OHF XDS Document Source

Architecture & API Documentation

Version 0.0.1

seknoop[AT]us[DOT]ibm[DOT]com | Sarah Knoop

Contents

1. Introduction

The Eclipse Foundation is a not-for-profit corporation formed to advance the creation, evolution, promotion, and support of the Eclipse Platform and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services. Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software.

www.eclipse.org

The Eclipse Open Healthcare Framework (EOHF) is a project within Eclipse formed for the purpose of expediting healthcare informatics technology. The project is composed of extensible frameworks and tools which emphasize the use of existing and emerging standards in order to encourage interoperable open source infrastructure, thereby lowering integration barriers.

www.eclipse.org/ohf

The Integrating the Healthcare Enterprise (IHE) is an initiative by healthcare professionals and industry to improve the way computer systems in healthcare share information. IHE promotes the coordinated use of established standards such as DICOM and HL7 to address specific clinical needs in support of optimal patient care. Systems developed in accordance with IHE communicate with one another better, are easier to implement, and enable care providers to use information more effectively.

www.ihe.net

The IHE Technical Frameworks are a resource for users, developers and implementers of healthcare imaging and information systems. They define specific implementations of established standards to achieve effective systems integration, facilitate appropriate sharing of medical information and support optimal patient care. They are expanded annually, after a period of public review, and maintained regularly by the IHE Technical Committees through the identification and correction of errata.

http://www.ihe.net/Technical_Framework/index.cfm

This document describes the current release of the Eclipse OHF plugin implementation of the IHE ITI Technical Framework XDS Profile Document Source Actor. This implementation supports the following IHE Transactions: ITI-15: Provide and Register Document Set Transaction.

2. Getting Started

2.1 Platform Requirements

Verify that the following platform requirements are installed on your workstation, and if not follow the links provided to download and install.

Eclipse SDK 3.2, or later	http://www.eclipse.org/downloads/
Java JDK 1.4.2, or later	http://java.sun.com/javase/downloads/index.jsp
Eclipse Modeling Framework 2.2.0	http://www.eclipse.org/emf/

2.2 Source Files

Information on how to access the Eclipse CVS technology repository is found on the eclipse wiki:

http://wiki.eclipse.org/index.php/CVS_Howto

Download from dev.eclipse.org/technology/org.eclipse.ohf/plugins

- org.eclipse.ohf.ihe.xds.source

For details regarding plugin contents, see the README.txt located in the resources/doc folder of each plugin.

2.3 Dependencies

2.3.1 Other OHF Plugins

Plugin dependencies include the following from dev.eclipse.org/technology/org.eclipse.ohf/plugins

- org.apache.log4j — Debug, warning and error logging
- org.apache.xerxes — Support serialization of XML messages
- org.eclipse.ohf.ihe.atna.audit — Support of audit events for relevant transactions
- org.eclipse.ohf.ihe.common.ebXML._2._1 — Model of ebXML v2.1: rim, query and rs
- org.eclipse.ohf.ihe.xds.metadata — Model to represent XDS metadata
- org.eclipse.ohf.ihe.xds.metadata.extract — Model to render XDS metadata from other formats
- org.eclipse.ohf.ihe.xds.metadata.transform — Model to render XDS metadata to other formats
- org.eclipse.ohf.ihe.xds.soap — SOAP messaging support for transactions
- org.apache.axis — Apache Axis 1.3 to support the above plugin

2.3.2 External Sources

At this time, the org.eclipse.ohf.ihe.xds.soap plugin listed above as a dependency for the XDS Source is dependant on the Axis 1.3 API. This API does not provide an implementation of the javax.acitvation.DataHandler nor the javax.mail.internet.MimeMultipart packages needed for compilation and SOAP with attachment support. Java Mail 1.3.3 (mailapi.jar) and Java Activation Framework 1.0.2

(activation.jar) must be downloaded separately and incorporated into the build path or plugin dependencies list for this plugin. Take note of exact versions of these jars. These .jars can be found at

JAF v1.0.2:

http://java.sun.com/products/archive/javabeans/jaf102.html

JAVA MAIL v1.3.3:

http://java.sun.com/products/javamail/javamail-1_3_3.htmlResources

Additionally, we are experiencing problems with the Axis 1.3 support for SOAP Attachments (for explicit details see the following posting: http://mail-archives.apache.org/mod_mbox/ws-axisuser/200607.mbox/%3cb50b8b100607191155s4c7159e7i20f5c3b8bf4434c8@mail.gmail.com%3e). We have yet to hear a response to this posting, and for the time being the following steps are needed to enable attachment support (needed when using the XDS Document Source).

