# OHF XDS Document Source

# Architecture & API Documentation

**Version 0.2.0**

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.eclipse.ohf.ihe.atna.agent | Invocation point for audit events, message transport |
| • org.eclipse.ohf.ihe.atna.audit | Audit message support |
| • org.eclipse.ohf.ihe.atna.payload | Message transport payload |
| • org.eclipse.ohf.ihe.atna.transport | Message transport |
| • org.eclipse.ohf.ihe.common.atna | Common ATNA security and configuration libraries |
| • org.eclipse.ohf.ihe.xds.soap | SOAP messaging support for transactions |
| • org.apache.axis2 | Apache Axis 2.0 to support the above plugins |
| • org.apache.log4j | Debug, warning and error logging |
| • org.apache.xerxes | Support serialization of XML messages |
| • org.eclipse.ohf.ihe.common.cdar2 | Model of the HL7 CDA R2 |
| • org.eclipse.ohf.ihe.common.ebXML._2._1 | Model of ebXML v2.1: rim, query and rs |
| • org.eclipse.ohf.ihe.common.hl7v2 | Model of HL7 v2 supportive of XDS metadata |
| • 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.extract.cdar2 | Model to render XDS metadata from HL7 CDA R2 |
| • org.eclipse.ohf.ihe.xds.metadata.transform | Model to render XDS metadata to other formats |
| • org.eclipse.ohf.utilities | Basic OHF tools, base exception handling, etc. |
| • org.apache.commons.lang | Supportive plugin for org.eclipse.ohf.utilities |
| • org.xmlpull.v1 | Supportive plugin for org.eclipse.ohf.utilities |

### *2.3.2 External Sources*

At this time, the OHF XDS Source plugin does not depend on any external sources. However, we are currently experiencing minor difficulties related to SOAP Attachment support with the current release of Axis 2.0. For more information on this problem and for the intermediary fix, please see: https://bugs.eclipse.org/bugs/show_bug.cgi?id=181354.

## 2.4 Resources

### *2.4.1 Eclipse OHF websites*

- IHE Components Website: http://www.eclipse.org/ohf/components/ihe/index.php
- OHF Wiki: http://wiki.eclipse.org/index.php/OHF

### *2.4.2 Other OHF Plugin Documentation*

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

- OHF ATNA Agent
- OHF XDS Metadata Model

### *2.4.3 Automatic XDS Metadata Extraction from CDA R2 Documents*

The OHF XDS Document Source includes a Metadata Extractor that is able to extract XDS Metadata from HL7 CDA R2 schema conformant documents. Detailed documentation of this extraction process can be found in org.eclipse.ohf.ihe.xds.metadata.extract.cdar2/resources/doc/CDAR2Extractor.pdf.

