PHINMS Technical Reference Guide

Manual Procedures for Technical Users

Prepared by
U.S. Department of Health & Human Services
March 2005

Table of Contents

Introduction

This guide is a supplement to the five part documentation system that you need to you get, install, configure and run the Public Health Information Network Messaging System. This guide contains procedures you can perform manually, outside the graphical user interface. Use this guide along with the other parts of the documentation system which include:

1. **Quick How Tos**: Read these Quick How Tos first. They give a streamlined list and some details of the things you need to do to get, install and run the PHINMS software. Go to the PHINMS web site at www.cdc.gov/phn/phinms, click the PHINMS Installation link and then click on the Quick How to link. The steps are listed in order. Read these steps before you use the other four parts of the documentation system.

2. **Release Notes**: After reading the Quick How tos, go to the Release Notes and Installation Guides section on the PHINMS website at www.cdc.gov/phin/phinms, click the PHINMS Support link, then click on the Release Notes and Installation Guides link. Read the Release Notes that go with the version of software you are installating.

3. **Installation Guide**: After reading the Release Notes, go to the Release Notes and Installation Guides section on the PHINMS website to get the Installation Guide. You should not use a copy of the Installation Guide that someone else has given you. The Installation Guide on the web site is continually updated. If you get it from the web site, you ensure you are getting the most up to date copy. To get the most up to date copy, go to the PHINMS web site at www.cdc.gov/phin/phinms, click the PHINMS Support link, then click on the Release Notes and Installation Guides link. Use the Installation guide that goes with the version of software you are installing. It features sample ebXML and Java files in the appendix along with queue schemas.

4. **Online Help:** Use the PHINMS software online help along with the Installation Guide. The online help gives you screen shots and step-by-step instructions for configuring and using the software. Go to www.cdc.gov/phin/phinms, click the PHINMS Support link, then click on the PHINMS Software Online Help link. The online help launches in a new browser. Use the search feature or the Contents navigation to find the procedures you want.

5. **FAQs:** Check the list of FAQs on the PHINMS web site (from the PHINMS Support link) to find answers to your questions and please suggest some of your own.

We welcome Your Comments

We're very interested in your comments. Please send us your comments about the PHINMS software, support and documentation to our PHINMS web site at www.cdc.gov/phin/phinms and use the Contact PHINMS email link at the top of the home page.

To Manually Create a CPA

Creating the CPA:

- Establishes communication requirements, such as transport protocols and security settings, with trading partners.
- Uses Object Ids (OIDS) to create a unique CPA file name.
- Completes the CPAs PartyInfo segments.
- Deploys the CPA.

To establish a message exchange relationship with the CDC send an email through the PHINMS web site at www.cdc.gov/phin/phinms. A CDC technical assistance representative will help you create a CDC CPA. Sending messages to the CDC requires more than the establishment of a CPA. The sending party's system administrator establishes an account with the CDC's Secure Data Network to receive an SDN digital certificate. For more information on SDN and acquiring SDN digital certificates see the Getting a Digital Certificate and Enrolling in a CDC Program guide on the PHINMS web site: www.cdc.gov/phin/phinms > PHINMS Support > Release Notes and Installation Guidelines > Documents.

After you create the file, deploy it on both the sending and receiving sides.

To Create a Unique CPA File

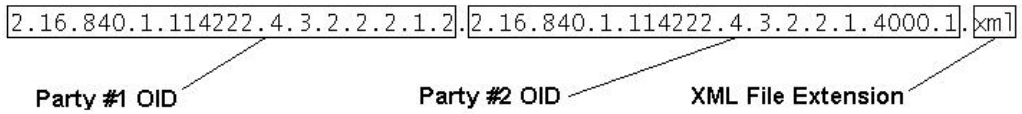
CDC CPA

If you are establishing communications with the CDC, the CDC technical assistant will create a unique CPA file for you. You are responsible for installing the CPA in your PHINMS or PHINMS compatible software. If you are establishing a trading relationship with a third party, both parties must work together to formalize the agreement.

CPA with a Third Party

After you have reached a verbal agreement with your trading partner, create a formal Collaboration Protocol Agreement. Every individual trading partner agreement requires the creation of a unique CPA and each unique CPA requires a unique file name. Standard operating procedure is to create a unique file name by concatenating unique identifiers of the trading partner. Use an object ID (OID) as a standard approach to establishing and maintaining unique party identifiers. The best way to get an OID is to request one from the CDC Help desk at the same time you request the software. Use the software request form at www.cdc.gov/phin/phinms > PHINMS Installation > Quick

How Tos. The following is an example CPA file name which is a concatenation of party one's OID and party two's OID with an XML file extension.



If you will send messages to the CDC you need to go through the PHINMS help desk as mentioned above. If you are sending messages to a trading party and will not use the CDC, you can create your own OIDS. Create a file with using the names of the respective organizations. Make sure the file has a unique name and that the name is relevant to both parties.

There is a sample CPA file, "SampleCPA.xml", in PHINMS's config/CPA folder. You can create a copy the file in the same config/CPA directory and follow the naming convention shown above. After you have created the unique file for these trading partner relationshipws update the file with your organization's information.

Complete PartyInfo segments of the CPA

CPAs are standard XML documents. You can use an XML editor or edit the file with a basic text editor such as Windows Notepad. Open the file and scroll down to the first PartyInfo entry in the file. Information pertaining to your organization is maintained in the PartyInfo tag. Information in the PartyInfo tag consists of identifying information, service location (URL), security and transport information. The following is a sample PartyInfo segment

```
<tp:PartyInfo>
  <tp:PartyId tp:type="DUNS">CDC</tp:PartyId>
  <tp:PartyRef xlink:href="http://www.cdc.gov/phin/messaging/index.html" />
  <tp:Transport tp:transportId="NO5">
        <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
        <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
        <tp:Endpoint tp:uri="www.cdc.gov/path/to/receiver" tp:type="allPurpose" >
        <tp:TransportSecurity>
                <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
                <tp:CertificateRef tp:certId="NO3" />
        </tp:TransportSecurity>
  </tp:Transport>
</tp:PartyInfo>
```

If you have established an OID for this PHINMS instance than the PartyId is the OID. You need to agree with the other party as to what unique identifier you are going to use.

Once you have completed your PartyInfo, your messaging trading partner must do the same. Send the CPA file to your trading partner and have them complete their PartyInfo segment. They should then send the completed CPA back to you. You should review the document to ensure your information has not been modified and ensure the trading partner's information is complete. Once you have reviewed the document for correctness and completeness, save the document in the PHINMS's config/CPA directory. Make sure the document maintains the xml extension.

Note: Your trading partner must also deploy the CPA on their respective ebXML endpoint. If they do not deploy the CPA you will not be able to send or receive messages to or from the partner.

Use the online help to configure the CPA. See www.cdc.gov/phin/phinms > PHINMS Support.

Configuring the Message Sender

To configure the Message Sender, the client, do the following::

1. Specify sensitive passwords in an XML password file in XML format. See the appendix for an example.

2. Using the **pbe.bat** utility described in this document, encrypt the password file in step 1 with a password that has at least eight characters, which are mixed case and a combination of alpha and numeric characters.

3. Set the polling configuration.

4. Create a CPA file for each of the routes to which you want to send messages.

5. Specify the end-point and security attributes within the CPA. See the appendix for an example.

6. Create a **routemap** entry for each CPA.

7. If you are using Secure Data Network, set up your client to perform SDN. Specify the SDN authentication properties in the CPA. See the appendix for an example.

To Use the Database Polling Configuration

To set up your client to use the database polling configuration, do the following:

1. In the **sender.xml** file, set **dbPoll** to **true**. Set **filePoll** to **false**.
2. Set up the Transport Queue database table.
3. In the **sender.xml** file, add the appropriate JDBC driver and the JDBC URL of the database that contains the Transport Queue database table.
4. Add the JDBC driver to the **lib** project directory.
5. Create the outgoing payload directory and then set the **dbPollDi**r to the full path name of this directory.
6. In **sender.xml** set the **databaseUser** and **databasePasswd** indices. Add these indices to your password XML file in XML format.
7. Set the other configurables in **sender.xml**.
8. Create a CPA file for every route you use to send messages.
9. Specify the end point and security attributes in the CPA.
10. Create a **routeMap** entry for each CPA.
11. If you are using Secure Data Network, set up your client to perform SDN. Specify the SDN authentication properties in the CPA. See the appendix A for an example.

To Use File- Based Polling

To set up your client to use the file-based polling configuration, do the following:

1. In the **sender.xml** file, set **filePoll** to **true** and set **dbPoll** to **false**.
2. Specify the **fileDescriptorFormat** as **XML** or **nameValue** and specify the **fileDescriptorDir**.
3. Create a file poll directory and specify it in the **sender.xml** file.
4. Create a CPA file for every route you use to send messages.
5. Specify the end point and security attributes in the CPA.
6. Create a **routeMap** entry for every CPA.

Message Sender Configuration File

The Message Sender reads its configuration file, **sender.xml**. The following table lists the **sender.xml** fields and their descriptions.

Field Name	Description
filePoll	If the value is **True**, the file system is polled for file descriptors.
dbPoll	If the value is **True**, the Transport Queue database table is polled for outgoing messages.
routeNotReadPoll	If the value is **True**, a route-not-read server is polled automatically at specified intervals for incoming messages.
test	If the value is **True**, a ping message is sent to CDC once every minute.
passwordFile	Full path name of the encrypted passwords file.
maxAttempts	Maximum number of times a message delivery is attempted.
filePollMode	Values are **loop** or **once**. In loop mode, files are continuously polled. In once mode, files are polled once.
filePollInterval	Number of seconds between file-based polls.
fileDescriptorDir	When the sending application uses file-based polling, it is the directory in which the Message Sender drops the file descriptors.
fileDescriptorFormat	File descriptor format: **XML** or **nameValue**.
dbPollMode	Database polling mode: **loop** or **once**.
dbPollDir	When the sending application uses the database polling mode, it is the directory containing outgoing payload files.
incomingDir	Directory to which incoming payload files are written.
dbPollInterval	Number of seconds between database polls.
dataReadTimeout	Maximum number of seconds the client, the Message Sender, waits to receive a response from the Message Receiver before timing out.
myPartyId	Message Sender's party ID.
routeMap	Full path name of the **Routemap** file.
cpaLocation	Full path name of the directory where Collaboration Protocol Agreements are stored.
connectionTimeout	Number of milliseconds the Message Sender waits before timing out the SSL connection attempt.
trustedCerts	Full path name of the certificate store, which contains the trusted certificates of authority.
trustedCertsPasswd	Name of the password, which enables access to the password store. The password store contains the passwords that enable access to the trusted certificates file.
keyStore	Full path name of the Message Sender's keystore.
delayedRetry	When true, retries to send failed records.