1. Download Axis 1.3 source code from: http://www.apache.org/dyn/closer.cgi/ws/axis/1_3
2. Unzip the content into a directory of your choosing.
3. In org/apache/axis/attachments/AttachmentsImpl.java comment out line 229 and lines 586 through 595 (Specifically, the operation being removed is "multipart = null;").
4. Re-compile this class and overwrite the existing class file in the axis.jar included in the org.apache.axis plugin for OHF.

2.4 Resources

2.4.1 Other OHF Plugin Documentation

The following OHF plugin documents are related to the OHF XDS Document Consumer:

- OHF ATNA Audit Client
- OHF XDS Metadata Model
- OHF XDS SOAP Client

2.4.2 Automatic XDS Metadata Extraction from CDA R2 Documents

The OHF XDS Document Source has the ability to be configured with Metadata Extractors for any given document type. The next version of the OHF XDS Document Source will include a Metadata Extractor that is able to extract XDS Metadata from HL7 CDA R2 schema conformant documents. Detailed documentation of this extraction process will be provided in a separate document at that time.

2.4.3 IHE ITI Technical Framework

Nine IHE IT Infrastructure Integration Profiles are specified as Final Text in the Version 2.0 ITI Technical Framework: Cross-Enterprise Document Sharing (XDS), Patient Identifier Cross-Referencing (PIX), Patient Demographics Query (PDQ), Audit trail and Node Authentication (ATNA), Consistent Time (CT), Enterprise User Authentication (EUA), Retrieve Information for Display (RID), Patient Synchronized Applications (PSA), and Personnel White Pages (PWP).

The IHE ITI Technical Framework can be found on the following website: http://www.ihe.net/Technical_Framework/index.cfm#IT.

Key sections relevant to the OHF XDS Document Consumer include (but are not limited to):

- Volume 1, Section 10 and Appendices A,B, E, J, K, L and M
- Volume 2, Section 1, Section 2, Sections 3.14, 3.15, and Appendices B, E, J, K and L

2.4.4 Newsgroup

Any unanswered technical questions may be posted to Eclipse OHF newsgroup. The newsgroup is located at news://news.eclipse.org/eclipse.technology.ohf.

You can request a password at: http://www.eclipse.org/newsgroups/main.html.

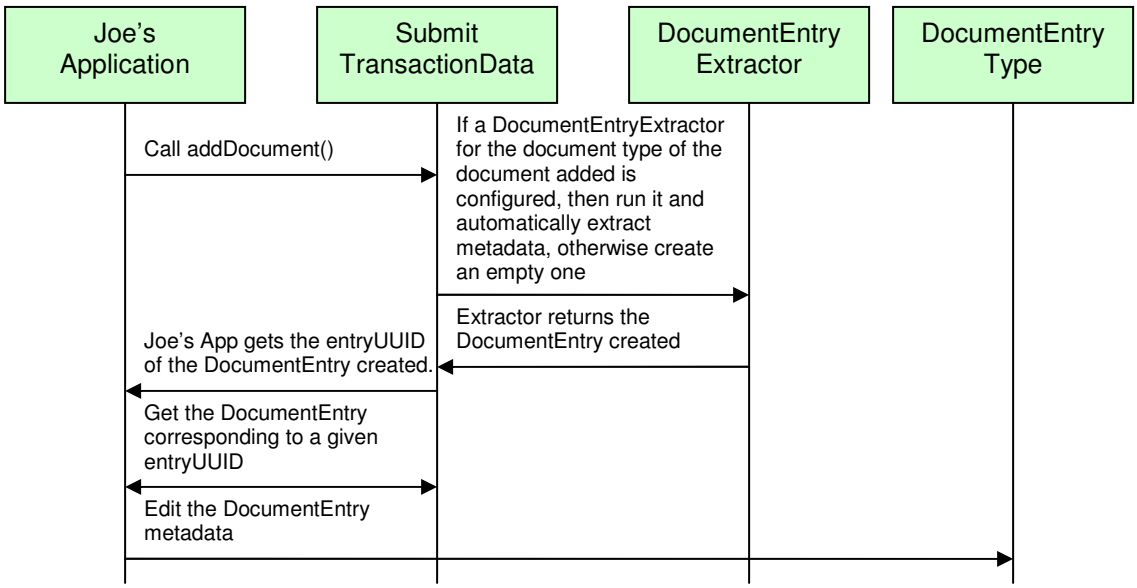
3. API Documentation

The XDS Document Source provides an API for the execution of the following IHE Transactions: ITI-15: Provide and Register Document Set Transaction. The API also provides utility functions for the construction of the payload data for this transaction (a SubmitTransactionData object) as well as the potential for automatic metadata extraction from any particular document type. In the use cases below, we first document the various ways to compose the payload data needed for a Provide and Register Document Set Transaction and then document how to execute this transaction. Support for automatic metadata extraction from HL7 CDA R2 is pending implementation. Compliance with the edits to the IHE Technical Framework for 2006 is pending.

