Using InterSystems UIMA

Version 2024.1
2024-07-02

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using InterSystems UIMA
InterSystems IRIS Data Platform Version 2024.1 2024-07-02

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

1

Using UIMA Support

Important: Implementation of UIMA within InterSystems products has been deprecated. It may be removed from future versions of InterSystems products. The following documentation is provided as reference for existing users only. Existing users who would like assistance identifying an alternative solution should contact the WRC.

Important: This version of InterSystems IRIS® does not include JAR files which are necessary to implement UIMA. Refer to Build Updated Apache UIMA JAR files for instructions on how to obtain these files.

UIMA is used to generate annotations for a source text. These annotations reference the source text by start and end position in that text. The annotations are separate from the source text and do not alter the source text.

InterSystems can perform the following operations:

- Use InterSystems IRIS Natural Language Processing (NLP) to generate UIMA text annotations.
- Access and use third-party technologies and user-written technologies that adhere to the UIMA standard to generate UIMA text annotations.
- Use multiple technologies in a pipeline architecture to apply annotations from multiple UIMA-compliant technologies in a single operation.
- Store and process annotations from multiple UIMA-compliant technologies, both annotations generated by InterSystems and annotations received from other vendors.

You can use InterSystems IRIS Natural Language Processing (NLP) independently of UIMA. This can be done concurrently with using NLP with UIMA. UIMA is an additional technology that can interface with NLP. It does not change or supersede older NLP indexing and processing.

This topic describes the following UIMA operations:

- Setting Environment Variables for dll Access
- Launching the Java Gateway
- Defining a UIMA Functional Index
  - Specifying UIMA analysis engines invoked by the Functional Index
  - Specifying a UIMA Annotation Store created by the Functional Index
- Using the Annotation Store SQL tables and their fields
- Modifying the Annotation Store by defining an XData block to add annotation tables, fields, and indexes
- Filtering which annotations are placed in the annotation table(s)

- Adding annotations manually to the annotation table(s)
- Using NLP as a UIMA analysis engine
- Using the REST interface to UIMA
- The InterSystems UIMA Type System

1.1 What is UIMA and What Can You Use It For

UIMA provides a standard for annotating unstructured data. By using this standard, different unstructured data analysis technologies, such as InterSystems IRIS Natural Language Processing (NLP), can annotate the same source text without the annotations interfering with each other or interfering with the ability to parse the source text. Thus the UIMA framework allows for the combining of annotations by different technologies that each focus on one data analysis task. For each technology, UIMA creates an analysis engine object as the interface to it; therefore, NLP interacts with UIMA through the NLP text analysis engine. Commonly, analysis engines work independently of one another, each producing its own annotations; however, UIMA provides the ability for an analysis engine to use the annotations created by another analysis engine.

The UIMA standard provides a framework for the implementation of UIMA-compliant technologies dedicated to a particular task. These technologies can support operations such as tokenizing, semantic analysis (InterSystems IRIS NLP), named-entity recognition (NER) tools for identifying names of persons and places, rule-based information extraction, speech-to-text conversion, and others.

An analysis engine produces annotations. An annotation is an encoding associated with a fragment of the source text, identified by a beginning and end character position, with a UIMA type string, and (optionally) other annotation-specific features. An analysis engine does not change the original source text.

UIMA is easy to use because deployment and scaling are handled by the UIMA framework. It handles setting up and invoking instances of these components in a possibly distributed architecture. It provides interoperability through this common annotation format.

InterSystems' implementation of UIMA is compliant with additional packages available from Apache UIMA, including Asynchronous Scaleout (UIMA-AS), Distributed UIMA Cluster Computing (UIMA-DUCC), and UIMA Ruta, a rule-based annotation workbench.

1.1.1 UIMA Glossary

Annotation: annotations within a source text inherit from the uima.tcas.AnnotationBase UIMA type and have three mandatory properties: an annotation type name (similar to a Java FQN), a start position property and an end position property, which are expressed as integer character positions from the beginning of the source text, counting from 0. Annotations that inherit from uima.tcas.TOP UIMA type are not associated with a particular section of the source text, but are instead associated with the source text itself. Annotations can have additional technology-specific properties, as defined by the implementor. Annotations can be nested and/or can refer to other annotations.

CAS: Common Analysis Structure, an in-memory object that provides cooperating UIMA components with a common representation and mechanism for shared access to the source text being analyzed. A CAS is a data structure that holds one or more Sofas; it always contains at least one Sofa.

Sofa: A CAS can have more than one "view" on the data to be processed, called a Sofa (Subject of Analysis). For example, a CAS for a web page can have one Sofa for the web page with HTML markup, and another Sofa for just the web page text. Commonly, a CAS has only one Sofa; if it has multiple Sofas, each presents a variant on the same data. For example, a source text translated into English, French, and Spanish would be one CAS with three Sofas. Annotations are always associated with an individual Sofa. A Sofa contains 1) the source text data being processed; 2) any annotations provided by Analysis Engines; 3) indexes to the annotations; 4) the type system for the annotations.

