OHF XDS Document Consumer

Architecture & API Documentation

Version 0.2.0

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

Contents

1. Introduction 4
2. Getting Started 5
  2.1 Platform Requirements 5
  2.2 Source Files 5
  2.3 Dependencies 5
    2.3.1 Other OHF Plugins 5
    2.3.2 External Sources 6
  2.4 Resources 6
    2.4.1 Eclipse OHF websites 6
    2.4.2 Other OHF Plugin Documentation 6
    2.4.3 IHE ITI Technical Framework 6
    2.4.4 Newsgroup 6
3. API Documentation 8
  3.1 Use Case 1 – Registry Stored Query: FindDocuments Query 8
    3.1.1 Flow of Execution 8
    3.1.2 API Highlights 8
    3.1.3 Sample Code 10
      3.1.3.1 Description 10
      3.1.3.2 Code 10
  3.2 Use Case 2 – Registry Stored Query: GetDocuments Query 11
    3.2.1 Flow of Execution 12
    3.2.2 API Highlights 12
    3.2.3 Sample Code 13
      3.2.3.1 Description 13
      3.2.3.2 Code 13
  3.3 Use Case 3 – Query Registry: FindDocuments Query 13
    3.3.1 Flow of Execution 14
    3.3.2 API Highlights 14
    3.3.3 Sample Code 16
      3.3.3.1 Description 16
      3.3.3.2 Code 16
  3.4 Use Case 4 – Query Registry: Raw SQL Query 17

3.4.1 Flow of Execution 18
3.4.2 API Highlights 18
3.4.3 Sample Code 19
3.4.3.1 Description 19
3.4.3.2 Code 19
3.5 Use Case 5 – Retrieve 20
3.5.1 Flow of Execution 20
3.5.2 API Highlights 20
3.5.3 Sample Code 21
3.5.3.1 Description 21
3.5.3.2 Code 21
4. Security 22
4.1 Node Authentication 22
4.2 Auditing 22
4.2.1 Auditing Example Code 22
4.2.1.1 Description 22
4.2.1.2 Code 22
5. Configuration 25
6. Debugging Recommendations 26
7. Pending Integration and API changes 27
8. Glossary 28

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

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.eclipse.ohf.ihe.common.ebXML._2._1	Model of ebXML v2.1: rim, query and rs
• org.eclipse.ohf.ihe.common.ebXML._3._0	Model of ebXML v3.0: rim, query, rs, lcm and cms
• 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.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 XDS Consumer 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. This problem will not affect the functionality of the XDS Consumer. 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 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

2.4.4 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

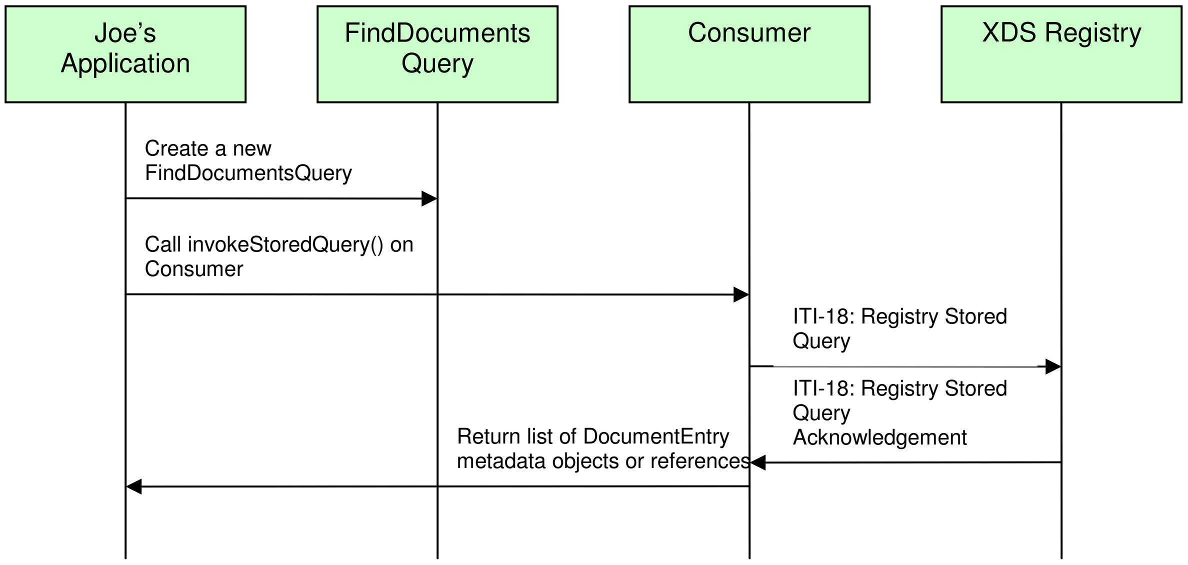
The XDS Document Consumer provides an API for the execution of the following IHE Transactions: ITI-16: Query Registry, ITI-17: Retrieve Document and ITI-18: Registry Stored Query. The API also provides utility functions for the construction of SQL queries for ITI-16 and for stored queries for ITI-18. The API currently supports automatic construction of the FindDocuments and the GetDocuments query for ITI-16 and for ITI-18.