Field Name	Description
delayedRetryInterval	When set to true, indicates the amount of time between sending failed records.
maxDelayedRetries	Maximum number of times to send a failed record.
useWebProxy	When true, the HTTP requests from the PHINMS client are sent through a proxy server.
proxyHost	Host name of the proxy server.
proxyPort	Proxy server port.
ProxyUser	Name of the proxy user. This field, which is used for authenticating to the proxy server, references an entry in the passwords file, which contains the proxy password.
proxyPasswd	This field, which is used for authenticating to the proxy server, references an entry in the passwords file, which contains the proxy password.
useLdapProxy	When true, LDAP requests, in the form of an ebxml message, are sent to an LDAP request handler.
ldapProxyRoute	Route used when proxy server sends the LDAP request.
ldapProxyService	ebXML service that proxies the LDAP request.
ldapProxyAction	EbXML action performed when proxying the LDAP request
keyStorePasswd	Name of the password which enables access to the password store. The password store contains the password that enables access to the keystore file.
logLevel	Logging level: **none**, **error**, **info**, **detail**, **messages**.
logDir	Directory where logging is done.
logArchive	If true, the log files are archived when they reach their maximum size
maxLogSize	Maximum log size in bytes.
processedDir	Full path name of the directory to which processed file descriptors are written.
jdbcDriver	Name of the JDBC Driver the Transport Queue database table uses.
databaseUrl	JDBC URL of the database that contains the Transport Queue database table.
messageTable	Name of the Transport Queue database table.
databaseUser	Name of the database user within the password store.
databasePasswd	Name of the database password within the password store.
encryptionTemplate	Full path name of the XML encryption template.
signatureTemplate	Full path name of the XML encryption template.
dbType	Database type. Values are: **oracle**, **sqlserver**, **access, mysql**. Access is the default.
responseToDb	When this field is set to **true**, the response file is written to the **responseContent** field in the Transport Queue instead of writing the content to

Field Name	Description
	disk. The default is set to **false**.
ldapKeyRetrieval	Specifies whether LDAP search should be used to retrieve the public key of the recipient for encryption. If set to true, LDAP is used. Otherwise, the **certificateURL** field in the Transport Queue is read for the URL of the recipient's certificate. The default is true.
ldapCache	If true, keys retrieved using an LDAP search are cached for efficiency.
ldapCacheTimeoutHours	Lifetime, in hours, of an LDAP cache entry.
ldapCachePath	Path of the LDAP cache, relative to the installation directory.
syncReply	When this field is set to true, a synchronous reply is requested from the Message Receiver. The default is true.
ackRequested	When this field is set to true, an acknowledgement is requested from the Message Receiver. The default is true.
signedAck	When this field is set to true a signed acknowledgement is requested from the Message Receiver. The default is false.
monitorTimerInterval	Amount of time between monitor refreshes.
service	If true, the client is configured as a service.
serviceKey	The key used to decrypt the seed, which obtains the password to the encrypted password file. The **serviceKey** is one of the inputs in the **substitute.bat** utility. The serviceKey is used only if service=true.
serviceSeed	The ciphertext obtained when the password to the password file is encrypted using the **substitute.bat** utility.
queueMap	Path, relative to the install directory, of the client-side queue map. This file defines the client-side worker queues.
serviceMap	Path, relative to the install directory, of the client-side service map. This file maps the arguments in the route-not-read poll responses to the worker queues.
XML Segment or Sub-Segment	**Description**
pollList	List of poll destinations.
pollList.destination	A **pollList** destination.
pollList.destination.route	Route name of the poll destination, as it appears in the **routemap**.
pollList.destination.service	Service used at the poll destination.
pollList.destination.action	Action used at the poll destination.
pollList.destination.recipient	Message Recipient for which the Message Sender polls the poll destination.
pollList.destination.pollInterval	Interval, in seconds, between polls.
databasePool	Pool of database connections, which are needed

Field Name	Description
	for client-side worker queues.
databasePool.database	A database pool block.
databasePool.database.databaseId	Database identifier - can be any text string with no spaces.
databasePool.database.dbType	Database type - sqlserver, oracle, or access.
databasePool.database.poolSize	Maximum number of simultaneous connections to the database.
databasePool.database.jdbcDriver	Name of the JDBC driver
databasePool.database.databaseUrl	URL of the database.
databasePool.database.databaseUser	Tag name of the database user name that is stored in the encrypted password file.
databasePool.database.databasePasswd	Tag name of the database password that is stored in the encrypted password file.

Encrypted Passwords File

The Message Sender and the Message Receiver use the encrypted password file to store sensitive passwords. The following is an example of the plain text version of the password file:

```
<?xml version="1.0"?>
<passwordFile>
      <cacertsPasswd1asfasdfkjlkjl</cacertsPasswd1>
      <sdnPassword1>mysdnpassword</sdnPassword1>
      <keyStorePasswd1>mykeystorepasswd</keyStorePasswd1>
      <dbUser1>scott</dbUser1>
      <dbPasswd1>tiger</dbPasswd1>
</passwordFile>
```

Entries in this file are referenced from other configuration files. For example, the CPA might have an entry as such:

```
<sdnPassword>sdnPassword1</sdnPassword>
```

The password XML file is encrypted using password based encryption, and the encrypted file is a resource for the Message Sender. A utility **cmds\pbe.bat** is provided with this library that can be used to encrypt and decrypt the passwords file. On startup of the Message Sender program, the user is prompted for the password that was used to encrypt this file. The decrypted file contents are stored in memory, not written to disk. The resulting in-memory password-map is used throughout the uptime of the Message Sender.

Example of the RouteMap File

The RouteMap maps routes to the Collaboration Agreement Protocol files. The RouteMap uses the following format:

```
<RouteMap>
	<Route>
		<Name>OKLAHOMA</Name>
		<Cpa>CPA_cdc_oklahoma</Cpa>
	</Route>
	<Route>
		<Name>NEBRASKA</Name>
		<Cpa>CPA_cdc_nebraska</Cpa>
	</Route>
</RouteMap>
```

Installing Java Trust Store

Java Trust Store is part of the Java Virtual Machine. In version JRE1.4 the file is **\jre1.4\liv\security\cacerts**. This file contains a list of certificates and their serial numbers, which belong to trusted certificate authorities. You can use the Java Trust Store as it is, without modifying it. To add a CA certificate or view a certificate, use the following procedures:

To View the Contents of a Trust Store

1. Type the following command:
   keytool –list –v keystore <trustorefile> -storepass <storepass>

2. Substitute **javabin** for the path to the location Java binaries are stored on your machine. For example: **d:\jdk1.4\bin**

3. Substitute **trustorefile** for the file containing the Java Trust Store such as **cacerts**.

4. Substitute **storepass** for the password to the Java Trust Store.

To Add a Certificate to a Trust Store

1. Type the following command:
   keytool –import –truscacerts –file <cert_file> -storepass <storepass> -keystore <truststore>

2. Substitute **Javabin** for the path to the location Java binaries are stored on your machine such as **d:\jdk1.4\bin**

3. Substitute **trustorefile** for the name of the file containing Java Trust Store such as **cacerts**.

4. Substitute **storepass** for the password to the Java Trust Store.

To Manually Set up the Client as a Service

1. In the client **sender.xml** file, set the **service** value to **true**. Specify the s**erviceKey** and **serviceSeed** values. Use the **substitute.bat** utility to derive these from your password file password.

   For example, the **sender.xml** entries may look like the following:

   ```
   <service>true</service>
   <serviceKey>2o3i23</serviceKey>
   <serviceSeed>151209139182126100100162</serviceSeed>
   ```

2. If you are using the UNIX platform, add an **inittab** entry for **cmds\spawn.sh** or for **cmds\routenotread.sh**.

3. If you are using Windows NT/2000 platforms, download and install **JavaService**, a third party tool, from the following URL:

   www.alexandriasc.com/software/JavaService:

   There are other similar tools; however, the following instructions are specific to the **JavaService** *tool.*

4. Run the following command (with appropriate path changes) to install the spawner as a service:

   ```
   call (installdirectory)\cmds\setenv.bat

   javaservice -install "spawner" (javahome)\jre\bin\server\jvm.dll
   \
   -Djava.class.path=%CLASSPATH%
   -start gov.cdc.nedss.applications.spawner.Spawner
   -params (installdirectory)\config\sender.xml
   ```

5. Run the following command (with appropriate path changes) to install the RNR poller as a service:

   ```
   call (installdirectory)\cmds\setenv.bat

   javaservice -install "poller" (javahome)\jre\bin\server\jvm.dll \
   -Djava.class.path=%CLASSPATH%
   -start gov.cdc.nedss.applications.rnrpoller.RNRPoller
   -params (installdirectory)\config\sender.xml
   ```

Additions to the sender.xml Configuration File

The following fields in the **sender.xml** configuration file on the Message Sender, the client, support route-not-read automatic polling:

- **routeNotReadPoll** - If using automatic polling, set to true.
- **queueMap** - Path name of the client-side queue map, relative to the install directory.
- **serviceMap** - Path name of client-side service map, relative to the install directory.

Polling Destinations

The following example is a segment of the **sender.xml** file, which defines the polling destinations:

```
<Sender>
….
<pollList>
 <destination>
  <route>CDC</route>
  <service>Router</service>
  <action>autopoll</action>
  <recipient>nedoh</recipient>
  <pollInterval>30</pollInterval>
 </destination>
</pollList>
…
</Sender>
```

Destination Nodes

The **pollList** element in the **sender.xml** file segment contains a list of polling **destination nodes**, which can vary in length. Each destination node contains the following tags:

Tag	Description
route	Name of the route to poll.
service	Name of the service to use in the poll request message.
action	Action to use in the poll request message.
recipient	Identifies the message recipient, which is specified in the poll request message.
PollInterval	Number of seconds between poll requests.

Database Connection

The following segment of the **sender.xml** file defines the database connection, which is required by the worker and error queues:

```
<Sender>
…..
<DatabasePool>
  <database>
    <databaseId>sqlserver1</databaseId>
    <dbType>sqlserver</dbType>
    <poolSize>1</poolSize>
  <jdbcDriver>com.microsoft.jdbc.sqlserver.SQLServerDriver</jdbcDriver>
<databaseUrl>jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=Phmsg</datab
aseUrl>
    <databaseUser>sqlServerDbUserLocal</databaseUser>
    <databasePasswd>sqlServerDbPasswdLocal</databasePasswd>
  </database>
  <database>
    <databaseId>sqlserver2</databaseId>
    <dbType>sqlserver</dbType>
    <poolSize>1</poolSize>
    <jdbcDriver>com.microsoft.jdbc.sqlserver.SQLServerDriver</jdbcDriver>
<databaseUrl>jdbc:microsoft:sqlserver://nedsssql3.cdc.gov:1433;DatabaseName=Phms
g</databaseUrl>
    <databaseUser>sqlServerDbUserRemote</databaseUser>
    <databasePasswd>sqlServerDbPasswdRemote</databasePasswd>
  </database>
  <database>
    <databaseId>oracleserver1</databaseId>
    <dbType>oracle</dbType>
    <poolSize>1</poolSize>
    <jdbcDriver>oracle.jdbc.driver.OracleDriver</jdbcDriver>
    <databaseUrl>jdbc:oracle:thin:@nedss-report:1521:ebxml</databaseUrl>
    <databaseUser>oracleServerDbUserRemote</databaseUser>
    <databasePasswd>oracleServerDbPasswdRemote</databasePasswd>
  </database>
</DatabasePool>
…
</Sender>
```

Database Pool Attributes

The following table describes the attributes for each of the database entries in the database pool:

Attribute	Description
databaseId	Unique identifier for the database - defined by the user.
dbType	Specifies the database type. Values: **oracle** , **sqlserver**.
poolSize	Defines the number of connections that a specific database needs to have open at a given time. For high volume applications, a larger connection pool may be needed, such as **poolSize=10**. For low volume applications, a small connection pool may be sufficient, such as **poolSize=2**.
jdbcDriver	JDBC driver name for the database.
databaseUrl	URL of the database.
databaseUser	Index to the tag within the encrypted passwords file containing the user ID to the database.
databasePasswd	Index to the tag within the encrypted passwords file, which contains the password to the database.