### *2.4.4 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

### 2.4.5 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**.**

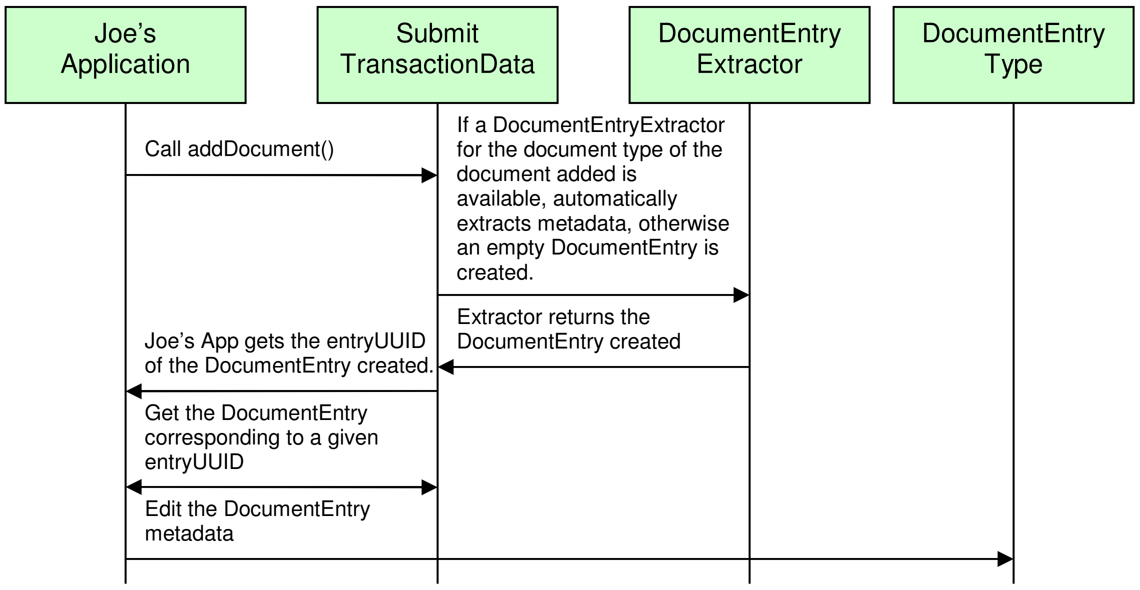
# 3. API Documentation

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.

## 3.1 Use Case 1 – Adding a Document to the SubmitTransactionData

Joe User, using his EMR Application, wants to submit documents to an XDS Repository. 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*


Joe's Application
Submit TransactionData
DocumentEntry Extractor
DocumentEntry Type
Call addDocument()
If a DocumentEntryExtractor for the document type of the document added is available, automatically extracts metadata, otherwise an empty DocumentEntry is created.
Extractor returns the DocumentEntry created
Joe's App gets the entryUUID of the DocumentEntry created.
Get the DocumentEntry corresponding to a given entryUUID
Edit the DocumentEntry metadata

### *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) throws MetadataExtractionException, SubmitTransactionCompositionException<br><br>Add the given document to the Document Submission Set.<br><br>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.<br>**Parameters:**<br>`document` - Document to add (@see com.ibm.ihii.xdssource.Document), must not be null<br>**Returns:**<br>a reference (the DocumentEntry.entryUUID) to the document entry metadata created for this document<br>**Throws:**<br>MetadataExtractionException - if an error occurred extracting the XDS metadata from the document.