AE: Analysis Engine, a technology that analyzes unstructured data. An analysis engine is a UIMA object that implements the UIMA interface which a technology (such as NLP) interacts with. Each technology has its own analysis engine. An analysis engine is configured as a UIMA component descriptor file in XML.

Pipeline: a linear series of Analysis Engines that are executed in the order specified. Analysis engines can create annotations independent of each other, or they can take as input annotations generated by an analysis engine earlier in the pipeline and generate annotations based on the work of another analysis engine. InterSystems supports a linear pipeline. Non-linear pipelines can be supported by an aggregate analysis engine; these are not directly supported by InterSystems but are compatible with InterSystems UIMA implementation.

Type System: a uniform standard for specifying annotation type names. An XML descriptor containing a section specifying the annotation vocabulary it uses internally and the types it emits. This XML descriptor is specified in the Functional Index AEDESCRIPTOR. When invoking an analysis engine you can limit recognition of annotation types to a subset of the types supported by that analysis engine.

1.1.2 Online Resources for UIMA

The Apache UIMA home page: http://uima.apache.org/index.html

http://uima.apache.org/d/uimaj-current/references.html#ugr.ref.json.overview

http://uima.apache.org/d/uimaj-current/references.html#ugr.ref.cas

UIMA-compliant annotators that can be downloaded from Apache Software Foundation: http://uima.apache.org/sandbox.html#UIMA%20Addons%20components

Apache cTAKES, an large set of annotators specifically designed for medical patient record unstructured data: http://ctakes.apache.org/

1.2 Overview of UIMA Support in the IRIS Data Platform

InterSystems IRIS® data platform support of UIMA provides the following:

- Simple invocation of UIMA, allowing for rapid development of UIMA-compliant technologies.
- Use of InterSystems IRIS Natural Language Processing (NLP) as a UIMA-compliant technology.
- A universal UIMA annotation store. Access to UIMA annotations from InterSystems IRIS using InterSystems IRIS methods, SQL queries, or REST operations.

InterSystems provides support for invoking UIMA in the following two ways:

- Through a UIMA Functional Index, which can be configured on a regular InterSystems IRIS table. Compiling the Functional Index creates an Annotation Store which store annotations independent of the table that contains the source text.
- Directly, by invoking UIMA on source text stored in a string variable, returning the annotations as a JSON array. You can perform this operation in either of two ways:
  - The **%UIMA.Utils.IndexNow()** method.
  - Using REST, supply a JSON object with the properties "data" and "descriptors" to retrieve annotations. Optionally, add a "markup" property to also retrieve marked-up versions of the Sofa source text.

In either scenario, InterSystems provides support for persisting the output in a UIMA Annotation Store. Providing a persistent independent Annotation Store is an InterSystems extension to the Apache UIMA standard.

1.2.1 The UIMA Functional Index

The UIMA functional index is an InterSystems SQL table index. It indexes the contents of a single column in an SQL source table. This index automatically loads data into a UIMA workflow from this source text column.

The InterSystems UIMA functional index class, %UIMA.Index implements the %Library.FunctionalIndex specification.

Once a functional index has been defined and compiled, adding records to the SQL source table results in the source text column data being processed by the specified UIMA analysis engine (or a linear series (pipeline) of analysis engines). An INSERT, UPDATE, DELETE, or a %BuildIndices() operation automatically revises the functional index, invoking the UIMA analysis engine (or engines).

Defining a Functional Index is described below.

1.2.2 The UIMA Annotation Store

Upon compiling a UIMA functional index, the system automatically generates several persistent classes to store UIMA annotations. This UIMA annotation store is an InterSystems extension to the UIMA framework. It stores UIMA output in persistent classes (SQL tables) that are separate from the source text tables, but linked to those tables. The InterSystems Annotation Store provides a flexible SQL-based annotation storage for later analysis of unstructured data across multiple source texts. This avoids the large XML annotation files created by standard UIMA.

The UIMA annotation store can be used as an annotation store for the annotations generated by InterSystems IRIS Natural Language Processing (NLP) and other analysis engines specified in the Functional Index.

The UIMA annotation store can also be used independent of the Functional Index to store additional manual annotations based on user review of the source text.

1.2.3 Using NLP as a UIMA Analysis Engine

You can use InterSystems IRIS Natural Language Processing (NLP) as a UIMA analysis engine, generating UIMA annotations for NLP Concepts and Relations. These annotations are fully compatible with UIMA annotations supplied by other UIMA technologies.

You can also use NLP independently of UIMA, generating InterSystems IRIS globals for NLP Concepts and Relations.