Sender-Side QueueMap

The **QueueMap** is a definition file that resides on the ebXML client configuration folder. It maps worker queues to database/table combinations. Its structure is identical to the queue map on the ebXML receiver.

Example:

```
<QueueMap>
  <workerQueue>
        <queueId>QID123</queueId>
        <databaseId>sqlserver2</databaseId>
        <tableName>workerqueue</tableName>
  </workerQueue>
  <workerQueue>
        <queueId>QID456</queueId>
        <databaseId>sqlserver1</databaseId>
        <tableName>workerqueue</tableName>
  </workerQueue>
  <workerQueue>
        <queueId>QID789</queueId>
        <databaseId>oracleserver1</databaseId>
        <tableName>workerqueue</tableName>
  </workerQueue>
</QueueMap>
```

The **Queuemap** contains the definition of three queues IDs:

- QID123
- QID456
- QID789

Each **QueueId** maps to a database and a table name. The **tablename** corresponds to a table that conforms to the worker queue schema. The databaseIDs are defined within the **sender.xml** file.

See the appendix for the complete **sender.xml** file.

Sender-Side Servicemap

The **ClientServiceMap** is a configuration file that resides in the configuration file on the ebXML sender. It contains two types of entries, **service** and **errorqueue**. The following table lists the attributes for the service entry:

Attribute	Description
Arguments	This field will be matched against the arguments that are returned as part of a route-not-read response. If they match, then the response is written to the queue indicated in the QueueId of this Service entry.
QueueId	Unique identifier for each worker queue
payloadToDisk	If this is true, then payload response is written to the *incomingDir* folder as defined in *sender.xml*, and the name of the file is entered into the *localPayload* field in the worker queue. If this is false, then the response payload is written to the database.
textPayload	If this is true, then the payload is written to the text field in the worker queue.

Example:

```
<ClientServiceMap>
<Service>
        <arguments>asdf</arguments>
        <queueId>QID123</queueId>
        <payloadToDisk>false</payloadToDisk>
        <textPayload>true</textPayload>
</Service>
<ErrorQueue>
        <QueueId>QID000</QueueId>
</ErrorQueue>
</ClientServiceMap>
```

Manually Installing the PHINMS Receiver

To install the Message Receiver do the following procedures. Step-by-step instructions are included.

1. Install Java Runtime Environment.
2. Install Tomcat 5.0.19
3. Start and stop Tomcat.
4. Test Tomcat Installation.
5. Configure SSL for Tomcat.
6. Install and configure the Message Receiver.
7. Set up the Message Receiver Directory.
8. (Optional) Install the File-Based Message Handler.
9. Create a Service Map for the Message Handler.

To Install Java Runtime Environment

For Windows:

1. Copy the **j2re-1_4_0_03-windows-i586.exe** file from **<XXXXXX>** on the Public Health Messaging Software's CD into a temporary location on disk. If the directory doesn't exist create it.
2. Double-click the installer's icon and follow the instructions.
3. Type **<installedPath>\java\j2re1.4.0_03** for the destination folder and then click **Next**.
4. Select **Microsoft Internet Explorer Java Plug-in (optional)** and then click **Next.**

5. **Note** You must have administrative permissions to install the Java 2 JRE on Microsoft Windows 2000 and XP

6. When you are finished with the installation you can delete the download file to recover disk space.

For Linux:

1. Copy the **j2re-1_4_0_03-linux-i586.bin** file from **<XXXXXX>** on the Public Health Messaging Software's CD to **<installedPath>/java** on disk. If the directory doesn't exist create it.
2. Type the following command to start the installer's script:

   ./<installedPath>/java/j2re-1_4_0_03-linux-i586.bin
3. Follow the instructions.
4. A message appears: "**Do you agree to the above license terms?**" Type **yes** and then press **Enter**.
5. When you are finished with the installation you can delete the download file to recover disk space.

Note All Java configurations are handles via environment and startup scripts, which eliminate potential conflicts with multiple versions of Java installed on the server.

Installing Tomcat 5.0.19

To obtain a list of approved application servers contact your PHINMS representative. For any additional questions about Tomcat configuration see the on-line documentation at: **http://jakarta.apache.org/tomcat/**

To Install Tomcat

To install Tomcat 5.0.19, do the following:

1. Copy the **jakarta-tomcat-5.0.19.zip** file from **<XXXXXX>** on the PHINMS software CD to **<installedPath>** location on disk. The directory structure **jakarta-tomcat-5.0.19** is be created when you expand the file.
2. Using **WinZip** or **PKZIP** extract **jakarta-tomcat-5.0.19.zip** into you <**installedPath**> directory.
3. For Windows, edit the **catalina.bat** file and for Linux edit the **catalina.sh** file in **<installedPath>\jakarta-tomcat-5.0.19\bin\.** Modify the following line to reference your installation of the Java Runtime Environment. If the entry doesn't exist, add it.

   For Windows: set JAVA_HOME=<installedPath>\java\j2re1.4.0_03

   For Linux: JAVA_HOME=<installedPath>/java/j2re1.4.0_03

4. The basic Tomcat 5.0.19 installation is now complete. Follow the steps in **To Start Tomcat.**

To Start Tomcat

Execute the following script to start Tomcat 5.0.19.

For Windows:

```
<installedPath>\jakarta-tomcat-5.0.19\bin\catalina.bat start
```

For Linux:

```
<installedPath>/jakarta-tomcat-5.0.19/bin/catalina.sh start
```

Follow the steps in **To Stop Tomcat**.

To Stop Tomcat

Execute the following script to stop Tomcat 4.0.

For Windows:

```
<installedPath>\jakarta-tomcat-5.0.19\bin\catalina.bat stop
```

For Linux:

```
<installedPath>/jakarta-tomcat-5.0.19/bin/catalina.sh stop
```

Follow the steps in **To Test Your Installation**.

To Test Your Tomcat Installation

Go to the following URL. If the page displays, your installation of Tomcat 5.0.19 was successful:

```
http://localhost:8080/
```

For more information about configuring and running Tomcat 5.0.19 go to the following web site:

http://jakarta.apache.org/tomcat/

Follow the steps in **To Configure SSL for Tomcat**.

To Configure SSL for Tomcat

To configure SSL support on Tomcat 5.0.19, do the following:

1. Execute the following command to create a certificate keystore:

   For Windows:

   <installedPath>\j2re1.4.0_03\bin\keytool -genkey -alias tomcat -keyalg RSA \
   -keystore <installedPath>\keystores\tomcat

   For Linux:

   <installedPath>\j2re1.4.0_03/bin/keytool -genkey -alias tomcat -keyalg RSA \
   -keystore <installedPath>\keystore\tomcat

2. To specify a different location or filename, add the **-keystore** parameter, followed by the complete pathname to your **keystore** file, to the **keytool** command in step 1.

3. To simplify keystore management, keep all keystores in the directory **<installedPath>/keystores.** Put this new location in the **server.xml** configuration file as described below. If the **–keystore** parameter is omitted, the keystore will be created under the JRE security directory.

   Note *:* Use the **keytool** utility to create a certificate only during testing. You need to get an official certificate from a certified certificate authority (CA) when you move your application to a production environment.

4. After executing the command in step 1, you will be prompted for the keystore password. The default password is **changeit.** All letters are lower case. You can create a custom password if you want. If you create a custom password, you need to specify the it in the **server.xml** configuration file as described later.

5. You will be prompted for general information about the certificate, such as company, contact name, and so on. This information is displayed to users who access a secure page in your application, so make sure the information is complete, clear and specific. Do not use abbreviations. Completely spell the names of cities, States and so on.

6. You will be prompted for the **key password**, the password specifically for this Certificate. (Other certificates may be stored in the same keystore file.) The **key password** must be the same as the **keystore** password.

   You now have a **keystore** file with a Certificate that can be used by your server.

> **Note** The certificate will be extracted and loaded into the sender's **TtrustedStore keystore**. See the Client Installation and Configuration guide for details.

7. Uncomment the **SSL HTTP/1.1 Connector** entry in **<installedPath>/conf/server.xml**.

```
<!--
    <Connector
className="org.apache.catalina.connector.http.HttpConnector"
               port="8443" minProcessors="5" maxProcessors="75"
               enableLookups="true"
      acceptCount="10" debug="0" scheme="https" secure="true">
      <Factory
className="org.apache.catalina.net.SSLServerSocketFactory"
               clientAuth="false" protocol="TLS"
               keystoreFile="<installedPath>/keystores/tomcat"/>
    </Connector>
    -->
```

8. Restart the Tomcat 5.0.19 server using the stop and start procedures previously listed.

9. Test the certificate installation by browsing the following URL: **https://localhost:8443/**

   Because you generated a test certificate without using a Certificate Authority known by the browser, a pop-up window warning appears which reads **Security Alert** and asks if you want to proceed. Accept this certificate to display the secure page.

   After you have received a valid certificate from a known Certificate Authority this warning will not be displayed. When the Tomcat 5.0.19 home page is displayed you have successfully installed SSL within Tomcat and the Tomcat installation is complete.

Installing the Message Receiver Servlet

The message receiver is a servlet that runs on a J2EE compliant application server. The message receiver receives the ebXML message and after processing the message

envelope, performs message decryption and signature verification. After verification is complete, the Message Receiver will either write the message to a queue or forward the message payload onto an appropriate message handler.

To Install and Configure the Message Receiver

To install and configure the Message Receiver do the following:

1. Copy the **ebxml.war** file from **<XXXXXX>** on the Public Health Messaging Software's CD to the **<installedPath>\jakarta-tomcat-5.0.19\webapp** directory. Tomcat 5.0.19 will automatically expand the war file in a newly created **ebxml** directory under **webapps**.

2. Edit the **web.xml** file in **<installedPath>\jakarta-tomcat-5.0.19\webapps\ ebxml\WEB-INF**. Modify the **<param-value>** to point to the message receiver's configuration file. This file is usually found in the **<installedPath>\config** directory.

```
   <servlet>
        <servlet-name>
            receiver
        </servlet-name>
        <servlet-class>
            gov.cdc.nedss.services.filerecv.ReceiveFileServlet
        </servlet-class>
        <init-param>
           <param-name>receiverConfig</param-name>
           <param-value>{installedPath}\config\receiver.xml</param-value>
        </init-param>
<load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
       <servlet-name>
            receiver
       </servlet-name>
       <url-pattern>
           /receiver
       </url-pattern>
    </servlet-mapping>
```

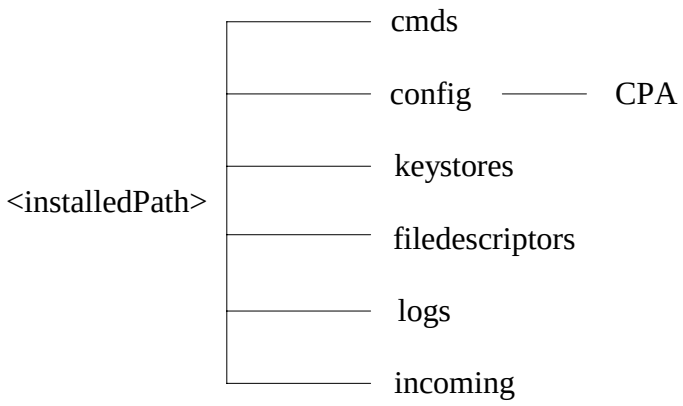
3. The **web.xml** file contains a servlet-mapping entry for the message receiver. The URL default value is **/receiver**. Update this value if you require a different URL mapping.