<br>SubmitTransactionCompositionException - if an error occurred in composing the submit transaction data |

### *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

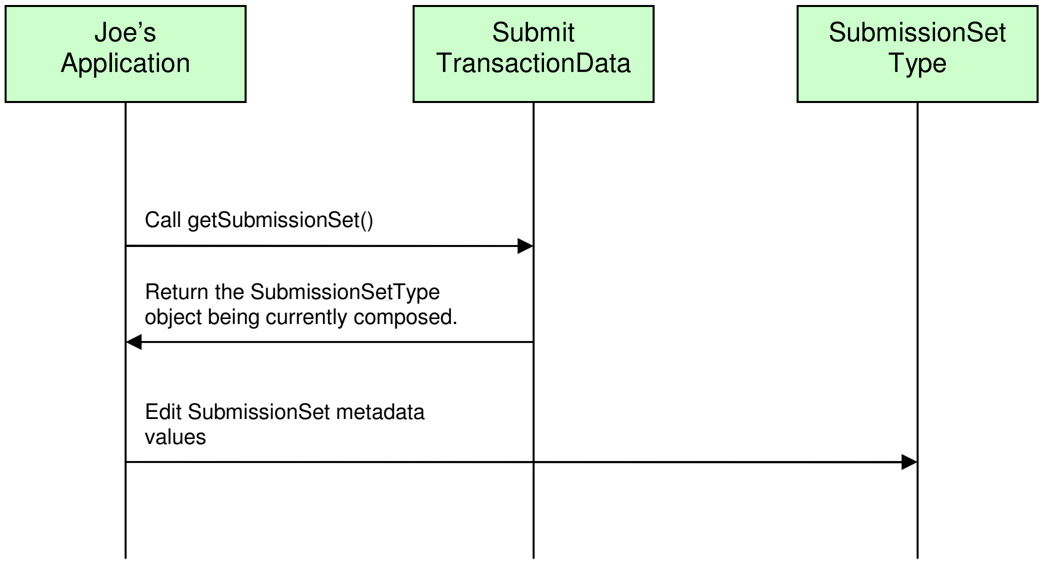
```
////////////////////////////////////////////////////////////////////////////////
//Create a SubmitTransactionData for a single Provide and Register Document Set
//Transaction.
////////////////////////////////////////////////////////////////////////////////

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 create 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 made available for the SubmitTransactionData object. If no
//DocumentEntryExtractor is available, an empty DocumentEntryType is created. In
//either case the DocumentEntryType is assigned an entryUUID and this is
//information 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



Joe's Application
Submit TransactionData
SubmissionSet Type
Call getSubmissionSet()
Return the SubmissionSetType object being currently composed.
Edit SubmissionSet metadata values

### 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`**`()`<br>Returns a reference to this Submission Set's metadata<br><br>**Returns:**<br>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.

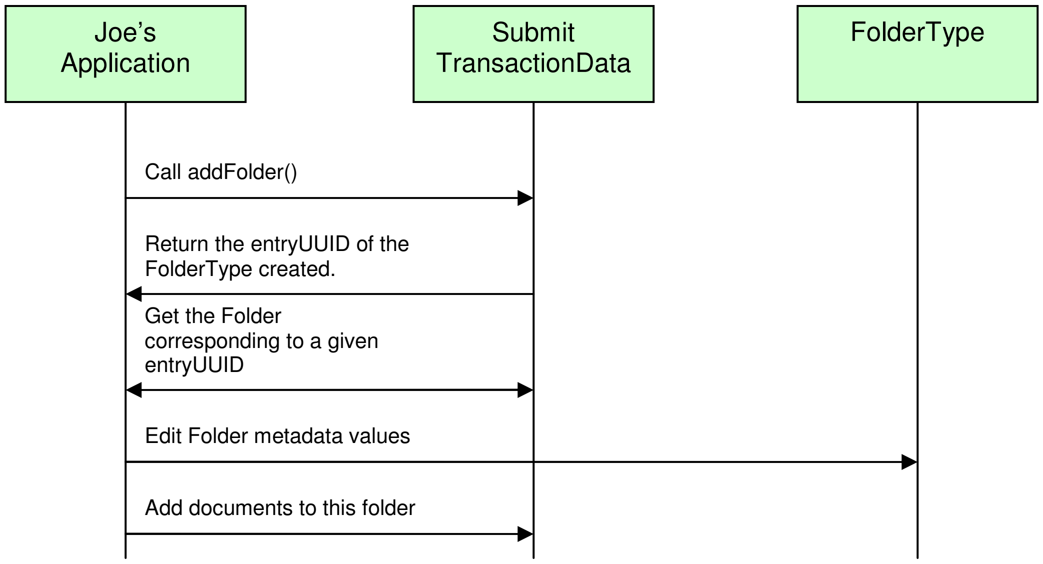
#### 3.2.3.2 Code

