EMC® Documentum® XDS Registry

Version 1.9

Installation Guide

EMC Corporation
Corporate Headquarters
Hopkinton, MA 01748-9103
1-508-435-1000
www.EMC.com

Table of Contents

Revision History

Revision Date	Description
February 2016	Initial publication.

Chapter 1

About XDS Registry Server

This chapter provides an overview of Cross-enterprise Document Sharing (XDS) Registry Server, its architecture, workflow, and endpoints.

Overview

The XDS Registry Server provides a central directory for a healthcare community that contains information about patient healthcare records. The registry does not store the records themselves, but instead contains information about each record, such as the patient ID, the document type, the physician name, the procedure involved, and the location of the record. Healthcare providers query the registry to obtain a list of patient healthcare records and their locations.

Architecture

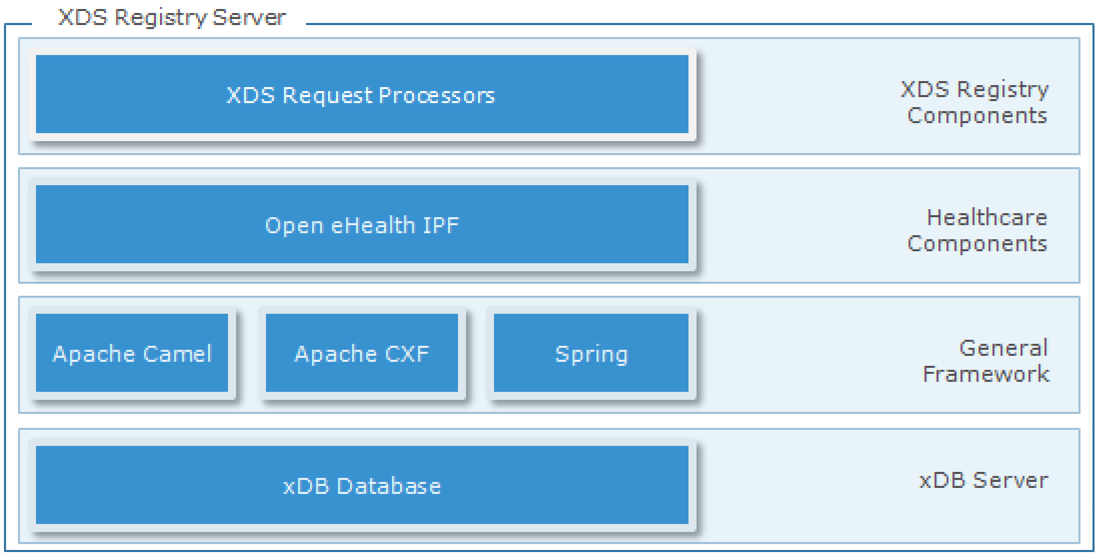
The HIP XDS Registry consists of the following components:

- HIP XDS Registry Server
- xDB

The XDS Registry Server is a J2EE web application built on the Apache Camel open-source routing and mediation framework. It queries and retrieves XML metadata from the Documentum xDB healthcare database.

The Documentum xDB healthcare database stores registry data in the `/registry/objects` library, and the XDS Registry Server configuration file in the `/registry` library. The server environment properties for the XDS Registry Server are stored in `registry.properties` file, which resides on the server in the HIP configuration directory.

The following figure shows the XDS Registry Server architecture:



The top layer contains XDS Registry Request processors that handle and process the Registry SOAP request messages. These processors handle messages for registering patient records and for retrieving the patient record metadata from the XDB database.

The second layer contains healthcare components from the Open eHealth Integration Platform (IPF). These are Apache Camel specific components for the XDS Registry that include request validators, SOAP components, and message convertors.

The third layer contains the general application framework which includes Apache Camel, Apache CXF and Spring. Apache CXF is an Apache Camel component that handles message requests formats from a wide variety of formats. This layer also uses the Spring ApplicationContext to provide configuration properties to the application.

The fourth layer consists of the Documentum xDB healthcare database on a Documentum xDB server.

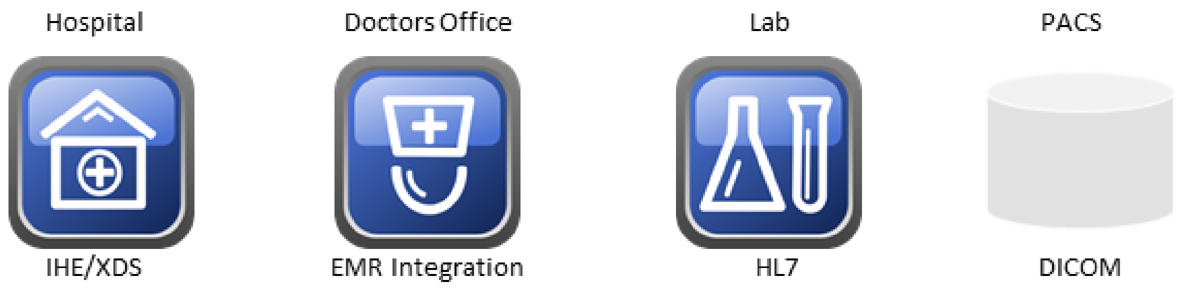
Workflow

The XDS Registry Server is a component of the HIP XDS Archive. It can be paired with the HIP XDS Repository Server or any Integrating the Healthcare Enterprise (IHE)-enabled XDS Repository. The registry implements the IHE XDS.b Profile registry actor which enables you to share healthcare records with the hospitals and organizations that comprise your healthcare community.

A healthcare community consists of different healthcare consumers and providers that need to access and share a patient's *Healthcare records*. Healthcare records include administrative records (patient information) and patient medical records (X-rays, doctor reports, lab results).

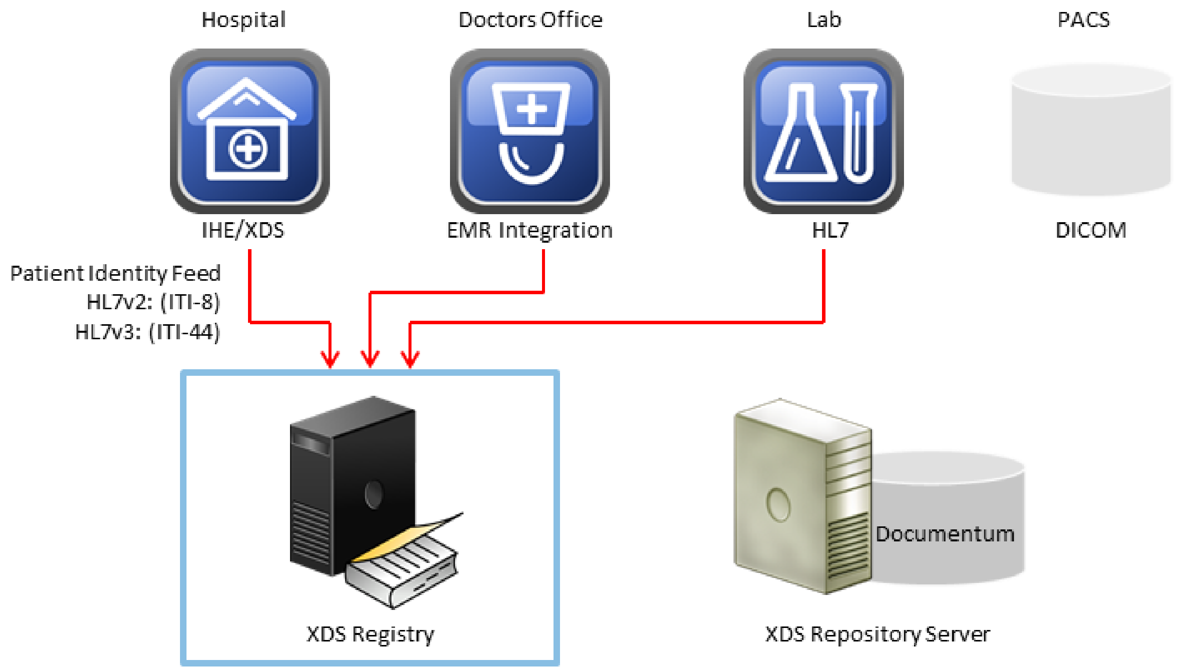
There are many potential consumers and providers in a healthcare community, some common examples are: hospitals, physician's offices, labs, pharmacies, insurance companies, and Picture Archiving and Communications Systems (PACS).

The following figure shows a few examples of consumers and providers in a healthcare community:



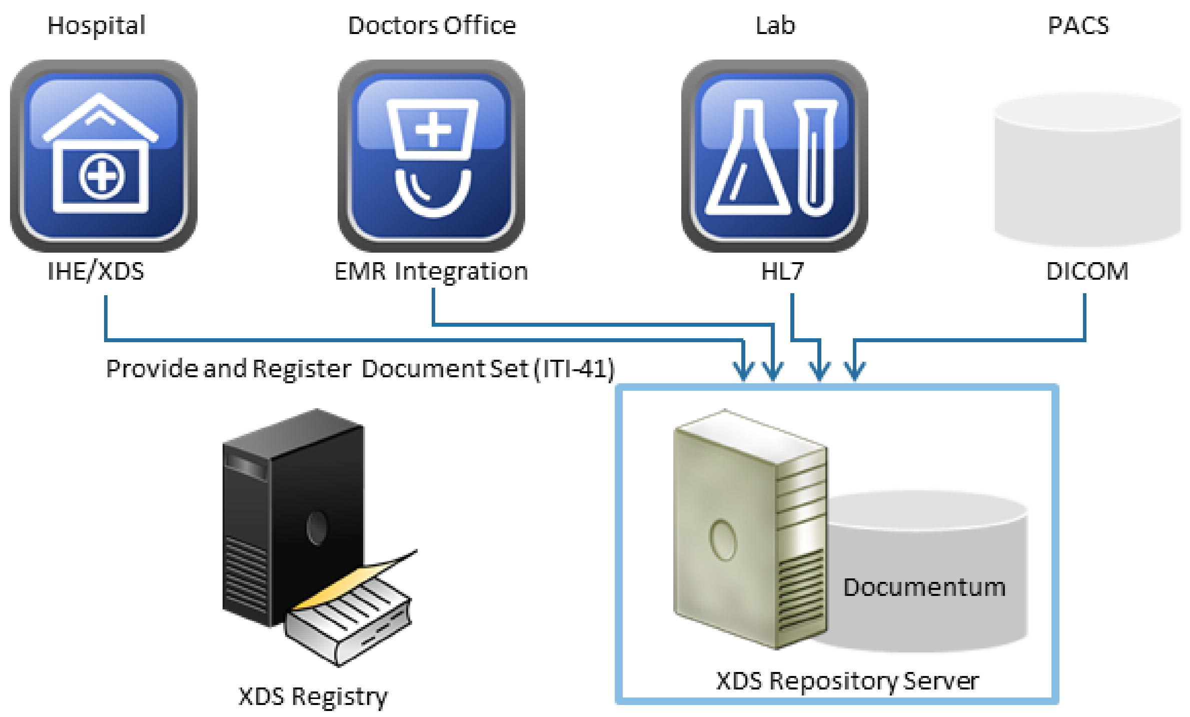
Healthcare providers create patient identities and supply that information to the registry through Patient Identity Feed transactions.

The following figure shows the Patient Identity Feed transactions:



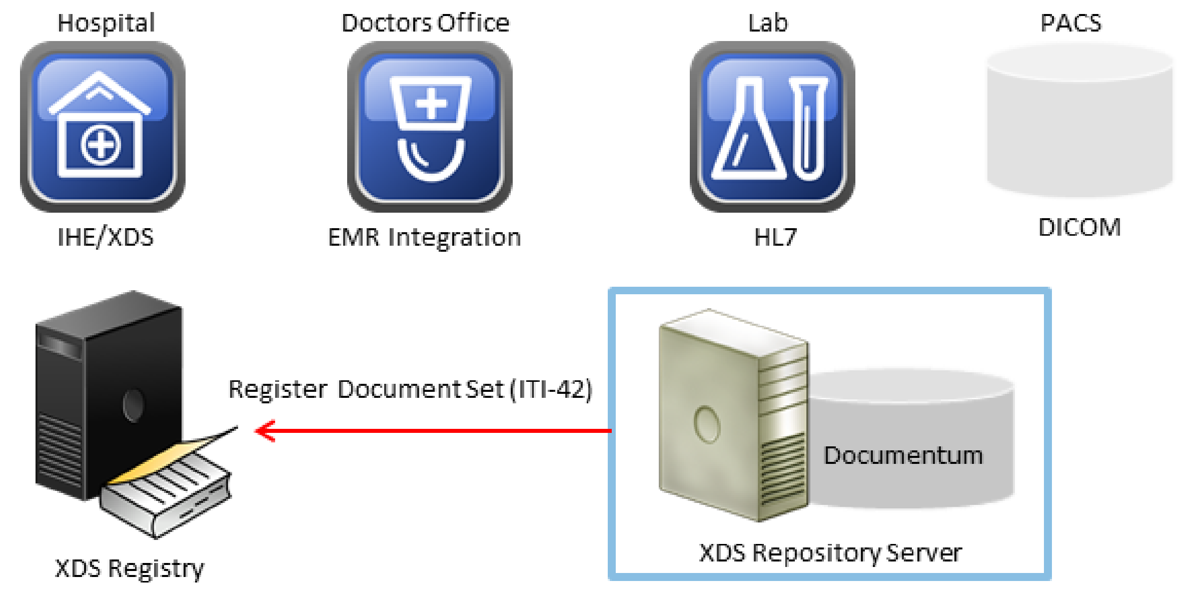
Healthcare providers submit new Healthcare records to an XDS Repository through the XDS Repository Server with the Provide and Register Document Set transaction. The XDS Repository Server stores the submitted Healthcare records in the XDS Repository.