Because UIMA provides UIMA-compliant indexing for NLP results, but does not change the NLP engine itself, these two uses of NLP can be performed concurrently on the same unstructured data sources. Refer to Using InterSystems IRIS Natural Language Processing (NLP) for further details.

1.3 How to Use the UIMA Integration

1.3.1 Build Updated Apache UIMA JAR files

This version of DOCBOOKMACRO(prod) does not include JAR files which are necessary for UIMA integration.

Before you can use the UIMA integration capabilities for this version of DOCBOOKMACRO(prod), you must obtain updated versions of these JAR files from the `main-v2` branch of the Apache UIMA SDK repository.

If you have Apache Maven and git installed on your system, you can perform the following steps:

1. From a command line, navigate to a directory of your choosing and clone the `main-v2` branch of the Apache UIMA SDK GitHub repository into it by issuing the following command:

   ```
   git clone -b main-v2 https://github.com/apache/uima-uimaj.git
   ```

2. Navigate into the uima-uimaj/ directory.
3. Build the SDK by issuing the following command:

   ```
   mvn install
   ```

4. Once the build is complete, copy the necessary JAR files from the SDK kit into the appropriate location in your DOCBOOKMACRO(prod) installation directory. To do so, you can issue the following commands, replacing *<installDir>* with the base directory for your DOCBOOKMACRO(prod) instance:

   ```
   cp uimaj-core/target/uima-core.jar <installDir>/dev/java/lib/uima/uimaj-core-2.10.3.jar
   cp uimaj-json/target/uimaj-json*.jar <installDir>/dev/java/lib/uima/uimaj-json-2.10.3.jar
   ```

1.3.2 Library Settings

Before launching the Java Gateway, you must make certain dll library files available to the system loader by adjusting environment variables.

Library settings for running the InterSystems IRIS Natural Language Processing (NLP) as a UIMA analysis engine: The NLP engine is written in C++ and is exposed as a UIMA Analysis Engine through the UIMACPP bridge. (See https://github.com/apache/uima-uimacpp). This allows the Java-based UIMA framework to call on C++ libraries using JNI, which requires a few platform-specific settings. See also https://uima.apache.org/d/uimacpp-current/docs/overview_and_setup.html.

- Windows platforms: On Windows, make sure the PATH environment variable contains the /bin directory of your InterSystems IRIS install, either by setting it as a system environment variable, or by configuring it in the terminal window from which the Java Gateway is launched.
- Linux & UNIX platforms: On Linux & UNIX, the /bin subdirectory of your InterSystems IRIS install needs to be appended to both the PATH and LD_LIBRARY_PATH environment variables, either at the system level or in the terminal session from which the Java Gateway is launched.
- macOS platforms: macOS' SIP (Security Integral Protection) puts additional constraints on what libraries can be loaded dynamically. More specifically, when Java was installed in a system folder, SIP will not allow it to load non-system libraries, such as the UIMACPP libraries the NLP Analysis Engine relies on. To avoid this problem, install or copy Java into a non-system folder and use that non-system path to launch the Java Gateway, setting the DYLD_LIBRARY_PATH as described above. Note that macOS is only supported as a development platform and is not supported for production.

To locate the /bin directory, call the **BinaryDirectory()** method of the %SYSTEM.Util class.

1.3.3 Launching the Java Gateway

You must have the Java Gateway running on port 5555 to compile a UIMA functional index. This Java Gateway provides access to the InterSystems UIMA Java Gateway.

The Java Gateway is an InterSystems IRIS feature that allows invocation of Java classes from ObjectScript. It runs as a daemon process listening on a TCP/IP port. It spawns off threads to execute the Java code as requested through messages sent from ObjectScript through %Net.Remote.Gateway code.

The UIMA Java Gateway com.intersystems.uima.Gateway is a Java class that holds the public methods our UIMA integration presents to ObjectScript. It is exposed to InterSystems IRIS through the Java Gateway.

The Java Gateway needs to be running on the same host as the InterSystems IRIS instance (localhost), listening on port 5555. No additional classpath settings should be provided when launching the JVM (Java Virtual Machine), but regular JVM parameters for setting minimal and maximum memory may be supplied.

On a Windows system you can start the Java Gateway by executing the following command from the Windows **Run** interface:

```
%JAVA_HOME%\bin\java -classpath
"C:\InterSystems\IRIS\dev\java\lib\JDK18\*;C:\InterSystems\IRIS\dev\java\lib\jackson\*;C:\InterSystems\IRIS\dev\java\lib\uima\*"
 com.intersystems.gateway.JavaGateway 5555
```

For this command to function, you must have defined JAVA_HOME. If JAVA_HOME is not defined on your Windows system, go to the Control Panel, System option, Advanced system settings. Select the Environment Variables button. Define a new system variable named JAVA_HOME. Browse to the path to your Java bin directory, and assign this path as the JAVA_HOME value. For example, `C:\Program Files\Java\jre1.8.0_141`.