```
///////////////////////////////////////////////////////////////////////////////
//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


Joe's Application
Submit TransactionData
FolderType
Call addFolder()
Return the entryUUID of the FolderType created.
Get the Folder corresponding to a given entryUUID
Edit Folder metadata values
Add documents to this folder

---

### 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()`<br>Create and Add a folder to the submission set<br>**Returns:**<br>Returns the entryUUID of the FolderType created |

| **SubmitTransactionData.addDocumentToFolder()** | |
|---|---|
| `void` | `addDocumentToFolder(String documentEntryUUID, String folderEntryUUID)`<br>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.

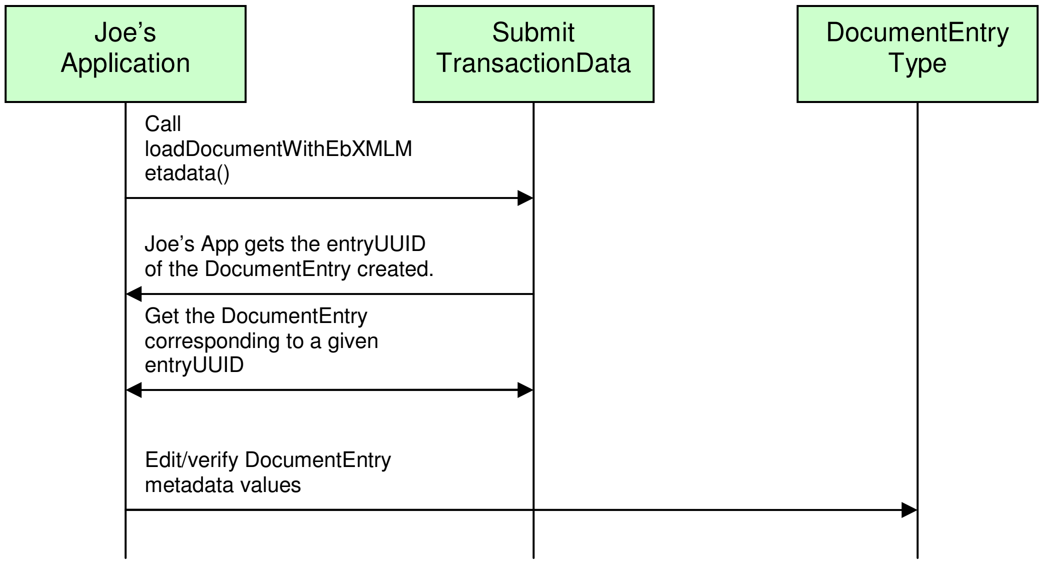
#### 3.3.3.2 Code

```
//////////////////////////////////////////////////////////////////////////////
//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


Joe's Application
Submit TransactionData
DocumentEntry Type
Call loadDocumentWithEbXMLMetadata()
Joe's App gets the entryUUID of the DocumentEntry created.
Get the DocumentEntry corresponding to a given entryUUID
Edit/verify DocumentEntry metadata values

---

### 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, java.io.InputStream metadata) throws MetadataExtractionException, SubmitTransactionCompositionException` 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_1InputStreamDocumentEntryExtractor` Also, the |

| | |
|---|---|
| | document will be added. Note: Each document may appear only once per submission set. If the same document object is attempted to be added more than once, it will not be added to the document set. The DocumentEntry.entryUUID returned will be the existing object instance this document.<br>**Parameters:**<br>`metadata` - InputStream containing ebXML compliant metadata<br>**Returns:**<br>the current entryUUID of this document entry metadata<br>**Throws:**<br>`MetadataExtractionException` - if an error occurred extracting the XDS metadata from the document.<br>`SubmitTransactionCompositionException` - if an error occurred in composing the submit transaction data |

### 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

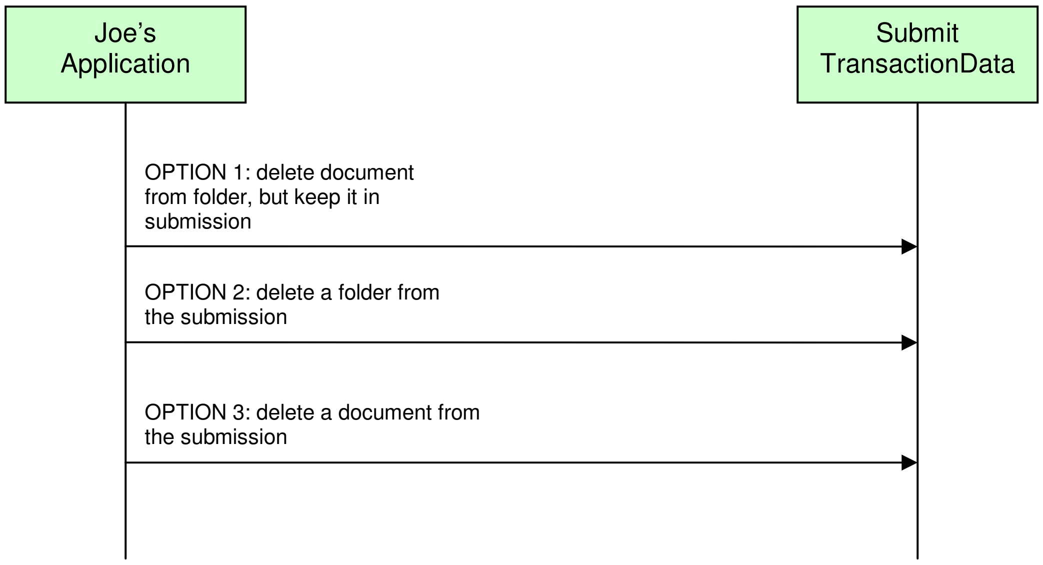
```
////////////////////////////////////////////////////////////////////////////////
// Assume we have a SubmitTransactionData object, ‘txnData’ as created in
//3.1.3.2. Suppose we have an HL7 CDA R2 schema compliant, XML, clinical
//document we want to submit. Suppose this file is located at
//“C:/temp/input2.xml”. 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");

FileInputStream fis = new FileInputStream(docEntryEbXMLFile);

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

////////////////////////////////////////////////////////////////////////////////
//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


Joe's Application
Submit TransactionData
OPTION 1: delete document from folder, but keep it in submission
OPTION 2: delete a folder from the submission
OPTION 3: delete a document from the submission

### 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)`<br>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 current 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.

