EMC® Documentum® XDS Registry

Version 1.7

Installation Guide

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

Documentation Feedback

Your opinion matters. We want to hear from you regarding our product documentation. If you have feedback about how we can make our documentation better or easier to use, please send us your feedback directly at IIGDocumentationFeedback@emc.com

Table of Contents

Revision History

Revision Date	Description
March 2015	Initial publication.

Chapter 1

About XDS Registry Server

This chapter contains the following topics:

Overview

The Cross-enterprise Document Sharing (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.

Components

HIP XDS Registry consists of the following components:

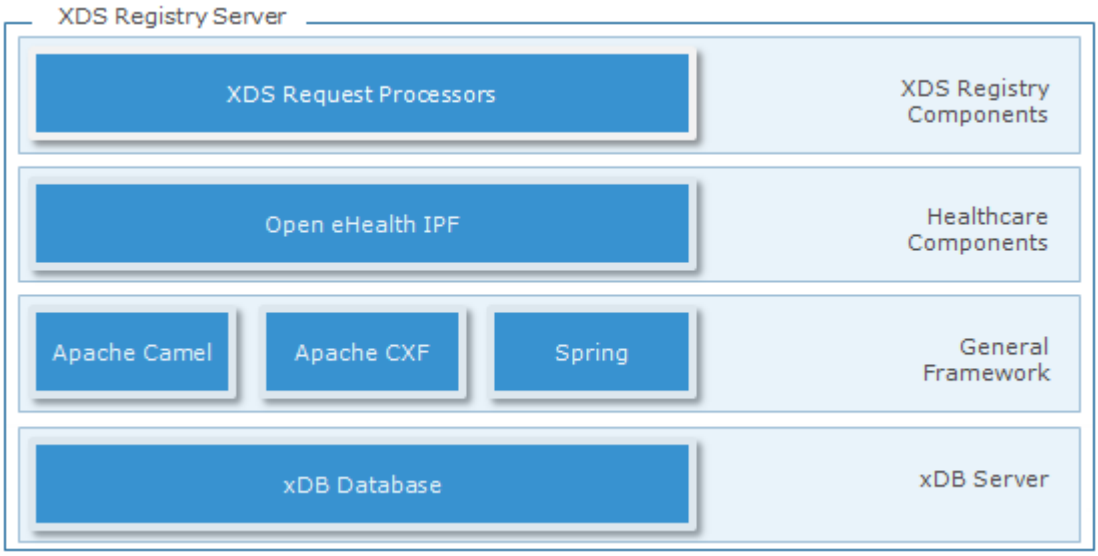
- HIP XDS Registry Server
- xDB

Architecture

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

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.

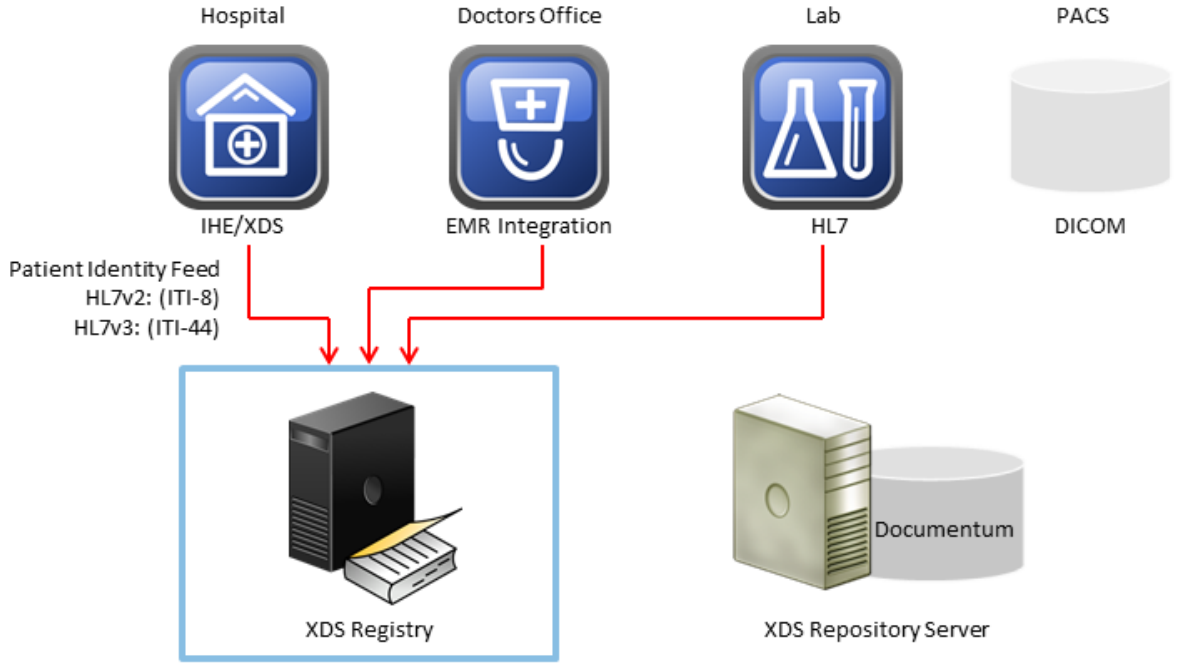
A healthcare community consists of different healthcare consumers and providers that need to access and share a patient's *Healthcare records*. Healthcare records include administrative records (patient information) and patient medical records (X-rays, doctor reports, lab results).

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:

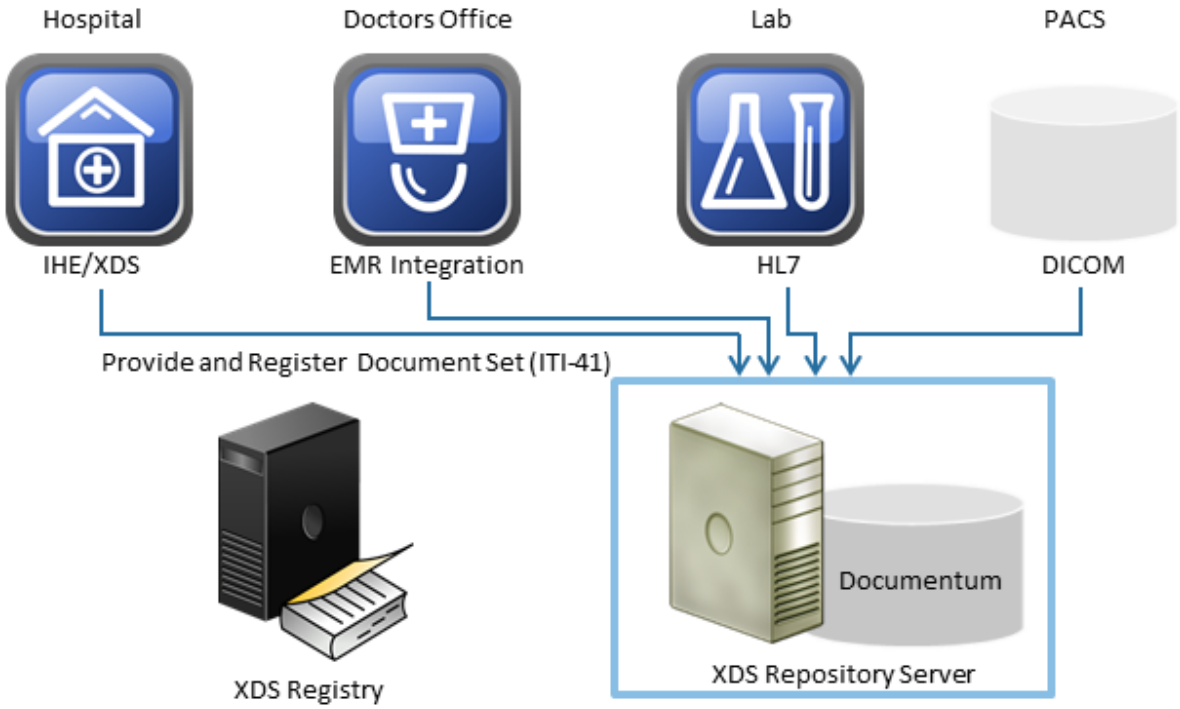
Healthcare providers create patient identities and supply that information to the registry through Patient Identity Feed transactions.

