Patient Identifier Cross-reference Consumer

Architecture & API Documentation

Version 0.1.1

srrenly{at}us{dot}ibm{dot}com | Sondra R Renly

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 (Eclipse OHF) 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 documentation addresses the beta release of the Eclipse OHF plugin implementation of the IHE ITI Technical Framework actor Patient Identifier Cross-reference Consumer for the implementation of the ITI-9 PIX Query 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	http://www.eclipse.org/downloads/
Java JDK 1.4.2	http://java.sun.com/javase/downloads/index.jsp

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.common.hl7v2.client
- org.eclipse.ohf.ihe.pix.consumer

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

2.3 Dependencies

The Patient Identifier Cross-reference Consumer has dependencies on other OHF plugins and external sources.

2.3.1 Other OHF Plugins

Patient Identifier Cross-reference Consumer plugins are dependent on additional org.eclipse.ohf project plugins. You also need to check-out the following:

• org.eclipse.ohf.hl7v2.core org.eclipse.ohf.utilities org.apache.commons org.apache.axis org.xmlpull.v1	HL7v2 message object plugins and dependencies
• org.eclipse.ohf.ihe.common.mllp	Minimum Lower Level Protocol
• org.eclipse.ohf.ihe.atna.audit	Auditing for messages sent and responses received
• org.eclipse.ohf.ihe.common.hl7v2	HL7v2 segment and field definitions (temporary)
• org.apache.log4j	Debug, warning, and error logging

2.3.2 External Sources

For the purpose of message object creation and verification, either a licensed copy of the HL7 access database (hl7_58.mdb) or a free javaStream file (hl7Definitions.javaStream) is necessary. You can obtain the free javaStream file from:

org.eclipse.ohf.hl7v2.ui > Resources > hl7Definitions.javaStream.

If you would like to obtain a copy of the HL7 access database file, refer to http://www.hl7.org.

To complete message verification, an XML conformance profile is necessary. A sample Q23 Get Corresponding IDs (HL7v2.5) conformance profile is included with this plugin in the /resources/conf folder.

2.4 Resources

The following resources are recommended.

2.4.1 Other OHF plugin documentation

The following OHF plugin documents are related to the Patient Identifier Cross-reference Consumer:

- OHF ATNA Audit Client (http://wiki.eclipse.org/index.php/OHF_IHE_Client_plugins)

2.4.2 HL7 Standard 2.5

The Patient Identifier Cross-reference Consumer references standards HL7 version 2.5.

http://www.hl7.org.

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.

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 Patient Identifier Cross-reference Consumer client supports three formats for input. The client will accept:

- a raw HL7 message (String)
- an HL7v2 message object (org.eclipse.ohf.hl7v2.core Message)
- a ITI-9 PIX Query message supporting the message construction of event:

  QBP^Q23 – Get Corresponding Identifiers

Examples for the three types of inputs are found in the org.eclipse.ohf.ihe.pix.consumer plugin.

org.eclipse.ohf.ihe.pix.consumer > src_tests > org.eclipse.ohf.ihe.pix.consumer.tests > HL7PixQuery.java
org.eclipse.ohf.ihe.pix.consumer > src_tests > org.eclipse.ohf.ihe.pix.consumer.tests > MSGPixQuery.java
org.eclipse.ohf.ihe.pix.consumer > src_tests > org.eclipse.ohf.ihe.pix.consumer.tests > ClientPixQuery.java

The files in src_tests use a TestConfiguration.java file for extracting the various file locations and MLLP connection parameters. Update this file with your settings before running the sample code.