The following figure shows the Provide and Register Document Set transaction:



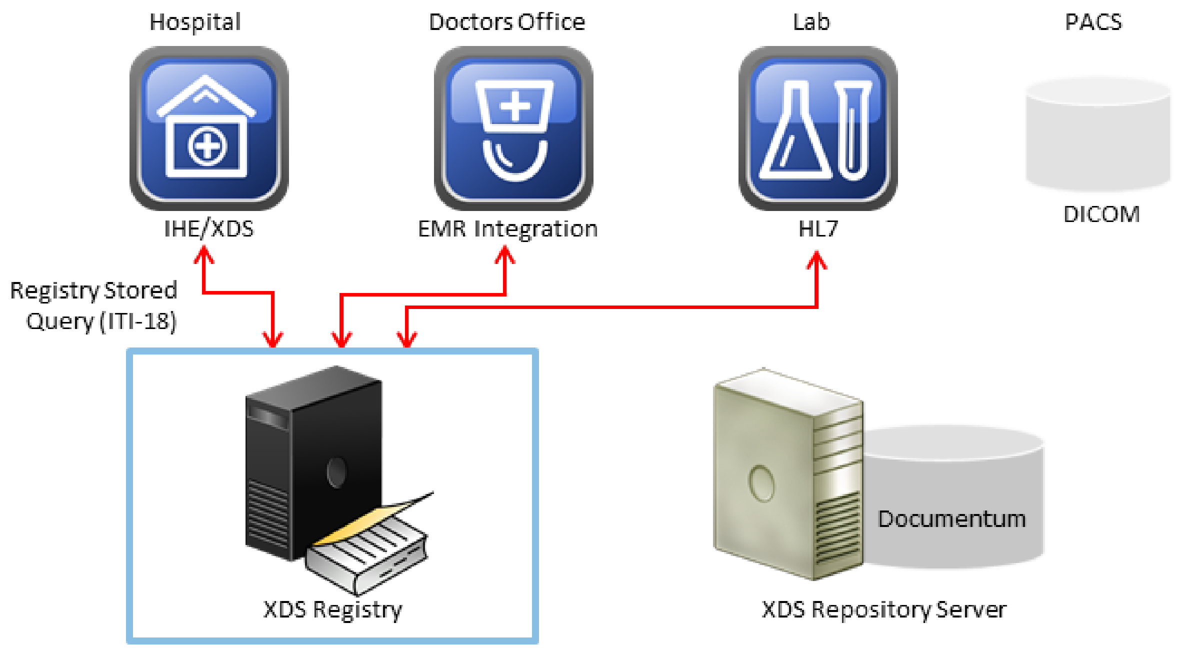
The XDS Repository Server then automatically registers the relevant document metadata with the XDS Registry with the Register Document Set transaction. Registering a healthcare record with the XDS Registry enables other healthcare providers to find the record.

The following figure shows the registration of document metadata.



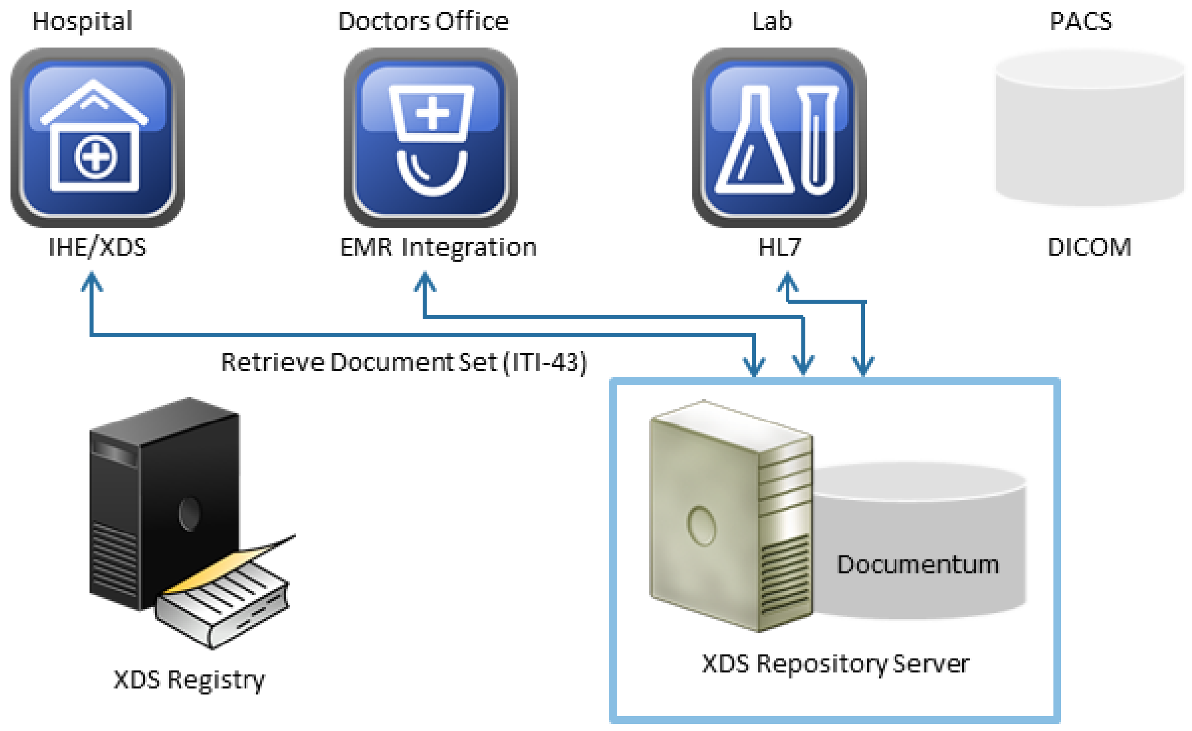
When healthcare providers need to obtain a patient's healthcare records, they query the registry with the Registry Stored Query request. The registry provides the location of the healthcare record. Healthcare providers operate as a Document Consumer when querying the XDS Registry.

The following figure shows the process of querying the registry:



Document Consumers can then retrieve the content from the Repository using the Retrieve Document Set transaction.

The following figure shows the Retrieve Document Set transaction:



Endpoints

The following table lists the endpoints for the XDS Registry Server:

IHE Transaction	Description	Endpoint
ITI-8	Patient Identity Feed HL7 v2.3.1	MLLP://*<host>*:*<port>*
ITI-18	Registry Stored Query Registry Stored Query–Asynchronous Registry Stored Query–Trusted host access-only	HTTPS://*<host>*:*<port>*/registry/services/xds-iti18 HTTPS://*<host>*:*<port>*/registry/services/xds-iti18as HTTPS://*<host>*:*<port>*/registry/services/trustedhosts/xds-iti18
ITI-42	Register Document Register Document–Asynchronous	HTTPS://*<host>*:*<port>*/registry/services/xds-iti42 HTTPS://*<host>*:*<port>*/registry/services/xds-iti42as
ITI-44	Patient Identity Feed HL7 v3	HTTPS://*<host>*:*<port>*/registry/services/xds-iti44
ITI-51	Multi-Patient Registry Stored Query Multi-Patient Registry Stored Query–Asynchronous	HTTPS://*<host>*:*<port>*/registry/services/xds-iti51 HTTPS://*<host>*:*<port>*/registry/services/xds-iti51as
ITI-61	Register On-Demand Document Register On-Demand Document–Asynchronous	HTTPS://*<host>*:*<port>*/registry/services/xds-iti61 HTTPS://*<host>*:*<port>*/registry/services/xds-iti61as
ITI-57	Update Document Update Document–Asynchronous	HTTPS://*<host>*:*<port>*/registry/services/xds-iti57 HTTPS://*<host>*:*<port>*/registry/services/xds-iti57as
ITI-62	Delete Document Delete Document–Asynchronous	HTTPS://*<host>*:*<port>*/registry/services/xds-iti62 HTTPS://*<host>*:*<port>*/registry/services/xds-iti62as

For client applications such as XDS consumers that support only single endpoint for XDS operations, HIP provides an additional single SOAP endpoint for IHE transactions. This additional endpoint dispatches request to original endpoint (as defined in SOAPEndpointRouteBuilder) based on the SOAP operation name in the request.

The following table lists the single endpoints (applicable for both http and https) provided by the XDS Registry Server for ITI-42 and ITI-18 transactions:

Type of Operation	**Unified Endpoint URL**	**Supported Transactions**
Synchronous operations	http://<*host:port*>/registry/services/xds-reg-svc	• **ITI-42**: http://<*host:port*>/registry/services/xds-iti42 • **ITI-18**: http://<*host:port*>/registry/services/xds-iti18
Asynchronous operations	http://<*host:port*>/registry/services/xds-reg-svc-as	• **ITI-42as**: http://<*host:port*>/registry/services/xds-iti42as • **ITI-18as**: http://<*host:port*>/registry/services/xds-iti18as

Chapter 2

Features

This chapter describes the features of XDS Registry Server.

ITI-8 Patient Identity Notifications

ITI-8 Patient Identity Feed is the transaction in which a **Patient Identity Source** sends a message to the **Patient Identity Cross Reference Manager** and **Document Registry** whenever a patient is admitted, pre-admitted, registered, or when the patient demographic data is modified.

The Patient Identity Feed transactions are done by the HL7 ADT messages.

The following are the HL7 Versions supported for inbound ADT messages:

```
ADT^A34(V23),ADT^A40(V231,V24,V25,V251)
```

For inbound messages, XDS Registry supports only the Merge Patient Identity notification (`ADT^A40`) and simple MDM messages for new document notifications.

The Registry Server listens to the ITI-8 Patient Identity feeds through two Minimal Lower Layer Protocol (MLLP) ports, one secure and the other, non-secure. The HTTPS properties must be set if you want to enable secure HTTPS MLLP port.

You must edit the `registry.properties` file to configure these ports.

New Patient Identity Notification

The following are the **New Patient Identity Notifications** that a Patient Identity Source triggers whenever an Admit/Register or an Update event occurs:

- **A01**: Admission of an inpatient into a facility
- **A04**: Registration of an outpatient for a visit of the facility
- **A05**: Pre-admission of an inpatient (that is, registration of patient information ahead of actual admission)
- **A08**: Update to an existing patient record

Merge Patient Identities Notification

The Patient Identity Source generates the **Merge Patient–Internal ID** notification denoted by `(ADT^A40)` whenever two patient records are merged. Two records are merged when they reference the same patient in the Patient Identifier Domain. The Patient Identity Source sends the generated Merge Patient - Internal (A40) message to Patient Identifier Cross-reference Manager and Document Registry.

Message Segments:

An ADT Patient Merge (`ADT^A40`) message consists of the following segments:

- **MSH**—Message Header
- **EVN**—Event Type
- **PID**—Patient Identity Information
- **MRG**—Merge Patient Information
- **PV1** (optional)—Patient Visit Information

For Patient Identity Merge, the main segments, fields, and components that are of interest are the following:

PID: Patient Information

  PID-3: Patient Identifier List–Internal

    PID-3.1: Patient Identifier Value

MRG: Merge Patient Information

  MRG-1: Prior Patient Identifier List–Internal

    MRG-1.1: Prior Patient Identifier Value

The Merge segment (MRG) contains information about the duplicate (secondary) patient identifier that needs to be dereferenced. MRG-1 indicates the subsumed patient identifier; the patient identifier whose use is being ended. The PID-3 indicates the surviving patient identifier; the patient identifier whose use continues.

Merging Process:

The Patient Identity Source merges the secondary patient identifier with the primary patient identifier, by populating the values of the following components of PID segment:

- PID-3.1: Patient Identifier Value
- PID-3.4: Assigning Authority

in the corresponding components of the MRG segment:

- MRG-1.1: Prior Patient Identifier Value
- MRG-1.4: Assigning Authority

That is, the same value of Patient Identifier component of PID-3.1 field is populated in the Prior Patient Identifier Value component of MRG-1.1 field. Similarly, the assigning authority of PID-3.4 component is populated in the assigning authority of the MRG-1.1 component.

After a merge, the patient identifier PID-3 represents all records formerly represented by either MRG-1 or PID-3. All other fields may be ignored. The secondary patient identifier should no longer be used to reference the patient. However, HL7 does not mandate that the secondary identifier be deleted.

After merging, the Patient Identity source sends an `ADT^A40` Merge Patient message to the following:

- Patient Identifier Cross-reference Manager
- Document Registry

Action taken by Patient Identifier Cross-reference Manager post merging: When the Patient Identifier Cross-reference Manager receives the `ADT^A40` message type, it replaces all references to the patient ID that was earlier existing in MRG-1.1 field with the patient ID in the PID-3.1 field. After the references are updated, the newly updated identifiers are made available to the PIX queries and the Patient Identifier Cross-reference Manager sends out a notification to the Patient Identifier Cross-reference Consumers using the PIX `Update Notification transaction (ITI-46).`

Action taken by the Document Registry post merging: When the Document Registry receives the `ADT^A40` message type, it merges the secondary patient identity (MRG-1.1) into the primary patient identity (PID-3.1) in the registry.

After merging,

- All document submission sets including the documents and folders beneath them associated with the secondary patient identity before merge points to the primary patient identity.
- The secondary patient identity is no longer referenced by the Registry for any future transactions.
- Any Register Document Set-b transaction referencing a subsumed identifier is rejected with an XdsUnknownPatientId error.
- Any Registry Stored Query transaction referencing a subsumed identifier returns no content.
- Registry Stored Query transactions referencing a surviving identifier successfully match the entire recorded merge chain and return appropriate metadata.

Note: The Document Registry performs merge only if `ADT^A40` message does not meet any of the following conditions:

- The subsumed patient identifier is not issued by the correct Assigning Authority according to the Affinity Domain configuration.
- The surviving patient identifier is not issued by the correct Assigning Authority according to the Affinity Domain configuration.
- The subsumed and surviving patient identifiers are the same.
- The subsumed patient identifier was subsumed by an earlier message.
- The surviving patient identifier was subsumed by and earlier message.
- Both the subsumed and surviving patient identifier must convey a currently active patient identifier known to the Registry Actor.

The changes resulting due to an A40 merge are irreversible.

XDS Registry Transactions

XDS Registry supports the following transactions:

- ITI-18 Registry Stored Query
- ITI-42 Register Document Set
- ITI-51 Multi-Patient Query
- ITI-57 Update Document Set
- ITI-61 Register On-Demand Document Entry
- ITI-62 Delete Document Set

Trusted Host ITI-18 Endpoint

The default ITI-18 endpoint provides Patient Privacy enforcement support to the EMR applications. The EMR applications can enable or disable PPIC. If enabled, only authorized users can access the metadata of patient records. However, to access the metadata, the EMR applications must provide sufficient authorization details in the request.

The applications such as Connector for Epic (C4E), Clinical Archiving, which do not support PPIC feature, but still need to access XDS Registry to register or deregister documents, cannot use the default ITI-18 endpoint because the default ITI-18 endpoint needs sufficient authorization details to process the request.

Therefore, to provide ITI-18-transaction access to such applications, you can enable an additional ITI-18 endpoint that:

- does not require authorization details to be sent over the request
- permits only a set of configured list of trusted hosts
- permits only secured transport (HTTPS)

This additional endpoint is enabled by setting the following property to **`true`** in the `registry.properties` file:

```
registry.trusted.hosts.enabled=true
```

By default, this property is set to **`false`**.

If this trusted host access endpoint is enabled, the applications that do not support PPIC feature can execute ITI-18 transactions without having to use the default endpoint that the other EMR applications use.