The following figure shows the Patient Identity Feed transactions:



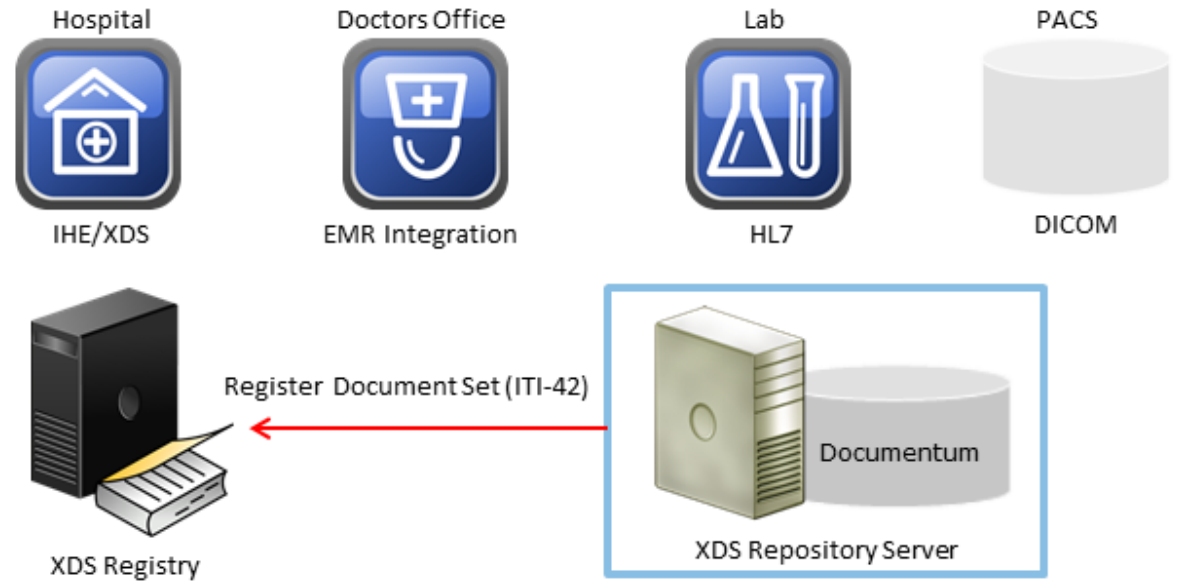
Healthcare providers submit new Healthcare records to an XDS Repository through the XDS Repository Server with the Provide and Register Document Set transaction. The XDS Repository Server stores the submitted Healthcare records in the XDS Repository.

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



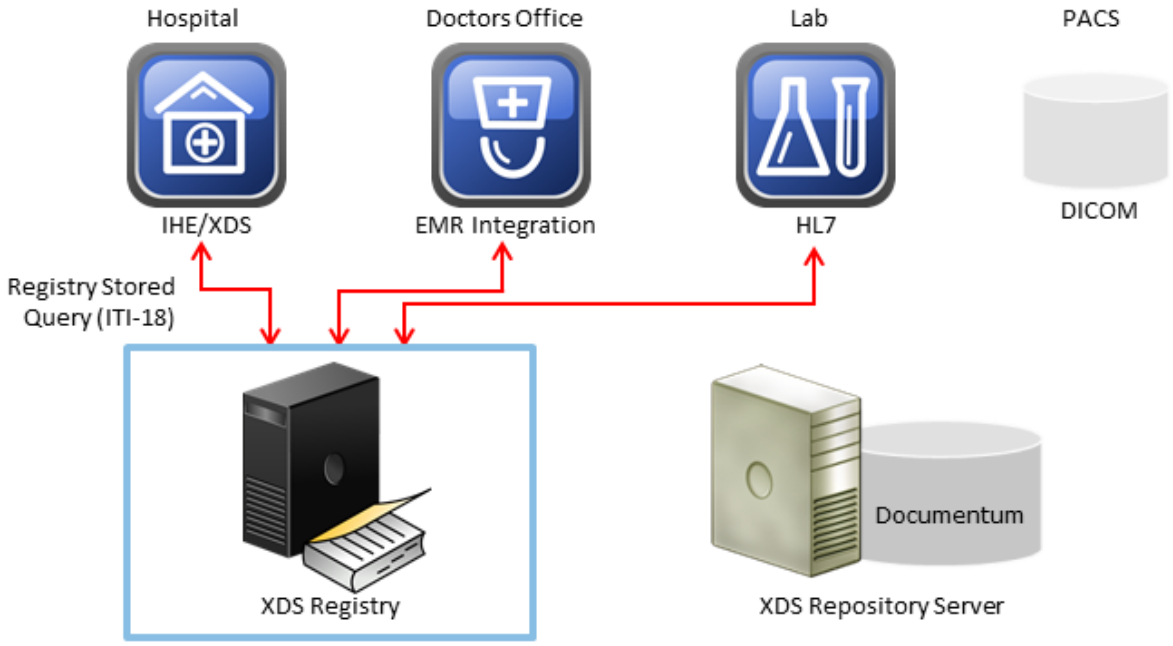
The XDS Repository Server then automatically registers the relevant document metadata with the XDS Registry with the Register Document Set transaction. Registering a healthcare record with the XDS Registry enables other healthcare providers to find the record.

The following figure shows the registration of document metadata.