A raw HL7 message string should be used as input when the originating application is fully capable of sending and receiving HL7 messages. In this case, the Patient Identifier Cross-reference Consumer client is simply providing auditing, communication with the PIX server, and optional message verification. Server responses are returned to the caller as raw HL7v2 message strings. (HL7PixQuery)

A message object should be used as input when the originating application is directly using the OHF HL7v2 component which the Patient Identifier Cross-reference Consumer client sits on top of. In this case, the application has taken full responsibility for message creation and reading the response. The Patient Identifier Cross-reference Consumer client is simply providing conversion to raw HL7, auditing, communication with the PIX server, and optional message verification. Server responses are returned to the caller as HL7v2 message objects. (MSGPixQuery)

A ITI-9 PIX Query message should be used as input when the originating application has neither support for rawHL7 nor message objects. The Patient Identifier Cross-reference Consumer client provides a friendly interface to set and read message fields as well as auditing, communication with the PIX server, and optional message verification. Server responses are returned to the caller as PixConsumerResponse objects. (ClientPixQuery)

ITI-9 PIX Query Message Class

- PixConsumerQuery

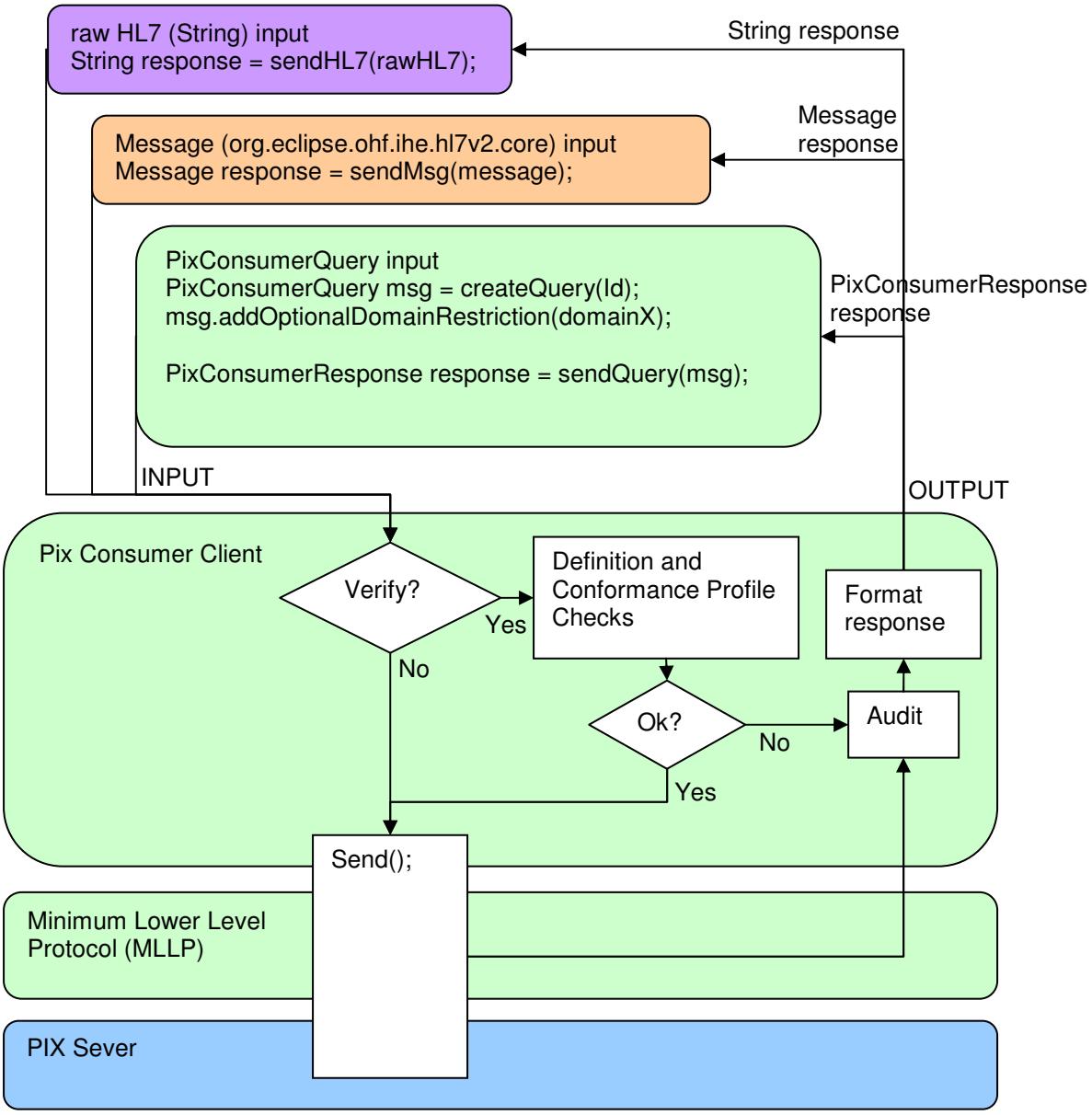
ITI-9 PIX Query Server Response Class

- PixConsumerResponse

3.1 Use Case - ITI-9 PIX Query

The Patient Identifier Cross-reference Consumer has one transaction. This use case demonstrates in step-by-step and with sample code the creation and use of the Patient Identifier Cross-reference Consumer Client, including the three input options. It includes example client construction of the event specific message object as input but not the creation of raw HL7 or HL7v2 Message object.

3.1.1 Flow of Execution



Create a Patient Identifier Cross-reference Consumer object:

1. Construct ITI-9 PIX Query
2. Construct and associate MLLP (Minimum Lower Level Protocol) Destination
3. Enable auditing.
4. Override the maximum level of validation error allowed before message submission is halted. The levels of error are constants in the OHF HL7v2 CPValidator.java file. The default is to allow up to the warning level.

Create a tailored HL7v2 message object:

1. Create Patient Identifier Cross-reference Consumer Message. Message field defaults are obtained first from the associated Conformance Profile. Required fields not found there default to settings for the IBM Dallas Server. (http://ibmod235.dal-ebis.ihost.com:9080/IBMIHII/serverInfoOHF.htm)
2. Change default settings.
3. Add optional field values.
4. As not all fields have a corresponding method, use the generic method to set these additional values. Use method .setField(alias, data).

The Patient Identifier Cross-reference Consumer supports populating data in MSH, QPD, and RCP segments. Information about the fields, components, and sub-components available in these segments is available in the HL7 Version 2.5 Standard document in Chapter 2 Section 2.15 Message Control Segments (MSH) and Chapter 5 Section 5.5 Query/Response Message Segments (QPD/RCP). Note that the ITI-9 PIX Query RCP segment is complete when the message is created and no further manipulation is required.

Send the message:

1. Send message

Read the response message:

1. Read response message fields.

3.1.2 Source Code

Create a Patient Identifier Cross-reference Consumer Query:

1. Construct ITI-9 PIX Query

   There are three ways to construct the ITI-9 PIX Query client. If you have HL7 input and are not using message validation, no constructor parameters are necessary. Otherwise, you are required to provide the HL7 definition file (the access database file or javaStream file) and optionally a conformance profile.

   In this sample code, TConfig refers to the TestConfiguration.java file mentioned in the beginning of this section. See this file for example formatting of these constants.

```
//pixQuery set-up: no supporting files
pixQuery = new PixConsumer();

//pixQuery set-up: supporting access database & conformance profile
String msAccessFile = TConfig.ACCESS_DATABASE_PATH;
InputStream cpaStream = new FileInputStream(TConfig.CPROFILE_PATH);
pixQuery = new PixConsumer(msAccessFile, cpaStream);

//pixQuery set-up: javaStream setup example
Stream javaStream = new Stream(new File(TConfig.SERIALISED_PATH));
InputStream cpStream = new FileInputStream(TConfig.CPROFILE_PATH);
pixQuery = new PixConsumer(javaStream, cpStream);
```

2. Construct and associate MLLP (Minimum Lower Level Protocol) Destination

   Un-Secure Connection:

```
MLLPDestination mllp =
new MLLPDestination(TConfig.MLLP_HOST, TConfig.MLLP_PORT);
pixQuery.setMLLPDestination(mllp);
```