1.3.4 Defining a UIMA Functional Index

You must define a functional index to allows users to use UIMA process unstructured data stored in an SQL table column. A UIMA functional index is an index of type %UIMA.Index.

The following example defines the NYT.Articles SQL table as a persistent class (a table of article texts from the New York Times) and defines a UIMA index on the FullArticle column:

Class Definition

```
Class NYT.Articles Extends %Persistent
{
Property NYTID As %Integer;
Property PubDate As %Date;
Property FullArticle As %String(MAXLEN=32000);
Index IdxNYTArticles On (FullArticle) As %UIMA.Index(
    AEDESCRIPTOR = "classpath:/com/intersystems/uima/annotator/iKnowEngine.xml"
    );
}
```

This Functional Index takes the following parameters:

- AEDESCRIPTOR — *Mandatory* — Specifies a UIMA component descriptor that designates which UIMA analysis engines need to be executed to process the column data. Specified as either:
  - The name of an XData block in the current class that contains the full XML for the descriptor. For example, `"iKnowEngine"`. For further details, refer to Defining and Using XData Blocks.
  - The path to a descriptor file accessible on the classpath, prefixed with `classpath:`. For example, `"classpath:/com/intersystems/uima/annotator/iKnowEngine.xml"`
  - The full path to a descriptor file. For example, on a Windows system `"C:\UIMA\Descriptors\MyAE.xml"` or on a UNIX system `"//user/UIMA/Descriptors/MyAE.xml"`

  You can specify a UIMA component descriptor for a single UIMA analysis engine as a quoted string, or specify multiple semicolon-separated UIMA component descriptor strings. Multiple UIMA analysis engines are executed in the order specified, creating a UIMA processing pipeline. You do not need to specify the Annotation Filer in the list of descriptors, it is automatically provided as the last analysis engine in the pipeline.
- ADDITIONALCLASSPATH — *Optional* — The additional class path(s) that specifies any additional libraries (jarfiles) that need to be loaded into the classpath in order to run the analysis engine(s) described in AEDESCRIPTOR. Multiple paths are specified as a semicolon-separated list.

  Alternatively, you can add the additional class paths to the Java Gateway classpath by adding them to the -classpath command-line argument when launching the Gateway.

- ANNOTATIONSTOREDEF — *Optional* — The name of an XData block defining additional features and indexes to configure for the Annotation Store. If not specified, the default is to use an XData block with the same name as the index. The XData block must be in the same class as the Annotation Store definition XML. For further details, refer to Defining and Using XData Blocks.

The table on which the functional index is applied needs to be compiled. This generates a number of methods on the table itself that are specific to the %FunctionalIndex framework. It also validates the Annotation Store descriptor XML (ANNOTATIONSTOREDEF) and, if it passes, generate the appropriate ObjectScript classes based on it.

Compiling the table also tests the AEDESCRIPTOR referenced from the functional index. This test is performed through a call to the Java Gateway. Therefore, by default, the Java Gateway needs to be running when you compile a table with a UIMA functional index.

You can perform this table compile without an active Java Gateway by setting the index parameter TESTONCOMPILE=0. This suppresses AEDESCRIPTOR testing. However, an active Java Gateway is required when you insert data into the table.

1.3.5 Invoking a UIMA Component Directly

You can invoke a UIMA component directly by using the **%UIMA.Utils.IndexNow()** method.

1.4 How to Use the UIMA Annotation Store

Compiling a UIMA functional index automatically generates a UIMA Annotation Store. A UIMA Annotation Store is a package containing a set of persistent classes (SQL tables).

The Annotation Store is configured through a piece of XML that is supplied through the functional index or directly through the annotation filer. Refer to %UIMA.Model.annotationStore for configuration options for the annotation store.

Annotation Store classes inherit from %UIMA.AnnotationStore.* superclasses and are generated by %UIMA.AnnotationStore.ClassGenerator.

1.4.1 The Annotation Filer

The UIMA Annotation Filer (com.intersystems.uima.filer.AnnotationFiler) is a Java class that implements the UIMA Analysis Engine interface and therefore acts as a regular UIMA component. The UIMA Annotation Filer is itself an analysis engine, the last analysis engine in a UIMA processing pipeline. Rather than adding annotations by itself, it reads all existing annotations from the supplied CAS (in-memory text object). It sends these annotations back to InterSystems IRIS for storing in the UIMA Annotation Store. This is sometimes referred to as a CAS Consumer in UIMA terminology.

Like any other analysis engine, the Annotation Filer is configured through a UIMA component descriptor file in XML, containing database connection parameters, an identifier for the Annotation Store to file the data into, and the XML description of the annotation store.

Note: All of these Annotation Filer parameters are configured automatically by the UIMA Functional Index.