You can access this endpoint by using the following URI:

```
https://<host:port>/registry/services/trustedhosts/xds-iti18
```

The default trusted host is localhost. You can configure a list of trusted hosts in the `registry-context-extension.xml` file.

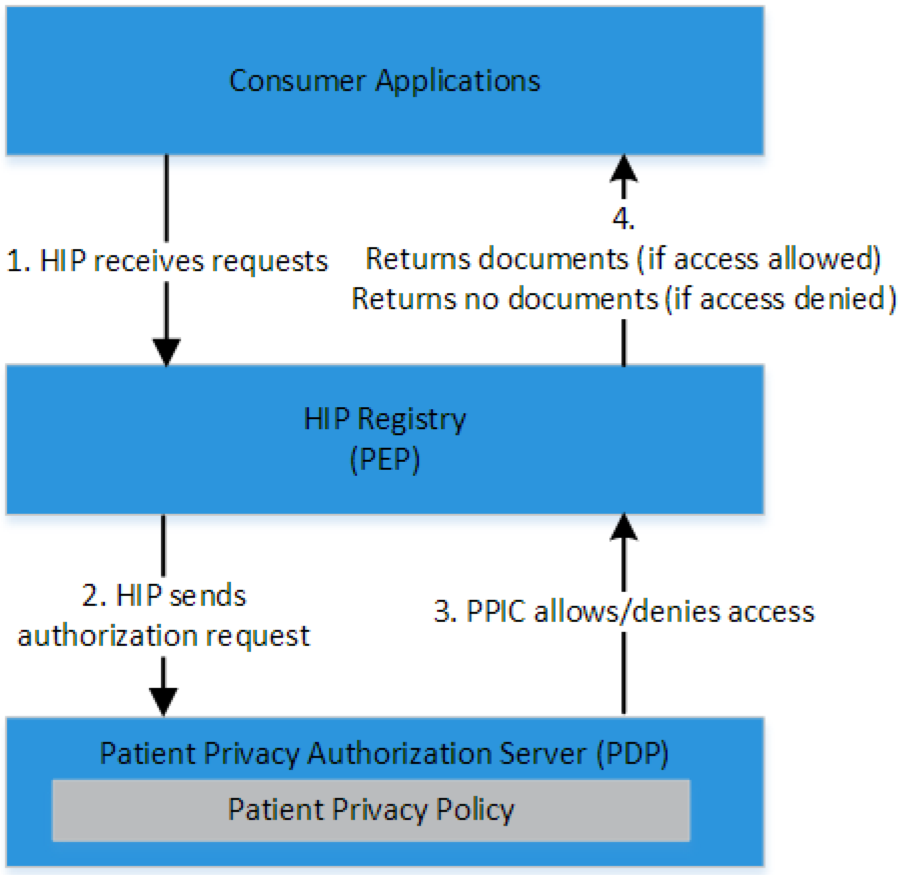
Configuring the Trusted Hosts, page 51 provides the steps to configure the trusted hosts.

Patient Privacy Policy Enforcement

Users require authorization to access the healthcare metadata of a patient and documents in an XDS Affinity Domain. Patient Privacy Policy (PPP) enforcement enables the XDS Registry Server to perform authorization for specific requests by user.

The Healthcare Registry Services act as a Policy Enforcement Point (PEP) in conjunction with a PDP Server (for example, Patient Privacy and Informed Consent (PPIC) Authorization Server) and controls the authorization. When the Registry is queried to fetch the documents, the authorization server checks if the user has permission to view all the documents that the query returns. The user is permitted to view only those documents for which permission is granted by the patient privacy policy.

The following figure shows the PPIC authorization flow:


Consumer Applications
1. HIP receives requests
4.
Returns documents (if access allowed)
Returns no documents (if access denied)
HIP Registry
(PEP)
2. HIP sends
authorization request
3. PPIC allows/denies access
Patient Privacy Authorization Server (PDP)
Patient Privacy Policy

The functions of PPIC Authorization Server and Registry Services are as follows:

- **PPIC Authorization Server**:

  PPIC Authorization Server maintains the Patient Privacy policies.

- **XDS Registry Services**:
  - XDS Registry Services receive `Registry Stored Query Request` (ITI-18) and perform records lookup in the Registry database.
  - If ITI-18 Request is for **Object Reference**, XDS Registry does not perform policy enforcement and returns all object references to the Document Consumer.

- If ITI-18 Request is for **Leaf Class**, the PEP component within the XDS Registry Services prepares Policy Decision Request containing Document Entry metadata such as entryUUID, healthcare facility code, and so on. The PEP component uses `hip-ppic-mapping.properties` to include all metadata attributes in the request. In an XUA-enabled environment, the PEP component retrieves Subject ID and Subject Role information from SAML token and passes them in the PDP Request.
- On receiving the Policy Decision Request, PPIC Server evaluates all policies and performs lookup for additional metadata that is required for policy evaluation, using the registered Policy Information Point (PIP) component.
- Based on the response received from PPIC Server, PEP component within the XDS Registry Services filters the documents and returns the metadata of only the authorized documents to the Document Consumer.

This feature can be enabled or disabled according to the requirement.

Configuring the PPIC Properties, page 46 provides details about configuring the PPIC properties.

The *EMC Documentum PPIC Installation Guide* provides details about installation and configuration of PPIC Server.

XDS Registry DSUB Notifications

With HIP 1.9 release, you can configure XDS Registry as IHE DSUB Notification Publisher Actor.

XDS Registry publishes Document Metadata Notification (ITI-54) whenever a successful register (ITI-42), update (ITI-57) or delete (ITI-62) transaction occurs. The Registry sends Document Metadata Notification (ITI-54) to the DSUB Notification Broker after each successful transaction. If the DSUB Notification Broker does not acknowledge the receipt of notification, the Registry re-sends the notification based on the number of retries that are configured.

Configuring Registry as DSUB Notification Publisher, page 48 provides more information about configuring HIP XDS Registry as IHE DSUB Notification Publisher.

Customization

The only customization available for the XDS Registry Server is the ability to override the Camel SoapRouteBuilder script. This script allows you to add custom route endpoints to filter or validate incoming requests and outgoing responses. You can do this configuration in the `registry.properties` file.

Business Continuance

The following information on business continuance is applicable only for the XDS Registry Server and not for xDB.

xDB documentation provides more details about business continuance for xDB.

The whitepaper *High Availability Configuration For a Multiple Region EMC Healthcare Integration Portfolio (HIP) Registry and Repository* provides more details on HADR.

Load Balancing and Scalability

The following are the two methods for load balancing and scaling the environment:

- Instantiate multiple instances of the XDS Registry Server and use a Load Balancer to route incoming requests to the XDS Registry cluster.

  In the event of a failure, the Load Balancer routes requests to available XDS Registry Servers to ensure the system remains available.

- Load balance the xDB Server for performance and scalability. The xDB documentation provides more information on load balancing.

Data Backup and Recovery

The XDS Registry Server maintains only initial XDS Registry application-specific configuration information. This information resides in the registry configuration file (`registry-config.xml`) and the server configuration properties files (`registry.properties`). These files are part of the XDS Registry Server and are either backed up with the entire server installation on a VM image or through a different method. There is no Recovery Point Objective for the XDS Registry Server because the server does not store transaction information or Repository content. The Documentum xDB Server stores XDS document metadata and should be backed up separately.

High Availability and Disaster Recovery

The High Availability Disaster Recovery (HADR) is achieved by implementing a backup strategy for each XDS server in your environment, including the xDB Server.

For example:

- **Spare Instance**: Create spare instances of the XDS server that you can manually start when an active XDS server fails. You can create a spare instance through VM images, ESXI servers, or other similar methods.
- **Active-Passive**: Configure a Universal Fail-Over Server (UFO) that is active and able to take over the functionality of the failing server.
- **Active-Active**: Configure multiple XDS Registry Servers to serve a single Documentum xDB Server. If one server fails, the other servers remain available. XDS Registry Servers may reside on different machines and different locations.

The whitepaper *High Availability Configuration For a Multiple Region EMC Healthcare Integration Portfolio (HIP) Registry and Repository* provides more details on HADR.

Usage Reporting

Usage Reporting is implemented to support the subscription pricing model for customers. That is, the customers can be billed based on the usage of the product against the licenses purchased. Usage Reporting provides a monthly report based on the usage of the product by the customer. The usage of Registry is calculated based on the number of unique patients registered in the Registry. A Usage Reporting web application provides the ability to view the monthly reports and to generate adhoc reports.

Standard Usage Reports are scheduled to be automatically generated at the end of each month. However, you can generate adhoc reports.

The monthly reports are stored in a library called UsageReports in xDB. However, ad hoc reports are not stored in any specific location; you must download them to your local system.

You can launch the Usage Reporting user interface by using the following URL:

```
localhost:port/registry/reports
```

The default username and password to log in to the web application are configured in the `registry.properties` file. You can modify the login configuration, if required.

Configuring the Usage Report Properties, page 47 provides information about configuring the Usage Reporting properties.

When you log in to Usage Reporting web application, you can see a list of generated reports. You can click a report to view its details.

Each report shows the following details:

- **Unique Patient Count**: Number of unique patients registered in the Registry.
- **Generated on**: Date and time when the report is generated.
- **Generated By**: Name of the user who generates the report.
- **Host**: IP of the host system that generates the reports.
- **Type**: Name of the product for which the reports are generated.
- **Version**: Version of the product for which the reports are generated.

The Usage Reporting user interface also provides options to download the reports as XML, HTML, or PDF files.

The **Generate Report Now** button enables you to create ad hoc reports that show the usage details from the day of product purchase. The names of the reports are the same as their IDs. The report ID consists of the timestamp when the reports are generated. That is, the report ID uses the format `yyyyMMddHHmmss`.

Chapter 3

Before You Install

Before beginning installation, ensure that your system meets the requirements.

The *EMC Documentum XDS Registry Release Notes* provides information on the system requirements for your product. This documentation is available from EMC Online Support.

Chapter 4

Pre-installation Tasks

This chapter describes the steps to set up the Documentum xDB database and install the dependent libraries that you require to successfully install the XDS Registry Server.

Setting up the Documentum xDB Healthcare Database

1. Install Documentum xDB on the system that hosts your Documentum xDB database.

   You may skip this step if you already have Documentum xDB in your environment. The *Documentum xDB Manual* provides the details of installation instructions.

2. Create the Documentum xDB Healthcare database.

   Use the Documentum xDB Administrator tool to create a Healthcare database to hold registry data. Allocate enough resources to the default and temporary segments to match your performance requirements. Record the name of database for use in later configuration steps. This installation guide mentions **`Healthcare`** as a sample database name. The *Documentum xDB Manual* provides database creation instructions.

3. Establish user access for the Documentum xDB Healthcare database.

   Create a non-privileged user account in the Documentum xDB Healthcare database. The XDS Registry Server uses this account to access the Documentum xDB Healthcare database. Record the user name and password for use in later configuration steps. This installation guide mentions **`HealthcareServer`** as a sample user name.

Installing the Dependent Libraries

You must install the following third-party dependent libraries to successfully deploy the Registry Server WAR file:

- xDB
- Camel
- HL7 Application Programming Interface (HAPI)
- Integrating the Healthcare Enterprise (IHE)

Obtaining the Dependent Libraries

To obtain the xDB JAR files:

1. Create a folder in the local path to copy the xDB JAR files.

   For example:

   `C:\jarfiles\xdb`

2. Go to the xDB installation directory.

   For example:

   `C:\Program Files\xDB\`

3. Copy the `lib` folder containing the JAR files from the `<xDB_install_dir>` folder to the `C:\jarfiles\xdb` folder.

   For example:

   `C:\jarfiles\xdb\lib`

The XDS Registry Server must use the xDB JAR files located in the `/lib` and `/lib/core` directories to enable the Registry to communicate with Documentum xDB. You have to manually add the JAR files to the XDS Registry `/lib` directory.

To obtain the Camel JAR files:

1. Create a folder in the local path to copy the Camel JAR files.

   For example:

   `C:\jarfiles\camel`

2. Go to www.camel.apache.org.

3. Download the following files:

   For Windows:

   `apache-camel-2.12.1.zip`

   For Linux:

   `apache-camel-2.12.1.tar.gz`

4. Extract the ZIP file to a local path.

   For example,

   `C:\apache-camel-2.12.1`

5. Go to the `<local path>\apache-camel-2.12.1` folder.

6. Copy the `lib` folder containing the JAR files to the `C:\jarfiles\camel` folder.

   For example:

   `C:\jarfiles\camel\lib`

To obtain the HAPI JAR files:

1. Create a folder in the local path to copy the HAPI JAR files.

   For example:

   `C:\jarfiles\hapi`

2. Go to www.sourceforge.net.
3. Download the `hapi-dist-2.0-all.zip` file.
4. Extract the ZIP file to a local path.

   For example,

   `C:\hapi-dist-2.0-all`
5. Go to the `<local path>\hapi-dist-2.0-all` folder.
6. Copy the `lib` folder containing the JAR files to the `C:\jarfiles\hapi` folder.

   For example:

   `C:\jarfiles\hapi\lib`

Due to licensing restrictions, HIP products do not deliver the required HAPI library JAR files used when parsing HL7 feeds.

To obtain the IHE JAR files:

1. Create a folder in the local path to copy the IHE JAR files.

   For example:

   `C:\jarfiles\ihe`
2. Go to www.projects.openhealthtools.org.
3. Download the `org.openhealthtools.ihe_2.0.0.zip` file.
4. Extract the ZIP file to a local path.

   For example:

   `C:\openhealthtools\`
5. Go to the `C:\openhealthtools` folder.
6. Copy the following JAR files to the `C:\jarfiles\ihe` folder:
   - `org.openhealthtools.ihe.atna.context_2.0.0.jar`
   - `org.openhealthtools.ihe.atna.nodeauth_2.0.0.jar`
   - `org.openhealthtools.ihe.utils_2.0.0.jar`
7. Go to www.repo.openehealth.org.
8. Copy the following JAR file to the `C:\jarfiles\ihe` folder:

   `org.openhealthtools.ihe.atna.auditor-2.0.0-p4.jar`

Bundling the Dependent Libraries

1. Download the `hip-registry-1.9.0.zip` file from the EMC Software Download Center.
2. Extract the `hip-registry-1.9.0.zip` file to a local folder.

   For example:

   `C:\hip-registry-1.9`

   You can find the `build.xml` file in the `C:\hip-registry-1.9` folder.
3. Go to the command prompt and navigate to the directory where `build.xml` is located.