   Secure Connection:

```
SecureTCPPort tcpPort = new SecureTCPPort();
tcpPort.setTcpHost(TestConfiguration.MLLP_HOST);
tcpPort.setTcpPort(TestConfiguration.MLLP_SECUREPORT);
tcpPort.setKeyStoreName(TestConfiguration.MLLP_KEYSTORE_NAME);
tcpPort.setKeyStorePassword(TestConfiguration.MLLP_KEYSTORE_PASSWORD);
tcpPort.setTrustStoreName(TestConfiguration.MLLP_TRUSTSTORE_NAME);
tcpPort.setTrustStorePassword(TestConfiguration.MLLP_TRUSTSTORE_PASSWORD);
tcpPort.setClientAuthNeeded(TestConfiguration.MLLP_NEEDS_CLIENT_AUTH);
tcpPort.setProtocol(TestConfiguration.MLLP_PROTOCOL);
tcpPort.setSslProviderName(TestConfiguration.MLLP_SSLPROVIDER_NAME);
tcpPort.setSslProviderClass(TestConfiguration.MLLP_SSLPROVIDER_CLASS);
MLLPDestination mllp = new MLLPDestination(tcpPort);
```

3. Enable auditing.

```
pixQuery.setDoAudit(true);
```

4. Override the maximum level of validation error allowed before message submission is halted. The levels of error are constants in the OHF HL7v2 CPValidator.java file. The default is to allow up to the warning level.

```
ITEM_TYPE_INFORMATION = 1;
ITEM_TYPE_WARNING = 2;
ITEM_TYPE_ERROR = 3;
ITEM_TYPE_FATAL = 4;

pixQuery.setMaxVerifyEvent(CPValidator.ITEM_TYPE_INFORMATION);
```

Create a tailored HL7v2 message object:

1. Create Patient Identifier Cross-reference Consumer Message.

   It is highly recommended to use the Patient Identifier Cross-reference Consumer to create the query rather than use the PixConsumerQuery constructor as the underlying manager is taken care of. The patient ID parameter is the QPD-3-1 field and the assigningAuthority parameters are within the QPD-3-4 field.

```
PixConsumerQuery msg = pixQuery.createQuery(patientId);
or
PixConsumerQuery msg = pixQuery.createQuery(patientId,
assigningAuthorityName, assigningAuthorityUniversalId,
assigningAuthorityUniversalIdType);
```

2. Change default settings.

```
msg.changeDefaultCharacterSet("UNICODE");
```

3. Add optional query values.
   You may add multiple domain restrictions by calling this method for each individual domain.

```
msg.addOptionalDomainRestriction("OHF");
```

4. As not all fields have a corresponding method, use the generic method to set these additional values. Use method .setField(alias, data).

```
msg.setField("QPD-3-1", patientId);
```

   Note that this method does not handle repetitions, such as required by the Domain Restriction QPD-4-4 field. If the additional values require repeat support, use the methods exposed through the Eclipse OHF HL7v2 Message object. See the PixConsumerQuery addOptionalDomainRestriction method for sample code.

Send the message:

1. Send message

```
String response = pixQuery.sendHL7(msg, isValidateOn, auditUser);
Message response = pixQuery.sendMsg(msg, isValidateOn, auditUser);
PixConsumerResponse
response = pixQuery.sendQuery(msg, isValidateOn, auditUser);
```

Read the response message:

1. Read response

```
//HL7v2 message object
msg.getElement("MSA-1").getAsString();    //message AckCode
msg.getElement("QAK-2").getAsString();    //message QueryStatus

//PixConsumerResponse object
response.getResponseAckCode(isExpandCodeToString);
response.getQueryStatus(isExpandCodeToString);
response.getErrorCode();
```

4. Security

4.1 Node Authentication

Transport Layer Security Protocol (TLS) is supported by creating a secure MLLP connection. Information required to instantiate a secure connection to the IBM Dallas Server is available in the sample code configuration file. For more information and use terms on the IBM Dallas Server, see http://ibmod235.dal-ebis.ihost.com:9080/IBMIHII/serverInfoOHF.htm

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.