1.4.2 Annotation Store Tables

Compiling a UIMA functional index automatically generates a UIMA Annotation Store. A UIMA Annotation Store is a set of InterSystems IRIS tables that stores all the annotations for a given dataset processed by a UIMA pipeline. This pipeline can consist of one or more UIMA analysis engines, with the UIMA Annotation Filer as the last engine in the pipeline.

By default, the Annotation Store is given the same name as the persistent class for which the index is defined. Therefore, in our example, compiling a functional index for a column of the table `NYT.Articles` would produce a corresponding Annotation Store package called `NYT_Articles`. This naming default is modifiable.

The UIMA Annotation Store consists of (at minimum) the following set of tables:

- A table for UIMA types. For example, `NYT_Articles.Type`
- A table for Sofas (the text objects). For example, `NYT_Articles.Sofa`
- A table for the annotations themselves. For example, `NYT_Articles.Annotation`

Because these are SQL tables, you can access their contents with standard SQL queries.

1.4.2.1 Type Table

The Type table consists of the following fields:

Field	Data Type	Indexed?	Purpose
name	String	unique index	The type of annotation.
parent	Integer		References Type table.

1.4.2.2 SofaTable

The Sofa table consists of the following fields:

Field	Data Type	Indexed?	Purpose
docID	Integer	bitmap index	References source text table.
hasManualAnnotations	Boolean	bitmap index	Specifies if there are any manual annotations.
mimeType	String		MIME type (also known as media type), an ISO standard description of the type of the data represented by the Sofa.
sofaID	String	bitmap index	
sofaString	String		The source text for this Sofa.

1.4.2.3 Annotation Table

The Annotation table consists of the following fields:

Field	Data Type	Indexed?	Purpose
begin	Integer	%pos standard index	The beginning character position for the annotation text. Positions are a count of Unicode characters, counting from 0.
coveredText	String		The annotation text. An exact copy of the section of source text identified by the `begin` and `end` character positions.
docID	Integer	bitmap index	References source text table.
end	Integer	%pos standard index	The ending character position for the annotation text.
isManual	Boolean	bitmap index	Flags a manual annotation.
sofaID	Integer		References Sofa table.
typeID	Integer	bitmap index	References Type table.

A Top Annotation table does not include the `begin`, `coveredText`, and `end` fields.

An annotation table can receive additional field values generated by the analysis engine. If no Annotation fields have been defined for these additional field values, they are stored in a generic features field as a JSON array of key:value pairs.

You can access annotations using the %UIMA.AnnotationStore.Store methods **GetAnnotations()** and **GetAnnotationsRS()**

1.4.3 Refining the Annotation Store

By default, the Annotation Store is given a package name that is the same as the persistent class for which the index is defined. You can optionally change the Annotation Store package name and add features to the annotation store by specifying an Xdata block. You can use this XData block to define additional annotation store features, including additional tables, columns, indexes, and filters. You can do this in either of two ways:

- Create an XData block with the same name as the functional index and set the XData block XMLNamespace keyword to a UIMA annotation store. For example:

```
Class NYT.Articles Extends %Persistent
{
Property NYTID As %Integer;
Property PubDate As %Date;
Property FullArticle As %String(MAXLEN=32000);
Index IdxNYTArticles On (FullArticle) As %UIMA.Index(
    AEDESCRIPTOR = "classpath:/com/intersys/uima/annotator/iKnowEngine.xml"
    );
XData IdxNYTArticles [ XMLNamespace = "http://www.intersystems.com/UIMA/annotationStore" ]
  { ... }
}
```

- Define an XData block with a different name and specify it in the ANNOTATIONSTOREDEF parameter of the Functional Index. For example:

```
Class NYT.Articles Extends %Persistent
{
Property NYTID As %Integer;
Property PubDate As %Date;
Property FullArticle As %String(MAXLEN=32000);
Index IdxNYTArticles On (FullArticle) As %UIMA.Index(
    AEDESCRIPTOR = "classpath:/com/intersys/uima/annotator/iKnowEngine.xml",
    ANNOTATIONSTOREDEF = "NYTExtra"
    );
XData NYTExtra [ XMLNamespace = "http://www.intersystems.com/UIMA/annotationStore" ]
  { ... }
}
```

The XData block contains XML-formatted data. For further details, refer to Defining and Using XData Blocks.

The UIMA Annotation Store may contain additional annotation tables, additional columns in those tables, and additional indexes on them.

