (,) or vertical bar (|), because these characters might interfere with the enterprise search query syntax.

Boost terms can have variants, such as acronyms or abbreviations. You can enumerate all variants in the boost word dictionary; however, if you plan to use a synonym dictionary as well as a boost word dictionary, and have already added terms and their variants to the synonym dictionary, you do not have to add these variants to the boost word list as well. Instead, you can simply set the attribute `useVariants` to true for the variant you add to the boost word dictionary. All variants of this term listed in the synonym dictionary that occur in any of the retrieved documents will influence the rank value assigned to these documents.

You do not need to enumerate normalizations of the term, such as the removal of accents or umlauts (enterprise search handles normalization automatically). For example, if you want to include the term météo as a boost word, you do not need to include the term METEO, too.

Procedure

To create a list of boost words for enterprise search:

1. Create an XML file. To avoid XML syntax errors, use an XML editor or XML authoring tool of your choice. The XSD schema for the XML file is called `boostTerms.xsd` and is contained in your enterprise search installation at *ES_INSTALL_ROOT*`/packages/uima/configuration_xsd/`.

2. Include your mappings in a <boostTerms xmlns="http://www.ibm.com/of/83/boostbuilder/xml"> element. The namespace (specified in the xmlns attribute) needs to be exactly as shown.
3. Add a <boostTermList> element for grouping all terms that share the specified boost value.

   The boost values can range from -10 to 10. For example, <boostTermList boost="-5"> or <boostTermList boost="5">.

   The importance of documents that contain the specified terms will be raised or lowered according to the specified boost value.
4. Add a <term> element for each term that uses the specified boost value.

   If you want to include variants of a boost word that are listed in the synonym dictionary, set the useVariants attribute in the <term> element to true. The default is false. If no variants can be found in the synonym dictionary, no error message is produced.
5. Repeat the preceding steps until you have specified all of the terms that are to be used as boost words when users search enterprise search collections.
6. Save and exit the XML file.

After you create the XML file, you must convert it to a boost word dictionary so that it can be added to the enterprise search system.

Creating a boost word dictionary

After you create or update a list of boost words in an XML file, you must convert the XML file to a boost word dictionary.

About this task

To create a boost word dictionary, use the command line tool called esboostworddictbuilder, which is supplied with OmniFind Enterprise Edition. The tool is in the ES_INSTALL_ROOT/bin directory.

The input to the tool is the XML file that lists your boost words, and the output from the tool is a boost word dictionary. The dictionary must have the suffix .dic. For example, c:\mydictionaries\productboostwords.dic.

The default location for both files is the directory where the script is invoked. If a dictionary with the same name exists, the script produces an error.

The maximum size of a .dic in enterprise search is 8MB.

Procedure

To create a boost word dictionary for enterprise search:

1. On the Index server, log in as the enterprise search administrator. This user ID was specified when OmniFind Enterprise Edition was installed.
2. Enter the following command, where *XML_file* is the fully qualified path to the XML file that contains the list of boost words and *DIC_file* is the fully qualified path to the boost word dictionary. If you want to use a synonym dictionary as well, add the fully qualified path to the synonym dictionary after the boost dictionary name. Naming a synonym dictionary is optional.

   UNIX®: esboostworddictbuilder.sh *XML_file DIC_file SYNDIC_file*
   Windows: esboostworddictbuilder.bat *XML_file DIC_file SYNDIC_file*

After you create a boost word dictionary, use the enterprise search administration console to add the dictionary to the enterprise search system and associate it with one or more collections.

Only the generated `.dic` file is uploaded to the enterprise search system. Ensure that the source XML file is kept in an access-controlled environment, with the appropriate backup strategy in place. You need this XML file to update your boost word dictionary.

Related tasks

"Creating a synonym dictionary" on page 60

Text analysis included in enterprise search

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

When a document is processed, enterprise search determines the language of that document and breaks up the stream of input text into distinct units or tokens.

During a search, the user or an application, 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 into:

- Basic nondictionary-based support. This includes white space and n-gram segmentation. Basic nondictionary-based support also contains sentence 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 that are recognized in the input text. Search quality is greatly enhanced by using advanced language processing.

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:

Table 10. Supported languages for automatic language identification

Afrikaans	Arabic	Balinese
Basque	Catalan	Chinese (Traditional and Simplified)
Czech	Danish	Dutch
English	Finnish	French
German	Greek	Hebrew
Icelandic	Irish (Gaelic)	Italian
Japanese	Korean	Malay
Norwegian (Bokmål)	Polish	Portuguese
Romanian	Russian	Spanish
Swedish	Tagalog	Thai
Turkish	Vietnamese	

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

In enterprise search, you can specify to detect the language of a document automatically or select a language to use.

If you select automatic language detection and the parser cannot determine the language of a document, the parser uses the language that you specify when you create the crawler in the enterprise search administration console.

If you do not select automatic language detection, the language that you specify is always used. You specify the document language by editing the crawler properties on the enterprise search administration console. The default language is English.

Documents for which there are no language-specific dictionaries are processed by using a basic language-independent technology such as white-space segmentation and n-gram segmentation.

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 that is used in the document. However, the analysis results are not always satisfactory.

The language of a document 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 in a multilingual document collection, you can limit the search results to include only documents that are written in French. Setting the language of your output documents is an advanced search option that you can select on the enterprise search administration console.

Related concepts

"Linguistic support for nondictionary-based segmentation"

Linguistic support for nondictionary-based segmentation

For documents in languages that are not supported by the 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.

When you create your collection, you can also optionally select to tokenize Chinese and Japanese documents using n-gram segmentation.

To remove any white space characters, for example new line or tab characters, during n-gram segmentation, you must turn on parameter settings in the file called

collection.properties in *ES_NODE_ROOT*/master_config/<CollectionID>.parserdriver before you begin to parse the documents. The parameters required to remove white space characters include:

- **removeCjNewLineChars**: If set to *true*, this parameter removes any sequence of new line and tab characters that occur between Chinese or Japanese characters. The default is removeCjNewlineChars=false.
- **removeCjNewLineCharsMode**: If set to *all*, this parameter removes white space characters irrespective of character context. For example, white space characters are also removed in English text. If you want to work with this option, you must add the parameter to the property file. Only removeCjNewlineCharsMode=all is valid, all other values are ignored.

Related concepts

"Language identification" on page 71

Tokenizing numerical characters as n-gram tokens

To tokenize numerical characters in addition to double-byte characters as n-gram tokens, you must change a parameter setting in the annotator descriptor file.

About this task

The default handling of numerical characters in the white space and n-gram tokenizer is to treat all numerical characters as white space segmented tokens. To tokenize numerical characters as n-gram tokens, you must change the n-gram mode setting in the annotator descriptor file. You cannot change this setting by using the enterprise search administration console.

Tip: Three modes for n-gram tokenization are available: normal, numeric, and full. This procedure discusses how to enable numeric n-gram tokenization. For information about how to configure support for full n-gram tokenization in enterprise search collections, and to learn about how characters are handled in collections that are configured for full n-gram support, see http://www.ibm.com/support/docview.wss?rs=63&uid=swg27011088.