Auditing is enabled with the doAudit boolean variable used within the Patient Identifier Cross-reference Consumer client.

```
boolean doAudit = true;

PixConsumer pixQuery = new PixConsumer();
pixQuery.setDoAudit(doAudit);
```

5. Configuration

There are two types of configuration in this release.

Create message default field values, such as message header fields, can now be read from the conformance profile field ConstantValue attribute. Currently this is only for single component fields. Overloading the string with component or sub-component delimited data is not supported.

```
<Field Name="MSH-1 Field Separator" … ConstantValue="|">
<Field Name="MSH-2 Encoding Characters" … ConstantValue="^~\&amp;">
<Field Name="MSH-3 Sending Application" … ConstantValue="OTHER_KIOSK">
<Field Name="MSH-4 Sending Facility" … ConstantValue="HIMSSSANDIEGO">
```

The sample code files in src_tests now use a TestConfiguration.java file for extracting the various file locations and MLLP connection parameters. Update the default values in this file as needed with your settings before running the code. Here are the fields that are configured in this file:

```
//basics
public static final String DATA_PATH
public static final String LOG4J_PATH

//HL7PixQuery - run from file
public static final String HL7FILE_PATH

//Enable examples using the Access DB File
//This file is licensed through HL7 and not provided as part of this
//plugin
public static final boolean HAS_ACCESSFILE
public static final String ACCESS_DATABASE_PATH

//Enable examples using the free javaStream file
public static final boolean HAS_JAVASTREAM
public static final String SERIALISED_PATH

//Conformance profile for second level HL7 verification
public static final String CPROFILE_PATH

//MLLP Connectivity:
//Default IBM Dallas IHII Server - more connection info available at:
//http://ibmod235.dal-ebis.ihost.com:9080/IBMIHII/serverInfoOHF.htm
public static final String MLLP_HOST

//Unsecure connection port
public static final int MLLP_PORT

//TLS: Secure connection parameters
public static final int MLLP_SECUREPORT
public static final String MLLP_KEYSTORE_NAME
public static final String MLLP_KEYSTORE_PASSWORD
public static final String MLLP_TRUSTSTORE_NAME
public static final String MLLP_TRUSTSTORE_PASSWORD
public static final boolean MLLP_NEEDS_CLIENT_AUTH
public static final String MLLP_PROTOCOL
public static final String MLLP_SSLPROVIDER_NAME
public static final String MLLP_SSLPROVIDER_CLASS
```

6. Debugging Recommendations

Log statements have been entered throughout the Patient Identifier Cross-reference Consumer plugin source code for assistance in monitoring the progress of the running client. To enable logging, there is a Log4j configuration file.

An example log4j configuration file is in the file /resources/conf/pixconsumer_log4j.xml. The default configuration creates the log file in the folder /resources/log. You may need to create a file named pixquery.log as the log folder does not have the file as received from CVS.

7. Release Fixes and Enhancements

The primary enhancement of this release is the addition of the Eclipse OHF HL7v2 message verification functionality. HL7v2 message verification relies on a two step process; a definition file and an XML conformance profile. The definition file comes from HL7 while the XML conformance profile is configurable to allow site specific limitations.

HL7v2 message verification is currently an incomplete implementation if you are using the string or Message object input options. Monitor Eclipse Bugzilla for the HL7v2 OHF component to determine when it will be advantageous to enable message verification.