1.4.3.1 Changing Names

You can specify a different package name for the UIMA Annotation Store in the XData block, as follows:

```
XData IdxNYTArticles [ XMLNamespace = "http://www.intersystems.com/UIMA/annotationStore" ]
  {<store package="NYT.ArticleAS">
    </store>
  }
```

You can change the name of the Annotation Table in the XData block, as follows:

```
XData IdxNYTArticles [ XMLNamespace = "http://www.intersystems.com/UIMA/annotationStore" ]
  {<store package="NYT.Articles">
   <tables>
     <table name="FilteredAnnotation">
     </table>
   </tables>
   </store>
  }
```

1.4.3.2 Specifying Additional Tables

You can specify additional Annotation Store tables to store annotations. For example, you could create annotation tables to store the annotations from two different analysis engines in different annotation tables. The following XData block creates two Annotation tables; the second annotation table has an additional field named normalizedValue:

```
XData IdxNYTArticles [ XMLNamespace = "http://www.intersystems.com/UIMA/annotationStore" ]
  {<store package="NYT.Articles">
   <tables>
     <table name="Annotation1">
     </table>
     <table name="Annotation2">
       <features storeOther="json" >
         <feature name="normalizedValue" path="normalizedValue" >
         <parameter key="MAXLEN">300</parameter>
         </feature>
       </features>
     </table>
   </tables>
    </store>
   }
```

In %UIMA.Model.annotationStore, specify additional annotation tables in the tables property. You can define a table using %UIMA.Model.table.

You may need to create an additional table for top level annotations. The topLevel boolean property allows you to specify whether an annotation table contains annotations within the source text (annotations on a unit of text defined with a beginning and an end character position), or whether it is a table of “top” annotations that apply to the entire source text.

1.4.3.3 Specifying Additional Columns

You may wish to add columns to an Annotation Table. For example, in a Top Annotation Table you may wish to add a field for the NLP Dominance score.

If the analysis engine generates annotation fields that do not correspond to existing Annotation Table fields, these values are stored in a generic features field as a JSON array of key:value pairs.

To add columns, specify each column as a feature property in the XData block supplied to the functional index. A feature must be defined within features within a table. Defining one or more additional columns also automatically defines a generic features column. The following example defines three additional fields, plus the generic features field:

```
XData IdxNYTArticles [ XMLNamespace = "http://www.intersystems.com/UIMA/annotationStore" ]
 {
 <store package="NYT.Articles">
 <tables>
 <table>
   <features storeOther="json" >
     <feature name="normalizedValue" path="normalizedValue" >
    <parameter key="MAXLEN">300</parameter>
    </feature>
    <feature name="occurrences" path="occurrences" type=":annotationList:Annotation" />
    <feature name="parent" path="_parent" type=":annotation" />
  </features>
 </table>
 </tables>
 </store>
 }
```

1.4.3.4 Specifying Additional Indexes

In %UIMA.Model.table, specify additional annotation table indexes in the indices property. You can define an index using %UIMA.Model.index.

These additional indexes must be supplied to an XData block that you specify in the %UIMA.Index ANNOTATIONSTORE-DEF parameter. An index must be defined within a table. The following example defines an additional field and indexes that field:

```
XData IdxNYTArticles [ XMLNamespace = "http://www.intersystems.com/UIMA/annotationStore" ]
 {
 <store package="NYT.Articles">
 <tables>
  <table>
   <features storeOther="json" >
     <feature name="normalizedValue" path="normalizedValue" >
     <parameter key="MAXLEN">300</parameter>
     </feature>
   </features>
   <indices>
     <index properties="normalizedValue" name="NV" />
   </indices>
  </table>
 </tables>
 </store>
 }
```

1.4.4 Adding Annotation Filters

By default, all generated annotations are stored in the Annotation Store. You can specify one or more filters to apply to the annotations generated by the analysis engines. These filters instruct the Annotation Filer to store only annotations of the UIMA types specified in the filter. Note that by applying a filter you automatically exclude all annotations other than those explicitly specified in the filter.

A filter is supplied to an XData block that you specify in the %UIMA.Index ANNOTATIONSTOREDEF parameter. The following is an example XData block specifying a filter. Note that a filter is defined within store, but not within a table:

```
XData IdxNYTArticles [ XMLNamespace = "http://www.intersystems.com/UIMA/annotationStore" ]
 {
 <store package="NYT.Articles">
   <filters>
     <include>
     <exclude pattern="org.apache.uima.alchemy.ts.entity.AlchemyAnnotation" />
     </include>
   </filters>
   <tables>
     <table name="Annotation1">
     </table>
   </tables>
 </store>
 }
```

See %UIMA.Model.filters and %UIMA.Model.filterRule.

1.4.5 Manual Annotations

After running an analysis engine on a source text, you may discover that there are additional annotations that you wish to include for that source text. You can supply these annotations directly to the Annotation Store as manual annotations.

You can use the **%UIMA.AnnotationStore.Store.FileAnnotation()** method to manually insert a single annotation into the Annotation Store.

You can use the **%UIMA.AnnotationStore.Store.FileAnnotations()** method to manually insert an array of annotations into the Annotation Store.

UIMA flags manual annotations in the Annotation table using the `isManual` Boolean field.