Procedure

The default n-gram mode setting is called normal, and treats numerical characters and SBCS characters as characters segmented by white space. To enable numerical n-gram mode:

1. Stop the parser for your collection.
2. Stop the runtime for your collection.
3. Open the annotator descriptor file called jtok.xml in the *ES_NODE_ROOT*/master_config/*collection_ID*.parserdriver/specifiers directory, where *collection_ID* is the ID that was specified for the collection (or that was assigned by the system) when the collection was created.
4. Change the **NgramMode** parameter setting from normal to numeric.
5. Restart the parser for your collection.
6. Restart the runtime.

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 includes 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 into their component parts. For example:

wouldn't is split into *would* + *not*
Horse's is split into *Horse* + *'s*

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 in the case of clitics are considered word parts, and they are considered full stops (or periods) in the case of unknown abbreviations. URLs, e-mail addresses and dates are split up into several tokens.

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:

Table 11. Supported languages

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

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

"Orthographic variants 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

"Word segmentation in Japanese"

Stop word removal

In enterprise search, all stop words, for example, common words, such as *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. For the other languages, enterprise search uses special lists.

No stop words are removed during query processing if:

- All of the words in a query are stop words. If all the query terms are removed during stop word processing, then the result set is empty. To ensure that search results are returned, stop word removal is disabled when all of the query terms are stop words. For example, if the word *car* is a stop word and you search for *car*, then the search results contain documents that match the word *car*. If you search for *car buick*, the search results contain only documents that match the word *buick*.
- The word in a query is preceded by the plus sign (+).
- The word is part of an exact match.
- The word is inside a phrase, for example, ″I love my car″.

Related concepts

“Character normalization”

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 half-width characters to full-width characters.

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 that contain *schoen* when searching for *schön*.

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

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

Ligature expansion
For example, finding documents that contain *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 76

Regular expression annotator

The regular expression annotator enables you to perform custom text analysis without the need to implement your own text analysis engine. Based on a set of rules (regular expressions) that you can define yourself, the regular expression annotator detects information structures in text documents and creates annotations of the detected information in the common analysis structure.

The regular expression annotator detects entities or units of information in text documents, for example, phone numbers, product codes, building and room numbers, or addresses, based on regular expressions. If one of the regular expressions matches parts of the document text, the regular expression annotator creates the corresponding annotations that cover the matched piece of information. These annotations are stored in the common analysis structure and can later be searched by mapping these analysis results to the enterprise search index, using a common analysis structure to index mapping file. Alternatively a common analysis structure to database mapping file can be created to store the annotations in a JDBC-capable database.

The set of rules (regular expressions) that you define are stored in an XML configuration file (also referred to as the rule set file). The regular expression annotator contains the analysis logic that processes these regular expressions. It supports the regular expression syntax in Java 1.4.

The type system description of the regular expression annotator must define the annotation types and features that are used and created by the regular expression annotator. Depending on the complexity of the application area of the regular expression annotator (for example, if more types are required than are defined in the supplied regular expression annotator), additional input and output capabilities must be defined in the regular expression annotator descriptor. The types used in the descriptor must match the types in the type system description of the annotator.

The regular expression annotator is included in enterprise search as a deployable PEAR (Processing Engine ARchive) file that is configured with sample rules to detect phone numbers, URLs, and e-mail addresses.

Easy semantic search using the regular expression annotator

Enterprise search includes the regular expression analysis engine pre-configured with a set of rules that enables it to detect telephone numbers, URLs and e-mail addresses in text documents.

You can use this sample configuration of the regular expression analysis engine to enable enterprise search to find actual phone numbers in documents without looking for the keyword *phone number* in documents. To query for the constructs that are detected by the regular expression annotator, a sample common analysis structure to index mapping file is also provided. Furthermore, a simple method is demonstrated by which you can issue powerful semantic queries through simple keywords. This method uses the enterprise search synonym support to automatically expand simple keyword queries into semantic queries. A sample synonym dictionary that illustrates this mechanism is provided. You can find all

the files that you need to use the regular expression annotator with the sample configuration at ES_INSTALL_ROOT/packages/uima/regex.

For many application scenarios, it may be sufficient to only slightly modify the regular expression rules that are provided with the sample configuration in order to tailor the regular expression annotator to meet your needs.

However, to fully customize the annotator, the use of the UIMA SDK is recommended. For this purpose, the regular expression annotator is also included in the enterprise search base annotator package located at ES_INSTALL_ROOT/packages/uima/.

Related tasks

"Enabling easy semantic search using the regular expression annotator"

"Customizing the regular expression annotator" on page 85

"Viewing base annotator and custom text analysis results" on page 11

Enabling easy semantic search using the regular expression annotator

To enable easy semantic search using synonyms, you must add the regular expression annotator, the common analysis structure to index mapping file, and the sample synonym dictionary to your enterprise search system and associate these resources with your collection.

Thereafter, the regular expression annotator will process your documents during the parsing phase, the indexer will add the custom analysis results to the index, and the search service can utilize the provided semantic synonym dictionary to search for the custom analysis results through simple keywords that are automatically expanded into semantic queries.

Procedure

To enable easy semantic search:

1. Add the regular expression custom text analysis engine called of_regex.pear at *ES_INSTALL_ROOT*/packages/uima/regex to the enterprise search system using the enterprise search administration console.
2. Associate the regular expression text analysis engine with your collection.
3. Add the common analysis structure to index mapping file called of_sample_regex_cas2index.xml in the directory *ES_INSTALL_ROOT*/packages/uima/regex. This maps the custom analysis results (annotations) that the regular expression annotator produces to searchable spans in the enterprise search index. Then you can use XML fragment or XPath queries to search for these spans.
4. Crawl, parse and index your collection. At this point, after indexing is finished, you could enter an XML search query using an XML fragment expression, for example, @xmlf2::'<#phonenumber>', using the search application. However, the purpose of enabling semantic search by synonyms, is to allow you to use queries like Barbara phone number and have the system translate the query to Barbara @xmlf2::'<#phonenumber>'.
5. Add the provided sample binary synonym dictionary called of_sample_synonym_dic.dic in the directory *ES_INSTALL_ROOT*/packages/uima/regex to the enterprise search system using the administration console. You can make changes to the source XML sample dictionary, or use it as a basis to create your own dictionary and then convert it to a new dictionary file by

using the essyndictbuilder tool. The XML sample synonym dictionary is called `of_sample_synonym_dic.xml`, also at *ES_INSTALL_ROOT*`/packages/uima/regex`.

6. Associate the synonym dictionary with your collection and start (or restart) the search service for your collection.
7. In the search application, select the option to automatically search for synonyms by using semantic expansion. After you enable this option, the search application rewrites your basic keyword queries to XML fragment queries and includes expressions that find the searchable spans that identify telephone numbers, e-mail addresses and URLs.
8. In the search application, enter a query requesting a telephone number, for example, `barbara telephone number`. The query searches for documents that contain the three keywords *barbara*, *telephone*, and *number*, as well as for documents that contain the keyword *barbara* and spans of numbers and characters in the documents that match the regular expressions defined for a telephone number. The keywords and the telephone numbers that are found are highlighted in the search results.