7.1 Release Fixes

1. Corrected a false dependency on org.apache.junit. This dependency has been removed.
2. Corrected the PixConsumerQuery changeDefaultAssigningAuthorityUniveralId and changeDefaultAssigningAuthorityUniversalIdType misspelling of the word Authority.
3. Corrected the PixConsumerResponse method getResponseAck, changing it to getResponseAckCode.
4. Corrected variable names containing “ID”, changing them to “Id” for consistency. For example, getControlID was changed to getControlId.
5. Corrected variable names containing “_”, changing them to Java standard for consistency. For example, patient_class was changed to patientClass.
6. Corrected an index out of bounds error that occurred when accessing the MSH Segment getSendingFacility method for a server response message.
7. Corrected the TestConfiguration.java path provided for the location of the javaStream file. It is now referencing the release location within org.eclipse.ohf.ihe.common.hl7v2.client.
8. Removed the dual-use of doAudit for both auditing and logging. Now logging is controlled fully by the log4j configuration file.
9. In response to an HL7v2 code change, the javaStream input type has been changed from Stream to PrivateFormat. Also, the filename has changed from hl7Definitions.javaStream to hl7Definitions.stream. This fix has removed the need to perform any of the workarounds previously listed at the end of this section.

7.2 Release Enhancements

1. Implementations using only rawHL7 strings with no intermediate verification are now able to instantiate the Patient Identifier Cross-reference Consumer without the HL7 definition file. This dependency has been removed. If no HL7 definition file is provided and verification is requested in the sendHL7 method, a log message is generated and the message sent without verification.

```
PixConsumer pixQuery = new PixConsumer();
pixQuery.setMLLPDestination(createMLLP());
String message = createMessage();
String response = pixQuery.sendHL7(message, false, auditUser);
readReturn(response);
```

2. Added Eclipse OHF HL7v2 message verification functionality. HL7v2 message verification supports a two step process, a required definition file and an optional XML conformance profile. The definition file comes from HL7 and can be either the HL7 access database file (licensed through HL7) or the javaStream file that is now provided free with the HL7v2 component. The XML conformance profile is configurable to allow site specific limitations. A sample Q23 conformance profile is included in the Patient Identifier Cross-reference Consumer component plugin.

   HL7 access database file example:

```
String msAccessFile = TConfig.ACCESS_DATABASE_PATH;
InputStream cpaStream = new FileInputStream(TConfig.CPROFILE_PATH);
PixConsumer pixQuery = new PixConsumer(msAccessFile, cpaStream);
String response = pixQuery.sendHL7(message, true, auditUser);
```

   javaStream file example:

```
Stream javaStream = new Stream(new File(TConfig.SERIALISED_PATH));
InputStream cpStream = new FileInputStream(TConfig.CPROFILE_PATH);
PixConsumer pixQuery = new PixConsumer(javaStream, cpStream);
String response = pixQuery.sendHL7(message, true, auditUser);
```

3. Create message default field values, such as message header fields, can now be read from the conformance profile field ConstantValue attribute. Currently this is only for single component fields. Overloading the string with component or sub-component delimited data is not supported.

```
<Field Name="MSH-1 Field Separator" … ConstantValue="|">
<Field Name="MSH-2 Encoding Characters" … ConstantValue="^~\&amp;">
<Field Name="MSH-3 Sending Application" … ConstantValue="OTHER_KIOSK">
<Field Name="MSH-4 Sending Facility" … ConstantValue="HIMSSSANDIEGO">
```

4. Stop the submission of a message based on the maximum level of allowed errors reported by HL7v2 message verification. The default is to only send a message that has received information or warning errors. The four levels of error include:

```
ITEM_TYPE_INFORMATION = 1;
ITEM_TYPE_WARNING = 2;
ITEM_TYPE_ERROR = 3;
ITEM_TYPE_FATAL = 4;

PixConsumer pixQuery = new PixConsumer(javaStream, cpStream);
pixQuery.setMaxVerifyEvent(Validator.ITEM_TYPE_INFORMATION); //block warnings
String response = pixQuery.sendHL7(message, true, auditUser);
```

5. Audit user is now entered at the transaction level rather than the client level.

   Before.
   ```
   PixConsumer pixQuery = new PixConsumer(msAccessFile, cpaStream);
   pixQuery.setAuditUser(auditUser);
   ```

   Now,
   ```
   PixConsumer pixQuery = new PixConsumer(msAccessFile, cpaStream);
   PixConsumerQuery msg = createMessage(pixQuery);
   PixConsumerResponse response =
   pixQuery.sendQuery(msg, true, auditUser);
   ```

   The files in src_tests now use a TestConfiguration.java file for extracting the various file locations and MLLP connection parameters. Update this file with your settings before running the sample code.