3.1 Use Case 1 – 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”

3.1.1 Flow of Execution



Joe’s Application
FindDocuments Query
Consumer
XDS Registry
Create a new FindDocumentsQuery
Call invokeStoredQuery() on Consumer
ITI-18: Registry Stored Query
ITI-18: Registry Stored Query Acknowledgement
Return list of DocumentEntry metadata objects or references

3.1.2 API Highlights

The most significant method portions in the above control flow are Consumer.invokeStoredQuery() and the construction of the FindDocumentsQuery (which extends the more generic StoredQuery 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.invokeStoredQuery()

`java.util.List`	**`invokeStoredQuery`**`(StoredQuery q, oolean returnReferencesOnly, java.lang.String initiatingUser) throws java.lang.Exception`
	
	Transaction ITI-18: Registry Stored Query
	Query for and retrieve a list of docoment metadata or metadata references based on the input attributes. This is the prefered version of the query method and should be used whenever possible.
	
	Parameters:
	
	`q` - fully populated AdhocQueryType object to be executed by the consumer `AdhocQueryType`
	
	`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`

org.eclipse.ohf.ihe.xds.consumer.storedquery.FindDocumentsQuery()

```
FindDocumentsQuery(CX patientID, CodedMetadataType[] classCodes,
DateTimeRange[] dateTimeRanges, CodedMetadataType[] practiceSettingCodes,
CodedMetadataType[] healthCareFacilityCodes, CodedMetadataType[] eventCodeList,
CodedMetadataType[] confidentialityCodes, CodedMetadataType[] formatCodes,
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) to query on include XDSDocumentEntry.creationTime, XDSDocumentEntry.serviceStartTime, XDSDocumentEntry.serviceStopTime, Choose slotName among metadata time slot name constants provided in {@link org.eclipse.ohf.ihe.xds.metadata.constants.DocumentEntryConstants, 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
`confidentialityCodes` - confidentialityCodes to query for, may be null
`formatCodes` - formatCodesCodes to query for, may be null
`status` - document status list, cannot be null

Throws:

`MalformedStoredQueryException`

3.1.3 Sample Code

3.1.3.1 Description

The following sample code illustrates how the XDS Consumer issues a FindDocumentsQuery Stored Query 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.
////////////////////////////////////////////////////////////////////////////////
String registryURL = "http://my.registry.url:8080";
Consumer c = new Consumer(registryURL);


////////////////////////////////////////////////////////////////////////////////
//Construct the parameters to our FindDocumentsQuery
////////////////////////////////////////////////////////////////////////////////
// Set up the patient ID: "JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO"
CX patientId = Hl7v2Factory.eINSTANCE.createCX();
patientId.setIdNumber("JM19400814");
patientId.setAssigningAuthorityUniversalId("1.3.6.1.4.1.21367.2005.1.1");
patientId.setAssigningAuthorityUniversalIdType("ISO");


// Set up the date-time range for creationTime between Dec 25, 2003 and Jan 01,
//2006
```