You can see which keywords translate into semantic queries in the supplied sample synonym dictionary.

```
<?xml version="1.0" encoding="UTF-8"?>
<synonymgroups xmlns="http://www.ibm.com/of/822/synonym/xml">
  <synonymgroup>
   <synonym>telephone number</synonym>
   <synonym>phone number</synonym>
   <synonym>telephone nbr</synonym>
   <synonym>phone nbr</synonym>
   <synonym>@xmlf2::'&lt;#phonenumber/&gt;'</synonym>
  </synonymgroup>
  <synonymgroup>
   <synonym>facsimile number</synonym>
   <synonym>fax number</synonym>
   <synonym>facsimile nbr</synonym>
   <synonym>fax nbr</synonym>
   <synonym>@xmlf2::'&lt;#phonenumber/&gt;'</synonym>
  </synonymgroup>
  <synonymgroup>
   <synonym>e-mail address</synonym>
   <synonym>email address</synonym>
    <synonym>@xmlf2::'&lt;#email/&gt;'</synonym>
  </synonymgroup>
  <synonymgroup>
   <synonym>URL</synonym>
   <synonym>unified resource locator</synonym>
   <synonym>Web address</synonym>
   <synonym>@xmlf2::'&lt;#url/&gt;'</synonym>
  </synonymgroup>
</synonymgroups>
```

Related concepts

"Easy semantic search using the regular expression annotator" on page 79

The rule set file

In the regular expression annotator, the XML rule set file defines the rules, in the form of regular expressions, which are used to parse the text document.

The rules specify, in sequential order, where in the document text, the annotator must look for what, and then what action to take if a match is found.

When the regular expression annotator is called, the XML rule set file that contains the regular expression patterns is compiled and matched against parts of the document text. If a match or a partial match is found, the annotation that is associated with the specific rule is created and stored in the common analysis structure.

The types used in the rules must be defined in the type system description of the regular expression annotator.

The regular expression annotator processes one rule at a time, beginning with the first rule in the XML rule set file. For each rule, the corresponding compiled regular expression is matched against the annotations are created in an earlier step, for example, annotations created by annotators that processed the document before the regular expression annotator. The annotations that match the rules must be of the same type as the input capability types specified in the regular expression annotator descriptor.

If a match is found, the annotation type created in the rule that fires must also be specified as a valid output capability type in the regular expression annotator descriptor. The new annotations that are created by an earlier rule can be used as input annotations for rules that fire later in the XML rule set.

Related tasks

"Defining regular expression rules"

Related reference

"The annotator descriptor" on page 86

"Logging" on page 89

Defining regular expression rules

The rule set defines the regular expressions that are matched against text in the document and the actions that the regular expression annotator must take if a pattern matches.

About this task

The XML rule set file must follow the rule syntax outlined in the following example. This is the rule set file for the sample regular expression annotator that recognizes telephone numbers, URLs and e-mail addresses.

The top level element is a `<ruleSet>` element which consists of one or more `<rule>` elements. Each `<rule>` element in turn defines a Java regular expression consisting of an attribute `regEx` as well as the `matchStrategy` and `matchType` attributes. The action is defined in the `<createAnnotation>` element which specifies the annotation ID and annotation type.

```
<?xml version="1.0" encoding="UTF-8"?>
<ruleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="ruleSet.xsd">
 <!-- Phone Number -->
 <!-- This rule matches different ways of writing telephone numbers,
   for example, 01234-12345, 01234 / 122-32, (001234)12345,
   +49 (0) 123412345, (123) 123 1234,
   1-800-IBM-4YOU -->
 <rule regEx="(?x)(\s|\b)(
0{1,2}[1-9]{1}[0-9]{1,5}\x20?[-/\\]\x20?[1-9]{1}([0-9]{1,8}-?)
  {1,3}[0-9]{1,}
|\(0[1-9]{1}[0-9]{1,3}\)\x20?[1-9]{1}[0-9]{2,8}
```

```
|\(00[1-9]{1}[0-9]{1,8}\)\x20?[1-9]{1}[0-9]{2,10}
|\((0\x20?[1-9]{1}[0-9]{1,3}|00\x20?[1-9]{1}[0-9]{1,8})\)\x20?[1-9]
   {1}[0-9] {1,3}(\x20[0-9]{2,4}){1,5}
|(0\x20?[1-9]{1}[0-9]{1,3}|00\x20?[1-9]{1}[0-9]{1,8})\x20?[/\\]\x20?
   [1-9]{1} [0-9]{1,3}(\x20[0-9]{2,4}){1,5}
|\(?\+[1-9]{1}[0-9]{0,3}\)?([-\x20]|\x20?\(0\))[-\x20]?[1-9]
   {1}[0-9]{1,10}
|\(?\+[1-9]{1}[0-9]{0,3}\)?([-\x20]|\x20?\(0\))[-\x20]?[1-9]
   {1}[0-9]{1,3}[-\x20]([0-9]{2,5}[-\x20]?){1,4}
|(1-)?[0-9]{3}-[0-9]{3}-[0-9]{4}
|\([1-9]{1}[0-9]{2}\)\x20[0-9]{3}[-\x20][0-9]{4}
|1-(800|888|877|866)-([A-Z0-9]{7}|[A-Z0-9]{3}-[A-Z0-9]
   {4}|[A-Z0-9]{4}-[A-Z0-9]{3})
)(?!(\d|\x20\d|-\d))(\s|\b)"
  matchStrategy="matchAll" matchType="uima.tcas.DocumentAnnotation">
  <createAnnotation id="phonenumber" type="com.ibm.es.uima.PhoneNumber">
   <begin group="0"/>
   <end group="0"/>
  </createAnnotation>
 </rule>
 <!-- potential Phone Number -->
 <!-- This rule matches numbers that resemble telephone numbers but could
 also be anything else. For example, 0123 1234 123,
 +123456789, 123 123 1234 -->
 <rule regEx="(?x)(\s|\b)(
  0[1-9]{1}[0-9]{1,3}\x20[1-9]{1}[0-9]*\x20?([0-9]{2,}\x20?)+
  |00\x20?[1-9]{1}[0-9]{0,3}\x20[1-9]{1}[0-9]{1,3}\x20?[1-9]
   {1}([0-9]{2,}\x20?)+
  |\+[1-9]{1}[0-9]{0,3}[1-9]{1}[0-9]{6,}
  |[1-9]{1}[0-9]{2}\x20[0-9]{3}\x20[0-9]{4}
  )(?!(\d|\x20\d|-\d))(\s|\b)"
  matchStrategy="matchAll" matchType="uima.tcas.DocumentAnnotation">
  <createAnnotation id="potential_phonenumber"
     type="com.ibm.es.uima.PotentialPhoneNumber">
   <begin group="0"/>
   <end group="0"/>
  </createAnnotation>
 </rule>
 <!-- URL Annotation -->
 <!-- This rule matches URLs, for example, http://www.ibm.com -->
 <rule regEx="(?x)(\s|\b)(
    http://[\w\-]+([\.][\w\-]+)+([/][\w\~\(\)\-\?=%\u0026\#]*)*
    |www.[\w\-]+([\.][\w\-]+)+([/][\w\~\(\)\-\?=%\u0026\#]*)*
   )(\s|\b)"
  matchStrategy="matchAll" matchType="uima.tcas.DocumentAnnotation">
  <createAnnotation id="url" type="com.ibm.es.uima.URL">
   <begin group="0"/>
   <end group="0"/>
  </createAnnotation>
 </rule>
 <!-- Email Annotation -->
 <!-- This rule matches e-mail addresses, for example, yourName@domain.com -->
 <rule regEx="(?x)(\s|\b)(
    [a-zA-Z0-9][\w\.-]*[a-zA-Z0-9]@[a-zA-Z0-9]([\.-]?\w)*\.[a-zA-Z]
    {2,3})(\s|\b)"
  matchStrategy="matchAll" matchType="uima.tcas.DocumentAnnotation">
  <createAnnotation id="email" type="com.ibm.es.uima.Email">
   <begin group="0"/>
   <end group="0"/>
  </createAnnotation>
 </rule>
</ruleSet>
```

Procedure

To create the XML rule set for the regular expression annotator that defines your custom regular expressions:

1. Create an XML file. To avoid XML syntax errors, use an XML editor or XML authoring tool of your choice. The XSD schema for the XML rule set file is called `ruleSet.xsd`, which you can find in your enterprise search installation in the *ES_INSTALL_ROOT*`/packages/uima/regex/` directory.
2. Include your mappings in a `<ruleSet xmlns="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="ruleSet.xsd">` element. The namespace is specified in the xmlns attribute, and must be exactly as shown.
3. Add a `<rule>` element that contains a `regEx` attribute containing the regular expression pattern, a `matchStrategy` attribute, and a `matchType` attribute.