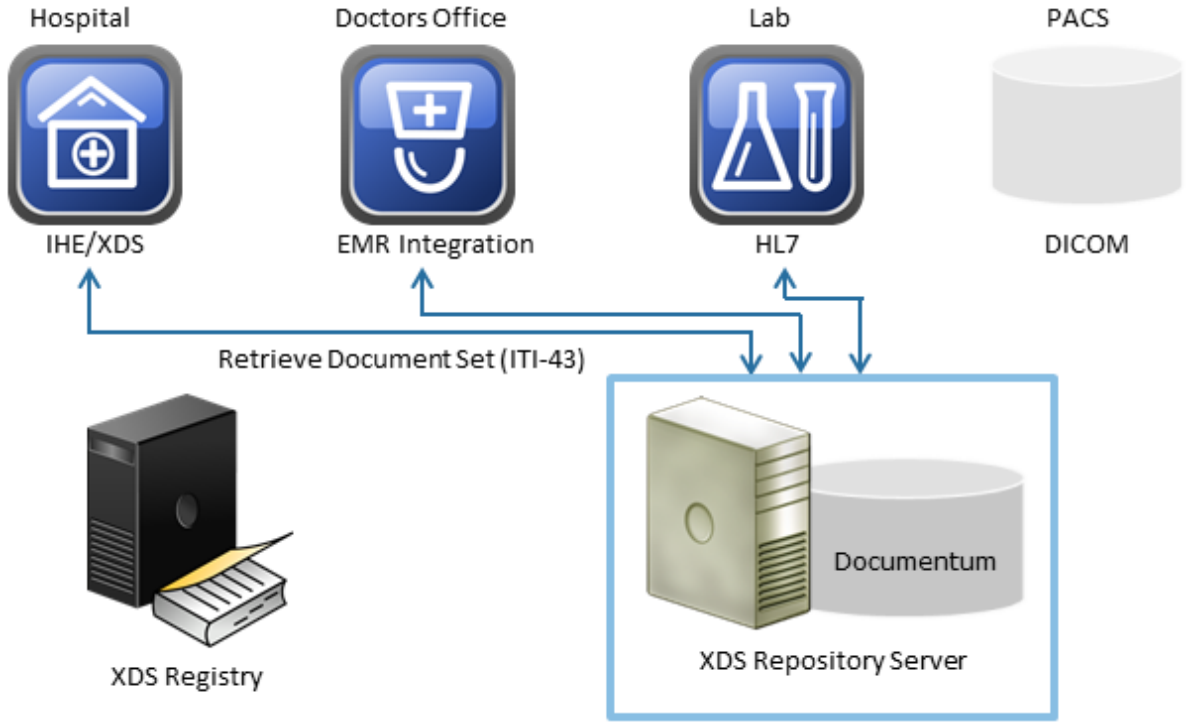
When healthcare providers need to obtain a patient's healthcare records, they query the registry with the Registry Stored Query request. The registry provides the location of the healthcare record. Healthcare providers operate as a Document Consumer when querying the XDS Registry.

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 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 HTTP://<*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 HTTP://<*host*>:<*port*>/registry/services/xds-iti61as

IHE Transaction	Description	Endpoint
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

Chapter 2

Features

This chapter contains the following topics:

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**: Transaction 18 is used by the Document Registry and Document Consumer actors.

  The Document Consumer requests a query by identifier (UUID), and passes parameters to the query. A parameter controlling the format of the returned data is passed; it selects either object references or full objects. The Document Registry services the query using its stored definitions of the queries defined for XDS.

- **ITI-42 Register Document Set**: Transaction [ITI-42] is used by the Document Repository Actor to register a set of documents with the Document Registry in XDS.b.

  The Document Repository submits document metadata to a Document Registry. The Document Registry receives and stores document metadata.

- **ITI-51 Multi-Patient Query**: Transaction ITI-51 is used by the Document Consumer and Document Registry actors.

  The Document Consume issues a Multi-Patient Stored Query to retrieve metadata based on criteria common to multiple patients. The Document Registry responds to a Multi-Patient Stored Query by providing the metadata or object references of registry objects which satisfy the query parameters.

- **ITI-57 Update Document Set**: Transaction 57 is used by the Document Administrator, Document Registry and Document Recipient actors.

  The Document Administrator Actor issues a collection of metadata updates to the Document Registry or Document Recipient Actor. The Document Registry or Document Recipient Actor accepts metadata updates.

- **ITI-61 Register On-Demand Document Entry**: Transaction 61 is used by the On-Demand Document Source and Document Registry actors.

  An On-Demand Document Source passes a Submission Request to a Document Registry Actor. The Submission Request contains metadata describing one or more On-Demand Document Entries. The Document Registry receives and stores metadata about available on-demand documents.

- **ITI-62 Delete Document Set**: Transaction 62 is used by the Document Administrator, Document Recipient and Document Registry actors. The Delete Document Set transaction deletes arbitrary metadata objects from a Document Registry or Document Recipient based on a list of entryUUID attribute describing the objects.

The Document Administrator issues metadata delete requests to Document Registry. The Document Registry or Document Recipient accepts metadata delete requests.

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.

Security

HIP Registry provides the following security features:

- TLS or Trusted Node Authentication
- Patient Privacy Policy enforcement
- User Authentication through XUA
- Auditing

Chapter 3, Security Configuration provides more information about the security features.

Business Continuance

The following information on business continuance is applicable only for 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

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 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.

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

A Usage Reporting web application provides the ability to view the monthly reports and to generate adhoc reports. The monthly reports are stored in a library called UsageReports in xDB. However, adhoc reports that users generate are not stored in any location; you can only 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 60 provides information on configuring the Usage Reporting properties.

When you log in to Usage Reporting, you can see a list of monthly reports that are generated. You can click a report to view the 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 machine 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 **Generate Report Now** button enables you to create adhoc reports that shows the usage details from the day of product purchase.

The name of the reports are same as their IDs. The report ID consists of the timestamp when the reports are generated. That is, the ID of a report is of the format yyyyMMddHHmmss.

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

Chapter 3

Security Configuration

This chapter contains the following topics:

Access Control Settings

Authentication Configuration

Trusted Node Authentication

XDS Registry Server supports Trusted Node Authentication using TLS certificates.

HIP provides system security through SSL/TLS standards. System security provides access to HIP systems through data encryption and data confidentiality. TLS performs mutual authentication of client and server systems and encryption of data through certificates issued by the assigning authority.

The TLS keystore and truststore credentials are configured in the `registry.properties` file. These credentials do not have any default value.

XDS Registry requires you to configure the connection credentials to the xDB database where the healthcare information is stored. The connection credentials do not have default values.

XDS Registry Server SOAP requests can be configured to be accessed through HTTP (non-secure), which allows these requests to be sent to the HIP XDS Registry Server without any trusted node authentication. However, this does not disable XUA. Enabling Trusted Node Authentication is optional.

User Authentication

The user authentication settings control the process of verifying the identity claimed by a user for accessing the product.

HIP provides user security through Cross Enterprise User Assertion (XUA).

The system security authenticates systems; however, the server enterprise system is not aware of users defined on the client enterprise systems. XUA provides the ability to implement cross-enterprise user authentication to prevent data manipulation and provide data integrity. HIP systems provide XUA through WS-Security and WS-Trust standards.

The user authentication credentials are configured in the `registry.properties` file.

Trusted Host Access Configuration

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.