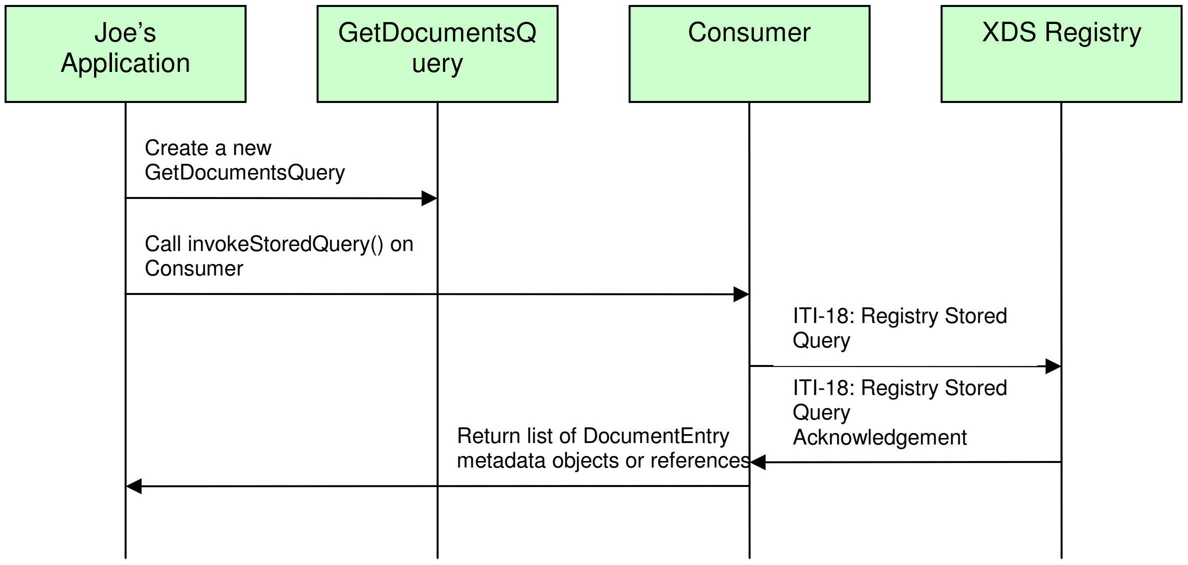
```
DateTimeRange[] creationTimeRange = {new
DateTimeRange(DocumentEntryConstants.CREATION_TIME, “20031225”, “20060101”};


//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”

CodedMetadataType hcfc1 = MetadataFactory.eINSTANCE.createCodedMetadataType();
hcfc1.setCode("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(
            patientId,
            null, // no classCodes
            new DateTimeRange[]{creationTimeRange},
            null, // no practiceSettingCodes
            new CodedMetadataType[]{hcfc1},
            null, // no eventCodes
            null, // no confidentialityCodes
            null, // no formatCodes
            status);


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

List docList = null;

try {
      c.invokeStoredQuery(query, false, "JOE USER");
} catch (Exception e) {
      System.out.println(e.toString());
      throw e;
}

if(docList == null){

      System.out.println("NO DOCUMENT FOUND");

}
```

3.2 Use Case 2 – Registry Stored Query: GetDocuments Query

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

- The document uniqueId is “1144362162012”

3.2.1 Flow of Execution


Joe’s Application
GetDocumentsQuery
Consumer
XDS Registry
Create a new GetDocumentsQuery
Call invokeStoredQuery() on Consumer
ITI-18: Registry Stored Query
ITI-18: Registry Stored Query Acknowledgement
Return list of DocumentEntry metadata objects or references

3.2.2 API Highlights

The most significant method portions in the above control flow are Consumer.invokeStoredQuery(), explained in the previous section, and the construction of the GetDocumentsQuery (which extends the more generic StoredQuery object). GetDocumentsQuery 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.

org.eclipse.ohf.ihe.xds.consumer.storedquery.GetDocumentQuery()

```
GetDocumentsQuery(java.lang.String[] docIDs,
                         boolean isUUID)
                  throws MalformedStoredQueryException
```

Constructor.