4. Follow the instructions in **To Set Up the Message Receiver Directory**.

> **Note:** The **receiver.xml** file is described in detail later in this document.

To Set Up the Message Receiver Directory

Use the following structure for the message receiver directory:



Directory	Description
cmds	Includes utilities required to support the operations of the system such as the password cipher and **pbe** encryption programs.
config	Includes all the configuration scripts. Also includes the CPA directory, which contains all the Collaborative Protocol Agreements.
keystores	Contains all the keystore files that support the encryption.
filedescriptors	Contains the file descriptors files used in file-based transfers.
logs	Contains the log files.
incoming	Contains the files that have been received in a file-based transfer.

Installing the File-Based Message Handler

You do not have to install the file-based Message Handler. It is optional. It is delivered with the Public Health Information Network Messaging System as an example Message Handle and you can use it to test the installation of the Message Receiver.

The file-based Message Handler receives the message from the Message Receiver and writes the payload to a file. The Message Handler should be installed on the same computer as the Message Receiver to eliminate an intruder's ability to eavesdrop on communication between them. Enforce firewall rules to protect the Message Handler from direct access from any internal or external computer.

To Install the File-Based Message Handler

To install and configure the Message Handler do the following:

1. Copy the **messagehandler.war** file from **<XXXXXX>** on the PHINMS software CD to the **<installedPath>\jakarta-tomcat-5.0.19\webapp** directory. Tomcat automatically expands the war file in a new **messagehandler** directory under **webapps**.

2. Edit the **web.xml** file in **<installedPath>jakarta-tomcat-5.0.19\webapps\messagehandler\WEB-IN**F. Modify the <**param-value**> to point to the Message Receiver's configuration file, which is usually in the **"<installedPath>\config**" directory.

```
   <servlet>
        <servlet-name>
            messagehandler
        </servlet-name>
        <servlet-class>
            gov.cdc.nedss.services.messagehandler.MessageHandler
        </servlet-class>
        <init-param>
           <param-name>receiverConfig</param-name>
           <param-value>{installPath}\config\receiver.xml</param-value>
        </init-param>
<load-on-startup>1</load-on-startup>
    </servlet>
```

The **web.xml** file contains a servlet-mapping entry for the Message Handler. The default value of the URL is **/messagehandler**. This URL will be used in the route map to route an incoming service/action to this Message Handler.

```
                <servlet-mapping>
                     <servlet-name>
                        messagehandler
                     </servlet-name>
                     <url-pattern>
                        /messagehandler
                     </url-pattern>
                 </servlet-mapping>
```

3. After installing and configuring the Message Handler, add an entry to the Message Receiver's **servicemap.xml** file to route messages to the Message Handler. Use the **To Create a Service Map for the Message Handler** procedures in the following section.

To Create a Service Map for the Message Handler

The **servicemap.xml** file maps an incoming service/action to specific Message Handlers. You need to add a service entry for the file-based Message Receiver. Do the following procedure to create a service map for the Message Handler servlet.

1. Add a unique name for your service using the <**Name**> tag. This name is used to log information in the log files.

2. Add a type for your service using the **<Type>** tag.

   The **Type** attribute tells the Message Receiver the type of Message Handler. The Message Receiver supports web-based Message Handlers such as servlets, JSPs, ASPs and CGI. The Message Receiver establishes a HTTP connection with the web Message Handler. For more details see the **Configuring Queues** section in this document.

3. Add an action name using the **<Name>** tag within the **<Action>** tag.

   The Message Receiver uses the <**Name**> tag in the **servicemap.xml** file to an action name to an associated action URL.

4. Add an URL using the <**Url**> tag within the **<Action>** tag.

   This tag defines the URL of the Message Handler that will be initiated when this Action is requested. In the following example a request with an action of **receiveFile** will be mapped to the Servlet's URL**, http://server:port/messagehandler.**

```
        <Service>
        <Name>FileReceive</Name>
        <Type>Servlet</Type>
        <Action>
              <Name>receiveFile</Name>
           <Url>http://server:port/messagehandler</Url>
           <Argument> </Argument>
        </Action>
    </Service>
```

5. Optionally you can add an argument to your **Action** using the **<Argument>** tag. It is used to pass name-value-based parameters to the Message Handler.

Message Receiver and File-Based Message Handler Configuration Files

Configurations for the Message Receiver and file-based Message Handler are maintained in five XML configuration files.

File Name	Description
web.xml	A standard Java configuration file for web modules. The Message Receiver servlet and servlet mapping entries may need modification to reflect specific environment settings such as install directory. Also, security constraint tags may need to be created if the web container security is implemented.
receiver.xml	The main configuration file for the Message Receiver. It contains references to application directories, settings, and certificates.
servicemap.xml	Maps incoming service requests with specified Message Handlers.
queuemap.xml	Maps a queue to a database and its associated queue table. Configuration of the **queuemap.xml** file is optional. It must be configured only when the Message Receiver is configured to write messages into queues. .
passwds.xml	Contains system passwords including passwords to access certificates in keystores. The **passwds.xml** file is encrypted before deployment on a production server. See the **Password Utilities** section in this document for more information on password encryption.

After configuring the software to accept and handle messages, configure the Collaboration Protocol Agreements (CPAs) for each partner that will send messages to the system.

Configuring the web.xml File

The **web.xml** file describes the Message Receiver's servlet attributes, mappings and, optionally, security constraints. The **web.xml** file is packaged within the **ebxml.war** file. In a Tomcat deployment, the file can be found under Tomcat's **webapps** directory in the expanded **/ebxml/WEB-INF** directory.

The deployed **web.xml** file contains a servlet entry for the Message Receiver. This entry maps a servlet name, **receivefile**, to the servlet class, **gov.cdc.nedss.services.filerecv.ReceiveFileServlet**. This is the actual Java class that will be instantiated by the web container. The Message Receiver servlet entry contains a parameter called **receiverConfig**. This parameter points to the Message Receivers' configuration file, **receiver.xml**.

```
<servlet>
     <servlet-name>
          receivefile
      </servlet-name>
     <servlet-class>
          gov.cdc.nedss.services.filerecv.ReceiveFileServlet
      </servlet-class>
      <init-param>
         <param-name>receiverConfig</param-name>
         <param-value>C:/tomcat4/ebxml/Config/receiver.xml</param-value>
      </init-param>
     <load-on-startup>2</load-on-startup>
 </servlet>
```

Servlet Mapping Entry

The **web.xml** file contains a servlet-mapping entry that maps a URL to a Message Receiver servlet. The server's end point for a message exchange will consist of protocol, server name, port and the URL entry in the **web.xml** servlet mapping. The default value is **receivefile**. Set the load on the startup parameter to a number greater than the Message Handler load on startup parameters. This allows servlet-based Message Handlers to initialize before the Message Receiver initializes.

```
<servlet-mapping>
<servlet-name>
receivefile
</servlet-name>
<url-pattern>
/receivefile
</url-pattern>
</servlet-mapping>
```

Security Constraints

Optionally, security constraints can be added to the **web.xml** file to support basic, form and certificate authentication. Ideally, use the web proxy layer to manage authorization. This level of abstraction ensures system integrity by isolating the Message Receiver (ebXML Receiver) layer. If you are unable to place the ebXML receiver software behind a web proxy and require configuration of authorization parameters at the application server layer, see Tomcat documentation to learn more about Tomcat authentication configuration.

Configuring the receiver.xml File

The **receiver.xml** file is the Message Receiver's main configuration file. This section covers the configuration of the following system settings:

- Deployment Directories
- System Logging Level
- Password File Cipher Key
- Location of CPAs
- Location of passwords file **
- Logging directory
- Queue Mappings
- Party Id of the receiver
- Location of servicemap.xml file.
- Keystore Information*
- Trusted Keystore Information*
- Domain Settings
- Timeout Settings
- Database Pools

Variables in the receiver.xml File

The following table describes the variables in the **receiver.xml** file:

Variable	Description
logDir	The directory in which the Message Receiver servlet writes log files. The directory must be an existing directory on the server. The default value is **<installPath>\logs**
logLevel	The logging level of messages within the log file. Supported logging levels include: **none**, **error**, **info**, **detail**, and **messages**. The amount of log information written to the file increases as the level moves from **none** to **messages**. At the **messages** level all possible logging messages are written including the contents of the messages. The default value is **info**.
maxLogSize	The maximum size of a log file. After the size limit is reached the process will stop writing to the log. The default value is **100 megs**.
logArchive	Values are **true** or **false**. If true, the process will archive log files when they reach their size limit and then start a new log file. The default value is **true**.
incomingDir	The system looks in this directory for incoming messages when the system is configured for file-based polling. If file-based polling is not being used the parameter can be left blank. The default value is **<installPath>\incoming**.
myPartyId	The Message Receiver's party Id. The value of the party Id must be the same as the Message Receiver's party Id in the CPAs. For example, the party Id for the Center for Disease Control is CDC. Make sure the party Id for the State or agency is unique.
cpaLocation	The directory to the CPAs. All CPAs must be retained in this directory. The directory can be any directory accessible by the servlet. The default directory is **<installPath>\config\CPA**.
serviceMap	The full path name to the service map. The service map contains mappings between actions and Message Handlers. The default directory and file name is **<installedPath>\config\servicemap.xml**.
passwordFile	Full path to the encrypted passwords file. The clear text

Variable	Description
	password file should not be deployed in a production environment. The default directory and file name is **<installedPath>\config\passwd**
keyStore	Full path of the Message Receiver's Java keystore. The key store maintains the Message Receiver's private key and associated certificate in PKCS12 format. This information is used to enable encryption between the Message Sender and the Message Receiver. See the Security section for more information on managing certificates and keystores.
keyStorePasswd	A pointer to the Message Receiver's Java Keystore password within the Message Receiver's encrypted password store. The value of this entry is not the keystore password itself. It is the name of the tag within the password file. The value of this tag within the passwords file contains the actual password. See appendix for configuration of the passwords file.
trustStore	Full path of the Message Receiver's trusted store. The trusted store maintains trusted public keys in JKS format for the sending parties that are using encryption and digital signatures. See the **Security** section for more information on the managing certificates and keystores.
trustStorePasswd	A pointer to the Message Receiver's trusted store password within the Message receiver's encrypted password store. The value of this entry is not the password to the trusted store itself; it is the name of the tag within the password file. The value of this tag within the passwords file contains the actual password. See appendix for configuration of the passwords file.
signatureRequired	Values are **true** or **false**. When set to true, the message must have a signature. If it doesn't have a signature the message will fail. When the variable is set to false, which is the default, signatures are verified when present and the messages without signatures are accepted.
signingCertsLocation	The directory in which signed certificates are stored. The directory can be any directory accessible by the servlet. The default directory is **<installPath>\config\CPA**. This directory is used when the server is communicating with a Message Sender that does not send signed certificates within digitally signed messages. In this case, the Message Receiver looks in the **signingCertsLocation** directory for the Message Sender's signed certificate. If a certificate is found, the Message Receiver uses the public key to decrypt the

Variable	Description
	signed hash. If the hash matches the hash computed on the Message Receiver side, the identity of the Message Sender is verified. You do not need to use the **signingCertsLocation** when communicating with the CDC's ebXML client software. The CDC's ebXML client software sends the client's certificate information in the **keyInfo** of the message. The Message Receiver uses this key to interrogate the computed hash.
queueMap	Full path to the queue map XML file. The queue map file maps a message to a queue by matching the message's argument to the queue name. The default directory and file name is **<installedPath>\config\queuemap.xml**
payloadToDb	Possible values are **true** or **false**. When true, the system places the payload in a BLOB field in a table. When false, the system writes the payload out to a file. The file will reside in the specified outgoing directory. The default value is **false**.
key	Substitution cipher key. The key and seed are used to mask the password store's password. Because the Message Receiver is a service, it is inefficient to require the system administrator to enter a password at every restart. This feature allows the system to access the encrypted passwords file without keeping any passwords in clear text on the production environment. See the appendix for more information about the password cipher functionality.
seed	Cipher text obtained by encrypting the password to the encrypted password store using the substitution cipher key. See the appendix for more information about the password cipher functionality.
usePersistentCache	When true, the file uses a persistent cache to detect duplicate messages.
applevelCaching	When true, **recordId** field is used to detect duplicates. When false, the **conversationId** is used to detect duplicates.
persistentCache.dbType	Type of database used by the persistent cache such as sqlserver or oracle.
persistentCache.jdbcDriver	JDBC driver for the persistent cache.