   The annotator fully supports the Java 1.4 regular expression syntax. For an introduction to regular expressions and to view the full syntax, refer to the Java documentation at http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html.

   `matchStrategy` specifies how to search, for example, if all matches must be found in the document or if the text match must be an exact match. Three different match strategies are available:

   - `matchFirst` stops at the first text sequence that matches the regular pattern
   - `matchAll` finds all text sequences in a document that match the regular pattern
   - `matchComplete` only matches text sequences that are an exactly match. For example, if we had a pattern "foo", only the term "foo" would be matched, "foobar" would not result in a match.

   `matchType` determines the annotation type the rule is matched against. This way, you can restrict your regular expression to match, for example, within an existing token annotation. This avoids matching too much content within a rule. Possible types are the allowed input annotation types to the annotator (defined in the annotator descriptor), such as uima.tt.DocumentAnnotation, uima.tt.ParagraphAnnotation, and user-defined types such as foo.bar.MyAnnotation. Sometimes, the output type of one rule is used as the input type of a subsequent rule. `matchType` allows you to restrict the search scope of certain rules.
4. Add a `<createAnnotation>` element that defines the action that the regular expression annotator is to take if a match is found.

   Each `createAnnotation` element has two attributes:

   - `id` uniquely identifies the annotation and is used to reference the annotation
   - `type` specifies the type of annotation that is created
5. Add the following component elements that define the match position for the `<createAnnotation>` element:
   - Mandatory: `<begin>` specifies where the match begins. This element has two attributes:
     - Mandatory: `group` identifies the Java capturing group. It can be assigned a value between 0 (complete text sequence match) and 9 (multiple capturing groups)
     - Optional: `location` indicates a position inside the match group (with respect to the positioning of the parentheses), either `start` (left parenthesis) or `end` (right parenthesis).
   - Mandatory: `<end>` specifies where the match ends. This element has two attributes:

- Mandatory: `group` identifies the capturing group. It can be assigned a value between 0 (complete text sequence match) and 9 (subsequent and ever smaller match groups)
- Optional: `location` indicates a position inside the match group (with respect to the positioning of the parentheses), either `start` (left parenthesis) or `end` (right parenthesis).

• Optional: `<setFeature>` creates a feature and assigns it to the annotation. This element has two attributes:
  - `name` is the name of the feature as you defined it in the type system description
  - `type` specifies the type of the feature value, either String, Integer, Float and Reference. The type must be the same as the range type defined for the feature in the annotator type system description.

    Features of type `Reference` are used to create a link between two annotations to model a semantic relation. The `<setFeature>` element content must be set to the `id` of the `<createAnnotation>` element you want to create a link to.

Related concepts

"The rule set file" on page 81

Customizing the regular expression annotator

You can customize the sample configuration of the regular expression annotator to detect new entities (for example, product serial numbers) or adapt the regular expression rules for existing entities (for example to detect company-specific phone numbers) by making smaller changes to the sample rule set and type system files.

The modified rule set file and type system description must be added to the regular expression processing engine archive file (the PEAR file). After you have updated the PEAR file, you can add the customized regular expression text analysis engine to the enterprise search system again.

For a more elaborate customization of the regular expression annotator, you are strongly recommended to use the UIMA SDK tools. These tools help you to create or update the type system description and descriptor files, to possibly combine the annotator with others to form an aggregate analysis engine and to create a new processing engine archive (PEAR file) that includes all the resources necessary to use the annotator in enterprise search. For information about the tools that are available to support you in these tasks, refer to the UIMA SDK documentation .

Procedure

To adapt the regular expression annotator by adding new rules and entities, or to change existing rules, you can update the provided sample regular expression annotator PEAR file as follows:

1. Create a new directory called `xml` in your system.
2. Copy the sample rule file `of_sample_regex_rules.xml` in the *ES_INSTALL_ROOT*`/packages/uima/regex/` directory to your `xml` directory and modify the file to include your custom pattern matching rules. To avoid XML syntax errors, use an XML editor or XML authoring tool of your choice.
3. Copy the corresponding type system description file `of_sample_typesystem.xml` from the directory `ES_INSTALL_ROOT/packages/`

uima/regex/ to your xml directory and modify the file to include the definitions for the types that your new rules require.

4. If you only add a few new rules or change existing rules, you are not required to change the annotator descriptor. If you plan to do other changes, or if you use additional custom analysis steps, check if the annotator descriptor must be modified.
5. Use an archive utility of your choice to update a copy of the regular expression annotator PEAR file to include your two updated files. For example, copy the of_regex.pear file from ES_INSTALL_ROOT/packages/uima/regex/ to the parent directory of the xml directory you created. Then use the Java jar command line tool (for example, part of the IBM Java SDK) to issue the following commands from that parent directory:

   ```
   "jar -uf of_regex.pear -C xml/ of_sample_regex_rules.xml"
   "jar -uf of_regex.pear -C xml/ of_sample_regex_typesystem.xml"
   ```
6. Use the enterprise search administration console to add the regular expression annotator as a custom text analysis engine to the enterprise search system and associate it with a test document collection.
7. Check the analysis results produced by the regular expression annotator by updating the document collection properties to produce readable XML output of the analysis results that are stored in the common analysis structure using the XCAS dump feature.
8. Process the test documents and use the XCAS Annotation Viewer to view the content of the XML files.
9. If you are satisfied with the annotations that are created by the annotator based on your custom regular expressions, edit the document collection properties again to disable the parser from producing readable XML output of the analysis results. If further modifications to the rule set file are required, you must repeat the steps that update the PEAR file.
10. Create the common analysis structure to index mapping file to index the analysis results, or the common analysis structure to database mapping file if you want to add the results to a database. You can use the provided sample common analysis structure to index mapping file as a starting point to create your common analysis structure to index mapping file.
11. Using the enterprise search administration console to add the mapping files and associate them with your full document collection.
12. Search your annotations using XML fragment or XPath queries, or alternatively using semantic expansion during synonym search.

Related concepts

"Easy semantic search using the regular expression annotator" on page 79

Related tasks

"Viewing base annotator and custom text analysis results" on page 11

The annotator descriptor

The regular expression annotator XML descriptor contains descriptive information about the regular expression annotator that is needed to run the annotator.