Parameters:

`docIDs` - list of ids of the documents (either uniqueId or entryUUID)

`isUUID` - set to 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:

`MalformedStoredQueryException`

3.2.3 Sample Code

3.2.3.1 Description

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

3.2.3.2 Code

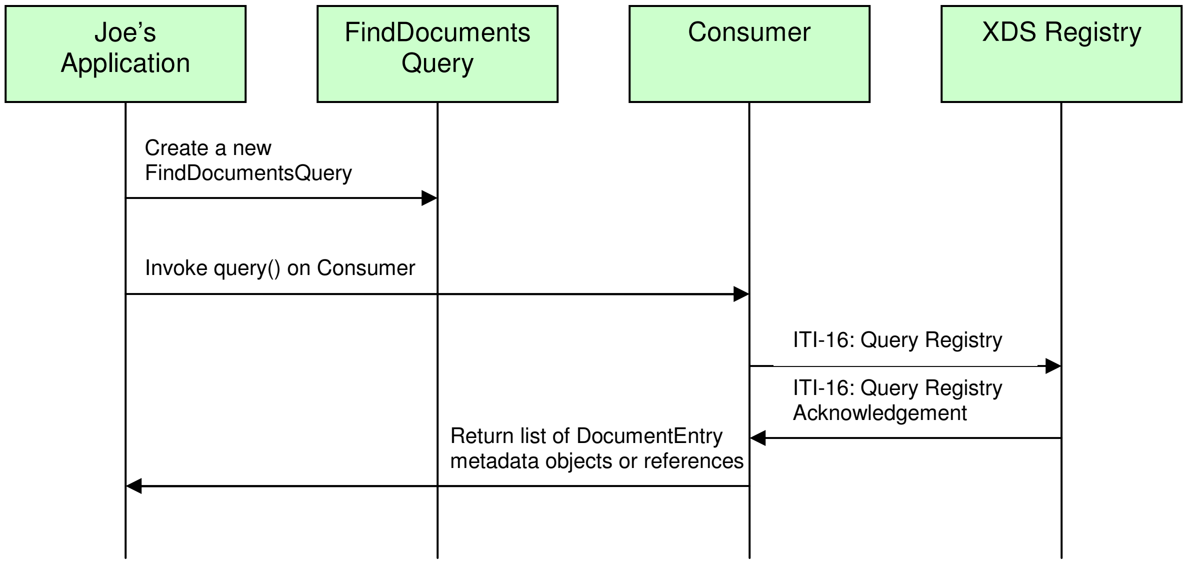
```
////////////////////////////////////////////////////////////////////////////////
//We assume a Consumer c has already been appropriately constructed as in
//Section 3.1.3.2
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//Construct our GetDocumentQuery for document with uniqueID "1144362162012";
//thus, the argument for the "isUUID" parameter is 'false'.
////////////////////////////////////////////////////////////////////////////////

GetDocumentQuery query = new GetDocumentQuery(new String[]{"1144362162012"},
false);


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

List docList = null;

try {
      c.invokeStoredQuery(query, false, "JOE USER");
} catch (Exception e) {
      System.out.println(e.toString());
      throw e;
}

if(docList == null){

      System.out.println("NO DOCUMENT FOUND");

}
```

3.3 Use Case 3 – 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"

3.3.1 Flow of Execution


Joe's Application
FindDocuments Query
Consumer
XDS Registry
Create a new FindDocumentsQuery
Invoke query() on Consumer
ITI-16: Query Registry
ITI-16: Query Registry Acknowledgement
Return list of DocumentEntry metadata objects or references

3.3.2 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   java.util.List query(Query q,
                                      boolean returnReferencesOnly,
                                      java.lang.String initiatingUser)
                                      throws java.lang.Exception
```

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 `Query`

`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`

org.eclipse.ohf.ihe.xds.consumer.query.FindDocumentsQuery()