To configure the list of trusted hosts:

1. Go to `<HIP_HOME>/registry`.

2. Open `registry-context-extension.xml`.
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>
```

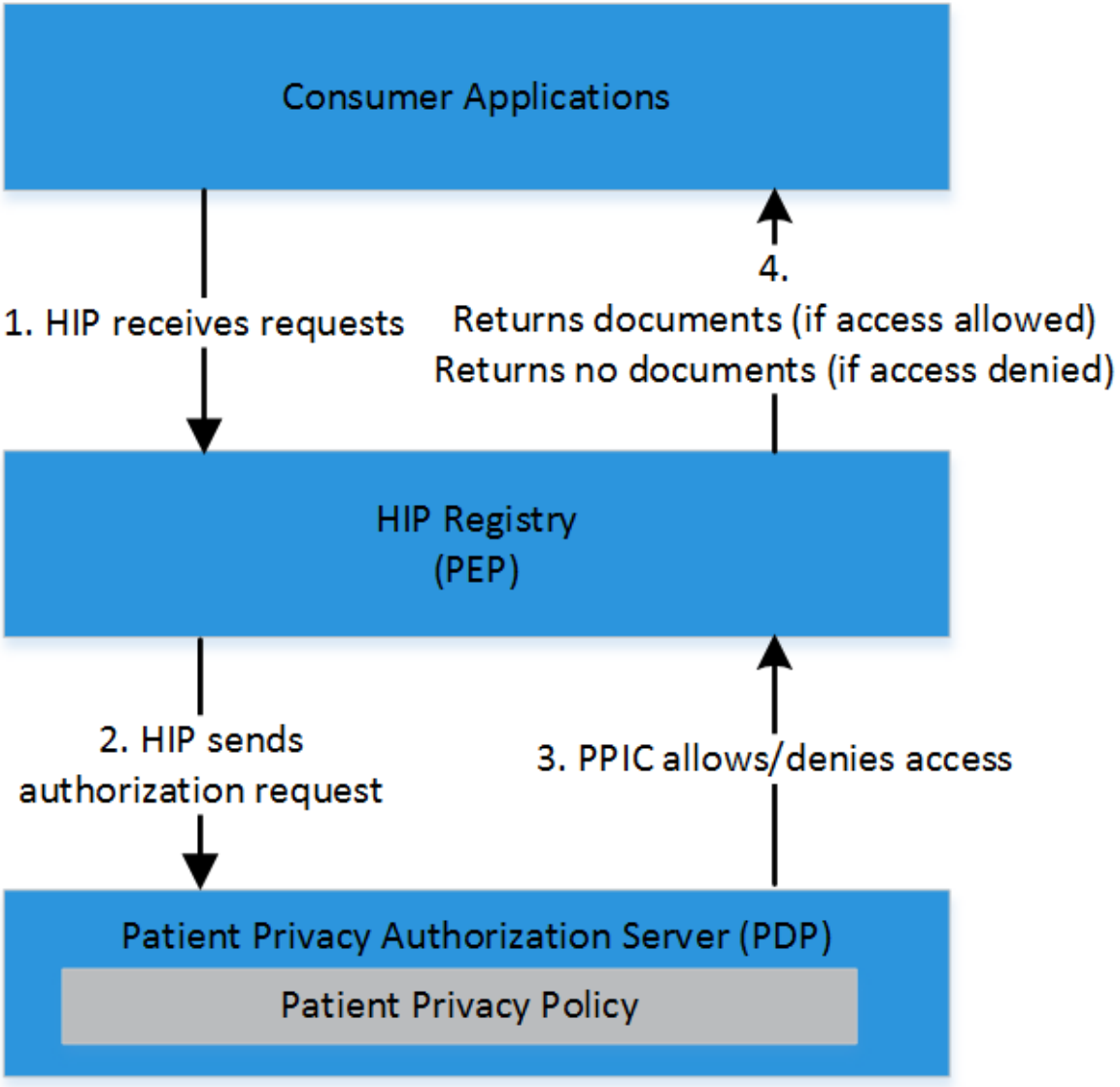
Configuring the IHE Endpoint for Trusted Hosts, page 56 provides the configuration information on enabling the IHE endpoint for trusted hosts.

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:



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 60 provides details about configuring the PPIC properties.

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

Communication Security Settings

The communication security settings enable the establishment of secure communication channels between the product components as well as between product components and external systems or components.

Port Usage

The HIP XDS Registry Server endpoint URLs are configured for each request type, which includes the optional port. No default port is provided.

XDS Registry Server creates audit messages for every request and response that it processes. The server prefixes audit messages with a unique audit source ID to easily identify the origin of the messages. The Audit Trail and Node Authentication Auditing (ATNA) properties in the `registry.properties` file provide the XDS Registry Server with connection information for an ATNA audit server.

The default port for ATNA audit server is 514 for ATNA SOAP requests.

The following table shows the ATNA port details:

Component	Service	Protocol	Port	Description
ATNA Audit Server	ITI-20 Audit Event.	SOAP	514	ATNA audit server

Network Encryption

The HIP Registry uses the HTTPS protocol for XDS Registry Server endpoints to achieve network encryption.

Data Security Settings

XDS Registry Server does not provide any data security settings, as all sensitive healthcare data is stored in the xDB database. The xDB database is responsible for securing this data internally.

Encryption of Data at Rest

XDS Registry Server data is stored in xDB and xDB is responsible for data encryption of the data at rest. The xDB database currently does not provide data encryption, but can be achieved by using encrypted file stores.

Secure Deployment Settings

You must follow the instructions given below to securely deploy the product:

- Use HTTPS for all network communications with the HIP XDS Registry Server SOAP requests.
- Configure TLS keystore and truststore credentials in the `registry.properties` file.
- Use XUA for user authentication with a Secure Token Server. Configure the STS server credentials in the `registry.properties` file.
- Use the HIP encryption password to encrypt sensitive user credentials.
- For the XDS Registry xDB healthcare database, follow the xDB recommended procedures to secure the xDB healthcare database.

Chapter 4

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 5

Installation

This chapter contains the following topics:

Pre-Installation Tasks

Before deploying XDS Registry Server, ensure that you have prepared the installation environment with the following components:

Component	Task
XDS Registry Installation Files	Download the following files from EMC Download Center: • Files for XDS Registry deployment: `hip-registry-1.7.0.zip` • HIP password encryption tool: `hip-encryptpassword-1.7.0.zip`
PPIC Server	Install HIP PPIC Server 1.7, if PPIC is enabled for XDS Registry. *EMC Documentum PPIC Server Installation Guide* provides the instructions for installing, and configuring PPIC Server.
J2EE Web Application	Install either one of the following application servers: • Apache Tomcat 7 – 7.0.42 • Oracle WebLogic 12.1.1 or Oracle WebLogic 12.1.2

Component	Task
Firewall Access	Provide the XDS Registry Server with access outside the firewall for communicating with other healthcare community members. When configuring the XDS Registry, create a record of the ports you use and the direction you want to allow connectivity.
xDB xHive Installation Files	Install xDB 10.5.5 or xDB 10.5.8.
Apache Ant	Install Apache Ant 1.9.3.
JDK	Install JDK 1.7.x

EMC Documentum XDS Registry Release Notes provides a complete list of system requirements.

Installing Documentum xDB Healthcare Database

To set up the Documentum xDB healthcare database:

1. Install Documentum xDB on the machine 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. 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 Third-party Library Dependencies

You must install the following third-party library dependencies for successful deployment of the Registry Server WAR file:

- xDB
- Camel
- HAPI
- IHE

Obtaining the Library Dependencies

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 `<xDB_install_dir>` to `C:\jarfiles\xdb`.
   For example:
   `C:\jarfiles\xdb\lib`

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 `<local path>\apache-camel-2.12.1`.
6. Copy the `lib` folder containing the jar files to `C:\jarfiles\camel`.
   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 `hapi-dist-2.0-all.zip`.
4. Extract the zip file to a local path.
   For example,
   `C:\hapi-dist-2.0-all`
5. Go to `<local path>\hapi-dist-2.0-all` folder.
6. Copy the `lib` folder containing the jar files to `C:\jarfiles\hapi`.
   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 `C:\openhealthtools`.
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-p1.jar`

To verify the jar files:

1. Check if the checksum values of the downloaded jar files match with the checksum values mentioned in the following table for corresponding files:

Filename	Checksum Value
apache-camel-2.12.1.zip	8aae88dec9558606cfd1b4bd6aaa2438
org.openhealthtools.ihe_2.0.0.zip	ac7bee0ac7d0db9becf71d29690d45d3
org.openhealthtools.ihe.atna.auditor-2.0.0-p1.jar	33c0fbc631ab539d70dd0bb6fd343fee
hapi-dist-2.0-all.zip	ee574cb7b458ea934a45d71a3c5da5e2

Installing the Library Dependencies

1. Download `hip-registry-1.7.0.zip` from EMC Download Center.
2. Extract `hip-registry-1.7.0.zip` to a local folder.

   For example:

   `C:\hip-registry-1.7`

   You can find the `build.xml` file in the `C:\hip-registry-1.7` folder.
3. Go to the command prompt and navigate to the directory where `build.xml` is located.

   For example:

   `C:\hip-registry-1.7`
4. Run `build.xml` using the following command:

   `ant -f build.xml`
5. Type the complete path of Camel home directory when the script prompts you to enter the Camel home directory.

   For example:

   `C:\jarfiles\camel`
6. Type the complete path of HAPI home directory when the script prompts you to enter the HAPI home directory.

   For example:

   `C:\jarfiles\hapi`
7. Type the complete path of xDB home directory when the script prompts you to enter the 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 machine, copy all jar files from the xDB lib folder to a local folder and rename the local folder to xDB home directory.
8. Type the complete path of OHT home directory when the script prompts you to enter the OHT home directory.

   For example:

   `C:\jarfiles\ihe`

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

`C:\hip-registry-1.7\install`

You can find the `hip-registry-1.7.0.war` file in the `install` folder.

Creating the HIP Configuration Directory

The HIP servers maintain configuration information in a directory called `HIP configuration directory` 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 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. Windows removes the trailing dot.

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

After creating the HIP configuration directory, copy the properties files to the HIP configuration directory.

To copy the property files to the HIP directory:

1. Go to the `install` folder obtained after running the build command described in Step 4.

   For example:

   `C:\hip-registry-1.7\install`
2. Extract `hip-registry-1.7.0.war`.
3. Go to `\hip-registry-1.7.0\config\`.
4. Copy the `registry` folder to `C:\Users\username\.hip\`.

   The resulting path should be `<hip_config_dir>/registry`.

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

Chapter 6, Post-Installation Configuration describes the configuration of property files required for successful installation.

Deploying the HIP Registry WAR File on Windows

Before deploying the WAR file, ensure that you have completed all configuration tasks described in Chapter 6, Post-Installation Configuration.

HIP supports the following:

- Deploying the HIP Registry WAR File Using Tomcat, page 39
- Deploying the HIP Registry WAR File Using WebLogic, page 39

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, perform the following tasks:

- Set the following environment variable:

  ```
  Name:  DOMAIN_HOME
  ```

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

To deploy the war file using WebLogic:

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-1.7.0.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 a successful deployment.

Deploying the HIP Registry WAR File in Linux

1. Log in as **root** user.
2. Copy `hip-registry-1.7.0.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-7.0.53
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
```

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.

Chapter 6

Post-Installation Configuration

This chapter contains the following topics:

Configuring Registry Properties File

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

registry.properties, page 87 shows a sample of this file.

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.

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

Configuring the Registry Property

The following table shows the configuration of registry property:

Property	Description
description	This optional property specifies a description for XDS Registry Server. The description is used only for display purposes. This property has no default value. For example: `description=XDS Registry`

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/write nodes.

Configuring the HADR Properties, page 45 provides more details on HADR property configuration.

The following three xDB properties are mandatory properties. If you fail to configure these properties, the Registry will fail to start.

The following table shows the configuration of Documentum xDB properties:

Property	Description
xdb.libraryPath	This property specifies the location in the Documentum xDB database where 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. For example: `xdb.libraryPath=/registry`

Property	Description
xdb.cachePages	This property specifies the number of cache pages for the page server defined in `xdb.bootstrapFileName`. For example: `xdb.cachePages=0`
xdb.maximumPoolSize	This property specifies the maximum pool size for the page server defined in `xdb.bootstrapFileName`. For example: `xdb.maximumPoolSize = 20`

Configuring the HADR Properties

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

HADR properties have been 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 will continue to operate even when the Master xDB (write node) is down. Similarly, the write operation will continue 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/write node parameters are comma separated list that can include multiple backup server nodes. The backup read/write node parameters are optional.

The following table shows the configuration of **primary read node parameters**:

Property	Description
xdb.readNode.bootstrapFileName	This mandatory property specifies the connection to a dedicated page server that runs behind the specified TCP/IP port. A bootstrap specifies a connection to a federation. For example: `xdb.readNode.bootstrapFileName=xhive://localhost:1235`

Property	Description
xdb.readNode.databaseName	This mandatory property specifies the name of the Documentum xDB database. You created and configured this database in Installing Documentum xDB Healthcare Database, page 34. This property has no default value. For example: `xdb.readNode.databaseName=XDS Registry`
xdb.readNode.userName	This mandatory property specifies the username that the XDS Registry Server uses to access the Documentum xDB database. You created and configured this user account in Installing Documentum xDB Healthcare Database, page 34. This property has no default value. For example: `xdb.readNode.userName=HealthcareServer`

xdb.readNode.password	This mandatory property specifies the password of the user that the XDS Registry Server uses to access the Documentum xDB.
	