If you are only using the regular expression annotator, and no additional custom analysis steps, you are only required to change the descriptor if:

- You want to change the file name of the rule set file (in the <externalResourceDependencies> element).
- You want to use more than one rule set file.

- You want to change the name of the type system description file.

If you are using additional custom analysis steps, you are required to change the descriptor if:

- You want your custom analysis to use annotations created by the regular expression annotator. In this case, you must update the output capabilities in the annotator descriptor.
- You have defined regular expression rules that must match annotation types created in earlier custom analysis steps. In this case, you must update the input capabilities in the annotator descriptor.

Use the UIMA SDK tools to create or update the annotator descriptor and recreate the processing engine archive (.pear file) that includes all of the resources required to use the annotator in enterprise search. For information on the tools that are available to support you in these tasks, see the UIMA SDK documentation at http://www.alphaworks.ibm.com/tech/uima/.

Configuration parameters

The regular expression annotator only has one configuration parameter called `String2NumberImpl` that must be set to the name of the class that implements the `com.ibm.uima.an_regex.String2Number` interface. The regular expression annotator must be supplied with an implementation of this class, otherwise an exception will occur. If you want to customize the regular expression annotator to meet your needs, you can provide your own implementation of the `String2Number` interface by passing your class name in the XML descriptor file.

The `String2Number` interface declares two methods, `toInt(String)` and `toFloat(String)`, which transform a string representation of an integer or float value to the corresponding integer or float value. These two methods are used to transform a number that contains separator characters into a valid Java Integer or Float value.

The default implementation of `com.ibm.uima.an_regex.String2Number_impl` considers a period (.) as a decimal separator and a comma (,) as a thousands separator. For example, if 1,999.00 is found in a text document, `toInt` converts it to 1999. `toFloat` returns 1999.00.

Sample

The configuration parameter section of the descriptor is as follows:

```
<configurationParameters>
 <configurationParameter>
  <name>String2NumberImpl</name>
  <description>Implementation of the
   com.ibm.uima.an_regex.String2Number interface</description>
  <type>String</type>
  <multiValued>false</multiValued>
  <mandatory>true</mandatory>
 </configurationParameter>

 <configurationParameterSettings>
  <nameValuePair>
   <name>String2NumberImpl</name>
   <value>
    <string>com.ibm.uima.an_regex.impl.String2Number_impl</string>
```

```
   </value>
  </nameValuePair>
 </configurationParameterSettings>
</configurationParameters>
```

Capabilities

The input and output capabilities of the regular expression annotator and the languages it supports are defined in the capabilities section of the annotator descriptor.

The input capabilities (input types) in the descriptor file must comply with the match types used in the rule set file. If the rules only use the `uima.tt.DocumentAnnotation` type, you do not have to declare any input capabilities because this type is always defined. All other types must be defined.

The annotation types created by the regular expression annotator are specified in the output capabilities section. These types must match the output types declared in the rule set file.

Because the regular expression annotator is language-independent, specify `x-unspecified`, which stands for any language.

Type system description

The type system description section in the regular expression annotator XML descriptor defines the type system used by the annotator. The types used in the rule set XML file, and mentioned in the input and output capabilities sections in the annotator descriptor must match the types defined in the type system description.

Sample

The type system description section of the descriptor imports the type system descriptor XML file:

```
<typeSystemDescription>
 <imports>
  <import location="./xml/of_sample_regex_typesystem.xml"/>
 </imports>
</typeSystemDescription>
```

External resources

The external resource section of the descriptor contains the files and classes required by the annotator.

The regular expression annotator requires the rule set file. The rule set file is made available to the regular expression annotator through the `com.ibm.uima.an_regex.FileResource` interface, which is implemented by the class `com.ibm.uima.an_regex.impl.FileResource_impl`. To pass your custom rules to the regular expression annotator, you must provide the name of the rule set file in the annotator descriptor and add the location of the file to your class path. The key that the regular expression annotator uses to access the rule set file is named `RuleSetDefinition`. Do not change this key, otherwise the regular expression annotator will not find the rule set and the annotator will not be able to initialize.

Custom annotators that you deploy for enterprise search cannot use the UIMA `datapath` setting to look up external resources. To look up external resources, specify the path names to the resources in the class path for the custom annotator. See the UIMA SDK documentation at http://www.alphaworks.ibm.com/tech/uima/ for information about using the PEAR Generation Wizard to specify custom annotator class path settings.

Sample

The external resource section of the descriptor is as follows:

```
<externalResourceDependencies>
 <externalResourceDependency>
  <key>RuleSetDefinition</key>
  <description>Rule set definition</description>
  <interfaceName>com.ibm.uima.an_regex.FileResource</interfaceName>
  <optional>false</optional>
 </externalResourceDependency>
</externalResourceDependencies>
<resourceManagerConfiguration>
 <externalResources>
  <externalResource>
   <name>of_samples_regex_rules</name>
   <description>Rule set definition file for room numbers</description>
   <fileResourceSpecifier>
    <fileUrl>file:of_samples_regex_rules.xml</fileUrl>
   </fileResourceSpecifier>
   <implementationName>
    com.ibm.uima.an_regex.impl.FileResource_impl</implementationName>
  </externalResource>
 </externalResources>
 <externalResourceBindings>
  <externalResourceBinding>
   <key>RuleSetDefinition</key>
   <resourceName>of_samples_regex_rules</resourceName>
  </externalResourceBinding>
 </externalResourceBindings>
</resourceManagerConfiguration>
```

Related concepts

"The rule set file" on page 81

Related reference

"Logging"

Logging

All log messages from the regular expression annotator are written to the current collection's log file.

The collection log files are located at `ES_NODE_ROOT/logs/` and have names of the form *<collection_id>_<current_date>*`.log`. You can view the log files using the esviewlogs.sh/.bat scripts.

There are seven possible log levels:

- Error
- Warning
- Info
- Config
- Fine
- Finer

- Finest

You cannot change the mapping for Error and Warning messages. By default, only Info, Warning and Error messages are written to the log file. These are the standard log levels used by enterprise search. The other log levels can be mapped for more detailed information.

To receive log messages from the regular expression annotator, the log level must be set at least to Config. At this level, the annotator logs configuration settings, such as, the rule set file that is used and the implementation class name for the `com.ibm.uima.an_regex.String2Number` interface.

If you set the log level to Finer for example, the annotator logs which annotations could not be created. This can help you to determine why not all of the annotations that you were expecting, were created. For example, there could be an error in one of your regular expressions, or an optional capturing group might not have matched any text in the document. Similarly, if you specify that a feature is to be set to the text sequence that matches a capturing group, and there is no matching text sequence, the feature is set to null.

For very detailed information, set the log level to Finest. At this level, the annotator logs the current regular expression pattern, the part of the document text that is currently being analyzed, and any annotations and features that were created. Using very detailed logging, especially the log levels Finer and Finest, has a negative impact on the overall performance of the annotator.

If you require detailed log level mapping, modify the configuration file called `tokenizer.properties` at *ES_NODE_ROOT*`/master_config/parserservice/` by changing the configuration setting `trevi.tokenizer.jedii.InformationalLevelMapping=Info` to `trevi.tokenizer.jedii.InformationalLevelMapping=Finest`, for example.

To activate the log level changes, you must stop all parser processes using the administration console. Then you must stop and then restart the parser service session from the command line by calling:

```
>esadmin session parserservice stop
>esdamin session parserservice start
```

Thereafter, the parse can be started again and you should now have the new log level. You must repeat these steps every time you change the log level.