By default, the Functional Index deletes all prior annotations for a source text before adding the annotations generated by the specified analysis engines. If you wish to preserve prior annotations, you must flag these as manual annotations.

1.5 How to Use NLP as a UIMA Analysis Engine

The InterSystems IRIS Natural Language Processing (NLP) engine is exposed as a UIMA annotator and is already loaded on the classpath, accessible at "classpath:/com/intersys/uima/annotator/iKnowEngine.xml". You can use NLP as a UIMA annotator by defining a UIMA functional index. This does not in any way limit the concurrent use of NLP independent of UIMA, which defines NLP indexes, which are stored as InterSystems IRIS globals.

When using NLP as a UIMA annotator, NLP places the results of its syntactic analysis into a compileable UIMACPP header file, then includes this data in an NLP UIMA wrapper, and sends it to the NLP engine. This allows NLP to perform processing on UIMA annotations as if they were InterSystems IRIS globals. The NLP engine itself is not changed.

NLP annotation types include Concepts, Relations, Negation, Positive Sentiment, Negative Sentiment. Top annotation types can include unique entities and dominance score.

The NLP UIMA wrapper code imports the UIMACPP (UIMA C++) library, which has dependencies on the Xerces and APR-1 libraries. Xerces is already part of the InterSystems IRIS code base. UIMACPP and APR-1 are provided as part of InterSystems UIMA implementation. Both are maintained by the Apache Software Foundation.

1.5.1 The NLP Annotation Type System

NLP supports the Sofa Unicode text data format, which has a metadata parameter: language. NLP uses this language parameter to choose the corresponding language model. A language is specified using the two-letter ISO language codes; for example, `en` for English. Because a Sofa is identified as containing source text in a single language, NLP Automatic

Language Identification (ALI) is not supported. The language identified for the Sofa is considered a constant for NLP UIMA indexing.

1.6 The UIMA REST API

The UIMA REST API, like its equivalents for Analytics and NLP, has a default generic web application that gets forwarded for various namespaces in %Api.UIMA, but an individual version of the API can be locked to another web app through the %UIMA.REST.v1 class.

1.6.1 REST API Basics

%UIMA.REST.v1 provides endpoints for accessing UIMA functionality over REST. You should set up a REST service at the following URL, using the <baseURL> for your instance:

```
https:<baseURL>/api/uima/v1/<namespace>
```

The following REST operations are supported:

- Retrieve Annotation Store information, including the names of the Annotation Store tables, whether a table contains standard annotations or top-level annotations, their columns, indexes, and filters. Optionally includes runtime statistics including the annotation count (row count) and a list of annotation types contained in the table:

  ```
  GET http://localhost:52773/api/uima/v1/<namespace>/store/<annotation store name>/info
  ```

  Both GET and POST are supported. Retrieves information based on AnnotationStoreInfo parameters that can be supplied through a POST object.

- Retrieve annotations:

  ```
  GET http://localhost:52773/api/uima/v1/<namespace>/store/<annotation store name>/annotations
  ```

  Both GET and POST are supported. Retrieves a set of annotations based on ASRequestObject parameters that can be supplied through a POST object. You can specify the maximum number of annotations to retrieve; the default is 500. You can specify what document IDs, annotation tables, columns, and annotation types to retrieve; the default is to retrieve all. Document IDs are specified as a comma-separated list of integer values; there is no default value for document IDs. Results are returned as an array with the same parameters as the **GetAnnotations()** method.

- Retrieve a document and its details:

  ```
  GET http://localhost:52773/api/uima/v1/<namespace>/store/<annotation store name>/document/<docID>
  ```

  Both GET and POST are supported. Retrieves the document source text. Can also retrieve the annotations of the document based on ASRequestObject parameters supplied through a POST object. Adding Markup parameters to the POST object highlights the text using the annotations. By default, Markup uses CSS (Cascading Style Sheet) classes.

- Processing a piece of source text:

  ```
  POST http://localhost:52773/api/uima/v1/<namespace>/process
  ```

  Specify a piece of source text and a UIMA component descriptor to process it. Through supplying ASRequestObject parameters and Markup parameters, the resulting annotations and marked-up version of the document can be requested.

1.6.2 Accessing the Swagger Reference Documentation

The UIMA REST API is fully documented using the OpenAPI Specification (also known as Swagger). The description in YAML is available from the "/swagger" endpoint and can be loaded directly into swagger-ui for convenient GUI capabilities on top of this API.

To use it, either install swagger-ui or go to http://petstore.swagger.io and point to this endpoint.

- Download swagger-ui
- Unzip the swagger-ui download.
- In the Swagger box specify either:
  - for InterSystems IRIS installed with Minimal security: `https:<baseURL>/api/uima/v1/samples/swagger` (substituting the desired namespace for `samples`).