	This property 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. `Enter clear text password`>Password
	`HIP_ENCR_PASS=JxwGkf59eneCKgVhZljyEA==`
	After generating the encrypted password, the `hip.keystore` file will be generated in `C:\EncryptPassword\` folder.
	3. Rename the `hip.keystore` file to `hip_readNode.keystore`.
	You can give any valid name for the file when you rename. `hip_readNode.keystore` is an example.
	4. Copy the `hip_readNode.keystore` file from the `C:\EncryptPassword\` folder to `{com.emc.healthcare.home}/registry/`.
	5. Set `xdb.readNode.keystore` property as follows:

Property	Description

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

xdb.readNode.keystore	This property is required if the HIP encrypted password is configured. For example: `xdb.readNode.keystore=${com.emc.healthcare .home}/registry/hip_readNode.keystore`

The following table shows the configuration of **primary write node parameters** (for single node deployment, set these four values to blank):

Property	Description
xdb.writeNode.bootstrapFileName	This property specifies the bootstrap filename for the Write node. For example: `xdb.writeNode.bootstrapFileName=xhive:/ /localhost:1245`
xdb.writeNode.databaseName	This property specifies the database name for the Write node. For example: `xdb.writeNode.databaseName=Healthcare`
xdb.writeNode.userName	This property specifies the user name for the Write node. For example: `xdb.writeNode.userName=HealthcareServer`

Property	Description

xdb.writeNode.password	This property specifies 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. `Enter clear text password>`**`Password`**
	`HIP_ENCR_PASS=JxwGkf59eneCKgVhZljyEA==`
	After password encryption, the `hip.keystore` file will be 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 `xdb.writeNode.keystore` property as follows:
	`xdb.writeNode.keystore=${com.emc.healthcare.home}/registry/hip_writeNode.keystore`
	After encryption, in `registry.properties` file, the `xdb.writeNode.password.property` appears as follows:
	`xdb.writeNode.password=HIP_ENCR_PASS=JxwGkf59eneCKgVhZljyEA==`

xdb.writeNode.keystore	This property is required if the HIP encrypted password is configured. For example: `xdb.writeNode.keystore=${com.emc.healthcare.home}/registry/hip_writeNode.keystore`

The following table shows the configuration of **backup read node parameters** (optional):

xdb.backup.readNode.bootstrapFileName	This property specifies the bootstrap filename for the backup Write node. For example: `xdb.backup.readNode.bootstrapFileName=xhive://10.8.55.239:1245`

xdb.backup.readNode .databaseName	This property specifies the database name for the backup Read node. For example: `xdb.backup.readNode.databaseName=Healthcare`
xdb.backup.readNode.userName	This property specifies the user name for the backup Read node. For example: `xdb.backup.readNode.userName=HealthcareServer`

xdb.backup.readNode.password	This property specifies 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. `Enter clear text password`>Password
	
	`HIP_ENCR_PASS=JxwGkf59eneCKgVhZljyEA==`
	
	After generating the encrypted password, the `hip.keystore` file will be generated in `C:\EncryptPassword\` folder.
	
	3. Rename the `hip.keystore` file to `hip _bkupreadNode.keystore`.
	
	You can give any valid name for the file when you rename. `hip_bkupreadNode.keystore` is an example.
	
	4. Copy the `hip_bkupreadNode.keystore` file from the `C:\EncryptPassword\` folder to `{com.emc.healthcare.home}/registry/`.
	
	5. Set `xdb.backup.readNode.keystore` property as follows:
	
	`xdb.backup.readNode.keystore=${com.emc .healthcare.home}/registry/hip_bkupreadNode .keystore`

	After encryption, in `registry.properties` file, the `xdb.backup.readNode.password.property` appears as follows: `xdb.backup.readNode.password=HIP_ENCR_PASS =JxwGkf59eneCKgVhZljyEA==`
xdb.backup.readNode.keystore	This property specifies the location of the backup read node keystore file. This property is required if the HIP encrypted password is configured. `xdb.backup.readNode.keystore=${com.emc .healthcare.home}/registry/hip_bkupreadNode .keystore`

The following table shows the configuration of **backup write node parameters** (optional):

xdb.backup.writeNode .bootstrapFileName	This property specifies the bootstrap filename for the backup Write node. For example: `xdb.backup.writeNode.bootstrapFileName=xhive:/ /10.8.55.229:1235`
xdb.backup.writeNode .databaseName	This property specifies the database name for the backup Write node. For example: `xdb.backup.writeNode.databaseName=Healthcare`
xdb.backup.writeNode.userName	This property specifies the user name for the backup Write node. For example: `xdb.backup.writeNode.userName=HealthcareServer`

xdb.backup.writeNode.password	This property specifies the password for the backup Write node.
	
	For example:
	
	`xdb.backup.writeNode.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. `Enter clear text password`>Password
	
	`HIP_ENCR_PASS=JxwGkf59eneCKgVhZljyEA==`
	
	After generating the encrypted password, the `hip.keystore` file will be generated in `C:\EncryptPassword\` folder.
	
	3. Rename the `hip.keystore` file to `hip_bkupwriteNode.keystore`.
	
	You can give any valid name for the file when you rename. `hip_bkupwriteNode.keystore` is an example.
	
	4. Copy the `hip_bkupwriteNode.keystore` file from the `C:\EncryptPassword\` folder to `{com.emc.healthcare.home}/registry/`.
	
	5. Set `xdb.backup.writeNode.keystore` property as follows:
	
	`xdb.backup.writeNode.keystore=${com.emc.healthcare.home}/registry/hip_bkupwriteNode.keystore`
	After encryption, in `registry.properties` file, the `xdb.backup.writeNode.password.property` appears as follows:
	
	`xdb.backup.writeNode.password=HIP_ENCR_PASS=JxwGkf59eneCKgVhZljyEA==`

xdb.backup.writeNode.keystore	This property specifies the location of the backup write node keystore file. This property is required if the HIP encrypted password is configured. For example: `xdb.backup.writeNode.keystore=${com.emc .healthcare.home}/registry/hip_bkupwriteNode .keystore`

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 following table shows the Registry configuration file properties that enable the Registry to find and use the Registry configuration file:

Property	Description
config.autoImport	This property 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. For example: `config.autoImport=false`
config.document	This property specifies 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. For example: `config.document=${com.emc.healthcare.home} /registry/registry-config.xml` 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 following table shows the configuration of MLLP parameters that identify and control the ports where the Registry Server listens for patient identity feeds:

Property	Description
mllp.port	This property specifies the non-secure port used to listen to the ITI-8 Patient Identity Feed messages. When set to `0`, the Registry Server will not open a listening port. The default value is `0`. For example: `mllp.port=9182`
mllp.securePort	This property specifies the optional secure port used to listen to the ITI-8 Patient Identity Feed messages. When set to 0, the Registry Server will not open a listening port. The default value is `0`. When not set to 0, you must also configure the HTTPS properties in this file. For example: `mllp.securePort=9183` **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.

mllp.routeBuilderScriptSource	This property specifies the optional property that enables you to override 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.
	
	If this property is not defined, Registry Server uses the default value, which is to use the script included with the Registry Server installation in the classpath.
	
	For example:
	
	`mllp.routeBuilderScriptSource=classpath:com/emc/healthcare/xds/registry/commons/MllpRouteBuilder.groovy`
	
	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`

Property	Description
	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

The following table shows the configuration of SOAP parameter:

Property	Description

soap.routeBuilderScriptSource	This optional property specifies whether you want to override the default SOAP RouteBuilder script provided with the Registry Server installation.
	
	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.
	
	For example:
	
	`soap.routeBuilderScriptSource=classpath:com /emc/healthcare/xds/registry/commons /SoapRouteBuilder.groovy`
	
	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`
	
	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.

Property	Description
	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.