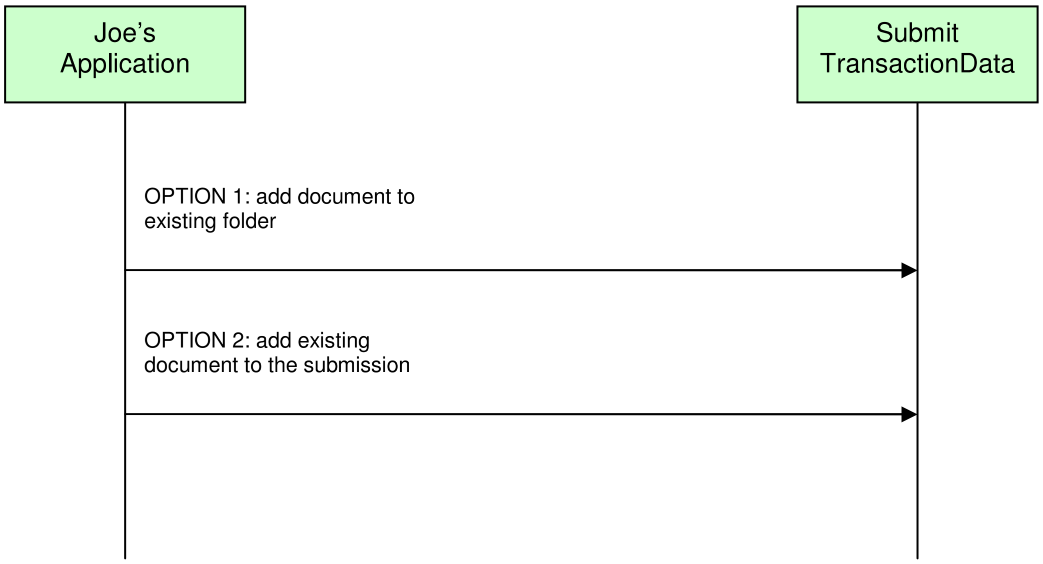
#### 3.5.3.2 Code

```
////////////////////////////////////////////////////////////////////////////////
//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 – Adding Existing Documents and/or Folders to the SubmitTransactionData

Joe User decides he wants to add a document to an existing folder or an existing document from the submission he is composing. An “existing” document or folder is one that already exists in the XDS Registry because it was registered there by a prior Provide and Register Document Set transaction.

### *3.6.1 Flow of Execution*


Joe’s Application
Submit TransactionData
OPTION 1: add document to existing folder
OPTION 2: add existing document to the submission

### *3.6.2 API Highlights*

The featured methods in the above control flow are the SubmitTransactionData.addExistingFolder(), SubmitTransactionData.addDocumentToFolder(), and SubmitTransactionData.addExistingDocument() methods. SubmitTransactionData.addDocumentToFolder() is described above in Section 3.3.2. The others 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.addExistingFolder()** | |
|---|---|
| `void` | `addExistingFolder(java.lang.String folderEntryUUID) throws MetadataExtractionException`<br><br>Add folder that already exists in the XDS Registry to the Document Submission Set.<br><br>**Parameters:**<br><br>`folderEntryUUID` - the Folder.entryUUID for the existing folder |

| | |
|---|---|
| | **Throws:**<br>MetadataExtractionException |
| | |

## SubmitTransactionData.addExistingDocument()

| | |
|---|---|
| void | **addExistingDocument**(java.lang.String documentEntryUUID) throws MetadataExtractionException<br><br>Add document that already exists in the XDS Registry to the Document Submission Set.<br>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.<br><br>**Parameters:**<br>documentEntryUUID - the DocumentEntry.entryUUID for the existing document<br><br>**Throws:**<br>MetadataExtractionException |

### *3.6.3 Sample Code*

#### 3.6.3.1 Description

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

#### 3.6.3.2 Code

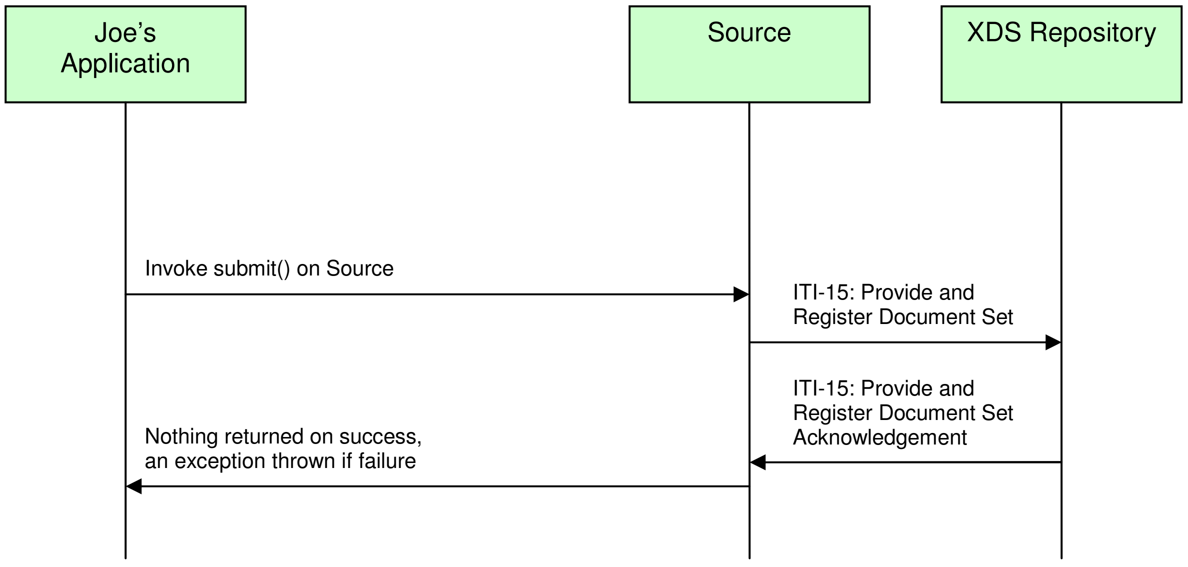
```
////////////////////////////////////////////////////////////////////////////////
//Assume we have an already constructed SubmitTransactionData object called
//'txnData' as in 3.1.3.2. Assume the existing folder and document we want to
//add are those referenced by 'folderEntryUUID_Existing' and
//'docEntryUUID_Existing. We obtained these uuids by doing a query or saving the
// ids in a local cache. Additionally, we want to place the document referenced
// by 'docEntryUUID_2' from section 3.4.3.2 into the existing folder.
////////////////////////////////////////////////////////////////////////////////