3.1 Use Case 1 – Adding a Document to the SubmitTransactionData

Joe User, using his EMR Application, wants to submit documents to an XDS Registry. First, he needs to compose a list of documents and corresponding metadata to send. He adds his first document, is returned a reference to the metadata already created and is able to obtain this DocumentEntryType and add any missing fields.

3.1.1 Flow of Execution



3.1.2 API Highlights

The featured method in the above control flow is the SubmitTransactionData.addDocument() method. This is described below in greater detail. The complete XDS Source javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

SubmitTransactionData.addDocument()

String	**addDocument**(Document document) Add the given document to the Document Submission Set. Note: Each document may appear only once per submission set. If the same document object is added more than once, it will still appear only once in the document set. The DocumentEntry.entryUUID returned will be the same object instance for each addition of the same document. Will automatically extract metadata from the document if a MetadataExtractor for the Document's DocumentDescriptor has been loaded. **Parameters:** document - Document to add (@see com.ibm.ihii.xdssource.Document) **Returns:** a reference to the document entry metadata created for this document **Throws:** org.eclipse.ohf.ihe.xds.metadata.extract.MetadataExtractionException - if an error occurred extracting the XDS metadata from the document.

3.1.3 Sample Code

3.1.3.1 Description

The following sample code illustrates how the API user can add a Document to the SubmitTransactionData object, as part of the process for composing data needed for the Provide and Register Document Set Transaction.

3.1.3.2 Code

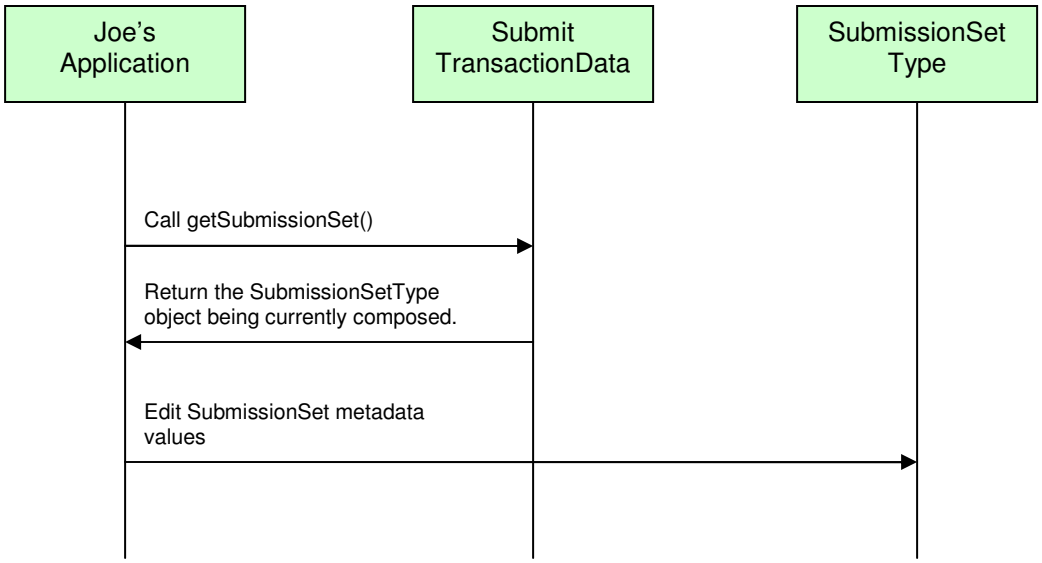
```
////////////////////////////////////////////////////////////////////////////
//Create a SubmitTransactionData for a single Provide and Register Document Set
//Transaction.
////////////////////////////////////////////////////////////////////////////

SubmitTransactionData txnData = new SubmitTransactionData();


////////////////////////////////////////////////////////////////////////////
//Suppose we have an HL7 CDA R2 schema compliant, XML, clinical document we want
//to submit. Suppose this file is located at "C:/temp/input.xml". We first need
//to creat a Document object for this file, with the appropriate
//DocumentDescriptor. The DocumentDescriptor is important because it is used to
//determine which available DocumentEntryExtractor to run for the given
//Document. It is strongly recommended to use the DocumentDescriptor constants
//provided.
////////////////////////////////////////////////////////////////////////////

Document clinicalDocument = new
Document("C:/temp/input.xml",DocumentDescriptor.CDA_R2);
```

```
////////////////////////////////////////////////////////////////////////////
//We can now add the document to the SubmitTransactionData object we are
//composing. XDS DocumentEntry Metadata will be automatically extracted from the
//Document if a DocumentEntryExtractor for the DocumentDescriptor.CDA_R2 has
//been configured for this SubmitTransactionData object. If no
//DocumentEntryExtractor is available, an empty DocumentEntryType is created. In
//either case the DocumentEntryType is assigned an entryUUID and this is
//returned.
////////////////////////////////////////////////////////////////////////////

String docEntryUUID = txnData.addDocument(clinicalDocument);


////////////////////////////////////////////////////////////////////////////
//Now obtain the DocumentEntry created for this clinical document and edit (or
//verify) the metadata values.
////////////////////////////////////////////////////////////////////////////

DocumentEntryType docEntry = txnData. getDocumentEntry(docEntryUUID);


////////////////////////////////////////////////////////////////////////////
// Add the patientId
////////////////////////////////////////////////////////////////////////////

CX patientId = MetadataFactory.eINSTANCE.createCX();

patientId.setIdNumber("1234567890");

patientId.setAssigningAuthorityUniversalId("1.3.6.1.4.1.21367.2005.1.1");

patientId.setAssigningAuthorityUniversalIdType("ISO");

docEntry.setPatientId(patientId);


//**NOTE: see OHF Metadata Model Documentation for more examples on editing XDS
//Metadata
```