Related concepts

"The rule set file" on page 81

Related reference

"The annotator descriptor" on page 86

Enterprise search documentation

You can read the OmniFind Enterprise Edition documentation in PDF or HTML format.

The OmniFind Enterprise Edition installation program automatically installs the information center, which includes HTML versions of the documentation for enterprise search. For a multiple server installation, the information center is installed on both search servers. If you do not install the information center, when you click help, the information center opens on an IBM Web site.

To see installed versions of the PDF documents, go to `ES_INSTALL_ROOT/docs/locale/pdf`. For example, to find documents in English, go to `ES_INSTALL_ROOT/docs/en_US/pdf`.

To access the PDF versions of the documentation in all available languages, see the OmniFind Enterprise Edition, Version 8.5 documentation site.

You can also access product downloads, fix packs, technotes, and the information center from the OmniFind Enterprise Edition Support site.

The following table shows the available documentation, file names, and locations.

Table 12. Documentation for enterprise search

Title	File name	Location
Information center		http://publib.boulder.ibm.com/infocenter/discover/v8r5/
Installation Guide for Enterprise Search	`iiysi.pdf`	`ES_INSTALL_ROOT/docs/locale/pdf/`
Quick Start Guide (This document is also available in hardcopy for English, French, and Japanese.)	`OmniFindEE850_qsg_two-letter locale.pdf`	`ES_INSTALL_ROOT/docs/locale/pdf/`
Administering Enterprise Search	`iiysa.pdf`	`ES_INSTALL_ROOT/docs/locale/pdf/`
Programming Guide and API Reference for Enterprise Search	`iiysp.pdf`	`ES_INSTALL_ROOT/docs/en_US/pdf/`
Troubleshooting Guide and Messages Reference	`iiysm.pdf`	`ES_INSTALL_ROOT/docs/locale/pdf/`
Text Analysis Integration	`iiyst.pdf`	`ES_INSTALL_ROOT/docs/locale/pdf/`

Accessibility features

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features

The following list includes the major accessibility features in OmniFind Enterprise Edition:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers

The OmniFind Enterprise Edition Information Center, and its related publications, are accessibility-enabled. The accessibility features of the information center are described at http://publib.boulder.ibm.com/infocenter/discover/v8r5m0/topic/com.ibm.classify.nav.doc/dochome/accessibility_info.htm.

Keyboard navigation

This product uses standard Microsoft® Windows navigation keys.

You can also use the following keyboard shortcuts to navigate and advance through the OmniFind Enterprise Edition installation program.

Table 13. Keyboard shortcuts for the installation program

Action	Shortcut
Highlight a radio button	Arrow key
Select a radio button	Tab key
Highlight a push button	Tab key
Select a push button	Enter key
Go to the next or previous window or cancel	Highlight a push button by pressing the Tab key and press Enter
Make the active window inactive	Ctrl + Alt + Esc

Interface information

The user interfaces for the administration console, sample search application, and search application customizer are browser-based interfaces that you can view in Microsoft Internet Explorer or Mozilla FireFox. See the online help for Internet Explorer or FireFox for a list of keyboard shortcuts and other accessibility features for your browser.

Related accessibility information

You can view the publications for OmniFind Enterprise Edition in Adobe Portable Document Format (PDF) using the Adobe Acrobat Reader. The PDFs are provided on a CD that is packaged with the product, or you can access them at

http://www.ibm.com/support/docview.wss?rs=63&uid=swg27010938.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

Glossary of terms for enterprise search

This glossary defines terms that are used in the enterprise search interfaces and documentation.

access control list (ACL)
In computer security, a list associated with an object that identifies all the subjects that can access the object and their access rights.

administrative role
A classification of a user that prescribes access to a user.

analysis engine
See text analysis engine.

analysis results
The information that is produced by annotators. Analysis results are written to a data structure called a common analysis structure. Analysis results produced by the custom text analysis engines (annotators) can be made available for search by inclusion in the enterprise search index.

annotation
Information about a span of text. For example, an annotation could indicate that a span of text represents a company name. In the Unstructured Information Management Architecture (UIMA), an annotation is a special kind of feature structure.

annotator
A software component that performs specific linguistic analysis tasks and produces and records annotations. An annotator is the analysis logic component in an analysis engine.

Boolean search
A search in which one or more search terms are combined by using operators such as AND, NOT, and OR.

boost class
An object that contains specifications that can influence the relative rank of a document in the search results.

boost word
A word that can influence the relative rank of a document in the search results. During query processing, the importance of a document that contains a boost word might be raised or lowered, depending on a score that is predefined for the word.

category tree
A hierarchy of categories.

certificate
In computer security, a digital document that binds a public key to the identity of the certificate owner, thereby enabling the certificate owner to be authenticated. A certificate is issued by a certificate authority and is digitally signed by that authority.

certificate authority
A trusted third-party organization or company that issues the digital

certificates used to create digital signatures and public-private key pairs. The certificate authority guarantees the identity of the individuals who are granted the unique certificate.

character normalization
A process in which the variant forms of a character, such as capitalization and diacritical marks, are reduced to a common form.

clitic A word that syntactically functions separately but is phonetically connected to another word. A clitic can be written as connected or separate from the word it is bound to. Common examples of clitics include the last part of a contraction in English (*would**n't*** or *you'**re***).

collection
A set of data sources and options for crawling, parsing, indexing, and searching those data sources.

common analysis structure (CAS)
A structure that stores the content and metadata of a document, and all analysis results that are produced by a text analysis engine. All data exchange during document analysis is handled by using the common analysis structure.

common analysis structure consumer (CAS consumer)
A consumer that does the final processing on the analysis results that are stored in the common analysis structure. For example, a consumer indexes the contents of the common analysis structure in a search engine or it populates a relational database with specific analysis results.

common communication layer (CCL)
The communication infrastructure that unites the various components (controller, parser, crawler, index server) of OmniFind Enterprise Edition.

concept extraction
A text analysis function that identifies significant vocabulary items (such as people, places, or products) in text documents and produces a list of those items. See also theme extraction.

crawl space
A set of sources that match specified patterns (such as Uniform Resource Locators (URLs), database names, file system paths, domain names, and IP addresses) that a crawler reads from to retrieve items for indexing.

crawler
A software program that retrieves documents from data sources and gathers information that can be used to create search indexes.

credential
Detailed information, acquired during authentication, that describes the user, any group associations, and other security-related identity attributes. Credentials can be used to perform a multitude of services, such as authorization, auditing, and delegation. For example, the sign-on information (user ID and password) for a user are credentials that allow the user to access an account.

custom text analysis engine
A text analysis engine that is created by using the Unstructured Information Management Architecture (UIMA) software development kit (SDK) and can be added to the set of standard enterprise search text analysis engines (also known as enterprise search base annotators). See also text analysis engine.

data source
Any repository of data from which documents can be retrieved, such as the Web, relational and nonrelational databases, and content management systems.

data source type
A grouping of data sources according to the protocol that is used to access the data.

data store
A data structure where documents are kept in their parsed form.

delta index build
In an enterprise search system, the process of adding new information to an existing index. Contrast with main index build.

dequeue
To remove items from a queue.

diacritic
A mark indicating a change in the phonetic value of a character or a combination of characters.

discoverer
A function of a crawler that determines which data sources are available for the crawler to retrieve information from.

distinguished name
The name that uniquely identifies an entry in a directory. A distinguished name consists of attribute:value pairs, separated by commas. Also, a set of name-value pairs (such as CN=person's name and C=country or region) that uniquely identifies an entity in a digital certificate.