The following table shows the configuration of request and response validator properties:

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**	These optional properties specify whether to disable the incoming message validation for the specified IHE transaction. These flags are enabled by default. For example: `registry.iti18.requestValidator.enabled=true`

Property	Description
Response Validator Properties	
registry.iti18.responseValidator .enabled **registry.iti42.reponseValidator .enabled** **registry.iti51.responseValidator .enabled** **registry.iti61.responseValidator .enabled** **registry.iti62.responseValidator .enabled**	These optional properties specify whether to disable the outgoing message validation for the specified IHE transaction. These flags are enabled by default. For example: `registry.iti18.responseValidator.enabled=true`

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.

The following table shows the configuration to enable the IHE endpoint for trusted hosts:

Property	Description
registry.trusted.hosts.enabled	This property specifies whether to enable the IHE endpoint for trusted hosts. The default value is **`false`**. For example: `registry.trusted.hosts.enabled=false`

Configuring the HTTPS Properties

The HTTPS properties enable 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 following table shows the configuration of HTTPS properties. The first four HTTPS configurations are required if XUA is enabled.

Property	Description
https.keyStore	This optional property specifies the location of the keystore on the system where you keep the private SSL certificates for this machine. If you are not using HTTPS, comment this property by prefixing it with a pound sign (#). This property has no default value. For example: `https.keyStore==${com.emc.healthcare.home}/keystore.jks`
https.keyStorePassword	This optional property specifies the password to access the keystore. If you are not using HTTPS, comment this property by prefixing it with a pound sign (#). This property has no default value. For example: `https.keyStorePassword=xxxxxx`
https.trustStore	This optional property specifies the location of the truststore on the system where you keep SSL certificates of machines trusted in TSL connections. This is where you keep the certificates of Document Consumers and XDS Repositories. If you are not using HTTPS, comment this property by prefixing it with a pound sign (#) . This property has no default value. For example: `${com.emc.healthcare.home}/truststore.jks`
https.trustStorePassword	This optional property specifies the password to access the truststore.If you are not using HTTPS, comment this property by prefixing it with a pound sign (#). This property has no default value. `https.trustStorePassword=xxxxxx`

Property	Description
https.ciphersuites	This optional property specifies the Cipher Suite used to encrypt the session. If you are not using HTTPS, comment this property by prefixing it with a pound sign (#). This property has no default value. For example: `https.ciphersuites=TLS_RSA_WITH_AES_128_CBC_SHA`
https.server.keyAlias	This property specifies the alias used for the server certificate in the keystore. If not specified, the first key read in the keystore is used. For example: `https.server.keyAlias=myservicekey`
https.server.privateKeyPassword	This property specifies the private key password for encrypting the data. For example: `https.server.privateKeyPassword=xxxxxx` **Note:** You must configure this property if MMLP secure port is enabled. The value of this property must be the private key password.

Configuring the ATNA Properties

The following table shows the configuration of ATNA properties:

Property	Description
audit.host	This property specifies the name of the machine that hosts the ATNA audit repository. For example: `audit.host=localhost`
audit.port	This property specifies the port number of the ATNA audit repository. The default value is **514**. For example: `audit.port=514`

Property	Description
audit.transport	This property specifies the transport type for the ATNA audit repository. For example, BSD, TLS, or UDP. The default value is **`UDP`**. For example: `audit.transport=UDP`
audit.sourceId	This property specifies the source ID of the event. This property has no default value. For example: `audit.sourceId=${description}`

Configuring the XUA Related Properties

The following table shows the configuration of XUA related properties:

Property	Description
registry.xua.enabled	This property specifies whether you want to enable XUA for the XDS Registry Server. The default value is **`false`**. For example: `registry.xua.enabled=false` After enabling XUA, you must configure the **xua** properties as described in Configuring the XUA Properties, page 66.

Configuring the PPIC Properties

The following table shows the configuration of PPIC properties:

Property	Description
ppic.enabled	This property 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. The default value is **`false`**. For example: `ppic.enabled=false` **Note:** You must configure the **ppic.pdpServiceUrl** property, if you enable this property.
ppic.pdpServiceUrl	This property specifies the URL for making PDP service call for PPIC. For example: `ppic.pdpServiceUrl=http://localhost:8080/ppic/pdp`

Configuring the Usage Report Properties

The following table shows the configuration of usage report properties:

Property	Description
usagereport.username	This property specifies the user name to log in to the Usage Reporting web application. For example: `usagereport.username=Administrator` The default username is Administrator
usagereport.password	This property specifies the password you need to type to log in to the Usage Reporting web application. For example: `usagereport.password=password` The default password is password.

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.

Configuring the Registry Configuration XML File

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.

The following table shows the configuration of `registry-config.xml` file:

Element	Description
strictAboutCodes	This property specifies how 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. For example: `<strictAboutCodes>true</strictAboutCodes>`

Element	Description
strictAboutPatientIds	This property specifies how 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. When set to **`True`**, the server rejects the register request. When set to **`False`**, the server accepts the request. For example: `<strictAboutPatientIds>true</strictAboutPatientIds>`
codeClassification	This property specifies 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. For example: `<codeClassification name="contentTypeCode" classificationScheme="urn:uuid:aa543740-bdda-424">`
assingingAuthority	This property specifies the assigning authority that provides IDs for patients and submits patient identities to the Registry. XDS Registry Server accepts the registration requests only from the assigning authorities that are defined here. For example: `<assigningAuthority id="1.19.6.24.109.42.1.3" /> <assigningAuthority id="1.20.7.25.90.22.7.1"/>`

Element	Description
Code	This property specifies a code and 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. For example: `<code code="Additional_Information" codeSystemName="Connect-a-thon associationDocumentation" displayName="Additional Information"/>`

Configuring the HIP PPIC Mapping Properties File

The following table shows the configuration of hip ppic mapping.properties:

Property	Description
documentEntry.patientId	This property specifies the XDS Affinity Domain Patient identifier. For example: `documentEntry.patientId =urn:oasis:names:tc:xacml:1.0:resource:patient-id`
documentEntry.typeCode	This property specifies the code of the precise kind of document (for example, Pulmonary History and Physical, Discharge Summary, Ultrasound Report). For example: `documentEntry.typeCode =urn:oasis:names:tc:xacml:1.0:resource:record-type`

Property	Description
documentEntry.classCode	This property specifies the high-level classification of documents that indicates the kind of document (for example, report, summary, note, consent.) For example: `documentEntry.classCode=urn:ihe:iti:xds-b:2007:document-entry:class-code`
documentEntry .healthcareFacilityCode	This property specifies the type of organizational setting of the clinical encounter during which the documented act occurred. For example: `documentEntry.healthcareFacilityCode=urn:ihe:iti:xds-b:2007:document-entry:healthcare-facility-type-code`
documentEntry.confidentialityCode	This property specifies the level of confidentiality of the document. For example: `documentEntry.confidentialityCode=urn:ihe:iti:xds-b:2007:confidentiality-code`
documentEntry.homeCommunityId	This property specifies the globally unique identifier for a community. For example: `documentEntry.homeCommunityId=urn:ihe:iti:xds-b:2007:home-community-id`
documentEntry.eventCode	This property specifies the main clinical acts, such as a colonoscopy or an appendectomy, being documented. For example: `documentEntry.eventCode=urn:ihe:iti:xds-b:2007:document-entry:event-code`
documentEntry.practiceSettingCode	This property specifies the clinical specialty where the act that resulted in the document was performed (for example, Family Practice, Laboratory, Radiology). For example: `documentEntry.practiceSettingCode=urn:ihe:iti:xds-b:2007:document-entry:practice-setting-code`

