OHF XDS Document Consumer

Architecture & API Documentation

Version 0.0.2

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 Consumer Actor. This implementation supports the following IHE Transactions: ITI-16: Query Registry, ITI-17: Retrieve Document and ITI-18: Registry Stored Query.

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.consumer

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

2.3 Dependencies

Document dependencies the source has (feel free to extend this section)

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.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.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 Consumer 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 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, Section 3.16, 3.17 and Appendices J and K
- Stored Query Supplement also available at: http://www.ihe.net/Technical_Framework/index.cfm#IT

2.4.3 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

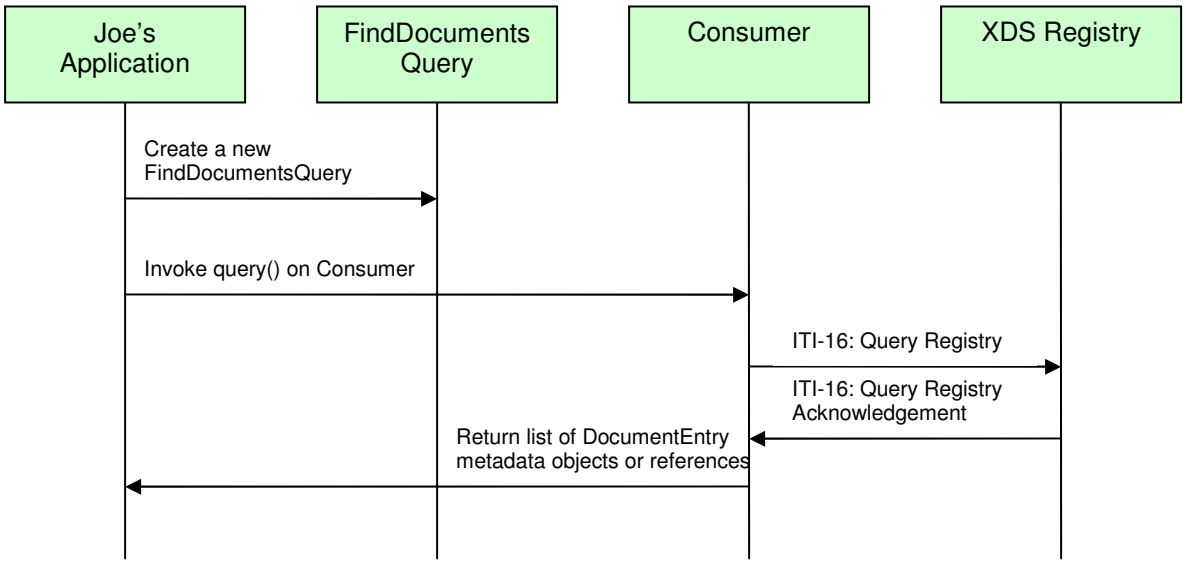
The XDS Document Consumer provides API for the execution of the following IHE Transactions: ITI-16: Query Registry, ITI-17: Retrieve Document and ITI-18: Registry Stored Query. The API also provides utility functions for the construction of SQL queries for ITI-16. The API currently supports automatic construction of the FindDocuments and the GetDocument query for ITI-16. Support for ITI-18 is pending implementation. Compliance with the edits to the IHE Technical Framework for 2006 is pending.

3.1 Use Case 1 – Query Registry: FindDocuments Query

Joe User, using his EMR Application, wants to find all documents where:

- patientID is "JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO"
- Only "Approved" documents are returned
- Creation time on the document is between 20031225 and 20060101
- Healthcare Facility Type Code is "Outpatient"

3.1.1 Flow of Execution



3.1.2 API Highlights

The most significant method portions in the above control flow are Consumer.query() and the construction of the FindDocumentsQuery (which extends the more generic Query object). These are described below in greater detail. The complete XDS Consumer javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