Document Object Model
A system in which a structured document, such as an XML file, is viewed as a tree of objects that can be programmatically accessed and updated.

Domino® Document Manager cabinet
A Domino Document Manager database that is used to organize documents. Cabinets hold Domino databases.

Domino Document Manager library
A Domino Document Manager database that is the entry point to Domino Document Manager.

Domino Internet Inter-ORB Protocol (DIIOP)
A server task that runs on the server and works with the Domino Object Request Broker to allow communication between Java applets that are created with the Notes® Java classes and the Domino server. Browser users and Domino servers use DIIOP to communicate and to exchange object data.

dynamic ranking
A type of ranking in which the terms in the query are analyzed with respect to the documents that are being searched to determine the rank of results. See also text-based scoring. Contrast with static ranking.

dynamic summarization
A type of summarization in which the search terms are highlighted and the search results contain phrases that best represent the concepts of the document that the user is searching for. Contrast with static summarization.

enqueue
To put a message or item in a queue.

enterprise search administrator
An administrative role that enables a user to administer the entire enterprise search system.

enterprise search base annotators
A set of standard text analysis engines used in enterprise search for default document analysis processing.

escape character
A character that suppresses or selects a special meaning for one or more characters that follow.

external data source
A data source for federation that is not crawled, parsed, or indexed by OmniFind Enterprise Edition. Searches of external data sources are delegated to the query application programming interface of those data sources.

feature path
A path that is used to access the value of a feature in a Unstructured Information Management Architecture (UIMA) feature structure.

feature structure
The underlying data structure that represents the result of text 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, much like a Java class.

federated search
A search capability that enables searches across multiple search services and returns a consolidated list of search results.

federation
The process of combining naming systems so that the aggregate system can process composite names that span the naming systems.

field An area into which a particular category of data or control information is entered.

fielded search
A query that is restricted to a particular field.

free-form text
Unstructured text consisting of words or sentences.

free text search
A search in which the search term is expressed as free-form text.

full-text index
A data structure that references data items to enable a search to find documents that contain the query terms.

fuzzy search
A search that returns words with spelling that is similar to that of the search term.

hybrid search
A combined boolean search and free text search.

identity management
A set of enterprise search APIs that control access to secure data and

enable users to search a collection without being required to specify a user ID and password for each repository in the collection.

index See full-text index.

index queue
A list of requests for main and delta index builds to be processed.

information extraction
A type of concept extraction that automatically recognizes significant vocabulary items, such as names, terms, and expressions, in text documents.

IP address
A unique address for a device or logical unit on a network that uses the IP standard.

Java Database Connectivity (JDBC)
An industry standard for database-independent connectivity between the Java platform and a wide range of databases. The JDBC interface provides a call-level API for SQL-based database access.

JavaScript™
A Web scripting language that is used in browsers and Web servers.

JavaServer Pages (JSP)
A server scripting technology that enables Java code to be dynamically embedded within Web pages (HTML files) and executed when the page is served, in order to return dynamic content to a client.

Java virtual machine (JVM)
A software implementation of a processor that runs compiled Java code (applets and applications).

Katakana
A character set that consists of symbols that are used in one of the two common Japanese phonetic alphabets, which is used primarily to write foreign words phonetically.

key database file
See key ring. key ring.

key ring
In computer security, a file that contains public keys, private keys, trusted roots, and certificates. See also keystore file.

keystore file
A key ring that contains both public keys that are stored as signer certificates and private keys that are stored in personal certificates.

language identification
In enterprise search, a search function that determines the language of a document.

lemma
The base form of a word. Lemmas are significant in highly inflected languages such as Czech.

lemmatization
A process that identifies the root form and different grammatical forms of a word. For example, a search for mouse also finds documents that contain the word mice, and a search for go also finds documents that contain going, gone, or went.

lexical affinity
The relationship of search words in a document that are close to each other in meaning. Lexical affinity is used to calculate the relevancy of a result.

library
A system object that serves as a directory to other objects. See also Domino Document Manager library.

ligature
Two or more characters that are connected so they appear as one character. For example, ff and ffi are characters that can be presented as ligatures.

Lightweight Directory Access Protocol (LDAP)
An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and that does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). For example, LDAP can be used to locate people, organizations, and other resources in an Internet or intranet directory.

linguistic search
A search type that browses, retrieves, and indexes a document with terms that are reduced to their base form (for example, so that *mice* is indexed as *mouse*) or expanded with their base form (as with compound words).

link analysis
A method that is based on the analysis of hyperlinks between documents and used to determine what pages in the collection are important to users.

local federator
In an enterprise search application, a client object created by the search and index APIs that enables users to search a set of heterogeneous collections and obtain a unified set of search results.

Lotus® QuickPlace® place
A Web venue that is provided by Lotus QuickPlace that enables geographically dispersed participants to collaborate on projects and communicate online in a structured and secure workspace.

Lotus QuickPlace room
A partitioned area of a Lotus QuickPlace place that is restricted to authorized members who share a common interest and a need to work collectively.

main index build
In enterprise search, the process of building the entire index. Contrast with delta index build.

masking character
A character that is used to represent optional characters at the front, middle, and end of a search term. Masking characters are normally used for finding variations of a term in an index. See also wildcard character.

MIME type
An Internet standard for identifying the type of object that is being transferred across the Internet.

monitor
An enterprise search user who has the authority to observe collection-level processes.

newline character
A control character that causes the print or display position to move down one line.

n-gram segmentation
A method of analysis that considers overlapping sequences of a given number of characters as a single word rather than using blank space to delimit words as in Unicode-based white space segmentation.

no-follow directive
A directive in a Web page that instruct robots (such as the Web crawler) to not follow links found in that page.

no-index directive
A directive in a Web page that instruct robots (such as the Web crawler) to not include the contents of that page in the index.

Notes remote procedure call (NRPC)
A communication mechanism of Lotus Notes® that is used for all Notes-to-Notes communication.

operator
An enterprise search user who has the authority to observe, start, and stop collection-level processes.

parametric search
A type of search that looks for objects that contain a numeric value or attribute, such as dates, integers, or other numeric data types within a specified range.

parser A program that interprets documents that are added to the enterprise search data store. The parser extracts information from the documents and prepares them for indexing, search, and retrieval.

parser driver
In enterprise search, a service that feeds the parser service with documents. There is one parser driver for each collection. A collection's parser driver service corresponds to the collection's parser in the enterprise search administration console.

parser service
The enterprise search service that handles all document parsing and text analysis processing across document collections. At least one parser service is running at all times.

place A virtual location that is visible in the portal where individuals and groups meet to collaborate. In a portal, each user has a personal place for private work, and individuals and groups have access to a variety of shared places, which can be either public places or restricted places. See also Lotus QuickPlace place.

popular ranking
A type of ranking that raises a document's existing ranking based on the document's popularity.