    This uses the <baseURL> for your instance.
  - for InterSystems IRIS installed with Normal security: `https:<baseURL>/api/uima/v1/samples/swagger?IRISUsername=myname&IRISPassword=mypassword` (substituting the desired namespace for `samples`, your InterSystems IRIS user name for `myname`, and your InterSystems IRIS account's password for `mypassword`).
- Select the Explore button.

1.7 Reference Material

1.7.1 InterSystems UIMA Type System

InterSystems supports two annotation type system: for annotations within source text, and for top annotations that apply to the entire source text:

1.7.1.1 Annotations within Text

- *uima.tcas.Annotation*:

  Subtypes:
  - *com.intersystems.uima.annotation.iknow.Base*: Abstract supertype for all regular annotations produced by NLP

    Subtypes:
    - *com.intersystems.uima.annotation.iknow.EntityOccurrence*: An occurrence of an Entity in the text itself

      { entity (com.intersystems.uima.annotation.iknow.Entity): Reference to the document-level Entity annotation occurring in the text. }
    - *com.intersystems.uima.annotation.iknow.Sentence*: A sentence as identified by NLP
    - *com.intersystems.uima.annotation.iknow.Path*: A semantically relevant sequence of entities, as a subset of a Sentence.

      { entities (uima.cas.FSArray[com.intersystems.uima.annotation.iknow.EntityOccurrence]): The entities making up this Path }

- *com.intersystems.uima.annotation.iknow.Attribute*: A semantic attribute affecting several entities in a Path or Sentence

  Subtypes:

  – *com.intersystems.uima.annotation.iknow.Negation*: Semantic Attribute expressing negation.

  – *com.intersystems.uima.annotation.iknow.PositiveSentiment*: Semantic Attribute expressing positive sentiment.

  – *com.intersystems.uima.annotation.iknow.NegativeSentiment*: Semantic Attribute expressing negative sentiment.

- *com.intersystems.uima.annotation.iknow.AttributeMarker*: The phrase implying a semantic attribute.

  { scope (com.intersystems.uima.annotation.iknow.Attribute): Reference to the semantic attribute annotation this marker implies. }

  Subtypes:

  – *com.intersystems.uima.annotation.iknow.NegationMarker*: Semantic Attribute marker expressing negation.

  – *com.intersystems.uima.annotation.iknow.PositiveSentimentMarker*: Semantic Attribute marker expressing positive sentiment.

  – *com.intersystems.uima.annotation.iknow.NegativeSentimentMarker*: Semantic Attribute marker expressing negative sentiment.

- *com.intersystems.uima.annotation.iknow.util.UDLabel*: Abstract supertype for user-defined labels.

  Subtypes:

  – *com.intersystems.uima.annotation.iknow.util.UDConcept*: User-defined label for what should be interpreted as a Concept by the NLP engine.

  – *com.intersystems.uima.annotation.iknow.util.UDRelation*: User-defined label for what should be interpreted as a Relation by the NLP engine.

  – *com.intersystems.uima.annotation.iknow.util.UDNegation*: User-defined label for what should be interpreted as a negation marker by the NLP engine.

  – *com.intersystems.uima.annotation.iknow.util.UDPositiveSentiment*: User-defined label for what should be interpreted as a positive sentiment marker by the NLP engine.

  – *com.intersystems.uima.annotation.iknow.util.UDNegativeSentiment*: User-defined label for what should be interpreted as a negative sentiment marker by the NLP engine.

1.7.1.2 Top Annotations

- *uima.cas.TOP*:

  Subtypes:

  – *com.intersystems.uima.annotation.iknow.TOP*: Abstract supertype for all document-level annotations produced by NLP

    Subtypes:

    - *com.intersystems.uima.annotation.iknow.Entity*: The main annotation type produced by NLP, identifying semantically relevant word groups. See subtypes Concept, Relation and PathRelevant

{ normalizedValue (uima.cas.String): Normalized value of the entity (lowercased and stripped of irrelevant punctuation)

dominance (uima.cas.Double): NLP-specific metric expressing the relevance of this entity within the document. }

Subtypes:

- *com.intersystems.uima.annotation.iknow.Concept*: Concepts are Entities that have a meaning by themselves, as expressed by the text author.
- *com.intersystems.uima.annotation.iknow.Relation*: Relations are entities expressing a link or relationship between Concepts (or PathRelevants) in the same Path or Sentence
- *com.intersystems.uima.annotation.iknow.PathRelevant*: PathRelevant terms are terms that have a role in a Path, but no semantic meaning by themselves. These entities will not have a dominance value assigned.

- *com.intersystems.uima.annotation.iknow.ProximityScore*: The proximity score for a given pair of entities, expressing how closely they are related semantically.

  { origin (com.intersystems.uima.annotation.iknow.Concept)

  destination (com.intersystems.uima.annotation.iknow.Concept)

  proximity (uima.cas.Double) }