3.2 Use Case 2 – Adding Submission Set Metadata to the SubmitTransactionData

Joe User now wants to add Submission Set Metadata values.

3.2.1 Flow of Execution



3.2.2 API Highlights

The featured method in the above control flow is the SubmitTransactionData.getSubmissionSet() method. This is described below in greater detail. The complete XDS Source javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

SubmitTransactionData.getSubmissionSet()	
`org.eclipse.ohf.ihe.xds.metadata.SubmissionSetType`	`getSubmissionSet()` Returns a reference to this Submission Set's metadata **Returns:** SubmissionSetType object holding the metadata for the transaction

3.2.3 Sample Code

3.2.3.1 Description

The following sample code illustrates how the API user can edit the SubmissionSet metadata for the SubmitTransactionData object, as part of the process for composing data needed for the Provide and Register Document Set Transaction.

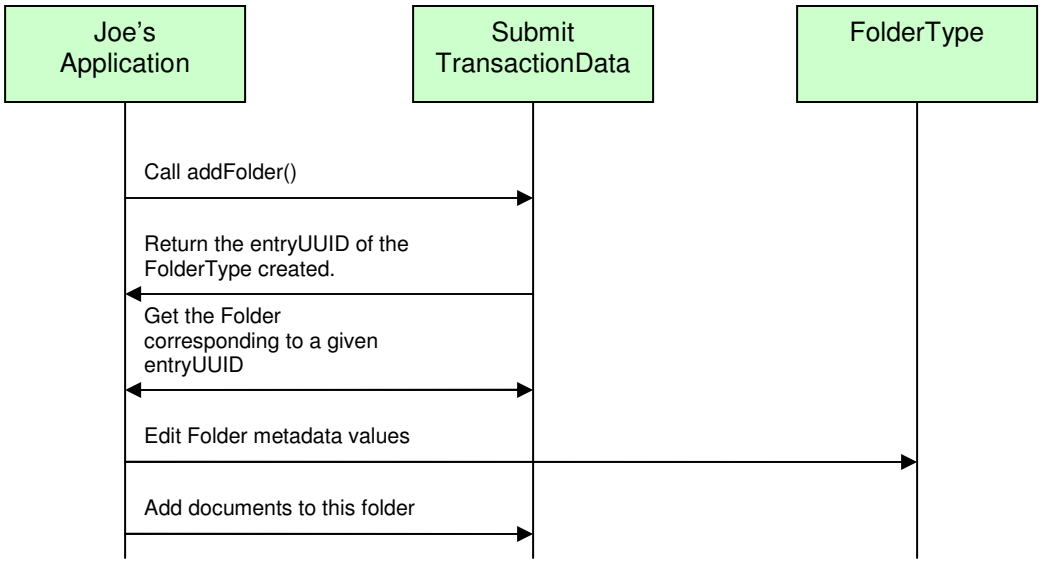
3.2.3.2 Code

```
////////////////////////////////////////////////////////////////////////////
//Now obtain the SubmissionSet created for this SubmitTransactionData and edit
//(or verify) the metadata values. Assume we have an already constructed
//SubmitTransactionData object called 'txnData' as in 3.1.3.2.
////////////////////////////////////////////////////////////////////////////

SubmissionSetType subset = txnData.getSubmissionSet();


////////////////////////////////////////////////////////////////////////////
//Set the contentTypeCode on the SubmissionSet
////////////////////////////////////////////////////////////////////////////

CodedMetadataType contentTypeCode =
MetadataFactory.eINSTANCE.createCodedMetadataType();

contentTypeCode.setCode("Consult");

contentTypeCode.setDisplayName("Consult");

contentTypeCode.setSchemeName("Connect-a-thon contentTypeCodes");

subSet.setContentTypeCode(contentTypeCode);


//**NOTE: see OHF Metadata Model Documentation for more examples on editing XDS
//Metadata
```

3.3 Use Case 3 – Adding Folders to the SubmitTransactionData

Joe User now wants to add a Folder to the submission he is composing. He additionally wants to place the document he has added in the newly created folder.