Portal Document Manager (PDM)
Allows users to have one central document repository for team collaboration. Administrators have the ability to effectively manage their documents and can control the way users interact with information.

processing engine archive
A .pear zip archive file that includes a Unstructured Information

Management Architecture (UIMA) analysis engine and all of the resources required to use it for custom analysis in enterprise search.

proximity search
A text search that returns a result when two or more matching terms occur within a certain distance from each other, such as in the same sentence or paragraph.

proxy server
A server that acts as an intermediary for HTTP Web requests that are hosted by an application or a Web server. A proxy server acts as a surrogate for the content servers in the enterprise.

quick link
An association between a Uniform Resource Identifier (URI) and keywords or phrases.

ranking
The assignment of an interger value to each document in the search results from a query. The order of the documents in the search results is based on the relevance to the query. A higher rank signifies a closer match. See also dynamic ranking and static ranking.

raw data store
A data structure where crawled documents are stored before they are sent to the parser. Crawlers write to the raw data store, and the parser reads from the raw data store. When documents have been parsed, they are removed from the raw data store. Not to be confused with data store.

regular expression annotator
A software component that detects entities or units of information in a text document, such as product numbers, based on regular expressions that describe the exact patterns that are searched in the document text. If one of the regular expressions matches parts of the document text, the regular expression annotator creates the corresponding annotations that cover the match or part of it. These annotated expressions are then stored, either in the enterprise search index using an index mapping file, or a JDBC-capable database using a database mapping file.

remote federator
A server federator that federates a set of searchable objects.

Robots Exclusion Protocol
A protocol that allows Web site administrators to indicate to visiting robots which parts of their site should not be visited by the robot.

room A program that allows users to create documents for others to read, respond to comments from others, and review project status and deadlines. Users can also chat with others who are in the same room. See also Lotus QuickPlace room.

rule-based category
Categories that are created by rules that specify which documents are associated with which categories. For example, you can define rules to associate documents that contain or exclude certain words, or that match a Uniform Resource Identifier (URI) pattern, with specific categories.

search application
In enterprise search, a program that processes queries, searches the index, returns the search results, and retrieves the source documents.

search cache
A buffer that holds the data and results of previous search requests.

search engine
A program that accepts a search request and returns a list of documents to the user.

search index files
The set of files in which an index is stored in the search engine.

search results
A list of documents that match the search request.

Secure Sockets Layer (SSL)
A security protocol that provides communication privacy. With SSL, client/server applications can communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery.

security token
Information about identity and security that is used to authorize access to documents in a collection. Different data source types support different types of security tokens. Examples include user roles, user IDs, group IDs, and other information that can be used to control access to content.

seed list page
In WebSphere Portal, an XML page that contains links to the pages that are available on a portal. Crawlers use the seed list to identify the documents to crawl. The seed list page also contains metadata that is stored with the crawled documents in the enterprise search index.

start Uniform Resource Locator (URL)
The starting point for a crawl.

segmentation
The division of text into distinct lexical units. Nondictionary-based processing includes white space and n-gram segmentation, while dictionary-based support includes word, sentence, and paragraph segmentation, and lemmatization.

semantic search
A type of keyword search that incorporates linguistic and contextual analysis. See also text analysis.

servlet
A Java program that runs on a Web server and extends the server's functionality by generating dynamic content in response to Web client requests. Servlets are commonly used to connect databases to the Web.

shingle
A string of consecutive tokens (words) that are taken from a sentence. For example, from "This is a very short sentence.", the 3-word shingles (or trigrams) are:

This is a
is a very
a very short
very short sentence

Shingles can be used in statistical linguistics. For example, if two different texts have a lot of common shingles, the texts are probably related somehow.

soft error page
A type of Web page that provides information about why the requested Web page cannot be returned. For example, instead of returning a simple status code, the HTTP server can return a page that explains the status code in detail.

static ranking
A type of ranking in which factors about the documents that are being ranked, such as date, the number of links that point to the document, and so on, augment the rank. Contrast with dynamic ranking.

static summarization
A type of summarization in which the search results contain a specified, stored summary from the document. Contrast with dynamic summarization.

stemming
See word stemming.

stop word
A word that is commonly used, such as *the*, *an*, or *and*, that is ignored by a search application.

stop word removal
The process of removing stop words from the query to ignore common words and return more relevant results.

summarization
The process of including non-redundant sentences in search results to briefly describe the content of a document. See also dynamic summarization and static summarization.

synonym dictionary
A dictionary that enables users to search for synonyms of their query terms when they search a collection.

taxonomy
A classification of objects into groups based on similarities. In enterprise search, a taxonomy organizes data into categories and subcategories. See also category tree.

text analysis
The process of extracting semantics and other information from text to enhance the retrievability of data in a collection. See also semantic search.

text analysis engine
A software component that is responsible for finding and representing context and semantic content in text.

text-based scoring
The process of assigning an integer value to a document that signifies the relevance of the document with respect to the terms in a query. A higher integer value signifies a closer match to the query. See also dynamic ranking.

text segmentation
See segmentation.

theme extraction
A type of concept extraction that automatically recognizes significant vocabulary items in text documents to extract the theme or topic of a document. See also concept extraction.

token The basic textual units that are indexed by enterprise search. Tokens can be the words in a language or other units of text that are appropriate for indexing.

tokenization
The process of parsing input into tokens.

tokenizer
A text segmentation program that scans text and determines if and when a series of characters can be recognized as a token.

trailing character
A character that holds the last position in a word.

type system
The type system defines the types of objects (feature structures) that may be discovered by a text analysis engine in a document. The type system defines all possible feature structures in terms of types and features. You can define any number of different types in a type system. A type system is domain and application specific.

Unicode-based white space segmentation
A method of tokenization that uses Unicode character properties to distinguish between token and separator characters.

Uniform Resource Identifier (URI)
A compact string of characters that identifies an abstract or physical resource.

Uniform Resource Locator (URL)
The unique address of an information resource that is accessible in a network such as the Internet. The URL includes the abbreviated name of the protocol used to access the information resource and the information used by the protocol to locate the information resource.

Unstructured Information Management Architecture (UIMA)
An IBM architecture that defines a framework for implementing systems for the analysis of unstructured data.

user agent
An application that browses the Web and leaves information about itself at the sites that it visits. In enterprise search, the Web crawler is a user agent.

Web crawler
A type of crawler that explores the Web by retrieving a Web document and following the links within that document.

weighted term search
A query in which certain terms are given more importance.

wildcard character
A character that is used to represent optional characters at the front, middle, or end of a search term.

word stemming
A process of linguistic normalization in which the variant forms of a word are reduced to a common form. For example, words like *connections*, *connective*, and *connected* are reduced to *connect*.

XML Path Language (XPath)
A language that is designed to uniquely identify or address parts of source XML data, for use with XML-related technologies, such as XSLT, XQuery, and XML parsers. XPath is a World Wide Web Consortium standard.

Notices and trademarks

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 other 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 grant 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 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 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 which 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.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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.

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

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Portions of this product are:

- Oracle® Outside In Content Access, Copyright © 1992, 2008, Oracle. All rights reserved.
- IBM XSLT Processor Licensed Materials - Property of IBM © Copyright IBM Corp., 1999-2008. All Rights Reserved.

Trademarks

See http://www.ibm.com/legal/copytrade.shtml for information about IBM trademarks.

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

Adobe, Acrobat, Portable Document Format (PDF), PostScript, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Linux is a registered trademark of Linus Torvalds 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.

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

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

Index

U

W

SC18-9674-02