Consumer.query()

java.util.List	**query**(Query q, boolean returnReferencesOnly, java.lang.String initiatingUser) Transaction ITI-16: Query Registry Query for and retrieve a list of document metadata or metadata references based on the input attributes. This is the preferred version of the query method and should be used whenever possible. **Parameters:** q - fully populated Query object to be executed by the consumer returnReferencesOnly - used to indicate the return type desired by the user. Set returnReferencesOnly = true if only references to document entries are desired. This option is desirable if the user is expecting many documents to satisfy the query, or if memory space is a concern. Set returnReferencesOnly = false if the complete document entry object is desired. initiatingUser - entity issuing the query. Can only be null or empty string if auditing is disabled for the Consumer **Returns:** A List of returned data. If returnReferencesOnly = false the returned data will be a List populated with DocumentEntryTypes, containing all metadata for each document satisfying the criteria specified in the argument map. If the returnReferencesOnly = true the returned data will be a list of DocumentEntry.entryUUID string values that reference the DocumentEntries in the registry that satisfy the query. Returns null if no documents are found. **Throws:** java.lang.Exception

FindDocumentsQuery()

FindDocumentsQuery(java.lang.String patientID, java.lang.String[] classCodes, DateTimeRange[] dateTimeRanges, java.lang.String[] practiceSettingCodes, java.lang.String[] healthCareFacilityCodes, java.lang.String[] eventCodeList, org.eclipse.ohf.ihe.xds.metadata.AvailabilityStatusType[] status)

Constructor with full set query of parameters.

Parameters:

patientID - id of patient, non null

classCodes, - classCodes to query for, may be null

dateTimeRanges, - date and time ranges (use DateTimeRange), may be null

practiceSettingCodes - practiceSetting codes to query for, may be null

`healthCareFacilityCodes` - healthcareFacility codes to query for, may be null

`eventCodeList` - eventCodes to query for, may be null

`status` - document status list, cannot be null

Throws:

`MalformedQueryException`

3.1.3 Sample Code

3.1.3.1 Description

The following sample code illustrates how the XDS Consumer issues a FindDocumentsQuery to the XDS Registry.

3.1.3.2 Code

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

Consumer c = null;

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

try {

      c = new Consumer(registryURL, true); // true = turn on auditing

} catch (Exception e1) {

      logger.error("Exception", e1);

      return ERROR;

}


////////////////////////////////////////////////////////////////////////////
//Construct the parameters to our FindDocumentsQuery
////////////////////////////////////////////////////////////////////////////

// et up the date-time range for creationTime between Dec 25, 2005 and Jan 01,
//2006

DateTimeRange[] ranges = {new
DateTimeRange(DocumentEntryConstants.CREATION_TIME, "20031225", "20060101"};


//Create a list of healthcare facility codes we want to search on. In this
//example we only want documents where the healthcare facility is "Outpatient"

String[] hcfCodes = {"Outpatient"};
```

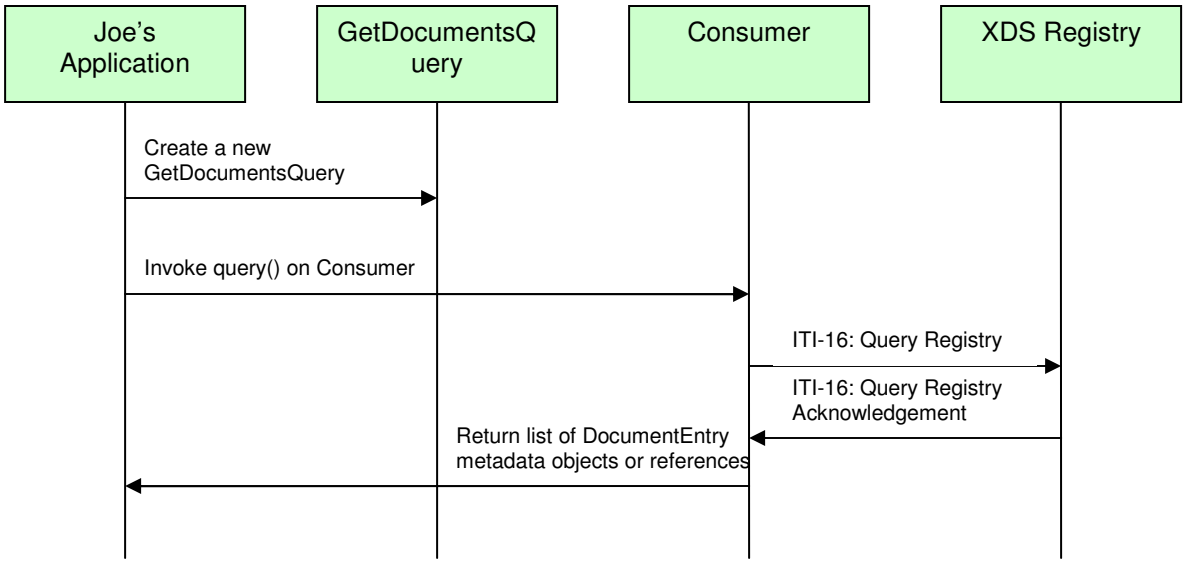
```
//Create a list of document status types we want to search on. In this example,
//we only want "Approved" documents.
AvailabilityStatusType status = {AvailabilityStatusType.Approved};