For example:

```
C:\hip-registry-1.9
```

4. Run `build.xml` using the following command:

   ```
   ant -f build.xml
   ```

   After you run the `ant -f build.xml` command, you obtain the `install` folder as follows:

   ```
   C:\hip-registry-1.9\install
   ```

   You can find the `hip-registry-1.9.0.war` file in the `install` folder.

5. When the script prompts you to enter the Camel home directory, type the complete path of Camel home directory.

   For example:

   ```
   C:\jarfiles\camel
   ```

6. When the script prompts you to enter the HAPI home directory, type the complete path of HAPI home directory.

   For example:

   ```
   C:\jarfiles\hapi
   ```

7. When the script prompts you to enter the xDB home directory, type the complete path of xDB home directory.

   For example:

   ```
   C:\jarfiles\xdb
   ```

   You must enter the xDB home directory where xDB is installed. If xDB is installed in a remote system, copy all JAR files from the xDB lib folder to a local folder and rename the local folder to xDB home directory.

8. When the script prompts you to enter the IHE home directory, type the complete path of IHE home directory.

   For example:

   ```
   C:\jarfiles\ihe
   ```

Chapter 5

Configuring HIP XDS Registry

This chapter describes the steps to configure HIP XDS Registry.

Configuring the Server Directory

Configuring the server directory consists of the following tasks:

- Creating the HIP configuration directory
- Deploying the properties files in the HIP configuration directory
- Securing the Registry properties file

Creating the HIP Configuration Directory

The HIP servers maintain configuration information in a directory called `hip` that is located outside the WAR file. This design enables users to easily upgrade to newer versions of the software by replacing the server WAR file.

By default, HIP uses the following directory:

```
<user.home>/.hip
```

where <*user.home*> is the home directory of the user who implements the XDS Registry Server.

For example:

```
C:\Users\username\
```

If this directory does not already exist, create a folder named **`.hip`** in your `user.home` directory by typing:

```
.hip.
```

Ensure that you place a dot at the end of the name. The Windows system removes the trailing dot.

The resulting folder must have the following name:

```
C:\Users\username\.hip
```

The **`com.emc.healthcare.com`** Java system property defines the location of the configuration directory. If you want to override the default location of the HIP server configuration information, override the **`com.emc.healthcare.com`** system property when you start your J2EE Web Application container.

Use the following syntax:

```
-Dcom.emc.healthcare.home=<hip_config_directory>
```

Deploying the Property Files in the HIP Configuration Directory

1. Go to the `install` folder that you obtained when you ran the build command described in Step 4.
   For example:
   ```
   C:\hip-registry-1.9\install
   ```
2. Extract the `hip-registry-1.9.0.war` file.
3. Go to the `\hip-registry-1.9.0\config\` folder.
4. Copy the `registry` folder to `C:\Users\username\.hip\`.
   The resulting path should be `<hip_config_dir>/registry`.
5. Ensure that the following files are present in the `C:\Users\username\.hip\registry` folder.
   - `hip-ppic-mapping.properties`
   - `registry.properties`
   - `registry-config.xml`
   - `registry-context-extension.xml`
   - `serviceKeystore.properties`
   - `ws-policy.xml`

   You must configure the property files according to your environment, for a successful installation.

Securing the Registry Properties File

The `registry.properties` file contains access information for the XDS Registry.

Secure the file as follows:

- Restrict access to the system where the XDS Registry Server and the `registry.properties` file reside.
- Restrict access to the `registry.properties` file by providing the read/write access only to the HIP Administrator. See www.wiki.apache.org/tomcat/FAQ/Password.
- Provide an encrypted Documentum user password. You can create an encrypted password during Documentum installation or through the **encryptPassword** utility.

  The *EMC Documentum Content Server Administration and Configuration Guide* provides the details on password encryption.

Enabling Remote xDB Instance Support

1. Open the `registry.properties` file.
2. Set the value of **xdb.readNode.bootstrapFileName** as follows:

   ```
   xdb.readNode.bootstrapFileName=xhive://<host-name>:1235>
   ```
   where `<host-name>` is the IP address of the remote xDB instance.
3. Edit the `xdb.properties` file located in `C:\Program Files\xDB\conf` as follows to ensure that the remote xDB instance accepts connections from other machines.

   ```
   # xDB server listen address, '*' to accept all connections,
   # 'localhost' to only accept local connections
   XHIVE_SERVER_ADDRESS=*
   # xDB webserver listen address, '*' to accept all connections,
   # 'localhost' to only accept local connections
   XHIVE_WEBSERVER_ADDRESS=*
   ```
4. Save and close the file.
5. Restart xDB and Registry.

Configuring the Registry Properties File

The `registry.properties` file contains user-definable properties that provide the XDS Registry Server with information about connecting to other systems.

During startup, the XDS Registry Server first evaluates the `registry.properties` file, and then evaluates the **System Properties**. Properties are set in the order they are encountered. If you set a property in multiple locations, the final occurrence of the property takes precedence.

The registry.properties, page 75 in Appendix A provides a sample content for the `registry.properties` file. You can refer this file for configuration examples.

To configure the `registry.properties` file, open the `<hip_config_dir>/registry/registry.properties` file, and configure the properties as described in the following topics.

Configuring the Registry Property

Property	Description
description	A description for the XDS Registry Server. This description is used only for display purposes. This property is optional and has no default value.

Configuring the Documentum xDB Properties

The xDB properties enable the Registry to connect to the Documentum xDB Healthcare database. These properties also include the HADR properties that define the primary read/write and backup read and write nodes.

The following xDB properties are mandatory properties. If you fail to configure these properties, the XDS Registry Server fails to start.

Property	Description
xdb.libraryPath	The location in the Documentum xDB database where the XDS Registry Server stores the data. This property is pre-configured with the following value, but you can change this value to specify a different location.
xdb.cachePages	The number of cache pages for the page server defined in `xdb.bootstrapFileName`.
xdb.maximumPoolSize	The maximum pool size for the page server defined in `xdb.bootstrapFileName`.

Configuring the HADR Properties

The HADR properties define the **primary read/write** and **backup read/write** nodes.

HADR properties are defined to avoid the exceptions occurring for both read and write transactions when only one xDB is available in XDS Registry for both read and write, and that xDB becomes unavailable. To avoid a transaction failure, the xDB database is separated into two—one for read and another for write.

The xDB master is set up as the write node and xDB replica is set up as the read node. In this design, the read operation continues to operate even when the Master xDB (write node) is down. Similarly, the write operation continues when the read node is down. Besides this, an additional xDB server with separate nodes for read and write is setup to act as backup node when the primary nodes are unavailable.

The primary read node parameters are mandatory where as the primary write node parameters are optional. Each takes a single parameter. The backup read and write node parameters are comma separated list that can include multiple backup server nodes. The backup read and write node parameters are optional.

Primary read node parameters:

Property	Description
xdb.readNode.bootstrapFileName	The connection to a dedicated page server that runs behind the specified TCP/IP port. A bootstrap specifies a connection to a federation. This property is mandatory.

Property	Description
xdb.readNode.databaseName	The name of the Documentum xDB database. You created and configured this database in Setting up the Documentum xDB Healthcare Database, page 27. This property is mandatory and has no default value.
xdb.readNode.userName	The username that the XDS Registry Server uses to access the Documentum xDB database. You created and configured this user account in Setting up the Documentum xDB Healthcare Database, page 27. This property is mandatory and has no default value.

xdb.readNode.password	The password of the user that the XDS Registry Server uses to access the Documentum xDB.
	
	This property is mandatory and has no default value.
	
	Note: Perform the following steps to encrypt the password:
	1. From the command prompt, execute the `HipEncryptPassword.bat` command as follows:
	`C:\EncryptPassword>HipEncryptPassword .bat`
	2. When the system prompts to enter the password, type the password as follows:
	`Enter clear text password>Password`
	The encrypted password is generated as follows:
	`HIP_ENCR_PASS=JxwGkf59eneCKgVhZljyEA==`
	After generating the encrypted password, the `hip.keystore` file is generated in `C:\EncryptPassword\` folder.
	3. Rename the `hip.keystore` file to a valid name, for example, `hip_readNode.keystore`.
	4. Copy the `hip_readNode.keystore` file from the `C:\EncryptPassword\` folder to `{com.emc.healthcare.home}/registry/`.
	5. Set the `xdb.readNode.keystore` property as follows:
	`xdb.readNode.keystore=${com.emc .healthcare.home}/registry/hip_readNode .keystore`

Property	Description
	After encryption, in the `registry.properties` file, the `xdb.readNode.password` property appears as follows: `xdb.readNode.password=HIP_ENCR_PASS =JxwGkf59eneCKgVhZljyEA==`
xdb.readNode.keystore	The path to the Read node keystore file. This property is mandatory if the HIP encrypted password is configured.

Primary write node parameters (for single node deployment, set these four values to blank):

Property	Description
xdb.writeNode.bootstrapFileName	The bootstrap filename for the Write node.
xdb.writeNode.databaseName	The database name for the Write node.
xdb.writeNode.userName	The user name for the Write node.

xdb.writeNode.password	The password for the Write node.
	Note: Perform the following steps to encrypt the password:
	1. From the command prompt, execute the `HipEncryptPassword.bat` command as follows:
	`C:\EncryptPassword>HipEncryptPassword.bat`
	2. When the system prompts to enter the password, type the password as follows:
	`Enter clear text password`>Password
	After password encryption, the `hip.keystore` file is generated in the `C:\EncryptPassword\` folder.
	3. Rename the `hip.keystore` file to `hip_writeNode .keystore` (or any valid file name).
	4. Copy the `hip_writeNode.keystore` file from `C:\EncryptPassword\` to `{com.emc.healthcare.home}/registry/`.
	5. Set the `xdb.writeNode.keystore` property as follows:
	`xdb.writeNode.keystore=${com.emc .healthcare.home}/registry/hip_writeNode .keystore`

Property	Description
	After encryption, in the `registry.properties` file, the `xdb.writeNode.password` property appears as follows: `xdb.writeNode.password=HIP_ENCR_PASS =JxwGkf59eneCKgVhZljyEA==`
xdb.writeNode.keystore	The path to the Write node keystore file. This property is mandatory if the HIP encrypted password is configured.

Backup read node parameters (optional):

xdb.backup.readNode .bootstrapFileName	The bootstrap filename for the backup Read node.
xdb.backup.readNode.databaseName	The database name for the backup Read node.
xdb.backup.readNode.userName	The user name for the backup Read node.

xdb.backup.readNode.password	The password for the backup Read node.
	For example:
	`xdb.backup.readNode.password=password`
	Note: Perform the following steps to encrypt the password:
	1. From the command prompt, execute the `HipEncryptPassword.bat` command as follows:
	`C:\EncryptPassword>HipEncryptPassword.bat`
	2. When the system prompts to enter the password, type the password as follows:
	`Enter clear text password>Password`
	After generating the encrypted password, the `hip.keystore` file is generated in `C:\EncryptPassword\` folder.
	3. Rename the `hip.keystore` file to a valid filename, for example, `hip_bkupreadNode.keystore`.
	4. Copy the `hip_bkupreadNode.keystore` file from the `C:\EncryptPassword\` folder to `{com.emc.healthcare.home}/registry/`.
	5. Set the `xdb.backup.readNode.keystore` property as follows:

	`xdb.backup.readNode.keystore=${com.emc .healthcare.home}/registry/hip_bkupreadNode .keystore`
	After encryption, in the `registry.properties` file, the `xdb.backup.readNode.password` property appears as follows:
	`xdb.backup.readNode.password=HIP_ENCR_PASS =JxwGkf59eneCKgVhZljyEA==`

xdb.backup.readNode.keystore	The path to the backup Read node keystore file. This property is mandatory if the HIP encrypted password is configured.

Backup write node parameters (optional):

xdb.backup.writeNode .bootstrapFileName	The bootstrap filename for the backup Write node.
xdb.backup.writeNode.databaseName	The database name for the backup Write node.
xdb.backup.writeNode.userName	The user name for the backup Write node.

xdb.backup.writeNode.password	The password for the backup Write node.
	Note: Perform the following steps to encrypt the password:
	1. From the command prompt, execute the `HipEncryptPassword.bat` command as follows:
	`C:\EncryptPassword>HipEncryptPassword.bat`
	2. When the system prompts to enter the password, type the password as follows:
	`Enter clear text password>Password`
	After generating the encrypted password, the `hip.keystore` file is generated in `C:\EncryptPassword\` folder.
	3. Rename the `hip.keystore` file to a valid filename, for example, `hip_bkupwriteNode.keystore`.
	4. Copy the `hip_bkupwriteNode.keystore` file from the `C:\EncryptPassword\` folder to `{com.emc.healthcare.home}/registry/`.
	5. Set the `xdb.backup.writeNode.keystore` property as follows:
	`xdb.backup.writeNode.keystore=${com .emc.healthcare.home}/registry/hip _bkupwriteNode.keystore`

	After encryption, in the `registry.properties` file, the `xdb.backup.writeNode.password` property appears as follows:
	`xdb.backup.writeNode.password=HIP_ENCR_PASS`
	`=JxwGkf59eneCKgVhZljyEA==`

xdb.backup.writeNode.keystore	The path to the backup Write node keystore file. This property is required if the HIP encrypted password is configured.

Note:

- Backup list can be blank and optional.
- Primary write node and read node can be configured with same values.

Configuring the Registry Configuration File Properties

The Registry configuration file properties enable the Registry to find and use the Registry configuration file.

Property	Description
config.autoImport	Specifies whether the system must automatically update the Documentum xDB-based `registry-config.xml` file each time it detects the changes made to the file system version.
config.document	The path and filename of the `registry-config.xml` file that resides in the file system. If this document does not already exist in the Documentum xDB database, Registry Server loads the file from the file system to the Documentum xDB database regardless of the value of the `config.autoImport` property. By default, the Registry Server stores the file in the HIP Configuration directory. The Documentum xDB database stores this file in the Documentum xDB library path that is defined in the `xdb.libraryPath` property.

Configuring the MLLP Parameters

The MLLP parameters identify and control the ports where the Registry Server listens for patient identity feeds.