3.3.1 Flow of Execution



3.3.2 API Highlights

The featured methods in the above control flow are the SubmitTransactionData.addFolder() and SubmitTransactionData.addDocumentToFolder() methods. These are described below in greater detail. The complete XDS Source javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

SubmitTransactionData.addFolder()

String	**addFolder**() Create and Add a folder to the submission set **Returns:** Returns the entryUUID of the FolderType created

SubmitTransactionData.addDocumentToFolder()

void	**addDocumentToFolder**(String documentEntryUUID, String folderEntryUUID) Adds the documentEntryUUID reference to the specified folder's list of associated documents. If the folder does not exist, the reference is not added. If the document is already in the folder, the reference is not added.

3.3.3 Sample Code

3.3.3.1 Description

The following sample code illustrates how the API user can add and edit the created Folder for the SubmitTransactionData object as well as place documents in this folder, as an optional part of the process for composing data needed for the Provide and Register Document Set Transaction.

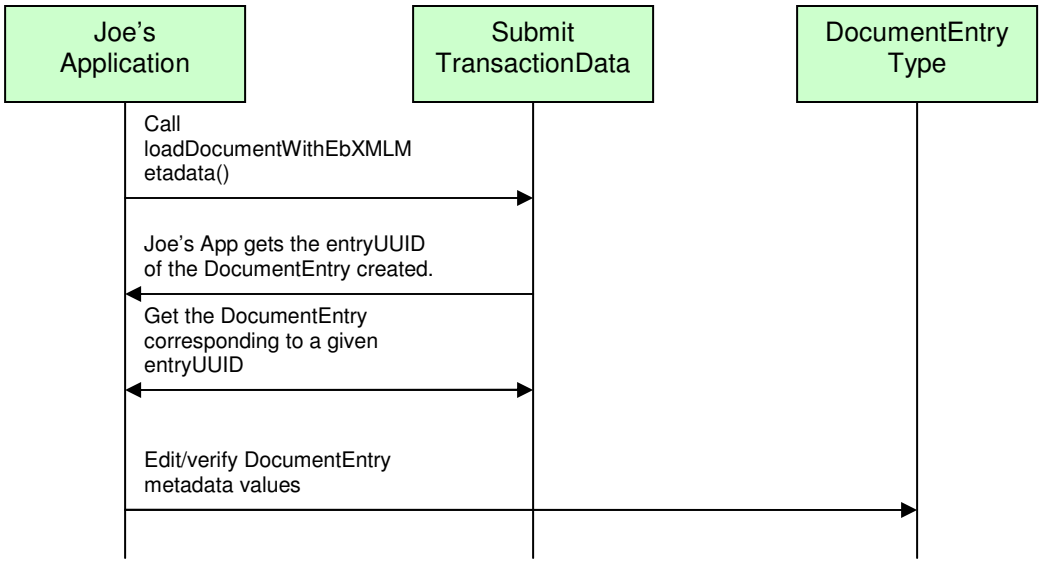
3.3.3.2 Code

```
////////////////////////////////////////////////////////////////////////////
//Now add a Folder to this SubmitTransactionData and edit (or verify) the
//metadata values. Assume we have an already constructed SubmitTransactionData
//object called 'txnData' as in 3.1.3.2.
////////////////////////////////////////////////////////////////////////////

String folderEntryUUID = txnData.addFolder();

FolderType folder = txnData.getFolder(folderEntryUUID);


////////////////////////////////////////////////////////////////////////////
//Set a code the on the Folder
////////////////////////////////////////////////////////////////////////////

CodedMetadataType code = MetadataFactory.eINSTANCE.createCodedMetadataType();

code.setCode("General Medicine");

code.setDisplayName("General Medicine");

code.setSchemeName("Connect-a-thon codeList");

folder.setContentTypeCode(code);


//**NOTE: see OHF Metadata Model Documentation for more examples on editing XDS
//Metadata

////////////////////////////////////////////////////////////////////////////
//Add the document from 3.1.3.2 to this folder.
////////////////////////////////////////////////////////////////////////////

txnData.addDocumentToFolder(docEntryUUID, folderEntryUUID);
```

3.4 Use Case 4 – Loading XDS Metadata from Files into the SubmitTransactionData

Joe User now has a clinical document and "pre-cooked" metadata in an ebXML formatted file that he wants to add, as is, to the growing SubmitTransactionData object.