// add the existing folder

txnData.addExistingFolder(docEntryUUID, folderEntryUUID_Existing);

// add the document referenced by 'docEntryUUID_2 to the existing folder
```

```
txnData.addDocumentToFolder(docEntryUUID_2, folderEntryUUID_Existing);
// add the existing document to the submission set
txnData.addExistingDocument(docEntryUUID_Existing);
```

## 3.7 Use Case 7 – 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 3.6, we see that what remains of Joe's submission is the submission set metadata composed in 3.2, the document with corresponding metadata added in 3.4. and the references to an existing folder and an existing document added in 3.6).

### *3.7.1 Flow of Execution*


Joe's Application
Source
XDS Repository
Invoke submit() on Source
ITI-15: Provide and Register Document Set
ITI-15: Provide and Register Document Set Acknowledgement
Nothing returned on success, an exception thrown if failure

### *3.7.2 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, java.lang.String initiatingUser) throws java.lang.Exception`<br>Submits the Provide and Register Document Set transaction to the XDS Repository |

| | |
|---|---|
| | Actor. Errors from the Repository are returned in a thrown exception.<br><br>**Parameters:**<br>`txnData` - transaction payload data: XDS metadata and corresponding documents<br>`initiatingUser` - initiating user for auditing purposes<br>**Throws:**<br>`java.lang.Exception` - If the transaction failed to complete |

### 3.7.3 *Sample Code*

#### 3.7.3.1 Description

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

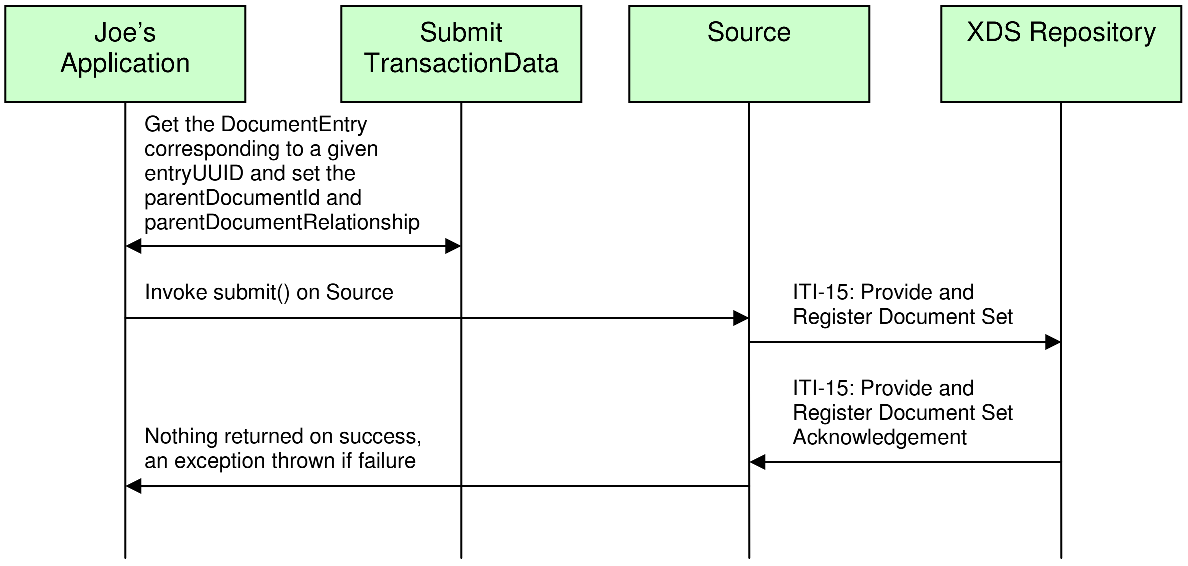
#### 3.7.3.2 Code