Property	Description
mllp.port	The non-secure port that the Registry Server uses to listen to the ITI-8 Patient Identity Feed messages. **Default value**: 0. When set to **0**, the Registry Server does not open a listening port.
mllp.securePort	The optional secure port that the Registry Server uses to listen to the ITI-8 Patient Identity Feed messages. **Default value**: 0. When set to 0, the Registry Server does not open a listening port. When not set to 0, you must also configure the HTTPS properties in this file. **Note:** The `https.server.privateKeyPassword` property should be configured if you enable the MLLP secure port. The value of this property must be the private key password.
hl7.inbound.message.encoding	The MLLP encoding format. **Default value**: UTF-8. HIP Repository supports only **ISO 8859-1** and **UTF-8** encoding.
mllp.routeBuilderScriptSource	If defined, overrides the default MLLP RouteBuilder script included in the Registry Server installation. The script is located within the classpath of one of the JAR files. Copies of the script are also located in the WAR file package and in the HIP configuration directory. This property is optional. If not defined, the Registry Server uses the default value, which is the script included with the Registry Server installation in the classpath. For example: `mllp.routeBuilderScriptSource=classpath:com/emc/healthcare/xds/registry/commons/MllpRouteBuilder.groovy`

Property	Description

	If you change the script located in the `/ROUTES` directory of the WAR file, use the following syntax to load the altered script.
	`mllp.routeBuilderScriptSource=ROUTES /MllpRouteBuilder.groovy`
	If you change the script located on the file system, load the altered script using the absolute path to the file.
	For example:
	`mllp.routeBuilderScriptSource=file:C:\ \absolute\\path\\myMllp.groovy`
	Ensure that you use a path that is appropriate for your operating system. This example shows a sample Windows path.

Configuring the Custom SOAP Routes Properties

Property	Description
soap.routeBuilderScriptSource	Specifies whether you want to override the default SOAP RouteBuilder script provided with the Registry Server installation. This property is optional. The default script is located within the classpath of one of the JAR files. Copies of the script are also located in the WAR file package and in the HIP configuration directory. If this property is not defined, the Registry Server uses the default value, which is to use the script included with the Registry Server installation in the classpath. If you change the script located in the`/ROUTES` directory of the WAR file, use the following syntax to load the altered script. For example: `soap.routeBuilderScriptSource=ROUTES /SoapRouteBuilder.groovy`

Property	Description

	Note: The `SoapRouteBuilder.groovy` file must have the `xuaEnabled` property defined in the file. The server fails to start if the property is missing in the file.
	
	If you change the script located on the file system, load the altered script using the absolute path to the file.
	
	For example:
	
	`soap.routeBuilderScriptSource=file:C:\`
	`\absolute\\path\\mySOAP.groovy`
	
	Ensure that you use a path that is appropriate to your operating system. This example shows a Windows path.

Configuring the Request and Response Validator Properties

The request and response validator flags are located in the spring configuration file in the deployed WAR. You must enable these flags to avoid problems that may occur if the request is not correct.

Property	Description
Request Validator Properties	
registry.iti18.requestValidator.enabled **registry.iti42.requestValidator.enabled** **registry.iti51.requestValidator.enabled** **registry.iti61.requestValidator.enabled** **registry.iti62.requestValidator.enabled**	Specifies whether to disable the incoming message validation for the specified IHE transaction. These flags are optional and are enabled by default.
Response Validator Properties	
registry.iti18.responseValidator .enabled **registry.iti42.reponseValidator .enabled** **registry.iti51.responseValidator .enabled** **registry.iti61.responseValidator .enabled** **registry.iti62.responseValidator .enabled**	Specifies whether to disable the outgoing message validation for the specified IHE transaction. These flags are optional and are enabled by default.

Configuring the IHE Endpoint for Trusted Hosts

The IHE endpoint for trusted hosts is provided for applications such as Connector for Epic (C4E), Clinical Archiving, which do not support PPIC feature, and cannot use the default ITI-18 endpoint to access XDS Registry to register/deregister documents.

Property	Description
registry.trusted.hosts.enabled	Specifies whether to enable the IHE endpoint for trusted hosts. **Default value**: false.

Configuring the HTTPS Properties

The HTTPS properties enable the XDS Registry Server to use the secure HTTPS communication. You must configure these properties if `mllp.securePort` is set to a non-zero value.

The first four HTTPS configurations are required if XUA is enabled.

Property	Description
https.keyStore	The location of the keystore on the system where you keep the private SSL certificates for this system. If you do not use HTTPS, comment this property by prefixing it with a pound sign (#). This property is optional and has no default value.
https.keyStorePassword	The password that is used to access the keystore. If you do not use HTTPS, comment this property by prefixing it with a pound sign (#). This property is optional and has no default value.
https.trustStore	The location of the truststore on the system where you keep SSL certificates of machines trusted in TSL connections. This location is where you keep the certificates of Document Consumers and XDS Repositories. If you do not use HTTPS, comment this property by prefixing it with a pound sign (#) . This property is optional and has no default value.
https.trustStorePassword	The password that is used to access the truststore.If you do not use HTTPS, comment this property by prefixing it with a pound sign (#). This property is optional and has no default value.

Property	Description
https.ciphersuites	The Cipher Suite that is used to encrypt the session. If you do not use HTTPS, comment this property by prefixing it with a pound sign (#). This property is optional and has no default value.
https.server.keyAlias	The alias that is used for the server certificate in the keystore. If not specified, the first key read in the keystore is used.
https.server.privateKeyPassword	The private key password that is used to encrypt the data. **Note:** You must configure this property if MLLP secure port is enabled. The value of this property must be the private key password.

Configuring the ATNA Properties

Property	Description
audit.host	The name of the system that hosts the ATNA audit repository.
audit.port	The port number of the ATNA audit repository. **Default value**: 514.
audit.transport	The transport type for the ATNA audit repository. For example, BSD, TLS, or UDP. **Default value**: UDP.
audit.sourceId	The source ID of the event. This property has no default value.

Configuring the XUA Properties

The `registry.properties` file contains user-definable properties for XUA on the server. The HIP XDS Repository Server and HIP XDS Registry Server share these configuration steps. If you use both the HIP XDS Repository and HIP XDS Registry Servers, you must enable XUA separately for each component.

Property	Description
registry.xua.enabled	Specifies whether you want to enable XUA for the XDS Registry Server. **Default value**: false.

After enabling XUA, you must configure the `registry.properties` file as described in the following topics:

Configuring the XUA Policy

The `ws-policy.xml` file enables the Web Service security for the server. This file defines and enables standard WS-Security features such as confidentiality (encryption), integrity (signing), and authentication (SAML token) for Web Services.

A sample `ws-policy.xml` file resides in the following XDS Registry directory:

`/webapps/registry/config/registry/`
Copy the sample `ws-policy.xml` file to `/webapps/registry/WEB-INF/classes/`.

Alternatively, you can place this file in a different folder and define the file location in the server classpath.

Configuring the XUA SAML Attribute Values

The XDS Registry Server uses the XUA SAML attribute properties to validate SAML Security Token attributes sent as part of a Registry request.

Property	Description
xua.service.endpoint	The endpoint regular expression. The XDS Registry Server compares this value against the audience restriction attribute provided in the token.
xua.crypto.provider	The crypto provider that you must use for encryption and decryption, and signature validation.
xua.supported.authentication.methods	A comma-separated list of authentication methods supported by the XDS Registry.
xua.purposeOfUse.codeSystem	The supported system that is used for the PurposeOfUseCode attribute.
xua.purposeOfUse.code.values	A comma-separated list of purpose of use codes that the XDS Registry supports.
xua.role.codeSystem	The supported code system value for the RoleCode attribute.

Property	Description
xua.role.code.values	A comma-separated list of roles that the XDS Registry supports.
xua.saml2.token.validator	The validator that validates the SAML token in the request.

Configuring the XUA Attribute Validation Property

The XUA attribute validation option can enable or disable some SAML attribute validations.

Property	Description
xua.authz.consent.option	Specifies whether to enable or disable the validation of the patient consent attribute value of SAML token. Default value: `false`.

Configuring the Trusted Assertion Provider Properties

This configuration is required if you enable XUA. These properties do not have any default value.

Property	Description
xua.assertion.provider.trustStore	The path to a truststore file.
xua.assertion.provider.trustStorePassword	The password to access the truststore file.

Configuring the PPIC Properties

Property	Description
ppic.enabled	Specifies whether to enable the PPIC server for user authorization. If enabled, the PPIC server checks if the user has permission to view the documents that a retrieve query returns. **Default value**: false. **Note:** You must configure the **ppic.pdpServiceUrl** property, if you enable this property.
ppic.pdpServiceUrl	The URL for making PDP service call for PPIC.

Configuring the Usage Report Properties

Property	Description
usagereport.username	The username that you use to log in to the Usage Reporting web application. The default username is **`Administrator`**.
usagereport.password	The password you use to log in to the Usage Reporting web application. The default password is **`password`**.

Configuring the Unified Endpoint Properties

Property	Description
unifiedEndpoint .routeBuilderScriptSource	The UnifiedRouteBuilder script path. **Default value**: classpath:com/emc/healthcare/xds /registry/commons/UnifiedEndpointRouteBuilder .groovy.
unifiedEndpoint.enabled	Specifies whether to enable or disable the unified endpoint on the XDS Registry server. **Default value**: true. **Note:** If **unifiedEndpoint.enabled** is set to **`true`**, it is mandatory to enable HTTP. If HTTP is not enabled, the requests to XDS Registry will fail.
unifiedEndpoint.xdsServer.port	The http port of the application server where the XDS Registry server is deployed. This property is mandatory and can take any value greater than zero. **Default value**: 0.
unifiedEndpoint.xdsServer .contextPath	The context name of the XDS Registry Server in the application server. **Default value**: /registry.
unifiedEndpoint.xdsServer .servicePath	The service path of the XDS Registry Server in the application server. **Default value**: **`/services/`**.

Configuring Registry as DSUB Notification Publisher

Property	Description
dsub.enabled	Specifies whether to enable or disable the DSUB broker. **Default value**: false If the value of **dsub.enabled** is set to **`true`**, it is mandatory to configure the RabbitMQ server properties.
notificationBroker.iti54Url	The ITI-54 endpoint of the Notification Broker where ITI-54 messages are published. If the URL is secured, you must configure the HTTPS properties.

Configuring the RabbitMQ Properties

The following properties need to be configured only if the **dsub.enabled** property is set to **`true`**.

Property	Description
rabbitmq.ssl.enabled	Specifies whether SSL is enabled between HIP servers and RabbitMQ server. **Default value**: False. You must configure the HTTPS properties if this property is enabled.
rabbitmq.host	The name of the system that hosts the RabbitMQ server.
rabbitmq.port	The RabbitMQ server port.
rabbitmq.username	The user name for the RabbitMQ server.
rabbitmq.password	The password for the RabbitMQ server.
rabbitmq.virtualhost	The virtual host name for the RabbitMQ server. This host must be pre-created in the RabbitMQ server to prevent deployment failure.
rabbitmq.maximumRedeliveries	The maximum number of times the XDS Registry attempts to redeliver the ITI-54 notification to the Notification Broker. • `0`: Indicates no redeliveries • `5`: Indicates five attempts to redeliver. **Default value**: 3

Property	Description
rabbitmq.redelivery.delay	The number of milliseconds the XDS Registry waits before the next re-delivery attempts to the Notification Broker. **Default value**: 5000
rabbitmq.iti54PublisherExchange	The RabbitMQ exchange where the published metatdata messages are stored by the XDS Registry for further processing. **Default value**: iti54PublisherExchange
rabbitmq.iti54PublisherQueue	The RabbitMQ queue where the ITI-54 notifications are queued which are further processed by XDS Registry by invoking DSUB Notification Broker ITI-54 endpoint. **Default value**: iti54PublisherQueue
rabbitmq.iti54PublishFailedExchange	The RabbitMQ exchange where the failed messages for ITI-54 are stored for further processing.
rabbitmq.iti54PublishFailedQueue	The RabbitMQ queue where the failed messages for ITI-54 are queued for further processing.

Configuring the Registry Configuration File

The XDS Registry Server configuration file resides in the xDB healthcare database in the `/registry` library. This file stores information about your Registry. The Registry configuration file resides in two places. The Documentum xDB database stores the main Registry configuration file that the server uses. Another version of the file resides on the file system to enable you to easily make changes to the file.

Element	Description
strictAboutCodes	Specifies how the XDS Registry Server responds when it receives register requests that do not contain valid coded document metadata. The server validates the coded metadata in the request against the appropriate code set defined in `registry-config.xml`. If set to **`True`**, the server rejects the request. If set to **`False`**, the server accepts the request even if it does not recognize the code.

Element	Description
strictAboutPatientIds	Specifies how the XDS Registry Server responds when it receives register requests where the patient ID is unknown or does not match the patient IDs already known to the Registry. The server validates the patient identifier in the request against the patient identifiers received from the patient identity feed source. If set to **`True`**, the server rejects the register request. If set to **`False`**, the server accepts the request.
codeClassification	All classification codes that the Registry expects when receiving a register request. This element contains the following two attributes: • **name**: The coded metadata name. • **classificationScheme**: The scheme value of the coded metadata.
assigningAuthority	The assigning authority that provides IDs for patients and submits patient identities to the Registry. The XDS Registry Server accepts the registration requests only from the assigning authorities that are defined here.
Code	A code, which is a sub-element of the codeClassification element. The element has the following three attributes: • **code**: The code value. • **codeSystemName**: The ID of the code system to which the code belongs. • **displayName**: The display name of the code.

Configuring the HIP PPIC Mapping Properties File

Property	Description
documentEntry.patientId	The XDS Affinity Domain Patient identifier.
documentEntry.typeCode	The code that indicates the precise kind of document (for example, Pulmonary History and Physical, Discharge Summary, Ultrasound Report).

Property	Description
documentEntry.classCode	The high-level classification of documents that indicates the kind of document (for example, report, summary, note, consent.)
documentEntry .healthcareFacilityCode	The type of organizational setting of the clinical encounter during which the documented act occurred.
documentEntry.confidentialityCode	The level of confidentiality of the document.
documentEntry.homeCommunityId	The globally unique identifier for a community.
documentEntry.eventCode	The main clinical acts, such as a colonoscopy or an appendectomy, being documented.
documentEntry.practiceSettingCode	The clinical specialty where the act that resulted in the document was performed. For example, Family Practice, Laboratory, Radiology.
request.subjectId	The logical identifier of the user performing the original service request.
request.subjectRole	The relevant user subject roles from a locally defined Code-Set.

Configuring the Web Container Heap Memory

You have to configure the initial and maximum heap size settings for your J2EE Web Application container according to the memory allocated to the server. You must also monitor the J2EE Web Application container during testing to ensure that the settings are sufficient.