```
public FindDocumentsQuery(CX patientID,
                          CodedMetadataType[] classCodes,
                          DateTimeRange[] dateTimeRanges,
                          CodedMetadataType[] practiceSettingCodes,
                          CodedMetadataType[] healthCareFacilityCodes,
                          CodedMetadataType[] eventCodeList,
                          CodedMetadataType[] confidentialityCodes,
                          CodedMetadataType[] formatCodes,
                          AvailabilityStatusType[] status)
                   throws MalformedQueryException
```

Constructor with full set of parameters.

Parameters:

`patientID` - id of patient, non null

`classCodes,` - classCodes to query for, may be null

`dateTimeRanges,` - date and time ranges (use DateTimeRange) to query on include XDSDocumentEntry.creationTime, XDSDocumentEntry.serviceStartTime, XDSDocumentEntry.serviceStopTime, Choose slotName among metadata time slot name constants provided in {@link org.eclipse.ohf.ihe.xds.metadata.constants.DocumentEntryConstants, 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

`confidentialityCodes` - confidentialityCodes to query for, may be null

`formatCodes` - formatCodesCodes to query for, may be null

`status` - document status list, cannot be null

Throws:

MalformedQueryException

3.3.3 Sample Code

3.3.3.1 Description

The following sample code illustrates how the XDS Consumer issues a FindDocumentsQuery SQL-based Query to the XDS Registry.

3.3.3.2 Code

```
////////////////////////////////////////////////////////////////////////////////
//If an instance does not already exist, create an instance of the XDS Consumer
//and provide the XDS Registry url and port.
////////////////////////////////////////////////////////////////////////////////
String registryURL = "http://my.registry.url:8080";

Consumer c = new Consumer(registryURL);


////////////////////////////////////////////////////////////////////////////////
//Construct the parameters to our FindDocumentsQuery
////////////////////////////////////////////////////////////////////////////////
// Set up the patient ID: "JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO"

CX patientId = Hl7v2Factory.eINSTANCE.createCX();
patientId.setIdNumber("JM19400814");
patientId.setAssigningAuthorityUniversalId("1.3.6.1.4.1.21367.2005.1.1");
patientId.setAssigningAuthorityUniversalIdType("ISO");


// Set up the date-time range for creationTime between Dec 25, 2003 and Jan 01,
//2006

DateTimeRange[] creationTimeRange = {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"

CodedMetadataType hcfc1 = MetadataFactory.eINSTANCE.createCodedMetadataType();
hcfc1.setCode("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};
```

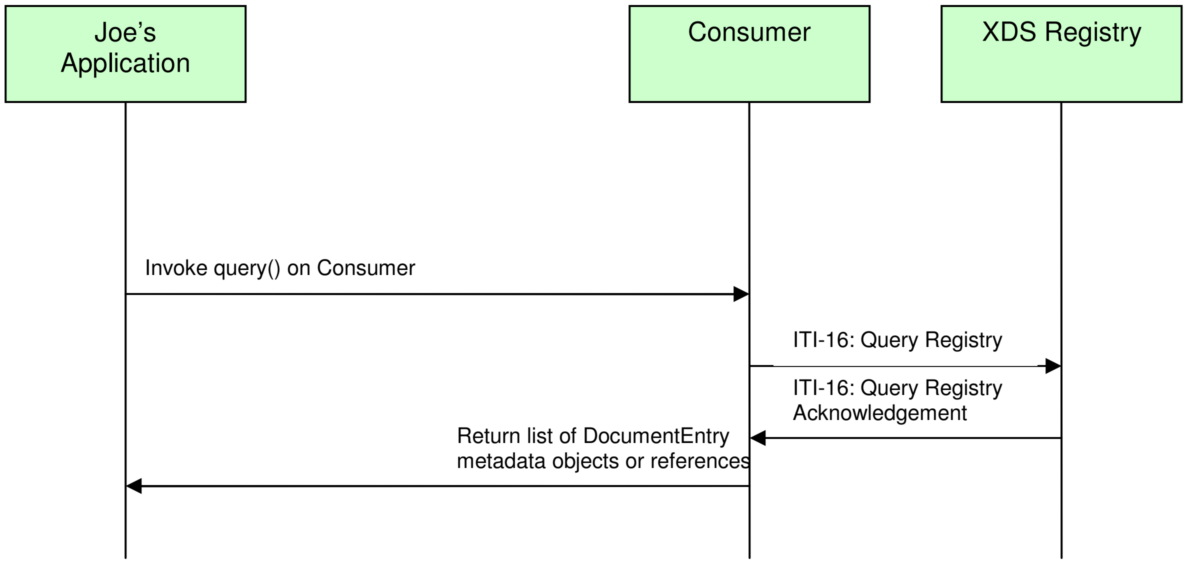
```
////////////////////////////////////////////////////////////////////////////////
//Construct our FindDocumentsQuery for patient
//"JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO"
////////////////////////////////////////////////////////////////////////////////
FindDocumentsQuery query = new FindDocumentsQuery(
            patientId,
            null, // no classCodes
            new DateTimeRange[]{creationTimeRange},
            null, // no practiceSettingCodes
            new CodedMetadataType[]{hcfc1},
            null, // no eventCodes
            null, // no confidentialityCodes
            null, // no formatCodes
            status);