3.4.1 Flow of Execution



3.4.2 API Highlights

The featured method in the above control flow is the SubmitTransactionData.loadDocumentWithEbXMLMetadata() method. This is described below in greater detail. The API also allows for the replacement of the Submission Set metadata with data contained in a file or an in-memory ebXML model. Additionally the API allows for the addition of folders to the SubmitTransactionData object using data from a file or an in-memory ebXML model. The process of doing so is similar to loading documents and corresponding, "pre-cooked" Document Entry metadata and will not be further elaborated on in this document. The complete XDS Source javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

SubmitTransactionData.loadDocumentWithEbXMLMetadata()	
String	**loadDocumentWithEbXMLMetadata**(Document document, File metadataFile) Loads the DocumentEntry metadata from the ebXML file specified, INCLUDING the entryUUID of the document entry, if present. It is advisible to set the id attribute in the ebXML file to the empty string, rather than replace the entryUUID assigned by this object, unless special circumstances exist. The DocumentEntry.entryUUID will be generated, if not present in the file. File format is to be conform to stipulations outlined in EbXML_2_1FileDocumentEntryExtractor Also, the document will be added. Note: Each document may appear only once per submission set. If the same document object is added more than once, it will still appear only once in the document set. The DocumentEntry.entryUUID returned will be the same object instance for each addition of the same document.

	Parameters: `metadataFile` - file containing ebXML formatted document entry metadata **Returns:** the current entryUUID of this document entry metadata **Throws:** `org.eclipse.ohf.ihe.xds.metadata.extract.MetadataExtractionException`

3.4.3 Sample Code

3.4.3.1 Description

The following sample code illustrates how the API user can add a Document and "pre-cooked" DocumentEntry metadata to the SubmitTransactionData object, as part of the process for composing data needed for the Provide and Register Document Set Transaction.

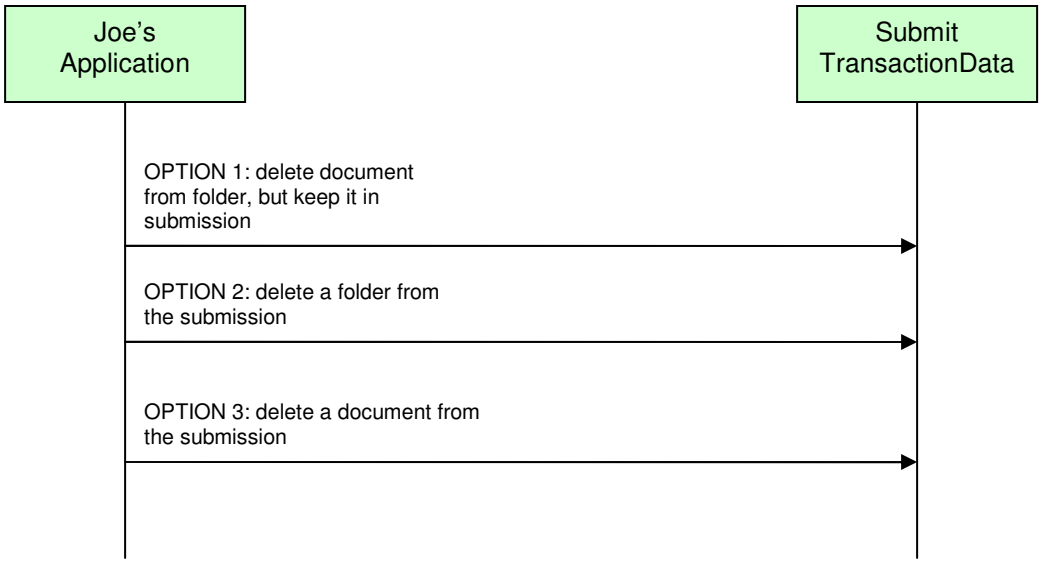
3.4.3.2 Code

```
////////////////////////////////////////////////////////////////////////////
// Assume we have a SubmitTransactionData object, 'txnData' as created in
//3.1.3.2. Suppose we have an HL7 CDA R2 schema compliant, XML, clinical
//document we want to submit. Suppose this file is located at
//"C:/temp/input2.xml". Suppose also we have DocumentEntry metadata in ebXML
//format for this CDA R2 clinical document located at "C:/temp/metadata2.xml".
////////////////////////////////////////////////////////////////////////////

Document clinicalDocument2 = new
Document("C:/temp/input2.xml",DocumentDescriptor.CDA_R2);

File docEntryEbXMLFile = new File ("C:/temp/metadata2.xml");

String docEntryUUID_2 = txnData.loadDocumentWithEbXMLMetadata(clinicalDocument2,
docEntryEbXMLFile);

////////////////////////////////////////////////////////////////////////////
//Now obtain the DocumentEntry created for this clinical document and edit (or
//verify) the metadata values.
////////////////////////////////////////////////////////////////////////////

DocumentEntryType docEntry2 = txnData.getDocumentEntry(docEntryUUID_2);

//**NOTE: see OHF Metadata Model Documentation for more examples on editing XDS
//Metadata
```

3.5 Use Case 5 – Deleting Documents and/or Folders From the SubmitTransactionData

Joe User decides he wants to delete a folder or a document from the submission he is composing.