For Apache Tomcat, EMC recommends:

Running the server as a service:

```
#Set initial heap size Xms and maximum heap size -Xmx JAVA_OPTS=”
-Xms512m -Xmx1024m -XX:MaxPermSize=512m”
```

Running the server as a standalone system:

```
Set “JAVA_OPTS”=-Xms256m -Xmx1g -XX:MaxPermSize=256m”
```

Configuring the Trusted Hosts

1. Go to the `<HIP_HOME>/registry` folder.
2. Open the `registry-context-extension.xml` file.
3. Add the IP address of all trusted hosts as follows:

   ```
   <util:list id="trustedHostsIPAddressList" list-class="java.util.ArrayList">
   <value>10.31.170.192</value>
   <value>10.31.170.193</value>
   </util:list>
   ```

By default, the additional ITI-18 endpoint can only be accessed over a **secured** transport.

4. If you need to provide access over non-secure channel, add the following bean:

```
<bean id="com.emc.healthcare.commons.core.cxf.CompositeSoapServiceAccessValidator"
     class="com.emc.healthcare.commons.core.cxf.CompositeSoapServiceAccessValidator">
     <constructor-arg index="0">
     <list>
     <--!<ref bean="com.emc.healthcare.commons.core.cxf.HttpsRequestAccessValidator"/>-->
     <ref bean="com.emc.healthcare.commons.core.cxf.WhitelistIPAddressAccessValidator"/>
     </list>
     </constructor-arg>
</bean>
```

Configuring the IHE Endpoint for Trusted Hosts, page 43 provides the configuration information on enabling the IHE endpoint for trusted hosts.

Configuring SSL

This section describes the steps to configure SSL for Apache Tomcat and Oracle WebLogic.

Configuring SSL for Tomcat

Add the paths for the keystore and truststore in the following file:

```
<Tomcat_install_dir>/conf/server.xml
```

For example:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
    SSLEnabled="true" maxThreads="150" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"
    keystoreFile="C:/Users/Administrator/.hip/keystore.jks"
    keystorePass="changeit"
    truststoreFile="C:/Users/Administrator/.hip/truststore.jks"
    truststorePass="changeit"/>
```

Configuring SSL for WebLogic

1. Log in to the WebLogic console.

   For example,

   `http://server:7001/console`

2. For each server, enable the **Configuration > General > SSL Listen Port Enabled** setting.

   Ensure that you use a correct and unused port.

3. For each server, change the KeyStore configuration as follows:

   a. From the **Configuration > Keystores** panel, click **Change**.

   b. Type the values for the following fields as given in the example below:

      Custom Identity Keystore=**`C:/Users/Administrator/.hip/keystore.jks`**

Custom Identity Keystore Type=`JKS`

Custom Identity Keystore Passphrase=`changeit`

Custom Trust keystore=`C:/Users/Administrator/.hip/truststore.jks`

Custom Trust Keystore Type=`JKS`

Custom Trust Keystore Passphrase=`changeit`

4. For each server, change the SSL configuration as follows:
   a. In the **Configuration > SSL** panel, configure the following fields:

      Private Key Alias=`serverXX`

      Private Key Passphrase=`changeit`
5. Click **Save**.

Chapter 6

Installing the XDS Registry Server

This chapter describes the steps to install the XDS Registry Server on Microsoft Windows system and Linux system.

Deploying the HIP Registry WAR File on Windows

HIP supports deploying the HIP Registry WAR file using Apache Tomcat or Oracle WebLogic.

Follow the procedure described for the application server you have installed.

Deploying the HIP Registry WAR File Using Tomcat

1. Stop the J2EE Web Application container.
2. Copy the XDS Registry Server WAR file to the following directory:

   `<tomcat_install_dir>/webapps/`

3. Rename the WAR file to `registry.war`.
4. Start the J2EE Web Application container to expand the WAR file.

The examples in this guide are provided with the assumption that the WAR file is deployed in `<tomcat_install_dir>/webapps/`.

Deploying the HIP Registry WAR File Using WebLogic

Before deploying the WAR file:

1. Set the following environment variable:

   ```
   Name:  DOMAIN_HOME
   ```

   ```
   Value:  ~\Oracle\Middleware\user_projects\domains\domain{domain used
   for deployment}
   ```

2. Set the following System Properties in the WebLogic startup script:

```
com.sun.xml.ws.spi.db.BindingContextFactory=com.sun.xml.ws.db.glassfish
.JAXBRIContextFactory javax.xml.bind.JAXBContext=com.sun.xml.bind.v2
.ContextFactory javax.wsdl.factory.WSDLFactory=com.ibm.wsdl.factory
.WSDLFactoryImpl
```

3. Set the HIP home location in the startup script to get the log files generated in the `<user.home>/.hip/` folder.

   Example:

   ```
   set JAVA_OPTIONS=%JAVA_OPTIONS% -Dcom.emc.healthcare.home=C:\Users
   \<username>\.hip
   ```

1. Log in to the WebLogic Admin console.
2. Go to **base_domain** > **deployment**.
3. Click **Install**.
4. From **Install Application Assistant**, click the **upload your file** link in the Note.
5. From **Deployment Archive**, browse and select the `hip-registry-1.9.0.war` file.
6. Click **Next**.
7. Select **Install as application**.
8. Click **Next**.
9. Click **Finish**.
10. Check if the deployment state of HIP Registry is Active.
    The Active state shows that the deployment is successful.

Deploying the HIP Registry WAR File on Linux

1. Log in as **root** user.
2. Copy `hip-registry-1.9.0.war` to the following location:
   ```
   $CATALINA_HOME/webapps
   ```
3. Run the following command to change Tomcat installation owner to **dmadmin**.
   ```
   chown -R $CATALINA_HOME dmadmin:dmadmin
   ```
4. Set **dmadmin** environment variables.
5. Set HIP Java options in `$CATALINA_HOME/bin/setenv.sh`.
   ```
   JAVA_OPTS="-Xms512m -Xmx1g -XX:MaxPermSize=512m -Dcom.emc.healthcare.home=/home/dmadmin/.hip"
   ```
6. As **dmadmin**, copy the `.hip` folder to the following location:
   ```
   /home/dmadmin
   ```
7. As root user, create Tomcat `startup/etc/init.d/tomcat` setting the values appropriately.
   For example:
   ```
   #!/bin/bash
   # description: Tomcat Start Stop Restart
   ```

```
# processname: tomcat
# chkconfig: - 90 10
# Source function library
. /etc/rc.d/init.d/functions

CATALINA_HOME=/app/apache-tomcat-7.0.42
TOMCAT_OWNER=dmadmin

#Set Startup Options for HIP
#See $CATALINA_HOME/bin/setenv.sh
#Check they have been used using ps-ef|grep tomcat

case $1 in
start)
        echo "Starting tomcat under dmadmin account..."
        echo "Note:xDB Must be Running or HIP Registry startup will fail..."
        su - $TOMCAT_OWNER -c "$CATALINA_HOME/bin/startup.sh"
        ;;
stop)
        echo "Stopping tomcat..."
        su - $TOMCAT_OWNER -c "$CATALINA_HOME/bin/shutdown.sh"
        ;;
restart)
        echo "Restarting tomcat under dmadmin account..."
        su - $TOMCAT_OWNER -c "$CATALINA_HOME/bin/shutdown.sh"
        sleep 2
        su - $TOMCAT_OWNER -c "$CATALINA_HOME/bin/startup.sh"
        sleep 2
        ;;
status)
        status tomcat
        ;;
*)
        echo "Usage: $0 {start|stop|restart|status}"
        exit 1
        ;;
esac
exit 0
```

8. Change permissions and set Tomcat to auto-start on reboot.

```
chmod +x /etc/init.d/tomcat
chkconfig tomcat on
```

Chapter 7

Verifying the Installation

This chapter describes the steps to verify the installation of HIP XDS Registry Server.

Verifying the Installation Using Tomcat

1. Ensure that the dependent libraries are installed.
2. Ensure that the `.hip` folder is available in the `C:\Users\<username>` folder.

   If you want to override the default location of the HIP configuration folder, override the `com.emc.healthcare.com` system property when you start the J2EE Web Application container.

   Use the following syntax: -

   ```
   Dcom.emc.healthcare.home=<hip_config_directory>
   ```
3. Start the xDB Server and ensure that the Healthcare database is operational.
4. Start the XDS Registry Server using the normal start procedure for the J2EE web application container. For example, start the server on Tomcat with the following command:

   ```
   [root]# service Tomcat start
   ```
5. Check the log file for errors.

   For example:

   ```
   /usr/share/apache-Tomcat-7.0.42/logs/catalina.out
   ```
6. Open a web browser and type the following URL:

   ```
   http://<host:port>/registry/services
   ```

   The Web Services Description Language (WSDL) page appears, which indicates that the server installation is successful.

Verifying the Installation Using WebLogic

1. Log in to WebLogic Admin console.
2. Ensure that the dependent libraries are installed.
3. Ensure that the `.hip` folder is available in `C:\Users\<username>` folder.

If the user does not have rights to access the `C:\Users\<username>` folder, perform the following steps:

a. Create `.hip` folder in any other location.

b. Update the `startWebLogic.cmd` file located at `~\Oracle\Middleware\user_projects\domains\domain{domain used for deployment}` by adding the following line:

```
set JAVA_OPTIONS=-Dcom.emc.healthcare.home=C:\.hip (".hip location")
```

4. Ensure that the HIP configuration properties files are present in the `.hip` folder.
5. Restart the system for the preceding changes to take effect.
6. Start the WebLogic server.
7. Deploy the registry WAR file.
8. Open a web browser and type the following URL:

```
http://<host:port>/registry/services
```

The WSDL page appears, which indicates that the server installation is successful.

Chapter 8

Upgrading HIP XDS Registry

This chapter contains the instructions to upgrade XDS Registry from version 1.8 to 1.9.

Upgrading XDS Registry from Version 1.8 to 1.9

1. Delete previous version of Registry WAR file from the deployed location.
2. Install the dependent libraries.

   The version of `org.openhealthtools.ihe.atna.auditor` jar file used by XDS Registry 1.9 is 2.0.0-p4. Ensure that you install the correct version.
3. Build a new Registry WAR from the `hip-registry-1.9.0.zip` file.
4. Go to the `\registry\config\` folder in the WAR file.
5. Copy the `registry` folder containing the properties file to HIP_HOME directory.
6. Configure the properties files in the HIP_HOME directory.

   XDS Registry 1.9 supports unified endpoints for XDS transactions. Ensure that the unified endpoint properties for Registry are configured in the `registry.properties` file.
7. Deploy the `hip-registry-1.9.0.war` file.
8. Verify the upgrade.

Chapter 9

Troubleshooting

This chapter describes the log settings, the XDS Registry installation issues, and their resolutions.

Log Settings

A log is a chronological record of system activities that is sufficient to enable the reconstruction and examination of the sequence of environments and activities surrounding or leading to an operation, procedure, or event in a security-relevant transaction from inception to final results.

Log Description

The log file for the XDS Registry Server is located in the `<hip_config_directory>\logs` folder.

For Apache Tomcat:

`C:\Users\<username>\.hip\logs\registry.log`

For Oracle WebLogic:

`C:\Users\<username>\.hip\logs\registry.log`

Log Management and Retrieval

The XDS Registry Server uses the Simple Logging Facade for Java (SLF4J) combined with a `logback` logging provider implementation.

The default log level setting is INFO.

You can increase the trace messages by setting the log level to DEBUG in the

`<tomcat installation directory>\webapps\registry\WEB-INF\classes\logback.xml` file.

For example:

```
<--!Set to DEBUG to see detailed HIP message information -->
<logger name="com.emc.healthcare">
<level value="DEBUG"/>
```

```
</logger>
```

Issues and Resolutions

This section describes the XDS Registry issues and their resolutions.

Context Initialization Failing when Deploying the Server WAR Files

Issue with HIP Configuration

Problem

When you try to install the Registry WAR files, you receive an error message as follows:

```
o.s.web.context.ContextLoader - Context initialization failed
org.springframework.beans.factory.BeanInitializationException:
Could not load properties; nested exception is java.io.FileNotFoundException:
  C:\Users\Administrator\.hip\registry\registry.properties
(The system cannot find the path specified)
 at org.springframework.beans.factory.config.PropertyResourceConfigurer.postProcessBeanFactory
(PropertyResourceConfigurer.java:87) ~[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
```

Cause

The `.hip` folder is not available in <user home>.

Resolution

Ensure that the HIP configuration properties are available in the `%HIP HOME%`.

Configuring the Server Directory, page 31 provides more details on HIP configuration.

Issue with Camel Jar Files

Problem

When you try to install the Registry WAR files, you receive an error message as follows:

```
10:57:01.592 [localhost-startStop-1] INFO  o.s.b.f.xml.XmlBeanDefinitionReader
 - Loading XML bean definitions from class path resource [META-INF/spring/xua-context.xml]
 10:57:01.895 [localhost-startStop-1] ERROR o.s.web.context.ContextLoader -
Context initialization failed
org.springframework.beans.factory.parsing.BeanDefinitionParsingException:
Configuration problem: Unable to locate Spring NamespaceHandler for XML schema namespace
[http://camel.apache.org/schema/spring] Offending resource: ServletContext resource
[/WEB-INF/spring/context.xml]
 at org.springframework.beans.factory.parsing.FailFastProblemReporter.error
(FailFastProblemReporter.java:68) ~[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
```

Cause

Camel JAR files are not added to the Tomcat classpath.

Resolution

Install Camel Library Dependencies.

Installing the Dependent Libraries, page 27 provides the steps to install camel dependent libraries.

Cannot Connect to the XDS Registry Server

Problem

You are unable to connect to the XDS Registry Server.

Cause

You are using incorrect URL or the Registry installation is incomplete.

Resolution

- Ensure that the endpoint (url:port) is correct.

  You can also validate the URL by accessing the XDS Registry Server WSDL.

  `http://localhost:<port>/registry/services`.

  If the WSDL loads then the XDS Registry Server is up.
- Ensure that the server is up and running.

  Chapter 7, Verifying the Installation provides the steps for verification.
- Ensure that the TLS certificates are valid and accessible.
- If you use XUA, ensure that the XUA configurations are correct.

  Configuring the XUA Properties, page 44 provides details on XUA configuration.

Cannot Access the xDB Server

Problem

You receiving the following error message when trying to connect to xDB:

```
o.s.web.context.ContextLoader - Context initialization failed
org.springframework.beans.factory.BeanCreationException:
Error creating bean with name 'com.emc.healthcare.commons.xdb.ManagedXhiveDriver'
defined in class path resource [META-INF/spring/hip-commons-xdb-beans.xml]:
Invocation of init method failed; nested exception is
com.xhive.error.XhiveException: CONNECTION_FAILED:
Connect to server at 127.0.0.1:1235 failed, Original message:
Connection refused: connect
```

Cause

Incorrect configuration of xDB Server properties.

Resolution

- Ensure that the xDB Server is running.
- Verify if the xDB bootstrap file name is correct.
- Ensure that the xDB username and password are correct.

Configuring the Documentum xDB Properties, page 34 provides details on xDB configuration.

Java Errors at Startup

Problem

You get an error message as follows in the startup.