```
////////////////////////////////////////////////////////////////////////////////
//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.repository.url:8080";
Source source = new Source(repositoryURL);
////////////////////////////////////////////////////////////////////////////////
//Assume we have a populated SubmitTransactionData object, 'txnData', as we left
//it in 3.6.3.2. Now we simply submit the document and the metadata it contains.
////////////////////////////////////////////////////////////////////////////////
try {
	source.submit(txnData, "JOE USER");
} catch (SourceException e) {
	System.out.println("Source.submit failed", e);
} catch (Throwable th) {
	System.out.println("Unexpected error", th);
}
```

## 3.8 Use Case 7 – Executing the Provide and Register Document Set Transaction for Document Replacement

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 3.6, we see that what remains of Joe's submission is the submission set metadata composed in 3.2, the document with corresponding metadata added in 3.4. and the references to an existing folder and an existing document added in 3.6). However, he wants to replace a document on that already exists in the XDS Registry with the new document he has in this submission set.

### *3.8.1 Flow of Execution*



Joe's Application
Submit TransactionData
Source
XDS Repository
Get the DocumentEntry corresponding to a given entryUUID and set the parentDocumentId and parentDocumentRelationship
Invoke submit() on Source
ITI-15: Provide and Register Document Set
ITI-15: Provide and Register Document Set Acknowledgement
Nothing returned on success, an exception thrown if failure

### *3.8.2 API Highlights*

The complete XDS Source javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

### *3.8.3 Sample Code*

#### 3.8.3.1 Description

The following sample code illustrates how the API user can replace a document in the XDS Registry using the Provide and Register Document Set Transaction.

#### 3.8.3.2 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.repository.url:8080”;

Source source = new Source(repositoryURL);
////////////////////////////////////////////////////////////////////////////////
//Assume we have a populated SubmitTransactionData object, ‘txnData’, as we left
//it in 3.6.3.2. Assume the document we want to replace is referenced by
//‘docEntryUUID_Parent. We obtained this uuids by doing a query or saving the
// id in a local cache. The document we want to replace it with is referenced by
//‘docEntryUUID_2’ from section 3.4.3.2.
////////////////////////////////////////////////////////////////////////////////
txnData.getDocumentEntry(docEntryUUID_2).setParentDocumentRelationship(
ParentDocumentRelationshipType.RPLC_LITERAL);

txnData.getDocumentEntry(docEntryUUID_2).setParentDocumentId(docEntryUUID_Parent
);


////////////////////////////////////////////////////////////////////////////////
// Now we simply submit the document and the metadata it contains.
////////////////////////////////////////////////////////////////////////////////
try {

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

} catch (SourceException e) {

      System.out.println("Source.submit failed", e);

} catch (Throwable th) {

      System.out.println("Unexpected error", th);

}
```