3.5.1 Flow of Execution



3.5.2 API Highlights

The featured methods in the above control flow are the SubmitTransactionData.deleteDocumentFromFolder(), SubmitTransactionData.deleteFolder(), and SubmitTransactionData.deleteDocument() methods. These are described below in greater detail. The complete XDS Source javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

SubmitTransactionData.deleteDocumentFromFolder()

void	**deleteDocumentFromFolder**(String documentEntryUUID, String folderEntryUUID) Removes (deletes) the documentEntryUUID reference to the specified folder's list of associated documents. If the folder does not exist, the referecnce is not deleted.

SubmitTransactionData.deleteFolder()

void	**deleteFolder**(String folderEntryUUID) Remove(delete) the folder to the submission set with the corresponding folderEntryUUID **Parameters:** folderEntryUUID - of the folder to remove (delete)

SubmitTransactionData.deleteDocument()	
void	**deleteDocument**(String documentEntryUUID) Removes the document and its associated metadata. Removes any references to the document from folders. If the document does not exist in the set, nothing is done. **Parameters:** documentEntryUUID - Reference to the DocumentEntryType object corresponding to the Document to remove

3.5.3 Sample Code

3.5.3.1 Description

The following sample code illustrates how the API user can delete existing Documents and Folders from the SubmitTransactionData object, as an optional part of the process for composing data needed for the Provide and Register Document Set Transaction.

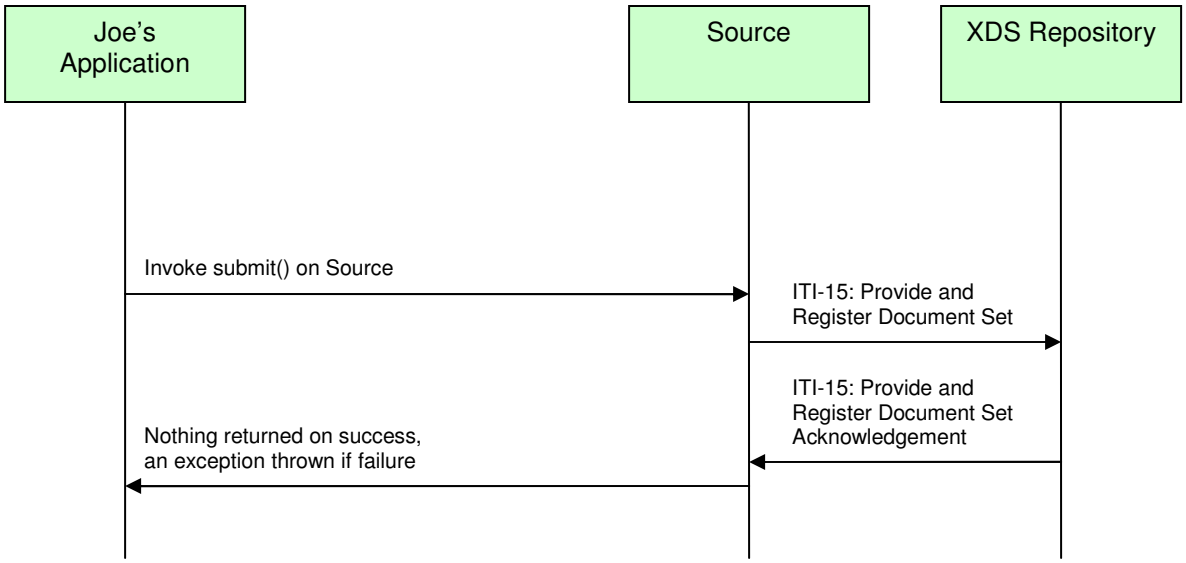
3.5.3.2 Code

```
////////////////////////////////////////////////////////////////////////////
//Assume we have an already constructed SubmitTransactionData //object called
//'txnData' as in 3.1.3.2. Assume the folder and document we want to delete are
//those referenced by 'folderEntryUUID' and 'docEntryUUID' in 3.3.3.2.
////////////////////////////////////////////////////////////////////////////

// disassociate the document with the folder

txnData.deleteDocumentFromFolder(docEntryUUID, folderEntryUUID);

// remove the folder from the submission

txnData.deleteDocumentFromFolder(docEntryUUID, folderEntryUUID);

// remove the document from the submission

txnData.deleteDocumentFromFolder(docEntryUUID, folderEntryUUID);
```

3.6 Use Case 6 – Executing the Provide and Register Document Set Transaction

Finally, Joe User is ready to submit the SubmitTransactionData he has so carefully composed. (NOTE: if we follow the sample code, collectively, from section 3.1 through section 3.5, we see that what remains of Joe's submission is the submission set metadata composed in 3.2 and the document with corresponding metadata added in 3.4.)