////////////////////////////////////////////////////////////////////////////////
//Execute the query.
/////////////////////////////////////////////////////////////////////////////
List docList = null;
try {
      c.query(query, false, "JOE USER");
} catch (Exception e) {
      System.out.println(e.toString());
      throw e;
}
if(docList == null){

      System.out.println("NO DOCUMENT FOUND");

}
```

3.4 Use Case 4 – 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'
```

3.4.1 Flow of Execution


Joe's Application
Consumer
XDS Registry
Invoke query() on Consumer
ITI-16: Query Registry
ITI-16: Query Registry Acknowledgement
Return list of DocumentEntry metadata objects or references

3.4.2 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`	`public java.util.List` **`query`**`(java.lang.String sqlQueryString, boolean returnReferencesOnly, java.lang.String initiatingUser) throws java.lang.Exception` 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 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`

3.4.3 Sample Code

3.4.3.1 Description

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

3.4.3.2 Code

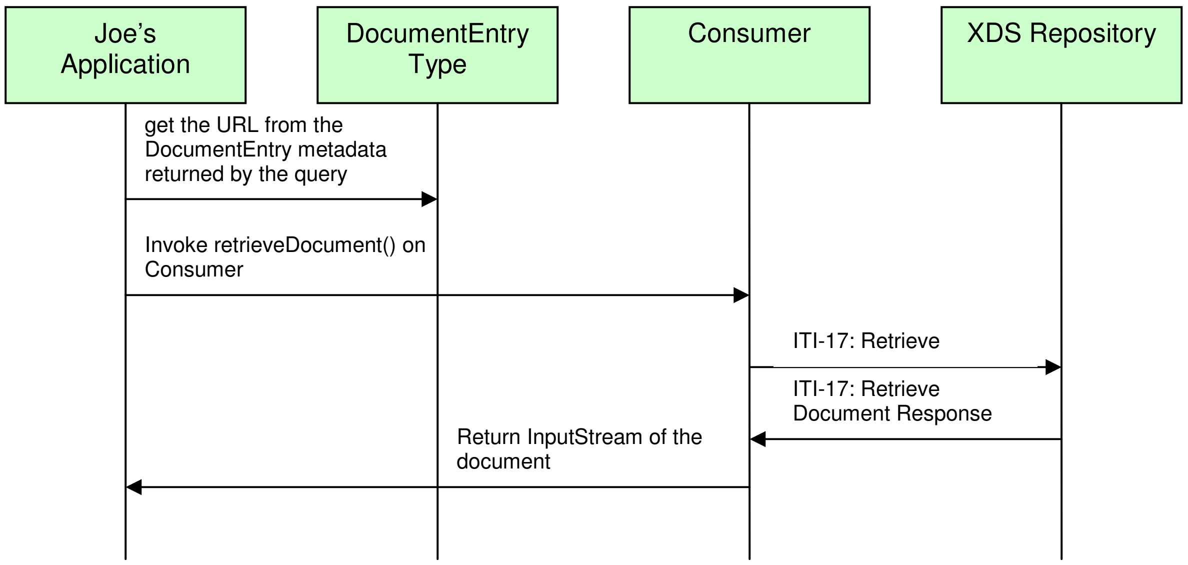
```
////////////////////////////////////////////////////////////////////////////////
//We assume a Consumer c has already been appropriately constructed as in
//Section 3.1.3.2
////////////////////////////////////////////////////////////////////////////////

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

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


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

List docList = null;

try {
      c.query(queryString, false, "JOE USER");
} catch (Exception e) {
      System.out.println(e.toString());
      throw e;
}

if(docList == null){

      System.out.println("NO DOCUMENT FOUND");
```