6. The MLLP constructor now assumes the start and end characters for MLLP. The buffer size is also automatically set to 4096, but configurable using the setBufferSize() method in TCPPort.java.

   Before,
   ```
   MLLPDestination mllp =
   new MLLPDestination(host, port, startChar, endChar, buffersize);
   ```

   Now,
   ```
   MLLPDestination mllp =
   new MLLPDestination(TConfig.MLLP_HOST, TConfig.MLLP_PORT);
   ```

7. The HL7v2 definition file is no longer fixed to a folder within the Client plugin. It is now a parameter of the Client constructor.
8. Javadoc and README files are now released with the plugin.
9. Secure MLLP connections are now available and sample code showing how to construct a secure connection to the IBM Dallas Server has been included.

   Secure Connection:
   ```
   SecureTCPPort tcpPort = new SecureTCPPort();
   tcpPort.setTcpHost(TestConfiguration.MLLP_HOST);
   tcpPort.setTcpPort(TestConfiguration.MLLP_SECUREPORT);
   tcpPort.setKeyStoreName(TestConfiguration.MLLP_KEYSTORE_NAME);
   tcpPort.setKeyStorePassword(TestConfiguration.MLLP_KEYSTORE_PASSWORD);
   tcpPort.setTrustStoreName(TestConfiguration.MLLP_TRUSTSTORE_NAME);
   tcpPort.setTrustStorePassword(TestConfiguration.MLLP_TRUSTSTORE_PASSWORD);
   tcpPort.setClientAuthNeeded(TestConfiguration.MLLP_NEEDS_CLIENT_AUTH);
   tcpPort.setProtocol(TestConfiguration.MLLP_PROTOCOL);
   tcpPort.setSslProviderName(TestConfiguration.MLLP_SSLPROVIDER_NAME);
   tcpPort.setSslProviderClass(TestConfiguration.MLLP_SSLPROVIDER_CLASS);
   MLLPDestination mllp = new MLLPDestination(tcpPort);
   ```

10. The PixConsumer createQuery method has been expanded to support immediate inclusion not just the patient ID but also the related assigningAuthority fields.

```
PixConsumerQuery msg = pixQuery.createQuery(patientId,
assigningAuthorityName, assigningAuthorityUniversalId,
assigningAuthorityUniversalIdType);
```

11. The ability to iterate through the Patient Identifier (PID-3) when multiple IDs are returned from the server has been added. There is a new method to return the count and the individual values are accessible via a 0 based index.

```
for (int i=0; i < response.getPatientCount(); i++) {
    int cnt = i+1;
    logger.debug(cnt + " - Patient ID: " +
          response.getPatientIdentifierIdNumber(i));
    logger.debug(cnt + " - Patient ID assigningAuthority: " +
          response.getPatientIdentifierAssigningAuthorityName(i));
    logger.debug(cnt + " - Patient ID universalID: " +
          response.getPatientIdentifierAssigningAuthorityUniversalId(i));
    logger.debug(cnt + " - Patient ID universalIDType: " +
          response.getPatientIdentifierAssigningAuthorityUniversalIdType(i));
}
```

12. A new method changeDefaultQueryName has been added to the PixConsumerQuery. This supports modification to the QPD-1 Query Name field.
13. Modified message header methods to support more component parameters. Now, the universalId and universalIdType are accepted.

    Before,
    `public void changeDefaultSendingApplication(String sendingApplication)`
    After,
    `public void changeDefaultSendingApplication(String namespaceId, String universalId, String universalIdType)`