Variable	Description
persistentCache.databaseUrl	Databae URL for the persistent cache.
persistentCache.databaseUser	Database user name for the persisten cache.
persistentCache.tableName	Table name for the persistent cache.
persistentCache.cacheEntryAgeHours	Maxium age for a persistent cache entry.

Database Pooling

To enhance system performance, the Message Receiver supports database pooling. The system can support multiple database connection pools. The **receiver.xml** file stores connection pool information in its **<databasePool>** tag, which can contain one or more **<database>** tags. Each **<database>** tag represents a database pool. The following values must be set for each database tag.

Database Tag Values

Tag Value	Description
databaseId	The unique name for the database connection pool. The database Id is referenced in the queue map. The service map uses the **databaseId** to map the queue to a specific database.
dbType	Designates the type of database. These databases are supported: oracle, sqlserver, mysql, access.
poolSize	The number of database connections to open. When setting the pool size make sure the system can handle the maximum client load while keeping enough memory available.
jdbcDriver	The type of JDBC driver. The JDBC driver should be appropriate for the type of database such as **com.microsoft.jdbc.sqlserver.SQLServerDriver** for Microsoft SQL Server and **oracle.jdbc.OracleDriver** for Oracle.
databaseUrl	The URL to the database. The format of the URL depends on the type of database and driver used such as **jdbc:microsoft:sqlserver://host:portnumber;DatabaseName=database** for Microsoft SQL Server and **jdbc:oracle://host:port:sid** for Oracle.
databaseUser	A pointer to the database user entry in the Message Receiver's encrypted password store. The value is not the database user. It is the name of the tag within the password file. The value of the tag within the passwords file contains the actual database user name. See the appendix for the password file configuration for both the **databaseUser** and **databasePasswd** entries.

Tag Value	Description
databasePasswd	A pointer to the database password entry in the Message Receiver's encrypted password store. The value is not the database password. It is the name of the tag within the password file. The value of the tag within the passwords file contains the actual database password. See the appendix for the password file configuration for both the **databaseUser** and **databasePasswd** entries.

Configuring the servicemap.xml File

The **ServiceMap** file maps a Service and Action attribute pair to the URL of a Message Handler. The **ServiceMap** file contains the name of the service, the service type, the action name, the URL of the Message Handler and the arguments that must be sent to this URL if the Service and Action attribute pair are invoked.

```
<ServiceMap>
     <Service>
       <Name>Router</Name>
       <Type>Servlet</Type>
       <Action>
            <Name>send</Name>
            <Url>http://158.111.1.164:8080/router/router</Url>
            <Argument>action=send</Argument>
       </Action>
    </Service>
</ServiceMap>
```

The Message Sender specifies the Service and Action in the message. The Message Receiver retrieves the Service and Action from the request then uses the service map to find the URL of the Message Handler and any arguments that need to be sent to this URL.

> **Note:** Install the Message Handler and the Message Receiver on the same computer to eliminate an intruder's ability to eavesdrop on communication between them. Enforce firewall rules to prevent internal or external computers from directly accessing the Message Handler.

The **Argument** tag allows the programmer to send additional information to the Message Handler. Use the HTTP query string format for arguments such as **arg1=data1&arg2=data2**. Use URL encoding for special characters such as underscores and dashes.

Configuring the Service Map

Specify **WorkerQueue** as the service type for the service entry in the service map.

```
<ServiceMap>

      <Service>
          <Name>Servicename</Name>
          <Type>WorkerQueue</Type>
          <payloadToDisk>true/false</payloadToDisk>
          <textPayload>true/false</textPayload>
          <Action>
              <Name>Receive</Name>
              <QueueId>QID123</QueueId>
              <QueueId>QID456</QueueId>
               …
          </Action>
       </Service>
</ServiceMap>
```

The entry of type **WorkerQueue** maps to one or more queue IDs. The **QueueID**s in the WorkerQueue are defined within a **QueueMap** shown below. There are two additional fields in the service element for the WorkerQueue service type, **payloadToDisk** and **textPayload**.

When the **payloadToDisk** flag is set to **true**, the incoming payload is written to disk instead of to the database field. In this case the name of the local file on disk is stored in the worker queue table.

When the **textPayload** flag is set to **true**, the payload is written to the **payloadTextContent** field. When the **textPayload** flag is set to **false**, the payload is written to the **payloadBinaryContent** field in the worker queue.

Configuring the Queue Map

The queue map is a definition file that resides on the Message Receiver's configuration folder and is used to map worker queues to database/table combinations.

The following example queuemap contains the definition of three queue-Ids: QID123, QID456 and QID789. Each **QueueId** maps to a database and a **tablename**. The tablename corresponds to a table that conforms to the worker queue schema. The databaseIDs are defined in the **receiver.xml** file.

```
<QueueMap>
    <workerQueue>
        <queueId>QID123</queueId>
        <databaseId>sqlserver2</databaseId>
        <tableName>workerqueue</tableName>
    </workerQueue>
    <workerQueue>
        <queueId>QID456</queueId>
        <databaseId>sqlserver1</databaseId>
        <tableName>workerqueue</tableName>
    </workerQueue>
    <workerQueue>
        <queueId>QID789</queueId>
        <databaseId>oracleserver1</databaseId>
       <tableName>workerqueue</tableName>
    </workerQueue>
</QueueMap>
```

See the Receiver Configuration section for more information about configuring database pools.

Worker Queue Table Schema

The worker queue table schema and scripts to create the worker queue table are provided in the appendix.

To Configure the Queues

To configure the queues for the PHINMS do the following:

1. Create the worker queues.

   See the DDL and SQL Server scripts in the appendix.
2. Add the **DatabasePool** entries to the **servicemap**.
3. Add the **QueueMap.**
4. Add the **ServiceMap** entries for the queues.
5. Add the database user and password entries to the encrypted passwords file and reference them from the **DatabasePool** entries.
6. Add the database drivers, JDBC, to the ebXML Receiver **WEB-INF\lib** folder for SQL Server or Oracle.

Installing the Route-Not-Read Message Handler

See the apendix for an example of the **routerconfig.xml** file.

Security

Security is a very important aspect of the Public Health Information Network Messaging System. Managing security is vital to ensure the messages are confidential and their integrity is preserved.

Recommended Security Practices

To maintain the security and integrity of the messages and to safeguard The Message System use the following practices:

Subject	Recommended Practice
Passwords	Do not store passwords as plain text files. When you use a plain text password file to generate an encrypted password file, promptly delete the plain text file.
KeyStores and Trusted Certificates	When creating passwords to the keyStores and trusted CA certificate files, choose a password with at least eight characters, including a mixture of alpha and numeric and upper and lower case.
Digital Signatures	To prevent the message from being rejected because of its origin, use digital signatures. Digital signatures require client certificates and a Public Key Infrastructure, such as LDAP, to manage the client certificates.
File System Permissions	Use file system permissions to prevent unauthorized users from accessing the configuration files and to prevent unauthorized users from accessing the incoming and outgoing payload directories.

Use the following guidelines to preserve your message's integrity and secrecy:

- Run the Message Receiver on an application server that accepts HTTPS requests. Connect the web proxy to the server using HTTPS to reduce wiretapping attacks on the DMZ.

- Permit access to the Message Receiver from the web proxy only. Block all other direct accesses to the Message Receiver.

- Permit access to the Message Handler from the Message Receiver only. Block all other direct accesses to the Message Handler.

- Protect the service maps on the Message Receiver by allowing only authorized users to modify them..

- Do not store passwords as plain text files on the Message Sender or Message Receiver. Promptly delete the plain text files that you use to generate encrypted password files.
- Use digital signatures for non-repudiation of message origin. You need a client certificate and a public key infrastructure, such as LDAP, to manage the client certificates.
- Use a combination of at least eight numbers and letters for passwords to the key stores and trusted CA certificates. Each user should have a unique password.
- Use only Message Handler services that are within the Message Receiver's Intranet. The service map on the Message Receiver should not contain any URLs for message handler services that point outside the Intranet.
- Using file system permissions, allow administrators only to modify the service map.
- If wire-tapping is a risk within the Message Receiver's Intranet, put the Message Handler and the Message Receiver on the same host because communication between the Message Receiver and the Message Handler is not encrypted.
- Using file system permissions, allow only authorized users to modify the configuration files on the client and on the server.
- Using file system permissions, allow only authorized users to read and write incoming and outgoing payload (file) directories.

Manual Management of Passwords

As the administrator, you need to protect the secrecy of several configuration variables which include passwords, keystore passwords, login passwords and so on. All of the passwords are kept in an encrypted password map so that you do not have to remember or enter several passwords.

During startup, the system prompts the user for the password to the encrypted password map. During the client's runtime, the password map is stored in the client's memory. Other configuration files make references to this password map. You can use the utility **pbe.bat** to encrypt plain text password maps and obtain their cipher text.
For example, the password map in plain text looks similar to the following:

```
<passwordFile>
<loginUser>raja</loginUser>
<loginPasswd>Kailar</loginPasswd>
</passwordFile>
```

The sender configuration file, **sender.xml**, references the encrypted password map and also references **loginUser** and **LoginPasswd**. The password used to encrypt the password map is supplied at startup time.

Java Keystores

A **keystore** is a repository of keys and certificates. The KeyStore class is an engine class that supplies well-defined interfaces to access and modify the information in a keystore. You can have several implementations at the same time, with a different implementation for each type of keystore.

Sun Microsystems provides a default implementation. It implements the keystore as a file, which utilizes a proprietary keystore format called **JKS**. It protects each private key with an individual password and protects the integrity of the entire keystore with another password, which may be different than the individual password.

The JKS keystore type is used to store all trusted certificated within the **TrustedStore**. Use the **keytool** utility, described later in this section, to manage information within the JKS keystore type.

RSA Laboratories, along with other vendors such as Apple, DEC, Microsoft, and Sun create the Public Key Cryptography Standards, PKCS, specifications. The functions of the PKCS specifications vary and they address issues related to security and cryptography. The **PKCS12 Keystore** type is used to store your site-specific keys and certificates. You can use either Internet Explorer or Netscape browsers to manage the information in this keystore type.

To Manage the PKCS12 KeyStore using Internet Explorer

1. If you haven't already, load your private key and certificate into the Internet Explorer or Netscape browser.

2. Select **Tools->Internet Options->Content->Certificates**. Select the certificate you want to export and then click **Export**.