////////////////////////////////////////////////////////////////////////////
//Construct our FindDocumentsQuery for patient
//"JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO"
////////////////////////////////////////////////////////////////////////////
FindDocumentsQuery query = new
FindDocumentsQuery("JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO"null, ranges,
null, hcfCodes, null, status);


////////////////////////////////////////////////////////////////////////////
//Execute the query.
////////////////////////////////////////////////////////////////////////////
List docList = null;
try {
      docList = c.query(query,false,"JOE USER");
} catch (Exception e){
      logger.error(e, e);
      return ERROR;
}
if(docList == null){
      logger.fatal("NO DOCUMENT FOUND");
      return ERROR;
}
```

3.2 Use Case 2 – Query Registry: GetDocument Query

Joe User, using his EMR Application, wants to find all documents where:

- The document uniqueId is "1144362162012"

3.2.1 Flow of Execution



3.2.2 API Highlights

The most significant method portions in the above control flow are Consumer.query() and the construction of the FindDocumentsQuery (which extends the more generic Query object). More detail on Consumer.query() can be found in Section 3.1.2. GetDocumentQuery is described below in greater detail. The complete XDS Consumer javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

GetDocumentQuery()

`GetDocumentQuery(java.lang.String docID, boolean isUUID)`

Constructor.

Parameters:

`docID` - id of the document (either uniqueId or entryUUID)

`isUUID` - set tu true if docID is the entryUUID (internal registry identifier) of the document and set to false if it is the uniqueID (external to registry) of the document. In most user cases, this should be set to false

Throws:

`MalformedQueryException`

3.2.3 Sample Code

3.2.3.1 Description

The following sample code illustrates how the XDS Consumer issues a GetDocumentQuery to the XDS Registry.

3.2.3.2 Code

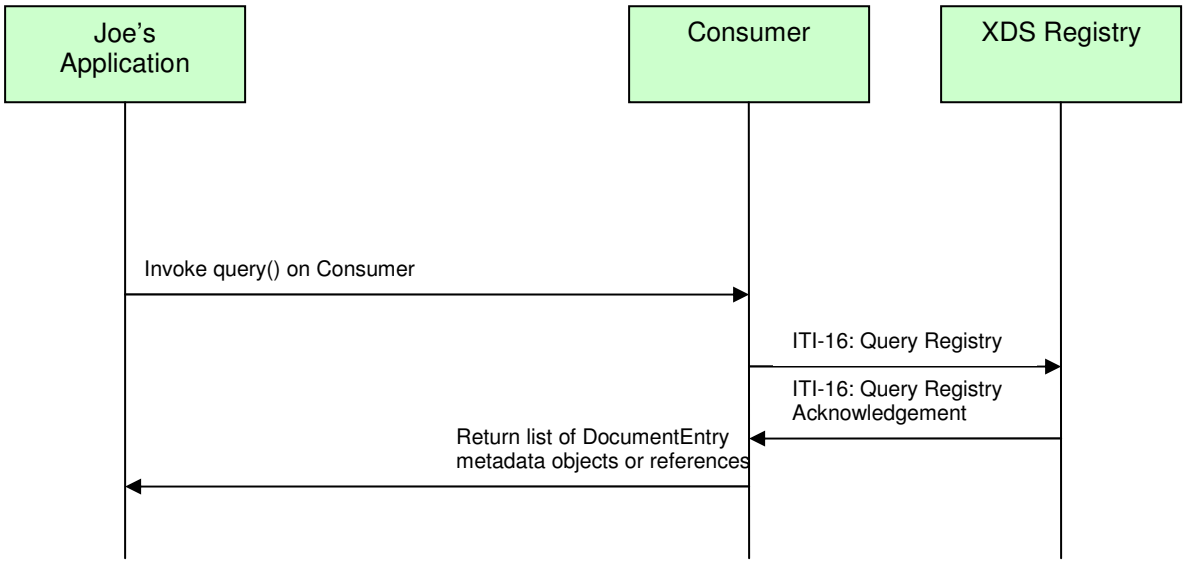
```
////////////////////////////////////////////////////////////////////////////
//We assume a Consumer c has already been appropriately constructed as in
//Section 3.1.3.2
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
//Construct our GetDocumentQuery for document "1144362162012"
////////////////////////////////////////////////////////////////////////////