3.6.1 Flow of Execution



3.6.2 API Highlights

The featured method in the above control flow is the Source.submit() method. This is described below in greater detail. The complete XDS Source javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

Source.submit()

void	**submit**(SubmitTransactionData txnData, String initiatingUser) Submits the Provide and Register Document Set transaction to the XDS Repository Actor. Errors from the Repository are returned in a thrown exception. **Parameters:** txnData - transaction payload data: XDS metadata and corresponding documents initiatingUser, - initiating user for auditing purposes **Throws:** Exception - If the transaction failed to complete

3.6.3 Sample Code

3.6.3.1 Description

The following sample code illustrates how the API user can submit documents for the Provide and Register Document Set Transaction.

3.6.3.2 Code

```
////////////////////////////////////////////////////////////////////////////
//If an instance does not already exist, create an instance of the XDS Source
//and provide the XDS Repository url and port. Also enable transaction auditing.
////////////////////////////////////////////////////////////////////////////

String repositoryURL = "http://my.registry.url:8080";

Source source = new Source(repositoryURL, true); // true = auditing enabled

////////////////////////////////////////////////////////////////////////////
//Assume we have a populated SubmitTransactionData object, 'txnData', as we left
//it in 3.5.3.2. Now we simply submit the document and the metadata it contains.
////////////////////////////////////////////////////////////////////////////

try {

      source.submit(txnData, "JOE USER");

} catch (SourceException e) {

      logger.error("Source.submit failed", e);

} catch (Throwable th) {

      logger.error("Unexpected error", th);

}
```

4. Security

4.1 Node Authentication

Node Authentication is pending integration with other OHF components providing this functionality. Currently, SOAP transactions are sent over an insecure connection. We are aware of this issue and are working on an alternative method to support mutual node authentication while integration is pending.

4.2 Auditing

Auditing to an Audit Record Repository is pending integration with other OHF components providing this functionality. Currently, a “place-holder” auditing API is in place in which auditable events are written to the local application log. For more information about this implementation please see the OHF ATNA Audit Client Document. We are aware of this issue and are working towards a solution.

5. Configuration

Logging is the only configurable aspect of this plugin at this time. This most likely will change when integration with other OHF plugins is complete. For more information about logging, consult Section 6 of this document. The following is a list of anticipated configurable aspects:

- Java keystore file for SSL communication
- Java keystore pass for SSL communication
- Java truststore file for SSL communication
- Java truststore pass for SSL communication
- Audit Record Repository url and port
- Enable/Disable auditing flag (potentially added to configuration, currently an API parameter)
- Initiating user ID, for auditing (potentially added to configuration, currently an API parameter)
- XDS Repository url and port (potentially added to configuration, currently an API parameter)
- Metadata Extractor implementations available to a particular Document Source

6. Debugging Recommendations

The XDS Source uses Apache Log4j. If you are experiencing bugs related to the Source, you may enable debug level logging by adding the following category to your log4j XML configuration file.

```
<category name="org.eclipse.ohf.ihe.xds.source">
   <priority value="debug" />
</category >
```

For more information about Log4j please see: http://logging.apache.org/log4j/docs/

For an example log4j XML configuration file and to see how it is loaded at run time see `org.eclipse.ohf.ihe.xds.source/resources/conf/submitTest_log4j.xml` and `org.eclipse.ohf.ihe.xds.source/src_tests/SubmitTest.java`, respectively. These can be obtained by downloading the plugin `org.eclipse.ohf.ihe.xds.source` from the Eclipse CVS technology repository. See 2.2 for details.

7. Pending Integration and API changes

Below is a laundry list of anticipated changes to the XDS Source:

1. Integration of Node Authentication and Auditing – It is not clear at this point how this will affect the current API of the XDS Source. We are working with the other OHF members supplying the functionality to ensure that changes are minimized. Much of this may potentially be handled via configuration rather than creating new methods in the API.
2. Support for Linking to Existing Metadata – Document submission in XDS allows a new submission to link to existing documents and folders on the XDS Registry. The SubmitTransactionData object will have to add API to expose this feature once enabled in the underlying XDS Metadata Model.
3. Support for DSG – Enabling submission of digitally signed documents also requires changes to the SubmitTransactionData object. We are awaiting resolution to internal IHE discussions on how to test DSG at Connectathon 2007 before considering these modifications.

8. Additional Sections – repeat as necessary

Any additional sections needed are added at this point.

9. Glossary

Define any non-common knowledge terms or acronyms here. Provide web-site reference if applicable.