```
java.lang.NoClassDefFoundError: Lca/uhn/hl7v2/parser/Parser;
at java.lang.Class.getDeclaredFields0(Native Method)
at java.lang.Class.privateGetDeclaredFields(Unknown Source)
at java.lang.Class.getDeclaredFields(Unknown Source)
at org.codehaus.groovy.vmplugin.v5.Java5.configureClassNode(Java5.java:313)
```

Cause

The server cannot find the HAPI JAR files because they were not deployed or were deployed incorrectly.

Resolution

Deploy the HAPI JAR files.

Chapter 6, Installing the XDS Registry Server provides more information.

XUA Policy File Error

Problem

You receive the following error in the log file during initialization:

```
Context initialization failed
org.apache.camel.RuntimeCamelException:
org.apache.cxf.ws.policy.PolicyException: Policy reference
classpath:ws-policy.xml could not be resolved.
```

Cause

The XDS Registry Server cannot find the `ws-policy.xml` file defined in the classpath.

Resolution

Ensure that you copy the sample `ws-policy.xml` file from

`/webapps/registry/config/registry/`

and place it in

`/webapps/registry/WEB-INF/classes/`

Alternatively, you can place this file in a different folder and define the file location in the server classpath.

servicesstore.jks File Not Found Error

Problem

You get an error message as follows in the log file:

```
java.io.FileNotFoundException: certificates\servicestore.jks (The system cannot
find the path specified)
```

Cause

The XDS Registry Server is unable to find the keystore file specified in the `serviceKeystore.properties` file.

If you use HTTPS to connect to the XDS Registry, the appropriate TLS certificates (keystore.jks, truststore.jks) must be located in the HIP_HOME directory.

Resolution

- Copy the keystore file to the location specified in `serviceKeystore.properties`.
- Verify if SSL and the paths for the keystores/truststores are configured in `tomcat\conf\server.xml` as follows:

  ```
  <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
  SSLEnabled="true"
  maxThreads="150" scheme="https" secure="true"
  clientAuth="false" sslProtocol="TLS"
  keystoreFile="C:/Users/Administrator/.hip/keystore.jks"
  keystorePass="changeit"
  truststoreFile="C:/Users/Administrator/.hip/truststore.jks"
  truststorePass="changeit"/>
  ```

Must Understand Headers Error

Problem

The XDS Registry Server writes the following error to the log file:

```
WARN  o.a.cxf.phase.PhaseInterceptorChain - Interceptor for {urn:ihe:iti:xds-b:2007}
DocumentRegistry_Service#{urn:ihe:iti:xds-b:2007}
DocumentRegistry_RegisterDocumentSet-b has thrown exception, unwinding now
org.apache.cxf.binding.soap.SoapFault: MustUnderstand headers:
[{http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd}Security]are not understood
```

Cause

An XUA-enabled XDS Registry Server received a request without a security header.

Resolution

With XUA enabled on the server, all requests must contain a security header. Either disable XUA on the server or instruct the sending application to send requests with security headers.

To disable XUA, open the `registry.properties` file and set the `registry.xua.enabled` property to false.

java.lang.OutOfMemoryError: PermGen space error

Problem

You receive an `java.lang.OutOfMemoryError:PermGen space` error while verifying the deployment of verifying the deployment of HIP Registry.

Cause

The permanent generation heap is full.

Resolution

Increase the Permgen space.

For Tomcat:

```
Set JAVA_OPTS=-Xms256m -Xmx512m -XX:PermSize=256m -XX:MaxPermSize=512m
```

For WebLogic:

Replace the following lines in the `setDomainEnv.cmd` files located at `C:\Oracle\Middleware \user_projects\domains\base_domain\bin`

Replace:

```
set WLS_MEM_ARGS_64BIT="-Xms256m -Xmx512m"
   set WLS_MEM_ARGS_32BIT="-Xms256m -Xmx512m"
```

With:

```
set WLS_MEM_ARGS_64BIT=-Xms256m -Xmx512m -XX:MaxPermSize=512m
  set WLS_MEM_ARGS_32BIT=-Xms256m -Xmx512m -XX:MaxPermSize=512m
```

Required Header Not Present Error

Problem

The XDS Registry Server writes the following error to the log file:

```
org.apache.cxf.interceptor.Fault: A required header representing a Message Addressing
Property is not present
```

Cause

An XUA-enabled XDS Registry Server received a request without a security header.

Resolution

If XUA is enabled on the server, all requests must contain a security header. You must either disable XUA on the server or instruct the sending application to send requests with security headers.

To disable XUA, open the `registry.properties` file and set the `registry.xua.enabled` property to **`false`**.

Unable to Connect to Documentum xDB

Problem

The XDS Registry Server is unable to connect to the xDB healthcare database and you receive the following error message in the log file:

```
ERROR Context initialization failed
org.springframework.beans.factory.BeanCreationException: Error creating bean
with name 'org.apache.cxf.bus.spring.BusApplicationListener' defined in class
path resource [META-INF/cxf/cxf.xml]: Initialization of bean failed; nested
exception is org.springframework.beans.factory.BeanCreationException: Error
creating bean with name 'camelContext': Invocation of init method failed; nested
exception is org.apache.camel.RuntimeCamelException:
org.springframework.beans.factory.BeanCreationException: Error creating bean with
name 'com.emc.healthcare.xds.registry.DirectRouteBuilder' defined in class path
resource [META-INF/spring/hip-xds-registry-commons-beans.xml]: Cannot create inner bean
```

Cause

The XDS Registry Server is unable to connect to the xDB healthcare database because:

- The Documentum xDB is not currently running
- The Documentum xDB database and log in information are incorrect

Resolution

- Connect to the Documentum xDB Admin Client and access the healthcare database to verify that Documentum xDB is running and accessible.
- Ensure that `registry.properties` file contains correct Documentum xDB database names and user names.

Configuring the Documentum xDB Properties, page 34 provides more information on xDB configuration.

o.s.web.context.ContextLoader - Context Initialization Failed

Problem

You get an error message as follows during context initialization:

```
09:18:28.719 [http-bio-80-exec-17] ERROR
o.s.web.context.ContextLoader - Context initialization failed
org.apache.camel.RuntimeCamelException:
org.apache.camel.FailedToCreateRouteException: Failed to create
route xds-iti8://0.0.0.0:9183: Route(xds-iti8://0.0.0.0:9183)
[[From[xds-iti8://0.0.0.0:9183... because of Failed to
resolve endpoint: xds-iti8://0.0.0.0:9183?clientAuth=MUST&codec=
%23iti8Hl7Codec&secure=true&sslContext=%23sslContext due to:
org.springframework.beans.factory.BeanCreationException:
Error creating bean with name 'sslContextFactory' defined
in class path resource [META-INF/spring/hip-xds-registry-
commons-beans.xml]: Cannot resolve reference to bean '
keyStore' while setting bean property 'keyManagerFactoryKeyStore';
nested exception is
org.springframework.beans.factory.BeanCreationException:
Error creating bean with name 'keyStoreFactory' defined in
class path resource [META-INF/spring/hip-xds-registry-commons-beans.xml]:
Error setting property values; nested exception is
org.springframework.beans.PropertyBatchUpdateException;
nested PropertyAccessExceptions (1) are:
PropertyAccessException 1:
org.springframework.beans.MethodInvocationException:
Property 'dataFile' threw exception;
nested exception is java.lang.NullPointerException
                at org.apache.camel.util.ObjectHelper.
wrapRuntimeCamelException(ObjectHelper.java:1344)
 ~[camel-core-2.12.1.jar:2.12.1]
                at org.apache.camel.spring.SpringCamelContext.
onApplicationEvent(SpringCamelContext.java:120)
 ~[camel-spring-2.12.1.jar:2.12.1]
```

Cause

You have not configured the HTTPS parameters that must be configured when you use secure MLLP port.

Resolution

If you use secure MLLP port, you must configure the following optional parameters in the `registry.properties` file.

For example, replace the following values according to your setup:

https.keyStore=`${com.emc.healthcare.home}/keystore.jks`

https.keyStorePassword=`changeit`

https.trustStore=`${com.emc.healthcare.home}/truststore.jks`

https.trustStorePassword=`changeit`

https.ciphersuites=`TLS_RSA_WITH_AES_128_CBC_SHA`

https.server.privateKeyPassword=`changeit`

The **https.server.privateKeyPassword** is mandatory for MLLP secure port.

CannotLoadBeanClassException: Error loading class

Problem

You receive an error message as follows:

```
Caused by: org.springframework.beans.factory.
CannotLoadBeanClassException: Error loading class
[com.emc.healthcare.xua.hanlder.XuaCallbackHandler]
for bean with name 'callbackHanlder' defined in
class path resource [META-INF/spring/xua-context.xml]:
problem with class file or dependent class; nested
exception is java.lang.UnsupportedClassVersionError:
com/emc/healthcare/xua/hanlder/XuaCallbackHandler :
Unsupported major.minor version 51.0 (unable to load
class com.emc.healthcare.xua.hanlder.XuaCallbackHandler)
                at org.springframework.beans.factory.
support.AbstractBeanFactory.resolveBeanClass
(AbstractBeanFactory.java:1278) ~[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.beans.factory.
support.AbstractAutowireCapableBeanFactory.createBean
(AbstractAutowireCapableBeanFactory.java:435)
 ~[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.beans.
factory.support.AbstractBeanFactory$1.getObject
(AbstractBeanFactory.java:295) ~
[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.beans.
factory.support.DefaultSingletonBeanRegistry.
getSingleton(DefaultSingletonBeanRegistry.java:223)
 ~[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.beans.
factory.support.AbstractBeanFactory.doGetBean
(AbstractBeanFactory.java:292)
~[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.beans.
factory.support.AbstractBeanFactory.getBean
```

```
(AbstractBeanFactory.java:194)
~[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.beans.
factory.support.BeanDefinitionValueResolver.
resolveReference(BeanDefinitionValueResolver.java:323)
 ~[spring-beans-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                ... 25 common frames omitted
```

Cause

You are using JDK 1.6 for the Application server.

Resolution

HIP servers require JDK 1.7. Use the JDK 1.7 version.

Apache Camel Shutting Down

Problem

You get an error message as follows:

```
09:27:55.568 [http-bio-80-exec-21]
INFO  o.a.camel.spring.SpringCamelContext
 - Apache Camel 2.12.1 (CamelContext: camelContext)
 is shutdown in 0.109 seconds
09:27:55.577 [http-bio-80-exec-21]
ERROR o.s.web.context.ContextLoader -
Context initialization failed
org.apache.camel.RuntimeCamelException:
java.net.BindException: Address already in use: bind
                at org.apache.camel.util.
ObjectHelper.wrapRuntimeCamelException
(ObjectHelper.java:1344) ~[camel-core-2.12.1.jar:2.12.1]
                at org.apache.camel.spring.
SpringCamelContext.onApplicationEvent
(SpringCamelContext.java:120) ~[camel-spring-2.12.1.jar:2.12.1]
                at org.apache.camel.spring.
CamelContextFactoryBean.onApplicationEvent
(CamelContextFactoryBean.java:301) ~[camel-spring-2.12.1.jar:2.12.1]
                at org.springframework.context.
event.SimpleApplicationEventMulticaster.
multicastEvent(SimpleApplicationEventMulticaster.java:96)
 ~[spring-context-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.context.
support.AbstractApplicationContext.publishEvent
(AbstractApplicationContext.java:334)
 ~[spring-context-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.
context.support.AbstractApplicationContext.
finishRefresh(AbstractApplicationContext.
java:948) ~[spring-context-3.2.4.RELEASE.
jar:3.2.4.RELEASE]
                at org.springframework.
context.support.AbstractApplicationContext.
refresh(AbstractApplicationContext.java:482)
 ~[spring-context-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.web.
context.ContextLoader.
configureAndRefreshWebApplicationContext(ContextLoader.java:389)
 ~[spring-web-3.2.4.RELEASE.jar:3.2.4.RELEASE]
                at org.springframework.web.context.
ContextLoader.initWebApplicationContext
(ContextLoader.java:294)
 ~[spring-web-3.2.4.RELEASE.jar:3.2.4.RELEASE]
```

Cause

Some of the MLLP ports used in HIP properties file are being used by some other application.

Resolution

Change the MLLP ports to the ports that are not used by other applications.

Appendix A

Sample Configuration Files

This appendix contains sample content of the following files:

- `registry.properties`
- `registry-config.xml`
- `hip-ppic-mapping.properties`

registry.properties

```
# User settable properties are listed in this file.
# The order of property evaluation is as follows:
# 1: ${com.emc.healthcare.home}/registry/registry.properties is consulted first
# 2: System.properties is consulted second.
#
# Property values are set in the order they are encountered. If the
# same property is defined in multiple locations, the final setter takes precedence
#
# All settable properties for this server are enumerated in this file.
# If a property defined and commented out this documents its default value.
#
#

# The description property is used to contain a description of this
# server instance for display purposes
# NO DEFAULT
description=XDS Registry

# ------------------------- xDB Related Parameters -------------------------
#
# The properties below that do not have values, have no defaults. The
# three properties without defaults listed below must be configured or
# the registry server will fail to start.
#
# The xdb.libraryPath has no default, however, it has been set to
# "/registry" in this property file so that the end user has one less configuration
# decision to make.

xdb.libraryPath=/registry

# xdb.cachePages=0
# xdb.maximumPoolSize=20

#--------------------------------------------------------------------------
# The following set of 5 parameters are used to define the primary READ XDB node.
# These settings are mandatory.
```

```
#----------------------------------------------------------------------------
xdb.readNode.bootstrapFileName=xhive://localhost:1235
xdb.readNode.databaseName=
xdb.readNode.userName=
xdb.readNode.password=
# xdb.readNode.keystore property is required if HIP encrypted password is configured

# DEFAULT=${com.emc.healthcare.home}/registry/hip.keystore
# xdb.readNode.keystore=${com.emc.healthcare.home}/registry/hip.keystore
#
# The same applies to xdb.writeNode.keystore, xdb.backup.readNode.keystore
#  and xdb.backup.writeNode.keystore.