GetDocumentQuery query = new GetDocumentQuery("1144362162012", true);


////////////////////////////////////////////////////////////////////////////
//Execute the query.
////////////////////////////////////////////////////////////////////////////

DocumentEntryType doc = null;

List docList = null;

try {

      docList = c.query(query,false,"JOE USER");

} catch (Exception e){

      logger.error(e, e);

      return ERROR;

}

if(docList == null){

      logger.fatal("NO DOCUMENT FOUND");

      return ERROR;

}

doc = (DocumentEntryType)docList.get(0);

if(doc == null){

      logger.fatal("NO DOCUMENT FOUND");

      return ERROR;

}
```

3.3 Use Case 3 – Query Registry: Raw SQL Query

Joe User, using his EMR Application, wants to find all documents where:

```
SELECT doc.id FROM ExtrinsicObject doc WHERE doc.id = 'urn:uuid:4c7aa6ba-4c72-7346-9639-000f1fd35f02'
```

3.3.1 Flow of Execution



3.3.2 API Highlights

The most significant method portions in the above control flow is Consumer.query(). This is described below in greater detail. The complete XDS Consumer javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

Method Summary

java.util.List	**query**(java.lang.String sqlQueryString, boolean returnReferencesOnly, java.lang.String initiatingUser) Transaction ITI-16: Query Registry Query for and retrieve a list of document metadata or metadata references based on the input attributes. This method should be used with extreme caution as the SQL query string is not validated in any way before being sent. **Parameters:**

	`sqlQueryString` - query to be executed by the Consumer `returnReferencesOnly` - used to indicate the return type desired by the user. Set returnReferencesOnly = true if only references to document entries are desired. This option is desireable if the user is expecting many documents to satisfy the query, or if memory space is a concern. Set returnReferencesOnly = false if the complete document entry object is desired. `initiatingUser` - entity issuing the query. Can only be null or empty string if auditing is disabled for the Consumer **Returns:** a List of returned data. If returnReferencesOnly = false the returned data will be a List populated with DocumentEntryTypes, containing all metadata for each document satisfying the criteria specified in the argument map. If the returnReferencesOnly = true the returned data will be a list of DocumentEntry.entryUUID string values that reference the DocumentEntries in the registry that satisfy the query. Returns null if no documents are found. **Throws:** `java.lang.Exception`

3.3.3 Sample Code

3.3.3.1 Description

The following sample code illustrates how the XDS Consumer issues a raw SQL query to the XDS Registry.

3.3.3.2 Code

```
////////////////////////////////////////////////////////////////////////////
//We assume a Consumer c has already been appropriately constructed as in
//Section 3.1.3.2
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
//Set up SQL query string
////////////////////////////////////////////////////////////////////////////

String query = "SELECT doc.id FROM ExtrinsicObject doc WHERE doc.id = " +
"\'urn:uuid:4c7aa6ba-4c72-7346-9639-000f1fd35f02\'";


////////////////////////////////////////////////////////////////////////////
//Execute the query.
////////////////////////////////////////////////////////////////////////////

List docList = null;

try {

      docList = c.query(query,false,"JOE USER");