# 4. Security

## 4.1 Node Authentication

Node Authentication is accomplished using Java keystores. For more information regarding the creation of keystores and truststores in Java see the OHF ATNA Agent documentation. Once you have created your java keystore containing your private key and a truststore containing all keys of your trusted partners, simply run the OHF XDS Source with the following JVM arguments:

```
-Djavax.net.ssl.keyStore=<location of your keystore file>
-Djavax.net.ssl.keyStorePassword=<password to your keystore file>
-Djavax.net.ssl.trustStore=<location of your truststore file>
-Djavax.net.ssl.trustStorePassword=<password to your truststore file>
```

## 4.2 Auditing

All auditing events surrounding the OHF XDS Source transaction is taken care of by this plugin. Auditing is enabled by default and audit messages, by default, are sent to `syslog://lswin10.dfw.ibm.com:515`.

To *disable* auditing for all OHF events call the following:

```
AtnaAgentFactory.getAtnaAgent().setDoAudit(false);
```

To switch the url of the Audit Record Repository to which the audit events are being sent, call the following:

```
AtnaAgentFactory.getAtnaAgent().setAuditRepository(new URI(
“syslog://new.audit.url));.
```

To *re-enable* auditing for all OHF events, call the following:

```
AtnaAgentFactory.getAtnaAgent().setDoAudit(true);
```

To create your own auditing message for application specific events, please see the OHF ATNA Agent documentation.

### *4.2.1 Auditing Example Code*

#### 4.2.1.1 Description

The following sample code illustrates how the above auditing commands can be used in conjunction with the OHF XDS Source.

#### 4.2.1.2 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.repository.url:8080”;

Source source = new Source(repositoryURL);
```

```
////////////////////////////////////////////////////////////////////////////////
//Switch Audit Record Repository end point
////////////////////////////////////////////////////////////////////////////////
String auditURL = “syslog://my.audit.url:515”;

AtnaAgentFactory.getAtnaAgent().setAuditRepository(new URI(auditURL));


////////////////////////////////////////////////////////////////////////////////
//Assume we have a populated SubmitTransactionData object, ‘txnData’, as we left
//it in 3.6.3.2. Now we simply submit the document and the metadata it contains.
////////////////////////////////////////////////////////////////////////////////
try {

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

} catch (SourceException e) {

      System.out.println("Source.submit failed", e);

} catch (Throwable th) {

      System.out.println("Unexpected error", th);

}


//Turn off auditing for a submission to a different repository (suppose we
//want to do this … for some reason).

AtnaAgentFactory.getAtnaAgent().setDoAudit(true);

String otherRepository = “http://other.repository.url:8080”
source.setRepositoryURL (otherRepository);
try {

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

} catch (SourceException e) {

      System.out.println("Source.submit failed", e);

} catch (Throwable th) {

      System.out.println("Unexpected error", th);

}
```

# 5. Configuration

The following is a list of configurable aspects:

- Java properties for TLS communication (set the following as JVM arguments):

  ```
  -Djavax.net.ssl.keyStore=<location of your keystore file>
  -Djavax.net.ssl.keyStorePassword=<password to your keystore file>
  -Djavax.net.ssl.trustStore=<location of your truststore file>
  -Djavax.net.ssl.trustStorePassword=<password to your truststore file>
  ```

- Audit Record Repository url and port:

  ```
  String auditURL = "syslog://my.audit.url:515";
  AtnaAgentFactory.getAtnaAgent().setAuditRepository(new URI(auditURL));
  ```

- Enable/Disable auditing flag

  ```
  AtnaAgentFactory.getAtnaAgent().setDoAudit(true); // enable
  AtnaAgentFactory.getAtnaAgent().setDoAudit(false); // disable
  ```

- Initiating user ID, for auditing (an API parameter on each transaction)
- XDS Registry url and port (an API parameter on the constructor of the OHF XDS Source)
- Logging is configurable. For more information about logging, consult Section 6 of this document.

# 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. 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 or 2008 before considering these modifications.
2. We anticipate supporting the new web-service version of the Provide and Register Transaction that is in development for the 2007-2008 season.

# 8. Glossary

No glossary terms at this time.