Property	Description
request.subjectId	This property specifies the logical identifier of the user performing the original service request. For example: `request.subjectId=urn:oasis:names:tc:xacml:1.0:subject:subject-id`
request.subjectRole	This property specifies the relevant user subject roles from a locally defined Code-Set. For example: `request.subjectRole=urn:oasis:names:tc:xacml:2.0:subject:role`

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 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 SSL for Tomcat

To configure Tomcat for SSL, add the paths for the keystores and truststores to 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. Ensure that SSL port enabled for each server.
   You can find the setting **SSL Listen Port Enabled** under **Configuration -> General** for each server. Enable this setting and make sure you use a correct and unused port.
3. For each server, change the KeyStore configuration as follows:
   a. Go to **Configuration -> Keystores**.
   b. In the **Keystores** section, click **Change**.
   c. 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. Go to **Configuration -> SSL**.
   b. Type the values for the following fields as given in the example below:
   Private Key Alias=s`erverXX`
   Private Key Passphrase=`changeit`
5. Click **Save**.

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 are using both the HIP XDS Repository and HIP XDS Registry Servers, you must enable XUA separately for each component.

For the HIP XDS Registry Server, open `registry.properties` and set the `registry.xua .enabled` property to **`true`**.

The default value is false.

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

- Configuring the XUA Policy, page 67
- Configuring the XUA SAML Attribute Values, page 67

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

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

The following table shows the configuration of XUA SAML attribute values:

Property	Description
xua.service.endpoint	This property specifies the endpoint regular expression. XDS Registry Server compares this value against the audience restriction attribute provided in the token. For example: `xua.service.endpoint=http://localhost:(\\d)*/hip-webapps-xua-registry-1.6/`
xua.crypto.provider	This property specifies the crypto provider to be used for encryption/decryption and signature validation. For example: `xua.crypto.provider=org.apache.ws.security.components.crypto.Merlin`
xua.supported.authentication.methods	This property provides a comma-separated list of authentication methods supported by the XDS Registry. For example: `xua.supported.authentication.methods=urn:oasis:names:tc:SAML:2.0:ac:classes:X509`

Property	Description
xua.purposeOfUse .codeSystem	This property specifies the supported system for the PurposeOfUseCode attribute. For example: `xua.purposeOfUse.codeSystem=1.3.6.1.4.1.21367.3000 .4.1`
xua.purposeOfUse.code.values	This property provides a comma-separated list of purpose of use codes supported by the XDS Registry. For example: `xua.purposeOfUse.code.values=99-101,99-102`
xua.role.codeSystem	This property specifies the supported code system value for the RoleCode attribute. For example: `xua.role.codeSystem=2.16.840.1.113883.6.96`
xua.role.code.values	This property provides a comma-separated list of roles supported by XDS Registry. For example: `xua.role.code.values=112247003,22515006`
xua.saml2.token.validator	This property specifies the validator that validates the SAML token in the request. For example: `xua.saml2.token.validator=com.emc.Healthcare.xua .validator.XuaValidator`

Configuring the XUA Attribute Validation Property

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

The following table shows the configuration of XUA attribute validation property:

Property	Description
xua.authz.consent.option	This property enables or disables the validation of the patient consent attribute value of SAML token. The default value is **`false`**. For example: `xua.authz.consent.option=false`

Configuring the Trusted Assertion Provider Properties

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

The following table shows the configuration of trusted assertion provider properties:

Property	Description
xua.assertion.provider.trustStore	This property points to a truststore file. For example: `truststore.jks`
xua.assertion.provider.trustStorePassword	This property value is the truststore password.

Chapter 7

Verifying the Installation

This chapter contains the following topics:

Verifying the Installation Using Tomcat

1. Ensure that the library dependencies are installed.

   Installing Third-party Library Dependencies, page 34 provides information on installing library dependencies.

2. Ensure that the `.hip` folder is available in `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 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-7.0.25/logs/catalina.out
   ```

6. 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.

Verifying the Installation Using WebLogic

1. Log in to WebLogic Admin console.
2. Ensure that the library dependencies are installed.
   Installing Third-party Library Dependencies, page 34 provides information on installing library dependencies.
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 above changes to take effect.
6. Start the WebLogic server.
7. Deploy the registry WAR file.
   Deploying the HIP Registry WAR File Using WebLogic, page 39 provides more details.
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

Upgrade

This chapter contains the following topic:

Upgrading XDS Registry from 1.6B250714 _update to 1.7

1. Delete previous version of Registry WAR file from the deployed location.
2. Build new Registry WAR from the `hip-registry-1.7.0.zip` file.
3. Go to the `\registry\config\` folder in the WAR file.
4. Copy the `registry` folder containing the properties file to HIP_HOME directory.
5. Configure the properties file in the HIP_HOME directory.
6. Deploy the `hip-registry-1.7.0.war` file.
7. Verify the upgrade.

Chapter 9

Troubleshooting

This chapter contains the following topics:

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 `<hip_config_directory>\logs`.

For Apache Tomcat:

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

For Oracle WebLogic:

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

Log Management and Retrieval

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

`<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 topic describes the following XDS Registry errors and their solutions:

Context Initialization Failing when Deploying the Server WAR Files

Issue with HIP Configuration

Problem

When you try to install the Registry WAR files, you get 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%`.

Creating the HIP Configuration Directory, page 37 and Deploying the Property Files in the HIP Configuration Directory, page 38 provide more details on HIP configuration.

Issue with Camel Jar Files

Problem

When you try to install the Registry WAR files, you get 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 Third-party Library Dependencies, page 34 provides the steps to install camel library dependencies.

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 by performing the steps in Chapter 7, Verifying the Installation.
- Ensure that the TLS certificates are valid and accessible.
- If you are using XUA, ensure that the XUA configurations are correct.

  Configuring the XUA Related Properties, page 59 provides details on XUA configuration.

Cannot Access the xDB Server

Problem

You get 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 that the xDB bootstrap file name is correct.
- Ensure that the xDB username and password are correct.

Configuring the Documentum xDB Properties, page 44 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 as described in Deploying the HIP Registry WAR File on Windows, page 39.

XUA Policy File Error

Problem

You get 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

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

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

If you are using 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

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 get `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

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

XDS Registry Server is unable to connect to the xDB healthcare database and you get 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

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.

The topic Configuring the Documentum xDB Properties, page 44 provides details 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 if you are using secure MLLP port.

Resolution

If you are trying to use secure MLLP port, you must include 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 get 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 the following sample files:

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 teh 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 tis property is not set, the default is used, which loads the
# MllpRouteBuilder.groovy source from a resoruce 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. Thie 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 equivilant 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 likelyhood 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 properites 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=
```

registry-config.xml

This sample has been edited for length. 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

S

T

U

V

W

X