} catch (Exception e){
```

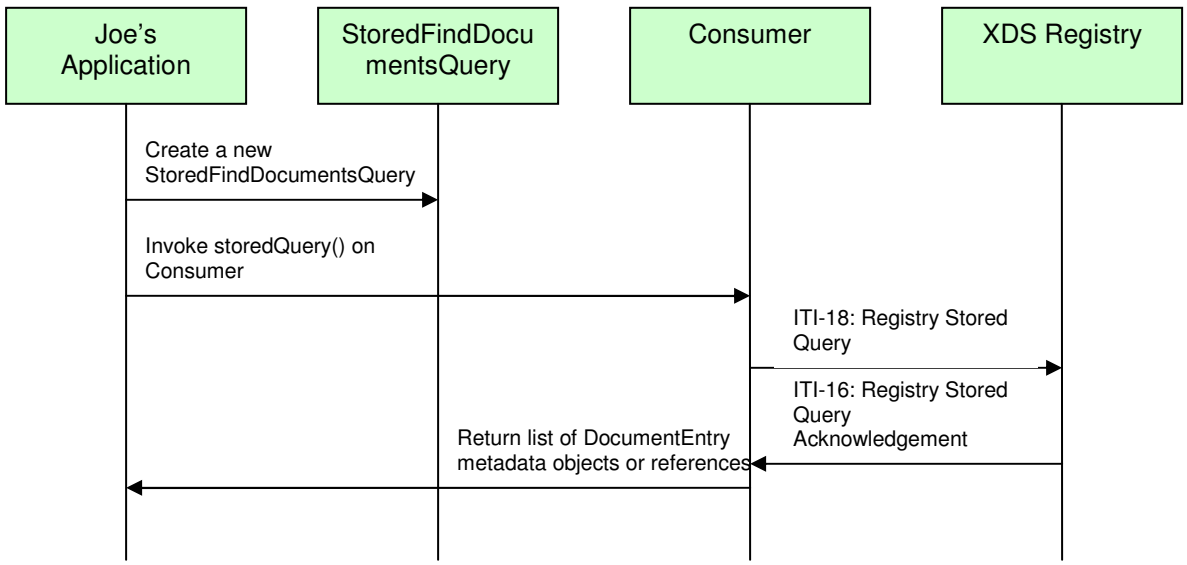
```
        logger.error(e, e);
        return ERROR;
}
if(docList == null){
        logger.fatal("NO DOCUMENT FOUND");
        return ERROR;
}
```

3.4 Use Case 4 – Registry Stored Query: FindDocuments Query

Joe User, using his EMR Application, wants to find all documents where:

- patientID is “JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO”
- Only “Approved” documents are returned
- Creation time on the document is between 20031225 and 20060101
- Healthcare Facility Type Code is “Outpatient”

3.4.1 Flow of Execution



3.4.2 API Highlights

NOTE 1: This API is not implemented yet because the IHE Profile Supplement for Stored Query is still in the editing process and not yet ready for trial implementation.

3.4.3 Sample Code

3.4.3.1 Description

Short description here.

3.4.3.2 Code

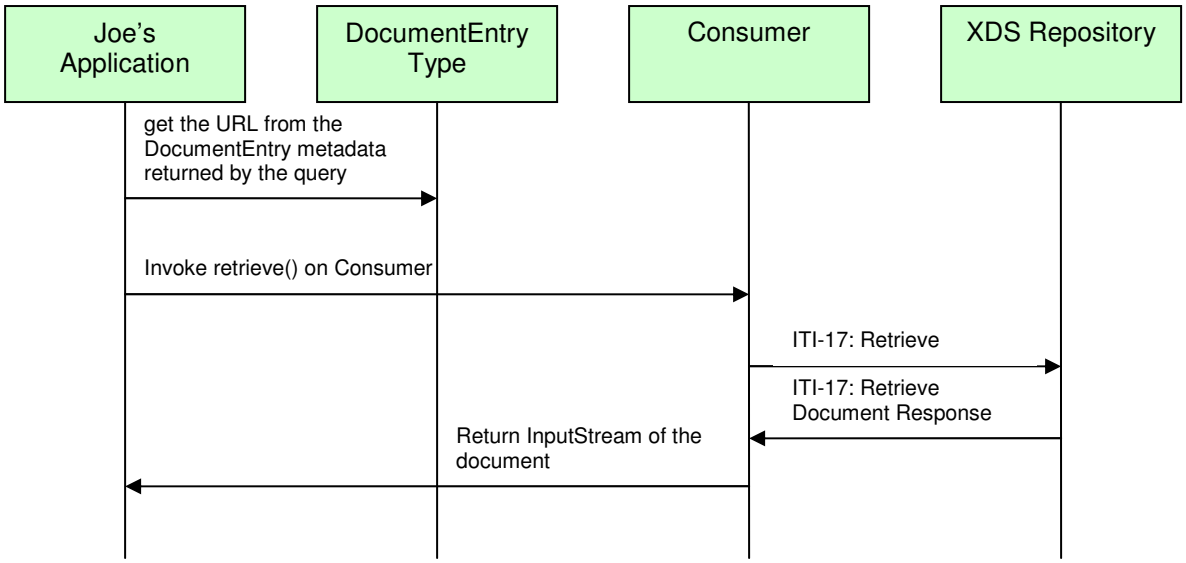
```
code here
```

3.5 Use Case 5 – Retrieve

Joe User, using his EMR Application, wants to retrieve a document he has just queried and found where:

- The document URL is "https://ibmod235.dal-ebis.ihost.com:9044/IHIIRepository/rtrv?ct=6-4-2006&id=714FFBF0&mt=text/xml"

3.5.1 Flow of Execution



3.5.2 API Highlights

The most significant method portions in the above control flow is Consumer.retrieveDocument(). This is described below in greater detail. The complete XDS Consumer javadoc can be downloaded from the Eclipse CVS technology repository. See 2.2 for details.

Consumer.retrieveDocument()	
`java.io.InputStream`	**`retrieveDocument`**`(java.lang.String uri,`

	java.lang.String initiatingUser) Transaction ITI-17: Retrieve Document Retrieve an input stream from which the document content can be read. **Parameters:** uri - The input value is the URI for the document, presumably returned from the getUri() function on an element returned from findDocuments. documentUid - the unique id of the document being imported, needed for auditing only. patientID - clinical affinity domain id of the patient affiliated with the document being imported, needed for auditing only initiatingUser - identity of the user initiating the export transaction. Used only for auditing. If not available specify null. **Throws:** org.eclipse.ohf.ihe.atna.audit.ATNAAuditClientException java.security.GeneralSecurityException java.io.IOException

3.5.3 Sample Code

3.5.3.1 Description

The following sample code illustrates how the XDS Consumer retrieves a document from the XDS Repository.

3.5.3.2 Code