#-----------------------------------------------------------------------------
# The following (optional) set of 5 parameters is used to define
# the primary WRITE XDB node.
# For Single node deployment, set these 4 values to blank
#-----------------------------------------------------------------------------
xdb.writeNode.bootstrapFileName=
xdb.writeNode.databaseName=
xdb.writeNode.userName=
xdb.writeNode.password=
#xdb.writeNode.keystore=   is required if HIP encrypted password is configured

#-----------------------------------------------------------------------------
# The following (optional) parameters are comma separated list for
# Backup READ XDB nodes setup.
#-----------------------------------------------------------------------------
xdb.backup.readNode.bootstrapFileName=
xdb.backup.readNode.databaseName=
xdb.backup.readNode.userName=
xdb.backup.readNode.password=
# xdb.backup.readNode.keystore=     is required if HIP encrypted password is
# configured

#-----------------------------------------------------------------------------
# The following (optional) parameters are comma separated list for
# Backup WRITE XDB nodes setup.
#-----------------------------------------------------------------------------
xdb.backup.writeNode.bootstrapFileName=
xdb.backup.writeNode.databaseName=
xdb.backup.writeNode.userName=
xdb.backup.writeNode.password=
#xdb.backup.writeNode.keystore=   is required if HIP encrypted password
# is configured

# ---------------------- Registry Configuration Parameters--------------------
# Indicates whether the xDB-based config document should be automatically
# updated when a change is detected to the copy on the file system.
# DEFAULT=false
# config.autoImport=false

# The path on the file system where the registry configuration document
# is stored. The final name portion of this document will be preserved in the
# xDB library. Using the defaults below, this means that the library path for
# this document will translate to:
#    ${xdb.libraryPath}/registry-config.xml
# Which translates to: /registry/registry-config.xml
#
#  Regardless of the value of the config.autoImport flag, the registry
# will attempt to load the document into xDB from the disk file copy if the
# library path for the document does not yet exist in xDB. After first
# load the value of the autoImport property is honored.
#
#
```

```
# DEFAULT=${com.emc.healthcare.home}/registry-config.xml
# config.document=${com.emc.healthcare.home}/registry/registry-config.xml

# -------------- Minimal Lower Layer Protocol (MLLP) Parameters ---------------#
# The MLLP parameters are used control the ports upon which the Registry
# server listens for ITI-8 Patient Identity feeds. The mllp.port and mllp.
# securePort both default to "0". When the port is  set to "0" the Registry
# server will not open a listening port.
#
# An optional unsecure port that will be used to listen for
# ITI-8 'Patient Identity Feed' messages.
# DEFAULT=0
# mllp.port=0

# An optional secure port that will be used to listen for ITI-8
# 'Patient Identity Feed' messages.
# If this property is set to a non zero value, the https properties
# must also be set.
# DEFAULT=0
# mllp.securePort=0

# An optional specification that allows users to override the default
# MLLP RouteBuilder script provided with the product. The actual location
# of this within the classpath is within one of the Jar files shipped
# with the product. A copy of this script is available inside this
# war package. Copies of the source used to define the routes have been
# copied into the ROUTES directory of this war.
#
#
# When this property is not set, the default is used, which loads the
# MllpRouteBuilder.groovy source from a resource on the class path. There
# are two other useful ways you could set this property.
#
#  1:  Load the copy that is shipped with the war file. The syntax for
#      doing this is shown below. The file name is path relative to the
#      exploded war directory
#
#      mllp.routeBuilderScriptSource=ROUTES/MllpRouteBuilder.groovy
#
#   2:  Load a copy of the source from the file system. Note that when
#       you load this file from the file system you must use the absolute
#       path of the file formatted in a way appropriate for your operating system.
#
#       mllp.routeBuilderScriptSource=file:/absolute/path/myMllp.groovy
#
#       Or for Windows:
#
#
#       mllp.routeBuilderScriptSource=file:C:/absolute/path/myMllp.groovy
#
#       This is equivalant to
#
#       mllp.routeBuilderScriptSource=file:C:\\absolute\\path\\myMllp.groovy
#
#       In Windows, the first "\" character is treated as an escape character
#       by the properties loader. Windows does allow the use of the
#       "/" character as a path separator.
#
# DEFAULT=classpath:com/emc/healthcare/xds/registry/commons/MllpRouteBuilder.groovy
# mllp.routeBuilderScriptSource=classpath:com/emc/healthcare/xds/registry/commons/MllpRouteBuilder.groovy


#---------------------------- Soap Route Builder ------------------------------#

# An optional specification that allows users to override the default
# Soap RouteBuilder script provided with the product. The actual location of
# this within the classpath is within one of the Jar files shipped with the
# product. See description for soap.routeBuilderScriptSource above
# for details on how this can be used.
#
# DEFAULT=classpath:com/emc/healthcare/xds/registry/commons/SoapRouteBuilder.groovy
# soap.routeBuilderScriptSource=classpath:com/emc/healthcare/xds/registry/commons/SoapRouteBuilder.groovy
#
# Optional flags to disable incoming message validation. These flags
# may be used in special situations to disable incoming message validation.
# These flags are intended mainly for Connectathon testing in case a testing
# partner does not pass message validation. Messages that do not pass
# validation have a high likelihood of failing for other reasons
# later in the processing cycle. These flags should always be set
# to "true" in production scenarios.
#
# DEFAULT=true
# registry.iti18.requestValidator.enabled=true
# DEFAULT=true
# registry.iti42.requestValidator.enabled=true
# DEFAULT=true
# registry.iti51.requestValidator.enabled=true
# DEFAULT=true
# registry.iti61.requestValidator.enabled=true
# DEFAULT=true
# registry.iti62.requestValidator.enabled=true
#----- Enables Additional IHE Endpoint for trusted Hosts Only ---------------#
registry.trusted.hosts.enabled=false
# --------------------------- HTTPS Related Properties -------------------- #
#
#   The following properties must be configured for secure communication
#   These are used for keystore and truststore configuration.
#   Keystore configurations (first 4 https configurations) are required
#   if XUA is enabled
#
#   The final property in this section, https.ciphersuites is used
#   to configure the suite used, a suitable value for this parameter is:
#   TLS_RSA_WITH_AES_128_CBC_SHA
#
#   The properties in this section have NO DEFAULTS


# https.keyStore
# https.keyStorePassword
# https.server.keyAlias
# https.server.privateKeyPassword
# https.trustStore
# https.trustStorePassword
# https.ciphersuites


# ---------------------  ATNA Audit Related Parameters --------------------- #

# Host name for the ATNA audit repository
# DEFAULT=localhost
# audit.host=localhost

# Port number for the ATNA audit repository.
# DEFAULT=514
# audit.port=514

# Transport type for the ATNA audit repository (BSD, TLS, UDP)
# DEFAULT=UDP
# audit.transport=UDP
# An optional source identifier for ATNA audit messages.
# NO DEFAULT
audit.sourceId=${description}

# ----------------------- XUA Related Properties ---------------------------#

# Flag to enable/disable Cross Enterprise User Assertion Validation
# Default value is false
# registry.xua.enabled=false

# The following properties are not required to be configured
# if XUA validation is disabled

# The validator to validate the SAML token in the request.
# The default value is com.emc.healthcare.xua.validator.XuaValidator
# xua.saml2.token.validator=com.emc.healthcare.xua.validator.XuaValidator

# The Crypto provider to be used for encryption/decryption and signature
# validation.
# The default value is org.apache.ws.security.components.crypto.Merlin
# xua.crypto.provider=org.apache.ws.security.components.crypto.Merlin

# The service endpoint regular expression to match against service endpoint
# attribute provided in the token.
# If not configured, the service endpoint provided in the token is not validated
# xua.service.endpoint


# The list of "," separated authentication methods supported by the XDS Registry.
# if not configured, Authentication method provided in the token is not validated
# xua.supported.authentication.methods

# The code system and the list of "," separated code values supported for
# the PurposeOfUseCode attribute provided in the token.
# if not configured, PurposeOfUseCode value provided in the token is not validated
# xua.purposeOfUse.codeSystem
# xua.purposeOfUse.code.values

# The code system and the list of "," separated code values supported
# for the Role attribute provided in the token.
# if not configured, Role value provided in the token is not validated
# xua.role.codeSystem
# xua.role.code.values

# The property to enable/disable Authorization Consent validation
# Default value is false
# xua.authz.consent.option=false

# Configuration of trusted assertion providers' certificates.
# It is a required configuration if XUA is enabled and has no default value.
# xua.assertion.provider.trustStore
# xua.assertion.provider.trustStorePassword

# ------------------------- PPIC Related Properties -----------------------#

# Flag to enable/disable PPIC
# Default value is false
# ppic.enabled=false

#The URL for making pdp service call for PPIC.
# Example: repository.pdpServiceURL=http://localhost:8080/ppic/pdp
ppic.pdpServiceUrl=http://localhost/ppic/pdp
#----------------------Usage Report properties---------------------------#

# User name to login to usage report User Interface
# Default is Administrator
#usagereport.username=

# User password to login to usage report User Interface
# Default is password
#usagereport.password=
#------------------- Unified EndPoint Related Properties-------------------#
 unifiedEndpoint.routeBuilderScriptSource=classpath:com/emc/healthcare/xds/registry/commons/UnifiedEndpointRouteBuilder.groovy
 unifiedEndpoint.enabled = true
 unifiedEndpoint.xdsServer.port=0
 unifiedEndpoint.xdsServer.contextPath=/registry
 unifiedEndpoint.xdsServer.servicePath=/services/
#-------------------  ------ DSUB Configurations -------------------------#
# Flag to enable/disable DSUB
# Optional, default value is false.
dsub.enabled=false

# DSUB ITI54 notification publish URL
# Required if dsub is enabled, no default.
notificationBroker.iti54Url=
#--------------------------- RabbitMQ Configurations -----------------------#
# The following RabbitMQ properties should be configured if dsub is enabled
# to publish metadata to the configured dsub broker

##The name of the machine that hosts RabbitMQ server.
#rabbitmq.host

##The RabbitMQ server port.
#rabbitmq.port

##The user name for RabbitMQ server.
#rabbitmq.username

##The password for RabbitMQ server.
#rabbitmq.password

##The virtual host name for RabbitMQ server.
#rabbitmq.virtualhost

##The RabbitMQ exchange where the published metatdata messages are stored
## by the notification broker for further processing.
#rabbitmq.iti54PublisherExchange

##The RabbitMQ queue where the incoming ITI-54 notifications are queued.
##rabbitmq.iti54PublisherQueue

##The RabbitMQ exchange where the failed messages for ITI-54 are stored
## for further processing.
#rabbitmq.iti54PublishFailedExchange

##The RabbitMQ queue where the failed messages for ITI-54 are queued
# for further processing.
#rabbitmq.iti54PublishFailedQueue

#The maximum number of times the XDS Registry attempts to redeliver the
## ITI-54 notification to the Notification Broker.
#rabbitmq.maximumRedeliveries=3

##The number of milliseconds the XDS Registry waits before the next
## re-delivery attempts to the Notification Broker.
#rabbitmq.redelivery.delay=5000
```

registry-config.xml

This sample is edited to reduce the length of the content. A full length sample resides in the `/registry` folder in your HIP Configuration Directory.

```
<?xml version="1.0" encoding="utf-8"?>
<registryConfig
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.emc.com/healthcare/xds/registry/commons/config"
    xsi:schemaLocation="http://www.emc.com/healthcare/xds/registry/commons/
     config registry-config.xsd">

    <strictAboutCodes>true</strictAboutCodes>
    <strictAboutPatientIds>true</strictAboutPatientIds>
    <codeClassification name="contentTypeCode" classificationScheme=
     "urn:uuid:aa543740-bdda-424e-8c96-df4873be8500">
     <code code="Communication" displayName="Communication"
           codeSystemName="Connect-a-thon contentTypeCodes"/>
     <code code="Evaluation and management" displayName="Evaluation and management"
           codeSystemName="Connect-a-thon contentTypeCodes"/>
        <code code="Conference" displayName="Conference" codeSystemName=
           "Connect-a-thon contentTypeCodes"/>
     <code code="Case conference" displayName="Case conference"
           codeSystemName="Connect-a-thon contentTypeCodes"/>
     <code code="Consult" displayName="Consult" codeSystemName=
           "Connect-a-thon contentTypeCodes"/>
     <code code="Confirmatory consultation" displayName="Confirmatory consultation"
           codeSystemName="Connect-a-thon contentTypeCodes"/>
     <code code="Counseling" displayName="Counseling" codeSystemName=
           "Connect-a-thon contentTypeCodes"/>
     <code code="Group counseling" displayName="Group counseling"
           codeSystemName="Connect-a-thon contentTypeCodes"/>
  </codeClassification>
    <codeClassification name="associationDocumentation"
    classificationScheme="urn:uuid:abd807a3-4432-4053-87b4-fd82c643d1f3">
    <code code="Additional_Information" codeSystemName=
      "Connect-a-thon associationDocumentation" displayName=
      "Additional Information"/>
    <code code="Corrected_Information" codeSystemName=
   "Connect-a-thon associationDocumentation" displayName="Corrected Information"/>
  </codeClassification>

    <assigningAuthority id="1.19.6.24.109.42.1.3"/>
    <assigningAuthority id="1.3.6.1.4.1.21367.2005.3.7"/>
    <assigningAuthority id="1.3.6.1.4.1.21367.2008.2.1"/>
    <assigningAuthority id="1.3.6.1.4.1.21367.2009.1.2.300"/>
    <assigningAuthority id="1.3.6.1.4.1.21367.2010.1.2.300"/>
    <assigningAuthority id="1.3.6.1.4.1.21367.13.20.1000"/>
    <assigningAuthority id="1.3.6.1.4.1.21367.13.20.2000"/>
    <assigningAuthority id="1.3.6.1.4.1.21367.13.20.3000"/>
</registryConfig>
```

hip-ppic-mapping.properties

```
documentEntry.patientId=urn:ihe:iti:xds-b:2007:patient-id
documentEntry.typeCode=urn:ihe:iti:xds-b:2007:document-entry:type-code
documentEntry.classCode=urn:ihe:iti:xds-b:2007:document-entry:class-code
documentEntry.healthcareFacilityCode=urn:ihe:iti:xds-b:2007:document-entry:healthcare-facility-type-code
documentEntry.confidentialityCode=urn:ihe:iti:xds-b:2007:confidentiality-code
documentEntry.homeCommunityId=urn:ihe:iti:xds-b:2007:home-community-id
documentEntry.eventCode=urn:ihe:iti:xds-b:2007:document-entry:event-code
documentEntry.practiceSettingCode=urn:ihe:iti:xds-b:2007:document-entry:practice-setting-code

request.subjectId=urn:oasis:names:tc:xacml:1.0:subject:subject-id
request.subjectRole=urn:oasis:names:tc:xacml:2.0:subject:role
```

Index