3. A wizard appears. Click **Next**.

4. Select the **Yes, export the private key** check box and then click **Next**.

5. Select the **Personal Information Exchange – PKCS12** option, and:

    - select the check box which reads: **Include all certificates in the certification path if possible**

    - deselect (uncheck) the **Enable strong protection** and **Delete the private key if…** check boxes, then click **Next**.

   The password dialog appears.

4. **Do Not** select the **Delete private key after export** check box. It will delete your private key from the store after you export it.

5. Click **Next**.

6. Specify the password for encrypting the file and then click **Next**.

   This password is the new **keystore** password. This password will be encrypted in the **passwds.xml** file using the **pbe** utility described in the **Security** section.

7. Specify the file name such as **<installedPath>\keystores\ServerCert** and then click **Next**.

   You will receive a message indicating the export was successful.

To Manage the PKCS12 Keystore using Netscape Communicator

1. From the browsers main window click **Security** icon.
2. Click the **Certificates->Yours** link.

   A window with a list of certificate serial numbers appears. If there is more than one, view each certificate and then select the correct one.

3. Click **Export**.

   A window appears.

4. Type the Netscape Communicator password for keystores. This is the same password you used when you initially installed this key/certificate on the Netscape browser.

   A window appears.

5. Enter the **keystore** password to protect the data you are exporting and when the next window appears, enter the password again to confirm it.

6. A browser window appears. Select the file name of the exported file and then click **Save**.

Managing the Java Trust Store with Keytool

Keytool is a key and certificate management utility. You can use it to administer your public/private key pairs and associated certificates, which are used in self-authentication. Self-authentication is a process in which a person uses digital certificates to authenticate his or her identity to other users, services, or data integrity and authentication services. You can also use keytool to cache the public keys, in the form of certificates, of your peers.

Viewing the Trusted CA Certificates in the Java Trust Store

By default, this command prints a certificate's MD5 fingerprint. When the **-v** option is specified, the certificate is printed in a format that can be read by humans and it contains information such as the owner, issuer, and serial number.

Command Syntax:

```
<javabin>keytool –list –v –keystore <truststorefile> –storepass
<storepass>
```

Where:

- **javabin**: Path indicating the location where the Java binaries are stored on your machine such as **d:\jdk1.4\bin**)
- **truststorefile**: File containing **truststore** such as **cacerts**.
- **Storepass**: Password to trust store

Example:

```
<installedPath>\java\j2re1.4.0_03\bin\keytool –list –keystore \
          <installedPath>\keystores\tomcat –storepass changeit
```

Changing the Password of the Java Trust Store

Change the password to the Java Trust Store to protect the integrity of the keystore contents. The new password is **newpass,** which must be at least 6 characters long.

Command Syntax:

```
<javabin>keytool -storepasswd -new <newpass> -keystore <truststorefile>
-storepass <oldpass>
```

Where:

- **javabin**: Path indicating the location where the Java binaries are stored on your machine such as **d:\jdk1.4\bin**)
- **truststorefile**: File containing **truststore** such as **cacerts**.
- **oldpass**: Old password to trust store.
- **newpass**: New password to trust store.

Example:

```
<installedPath>\java\j2re1.4.0_03\bin\keytool –storepasswd –new
changeit2 \
     –keystore  <installedPath>\keystores\tomcat –storepass changeit
```

Importing a Certificate Authority Certificate to the Java Trust Store

Use the following command to read the certificate or certificate chain from the file **cert_file** and store it in the keystore entry identified by **alias**. The certificate chain is supplied in a reply in **PKCS#7** format.

Command Syntax:

```
<javabin>keytool –import <alias> –trustcacerts –file <cert_file> -
storepass <storepass> -keystore <truststore>
```

Where:

- **javabin**: Path indicating location where Java binaries are stored on your machine such as **d:\jdk1.4\bin**
- **truststorefile**: File containing **truststore** such as **cacerts**.
- **storepass:** Password to trust store.

Example:

```
<installedPath>\java\j2re1.4.0_03\bin\keytool –import cacert1 –
trustcacerts –file \
      <installedPath>\keystores\changeit3.cer –storepass changeit  \
           –keystore <installedPath>\keystores\cacert1
```

Exporting a Certificate from a Java Key Store

The following command reads the certificate associated with **alias** from the keystore and stores it in the **cert_file** file.

Command Syntax:

```
<javabin>keytool –export <alias> –file <cert_file> -
storepass <storepass> -keystore <keystore>
```

Example:

```
<installedPath>\java\j2re1.4.0_03\bin\keytool –export  \
       –file <installedPath>\keystores\changeit3.cer –storepass
changeit  \
           –keystore <installedPath>\keystores\cacert1
```

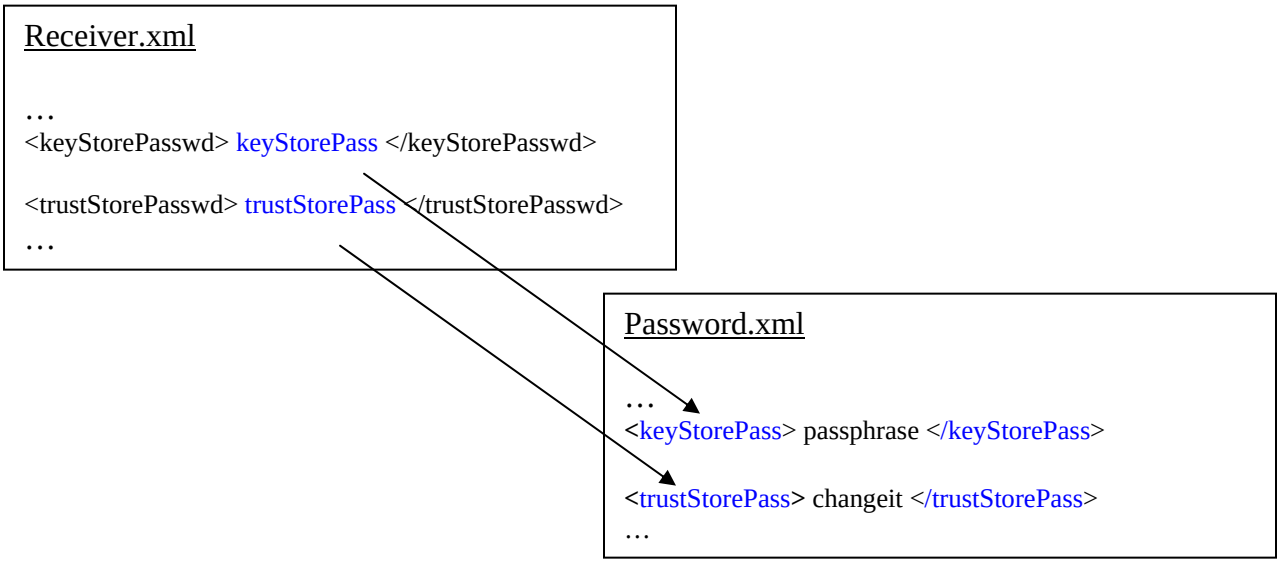
Managing Passwords

Receiver Password File

All passwords are stored in an encrypted password file. You define the name of the password file. It is referenced in the **receiver.xml** file with the **<passwordFile>** tag in XML format.

Passwords referenced in the **receiver.xml** file are pointers to the XML tag in the password file. The password itself is contained within the associated XML tags.

Example:



Because the Receiver is a servlet-based application and has no human intervention during initialization, the password for the password file must be protected. A cipher key generator is used to protect the password file's password. Use the **<key>** and **<seed>** tags are used to define the cipher text.

During the creation of the cipher text, enter a key value that will be used to generate the cipher text. This key value can be any number. The larger the number the larger the seed value generated. The **<seed>** tag references the actual cipher text for the password.

The Public Health Information Network Messaging System offers two utilities to manage passwords; **pbe.bat,** which encrypts and decrypts the password file, and **substitue.bat,** which creates the cipher text used to protect the password file. These commands are defined in more detail in the Password Utilities section.

Password Utilities

Password Based Encryption (pbe)

The Password-Based Encryption command line utility, **pbe.bat**, encrypts and decrypts resource files, such as the password files. These files hold references to all password information throughout the system including passwords for Trusted KeyStore, KeyStore, database, user name and password.

To Encrypt a File

1. Start the **pbe** utility by executing the following command:

   <installedPath>\cmds\pbe.bat
2. Type the letter **e** to encrypt the file and then press **Enter**.
3. Type the name of the file you want to encrypt and then press **Enter**. This file must be at least six characters long and contain at least one numeric digit.
4. Type the location you want to put the file and then press **Enter**.
5. Type the password and then press **Enter**.

To Decrypt a File

1. Start the pbe utility by executing the following command:
   <installedPath>\cmds\pbe.bat
2. Type the letter **d** and then press **Enter**.
3. Type the name of the file you want to decrypt and then press **Enter**. This file must be at lease 6 characters long and contain at lease one number.
4. Type the location you want to put the decrypted file and then press **Enter**.
5. Type the password and then press **Enter**.

Substitution Cipher Key Generator

Use the cipher key generator utility to generate cipher text for a password. The cipher text- based password is used to protect the Message Receiver's password file. The Cipher Key utility requires a unique numeric key value that is used to generate the unique cipher seed value. The key and seed values are placed in the **receivers.xml** configuration file.

To generate a cipher seed value do the following:

1. Start the cipher utility by executing the following command: **<installedPath>\cmds\substitute.bat**.

2. Type the substitution cipher key and then press **Enter**. The key value must be a numeric number at least 3 digit.

3. Type the password text you want to encrypt and then press **Enter**.

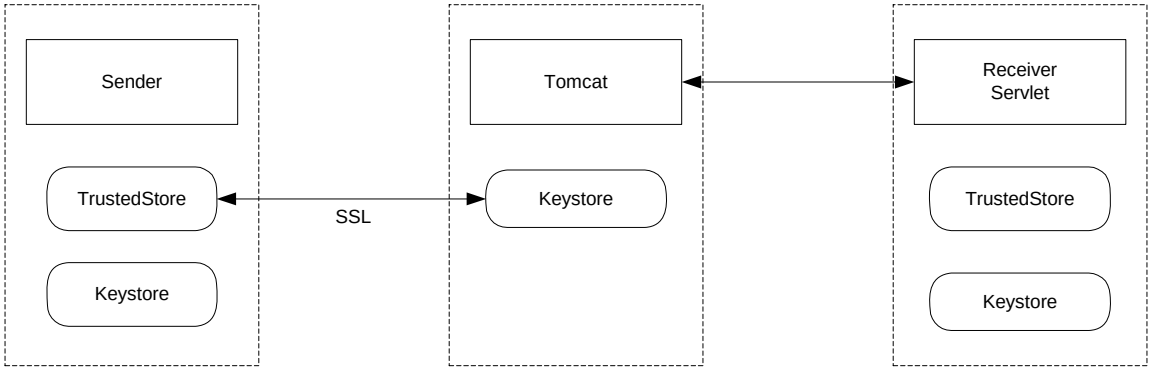
   The encrypted value is displayed on the command prompt. Cut and paste this value into the **receiver.xml <seed>** tag. The associated key value you enter during the generation of the key will also need to be included in the **receiver.xml** file using the <**key>** tag.

Encryption

Enabling SSL Authentication

SSL provides for encryption of a session, authentication of a server, and, optionally, a client and message authentication.