```
////////////////////////////////////////////////////////////////////////////
//We assume a Consumer c has already been appropriately constructed as in
//Section 3.1.3.2. Additionally, we assume a DocumentEntryType docEntry filled
//with XDS metadata for the desired document has be has already obtained from
//the issuing a query to the XDS Registry.
////////////////////////////////////////////////////////////////////////////


// do the retrieve
InputStream is;
try{
      is = c.retrieveDocument(docEntry.getUri()/*, null, null*/, "JOE USER");
}catch(Exception e){
      logger.fatal("Error when attempting to retrieve from: " +
      docEntry.getUri(), e);
```

```
        return ERROR;
}
logger.debug("DONE RETRIEVE: " + docEntry.getUri());
```

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 moved to configuration, currently an API parameter)
- Initiating user ID, for auditing (potentially moved to configuration, currently an API parameter)
- XDS Registry url and port (potentially moved to configuration, currently an API parameter)

6. Debugging Recommendations

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

```
<category name="org.eclipse.ohf.ihe.xds.consumer">
   <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 Consumer

1. Integration of Node Authentication and Auditing – It is not clear at this point how this will affect the current API of the XDS Consumer. We are working with the other OHF members supplying the functionality to ensure that changes are minimized. Much of this may potentially be handle via configuration rather than creating new methods in the API.
2. Stored Query Support – We anticipate adding a new method called “invokeStoredQuery()” to the API that will initiate ITI-18: Registry Stored Query. We also anticipate adding a generic StoredQuery Object with concrete implementations of (minimally) FindDocuments and GetDocument Stored Query.
3. IHE 2006 Change Proposal Integration – The most significant change proposal affecting the XDS Consumer implementation here is that for 2006 the GetDocument query has been changed to GetDocuments query; whereby a list of document uniqueIds or uuids can be provided, rather than just a single uniqueId or uuid.
4. General “Housekeeping” – Several smaller edits are potentially on tap to improve clarity and function of this API including having patientId parameters be metadata CX types rather than String, as well as processing of the error list returned from the query, even when the Registry indicates a “success” status.

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.