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 "http://my.repository.url:8080/IHIIRepository/rtrv?ct=6-4-2006&amp;id=714FFBF0&amp;mt=text/xml"

3.5.1 Flow of Execution


Joe's Application
DocumentEntry Type
Consumer
XDS Repository
get the URL from the DocumentEntry metadata returned by the query
Invoke retrieveDocument() on Consumer
ITI-17: Retrieve
ITI-17: Retrieve Document Response
Return InputStream of the document

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) throws java.io.IOException, ATNAAuditClientException, java.security.GeneralSecurityException` 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.
	`initiatingUser` - identity of the user initiating the export transaction. Used only for auditing.
	Throws:
	`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 already been obtained from
//the issuing a query to the XDS Registry.
////////////////////////////////////////////////////////////////////////////////


// do the retrieve

InputStream document = null;

try{

      document = c.retrieveDocument(docEntry.getUri(), "JOE USER");

catch(Exception e){

      // if an exception happens, the retrieve has failed

      System.out.println("Error when attempting to retrieve from: " +
      docEntry.getUri(), e);

      throw e;

}

if(document == null){
      // if null is returned, then the repository returned null, something else
      // is wrong.
      System.out.println("Document InputStream is null.");
      throw new Exception("Document InputStream is null.");
}
```

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 Consumer 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 any of the OHF XDS Consumer transactions are 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 Consumer.

4.2.1.2 Code

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

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

```
Consumer c = new Consumer(registryURL);


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

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


////////////////////////////////////////////////////////////////////////////////
//Construct the parameters to our FindDocumentsQuery
////////////////////////////////////////////////////////////////////////////////
// Set up the patient ID: “JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO”

CX patientId = Hl7v2Factory.eINSTANCE.createCX();
patientId.setIdNumber("JM19400814");
patientId.setAssigningAuthorityUniversalId("1.3.6.1.4.1.21367.2005.1.1");
patientId.setAssigningAuthorityUniversalIdType("ISO");


////////////////////////////////////////////////////////////////////////////////
//Construct our FindDocumentsQuery for patient
//“JM19400814^^^&1.3.6.1.4.1.21367.2005.1.1&ISO”
////////////////////////////////////////////////////////////////////////////////
FindDocumentsQuery q = new FindDocumentsQuery(patientId, new
AvailabilityStatusType[]{AvailabilityStatusType.APPROVED_LITERAL});


////////////////////////////////////////////////////////////////////////////////
//Execute the query.
////////////////////////////////////////////////////////////////////////////////
List docList = null;

try {
      c.query(queryString, false, "JOE USER");
} catch (Exception e) {
      System.out.println(e.toString());
      throw e;
}

if(docList == null){

      System.out.println("NO DOCUMENT FOUND");

      System.exit(0);

}

if(docList.size() == 0){
      System.out.println("NO DOCUMENT FOUND");

      System.exit(0);
}


/////////////////////////////////////////////////////////////////////////
// Get the first document entry URL and retrieve
/////////////////////////////////////////////////////////////////////////

DocumentEntryType docEntry = (DocumentEntryType)docList.get(0);
if(docEntry.getUri() == null){
      System.out.println("Malformed DocumentEntry.URI is null.");
      throw new Exception("Malformed DocumentEntry.URI is null.");
}

//Turn off auditing for the retrieve (suppose we want to do this … for some
//reason).
AtnaAgentFactory.getAtnaAgent().setDoAudit(true);

// finally get the document
InputStream document = c.retrieveDocument(docEntry.getUri(), "JOE USER");
if(document == null){
      System.out.println("Document InputStream is null.");
      throw new Exception("Document InputStream is null.");
}
```

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 Consumer)
- Logging is configurable. For more information about logging, consult Section 6 of this document.

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. Bugzilla bug 165105 (https://bugs.eclipse.org/bugs/show_bug.cgi?id=165105) outlines a potential API change for the next phase of this XDS Consumer.
2. In the next phase of the XDS Consumer, Stored Query support for the entire minimum query catalog specified in ITI-18 is anticipated to be enabled. We are considering expanding minimum query catalog support for the Registry Query transaction.
3. We anticipate supporting the new web-service version of the Retrieve Transaction that is in development for the 2007-2008 season.

8. Glossary

No glossary terms at this time.