SSL Authentication Flow



To Enable SSL

To enable SSL authentication do the following:

1. Execute the following command to export the Tomcat certificate, which was generated during the **Configuring SSL for Tomcat** section.

```
<installedPath>\java\j2re1.4.0_03\bin\keytool -export -file
tomcat.cer \
          -keystore <installedPath>\keystores\tomcat –alias
tomcat
```

2. Execute the following command to import the Tomcat certificate into your **sender's** Trusted Store.

```
<installedPath>\java\j2re1.4.0_03\bin\ keytool -import -
file tomcat.cer
      -keystore cacerts -alias tomcat
```

3. Restart the sender application.

Encryption

Asymmetric Encryption

Based on the configuration of the PHINMS Message Sender configuration, the Message Sender obtains the information that needs encrypting, along with the LDAP attributes of the message recipient, which include the LDAP server name, the base address, and the recipient's common name.

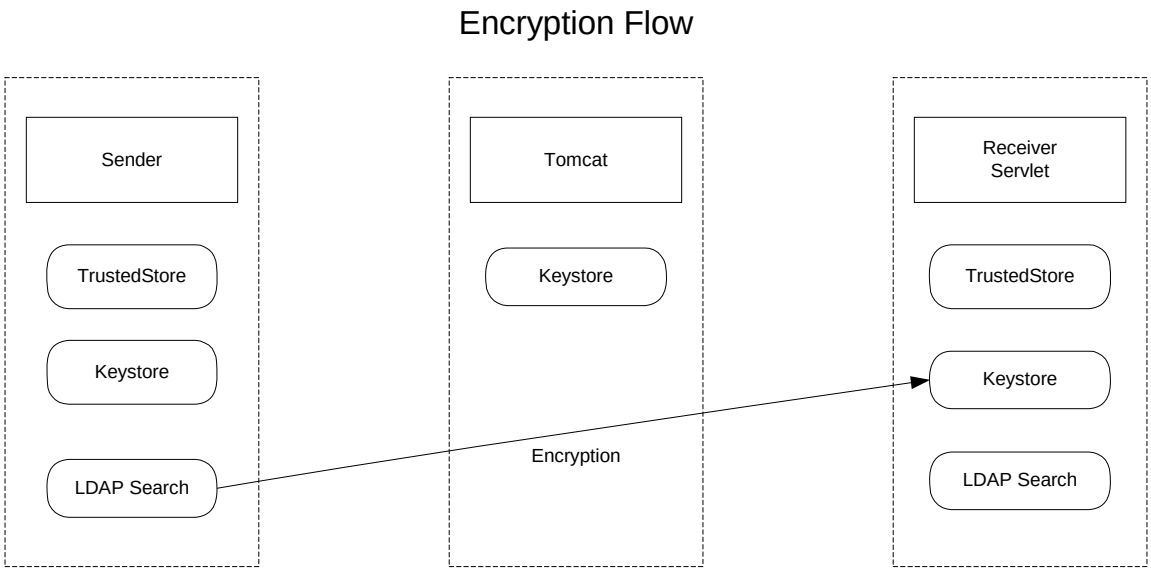
When the Message Sender receives the message from the database or file system it performs a cache lookup for the recipient's certificate. If a recent certificate for the Message Receiver exists in the cache, it retrieves it from the cache and uses it. If the key is not found in the cache, it queries the Verisign LDAP directory and obtains the Message Receiver's public key certificate.

After the Sender obtains the public key certificate, it retrieves the recipient's public key from the certificate .The Message Sender uses the XML Encryption library to perform asymmetric encryption on the plaintext supplied.

Asymmetric Decryption

During asymmetric decryption, the Message Receiver receives the XML encrypted message. The Message Receiver retrieves the private key of its asymmetric key pair from the local Java keystore, and uses this key to decrypt the ciphertext using XML encryption library.

.



Enabling Encryption

To enable encryption do the following:

1. Obtain a valid certificate either through the CDC's Secure Data Network (SND) request or directly with Verisign at http://www.verisign.com.

2. Follow the steps in **Managing PKCS12 Keystore** to export your new key into a PKCS12 keystore.

3. Copy the new keystore created in step 2 to <**installedPath>\keystores**.

4. Modify the **<keyStore>** tag in the **receiver.xml** file to the appropriate path and file name created in step 2.

5. Modify the **<keyStorePasswd>** tag in the **receiver.xml** file to the appropriate tag in the **password.xml** file that points to your keystore password. See the Password Management section of this document for more information.

6. Restart the Message Receiver servlet. There are several steps that must be completed on the Message Sender side before encryption can be completely enabled.

To Enable Digital Signatures

To enable digital signatures do the following:

1. Edit the **receiver.xml** file in **<installedPath>\config** and change the **<signatureRequired>** tags value to **true**.

2. If your Message Receiver will be supporting any third- party clients, modify the **<signingCertsLocation>** tag in the **receiver.xml** file to point to a location on disk that will store all third- party public key certificates.

3. Restart the Message Receiver servlet.

Message Receiver Interface

Example Message Handler Servlet

This is a code listing of an example Message Handler servlet which copies the payload file to disk:

```
/**
 * Title:        <p> MessageHandler
 * Description:  <p> Handles message passed in by ReceiveFileServlet
 * Copyright:    Copyright (c) CDC
 * Company:      CDC
 * @author       Raja Kailar, PhD.
 * @version 1.0
 */
package gov.cdc.nedss.services.messagehandler;

import java.util.Properties;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
// Import the nedss.common package for?
import gov.cdc.nedss.common.*;
// Incorporate logging mechanisms used in the message receiver
// by importing the nedss logging package.
import gov.cdc.nedss.services.logging.*;
// Import nedss xml utiltities to read receiver.xml file.
import gov.cdc.nedss.utilities.xml.*;
import gov.cdc.nedss.services.transport.message.*;
import gov.cdc.nedss.services.security.encryption.*;

public class MessageHandler extends HttpServlet {


    // static property to store incoming directory setting
    // system will write payload out to incoming directory
    private static String incomingDir = null;

    public void setIncomingDir(String s) { incomingDir = s; }
    public String getIncomingDir() { return incomingDir; }

    /**
     * Message Handler initialization routine
```

```
     * @param Servlet Configuration
     */
    public void init(ServletConfig servletConfig) throws
ServletException {

        super.init(servletConfig);
        System.out.println("MessageHandler started");
        System.out.println(Defines.VERSION);

        String configFile =
servletConfig.getInitParameter("receiverConfig");
        XMLReader xmlrdr = new XMLReader();
xmlrdr.readProperties(configFile);
        Properties props = xmlrdr.getProperties();

        // Set the log level, default to debug, development?
        // using CDC nedss log features
        Log.setDebug(true);
        Log.setLogDir(props.getProperty(Defines.RECEIVERLOG));
String logLevel = props.getProperty(Defines.RECEIVERLOGLEVEL);

        if (logLevel.equalsIgnoreCase("none")) {
Log.setLogLevel(Log.NONE);
        } else if (logLevel.equalsIgnoreCase("error")) {
Log.setLogLevel(Log.ERROR);
        } else if (logLevel.equalsIgnoreCase("info")) {
Log.setLogLevel(Log.INFO);
        } else if (logLevel.equalsIgnoreCase("detail")) {
Log.setLogLevel(Log.DETAIL);
        }

        Log.logIt("Started MessageHandler servlet", Log.INFO);
setIncomingDir(props.getProperty(Defines.RECEIVERINCOMINGDIR));
        Log.logIt("messageHandler, incomingDir=" + getIncomingDir(),
Log.INFO);
    }

   /**
    * Servlet post
    * The Servlet accepts the payload and parameters as an HTTP post.
    * @param Http Servlet Request object
    * @param Http Servlet Response object
    */
   protected void doPost(HttpServletRequest request,
HttpServletResponse response)
             throws ServletException, IOException {

      StringBuffer resp = null;
      try {
```

```
        // call processRequest to write file to disk
        if ((resp = processRequest(request)) == null) {
            System.out.println("In message handler, error processing
request");
            return;
        }
        PrintWriter out = response.getWriter();
        out.println(resp.toString());
        out.close();
      } catch (Exception e) {
        System.out.println("Error in doPost");
        e.printStackTrace();
      }
    }

    /**
    * Parses the MIME multipart request message and returns a response
    * @param Http Servlet Request object
    * @return Response string buffer
    */
    private StringBuffer processRequest(HttpServletRequest request) {

      BufferedReader in = null;
      String token = null;
      String fromPartyId = null;
      String manifest = null;

      // use the ? multi part parser
      HttpMultiPartParser hmp = new HttpMultiPartParser();

      try {
          // Splits mime message fields into text and payload
          if (!hmp.processHttpRequest(request)) {
              System.out.println("Error parsing multipart fields in
request");
              return null;
          }
      } catch (Exception e) {
        Log.logIt(e, "Error parsing message in messagehandler",
Log.ERROR);
      }

      // parse the text parameters
      // the message receiver can send multiple text parameters
      StringTokenizer st = new StringTokenizer(hmp.getTextPart(), "&");

      while (st.hasMoreTokens()) {
         token  = st.nextToken();
         if (token.startsWith("from")) {
```

```
            fromPartyId =
token.substring(token.indexOf("=")+1).trim();
         } else if (token.startsWith("manifest")) {
             manifest = token.substring(token.indexOf("=")+1).trim();
         }
      }

      // Do necessary processing of data here
      // e.g., Copy file to disk as below
      try {
        if (hmp.getPayloadPart() != null) {
            FileOutputStream fos = new
FileOutputStream(getIncomingDir() +

hmp.getFilename());

            if (Base64Converter.isBase64(hmp.getPayloadPart())) {

fos.write(Base64Converter.base64StringToByteArray(hmp.getPayloadPart())
);
            } else {
                fos.write(hmp.getPayloadPart().getBytes());
            }
            fos.flush();
            fos.close();
        }
      } catch (Exception e) {
e.printStackTrace();
      }

      // create response
      // Note – these routines will be implemented by the message
handler programmer
      // depending on the specific data processing requirements and
logic.
      String status   = getResponseStatus();// application status
      String error    = getResponseError();// application error
      String appdata  = getResponseAppData();// get application data
      String payload  = getResponsePayload();// get response payload
      String filename = getResponseFilename();          // get response
filename

      // compose mime multi-part response
      StringBuffer response = MimeComposer.composeMessage(status,
error,
                                                          appdata,
                                                          payload,
                                                          filename);
      return response;
```

```
    }

    // The following routines need to be implemented by the message
handler
    // programmer
    private String getResponseStatus() {
        return ("success");
    }

    private String getResponseError() {
        return ("noError");
    }

    private String getResponseAppData() {
        return ("applicationData");
    }

    private String getResponsePayload() {
        return null;    // UCC
    }

    private String getResponseFilename() {
        return null;    // UCC
    }

    /**
     * Dummy get method
     */
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

            PrintWriter out = resp.getWriter();
            resp.setContentType("text/html");
            out.println("<html>");
            out.println("ebXML Default MessageHandler 2.0<br>");
            out.println("Centers for Disease Control and
Prevention<br>");
            out.println(Defines.VERSION+"<br>");
            out.println("</html>");
    }
}
```

Appendix: Transport Queue Interface

Transport Queue Scripts

To create the transport queue you can cut and paste the following Oracle and SQL Server scripts into your program:

Script (DDL) for Oracle:

```
DROP TABLE TransportQ_out;
CREATE TABLE TransportQ_out (
recordId NUMBER (20) NOT NULL,
messageId VARCHAR2 (255) NULL,
payloadFile VARCHAR2 (255) NULL,
payloadContent BLOB,
destinationFilename VARCHAR2 (255) NULL,
routeInfo VARCHAR2 (255) NOT NULL,
service VARCHAR2 (255) NOT NULL,
action VARCHAR2 (255) NOT NULL,
arguments VARCHAR2 (255) NULL,
messageRecipient VARCHAR2 (255) NULL,
messageCreationTime VARCHAR2 (255) NULL,
encryption VARCHAR2 (10) NOT NULL,
signature VARCHAR2 (10) NOT NULL,
publicKeyLdapAddress VARCHAR2 (255) NULL,
publicKeyLdapBaseDN VARCHAR2 (255) NULL,
publicKeyLdapDN VARCHAR2 (255) NULL,
certificateURL VARCHAR2 (255) NULL,
processingStatus VARCHAR2 (255) NULL,
transportStatus VARCHAR2 (255) NULL,
transportErrorCode VARCHAR2 (255) NULL,
applicationStatus VARCHAR2 (255) NULL,
applicationErrorCode VARCHAR2 (255) NULL,
applicationResponse VARCHAR2 (255) NULL,
messageSentTime VARCHAR2 (255) NULL,
messageReceivedTime VARCHAR2 (255) NULL,
responseMessageId VARCHAR2 (255) NULL,
responseArguments VARCHAR2 (255) NULL,
responseLocalFile VARCHAR2 (255) NULL,
responseFilename VARCHAR2 (255) NULL,
responseContent BLOB,
responseMessageOrigin VARCHAR2 (255) NULL,
responseMessageSignature VARCHAR2 (255) NULL,
priority NUMBER (1) NOT NULL
);
ALTER TABLE TransportQ_out
ADD PRIMARY KEY (recordId);
CREATE SEQUENCE transport_record_count
 INCREMENT BY 1
 START WITH 10
 MINVALUE 1
 MAXVALUE 9999999999999999999999999999
```

```
NOCYCLE
NOORDER
CACHE 20;
```

Script for SQL Server:

```
if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[TransportQ_out]') and OBJECTPROPERTY(id, N'IsUserTable') =
1)
drop table [dbo].[TransportQ_out]
GO

CREATE TABLE [dbo].[TransportQ_out] (
[recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
[messageId] [char] (255) NULL,
[payloadFile] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[payloadContent] [IMAGE] NULL ,
[destinationFilename] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[routeInfo] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
[service] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
[action] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
[arguments] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[messageRecipient] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[messageCreationTime] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[encryption] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
[signature] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
[publicKeyLdapAddress] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[publicKeyLdapBaseDN] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[publicKeyLdapDN] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[certificateURL] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[processingStatus] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[transportStatus] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[transportErrorCode] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[applicationStatus] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[applicationErrorCode] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[applicationResponse] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[messageSentTime] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[messageReceivedTime] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[responseMessageId] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[responseArguments] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[responseLocalFile] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[responseFilename] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[responseContent] [IMAGE] NULL ,
[responseMessageOrigin] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
,
[responseMessageSignature] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
[priority] [int] NOT NULL
) ON [PRIMARY]
GO
```

XML File Descriptor

The following is an example of an XML File Descriptor:

```
<fileDescriptor>
      <recordId>22</recordId>
      <payloadFile>d:\projects\clebint\ebxmlvob\outgoing\test.txt</paylo
            adFile>
      <payloadContent></payloadContent>
      <destinationFilename>test.txt</destinationFilename>
      <routeInfo>OKLAHOMA</routeInfo>
      <service>Router</service>
      <action>send</action>
      <arguments>xyz</arguments>
      <messageRecipient>list56</messageRecipient>
      <messageCreationTime>time</messageCreationTime>
      <encryption>yes</encryption>
      <signature>yes</signature>
      <messageRecipient>list56</messageRecipient>
      <publicKeyLdapAddress>directory.verisign.com:389</publicKeyLdapAdd
            ress>
      <publicKeyLdapBaseDN>o=CDC</publicKeyLdapBaseDN>
      <publicKeyLdapDN>cn=Rajashekar Kailar</publicKeyLdapDN>
      <acknowledgementFile>
      d:\projects\clebint\ebxmlvob\filesend_acks\ack_send.xml
      </acknowledgementFile>
</fileDescriptor>
```

XML File Descriptor Response

The following is an example of an XML File Descriptor response.

```
<acknowledgement>
      <transportStatus>success</transportStatus>
      <transportError>none</transportError>
      <applicationStatus>retrieveSucceeded</applicationStatus>
      <applicationError>none</applicationError>
      <applicationData>targetTable=payroll</applicationData>
      <responseLocalFile>1018387200432</responseLocalFile>
      <responseFileName>test.txt</responseFileName>
      <responseSignature>unsigned</responseSignature>
      <responseMessageOrigin>LABCORPDUNSNUMBER</responseMessageOrigin>
</acknowledgement>
```

File-Based Transport Queue

When the interface, the **Transport Queue**, is implemented as a file system directory, the file descriptors may be name-value pairs or XML standard files. The fields used in the file system directory have the same name and semantics as the ones used in the relational database table.

Name-Value Based File Descriptor

The following is an example of a name-value based file-descriptor:

```
recordId=22
payloadFile=d:\\projects\\clebint\\ebxmlvob\\outgoing\\test.txt
```

```
destinationFilename=test.txt
routeInfo=OKLAHOMA
service=Router
action=send
arguments=xyz
messageRecipient=list56
```

Response from Sending a File

The following is an example of a response from a file send operation. The response is written to the **acknowledgementFile** specified in the outgoing file descriptor:

```
transportStatus=success
transportError=none
applicationStatus=retrieveSucceeded
applicationError=none
applicationData=TargetTable=payroll
responseLocalFile=1018379449158
responseFileName=test.txt
responseSignature=unsigned
responseMessageOrigin=LABCORP
```

Transport Level Status and Error Codes

When a message is delivered or processed a transport status code is sent back to the Transport Queue. If there was an error in the delivery or processing of the message an error code is also sent back to the Transport Queue. Applications that use the PHINMS read these codes and act on them.

The following table describes the transport status and error codes:

Transport Status Code	Description
success	Message send or receive operation was successful.
failure	Message send or receive operation failed.
Transport Error Code	**Description**
SecurityFailure	Error logging into Message Receiver.
DeliveryFailure	Failed to deliver message.
NotSupported	Format of the ebXML message or CPA is unsupported.
Unknown	Not a standard ebXML error.
NoSuchService*	Service/Action did not map to a service on the Message Receiver.
CheckSumFailure*	File checksum verification failure at the Message Receiver.

Custom Error Codes

The asterisk (*) symbol indicates a custom error code, which means, the code is not in the ebXML specifications.

NoSuchService* - Service/Action did not map to a service on the Message Receiver.

CheckSumFailure* - File checksum verification failure at the Message Receiver.

Appendix: Worker/Error Queue Interface

The following table describes the fields and data in the worker/error queue.

Field	Description	Data Type
recordId	Unique ID of the record in the table and the table's primary key.	Required SQL Server: Integer Identity=Yes, Identity Increment=1 Oracle: Integer (20) not null
messageId	Application level message identifier.	Optional SQL Server: varchar (255) null Oracle: varchar2 (255) null
payloadName	File name of the payload, specified by the Message Sender.	Optional SQL Server: varchar (255) null Oracle: varchar2 (255) null
payloadBinaryContent	Image/BLOB field that is written to by the receiver servlet.	Optional SQL Server: Image data type, null Oracle: BLOB data type, null
payloadTextContent	Text/CLOB field that is populated if **textPayload=true** in the **servicemap** entry.	Optional SQL Server: Text, null Oracle: CLOB, null
localFileName	Used when the file is written to disk instead of a database field. (If **payloadToDisk=true** in the **servicemap** entry)	Optional SQL Server: varchar (255,) null Oracle: varchar2 (255), null
service	ebXML service name.	Required SQL Server: varchar (255,) null Oracle: varchar2 (255), null
action	ebXML action.	Required SQL Server: varchar (255,) null Oracle: varchar2 (255), null
arguments	Arguments specified by the Message Sender.	Optional SQL Server: varchar (255,) null Oracle: varchar2 (255), null

Field	Description	Data Type
fromPartyId	PartyId of the Message Sender.	Required SQL Server: varchar (255,) null Oracle: varchar2 (255), null
messageRecipient	Message recipient's identifier, specified by the Message Sender in the **transportQ_Out** queue.	Optional SQL Server: varchar (255,) null Oracle: varchar2 (255), null
errorCode	Error code.	Optional SQL Server: varchar (255,) null Oracle: varchar2 (255), null
errorMessage	Error message.	Optional SQL Server: varchar (255,) null Oracle: varchar2 (255), null
processingStatus	Status of the record. When a record is created, the initial value of the **processingStatus** field is **queued**.	Optional SQL Server: varchar (255,) null Oracle: varchar2 (255), null
applicationStatus	Status of the application.	Optional SQL Server: varchar (255), null Oracle: varchar2 (255), null
encryption	The value is **yes** if the payload is encrypted and **no** if not encrypted.	Optional SQL Server: varchar (10) null Oracle: varchar2 (10), null
receivedTime	Time when payload was received, in UTC format such as **`2001-10-01T16:01:01`**.	Optional SQL Server: varchar (255), null Oracle: varchar2 (255), null
lastUpdateTime	Time when record was last updated, in UTC format such as **`2001-10-01T16:01:01`**.	Optional SQL Server: varchar (255), null Oracle: varchar2 (255), null
processId	Identifies the process that is processing or most recently processed the record.	Optional SQL Server: varchar (255), null Oracle: varchar2 (255), null

Worker Queue Script for SQL Server

```
CREATE TABLE [dbo].[<Tablename>] (
[recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
[messageId] [varchar] (255) NULL,
[payloadName] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[payloadBinaryContent] [IMAGE] NULL ,
       [payloadTextContent] [TEXT] NULL,
       [localFileName] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL ,
[service] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
[action] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
[arguments] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[fromPartyId] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[messageRecipient] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[errorCode] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
       [errorMessage] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
,
[processingStatus] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[applicationStatus] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
       [encryption] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
,
       [receivedTime] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
,
       [lastUpdateTime] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
[processId] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
) ON [PRIMARY]
```

Worker Queue Script (DDL) for Oracle

```
DROP TABLE <Tablename>;
CREATE TABLE <Tablename> (
recordId NUMBER (20) NOT NULL,
messageId VARCHAR2 (255) NULL,
payloadName VARCHAR2 (255) NULL,
payloadBinaryContent BLOB,
       payloadTextContent CLOB,
localFilename VARCHAR2 (255) NULL,
service VARCHAR2 (255) NOT NULL,
action VARCHAR2 (255) NOT NULL,
arguments VARCHAR2 (255) NULL,
       fromPartyId VARCHAR2 (255) NULL,
messageRecipient VARCHAR2 (255) NULL,
       errorCode VARCHAR2 (255) NULL,
       errorMessage VARCHAR2 (255) NULL,
processingStatus VARCHAR2 (255) NULL,
applicationStatus VARCHAR2 (255) NULL,
       encryption VARCHAR2 (10) NOT NULL,
receivedTime VARCHAR2 (255) NULL,
       lastUpdateTime VARCHAR2 (255) NULL,
       processId VARCHAR2 (255) NULL,
);
ALTER TABLE <Tablename>
ADD PRIMARY KEY (recordId);
CREATE SEQUENCE <Tablename>_record_count
 INCREMENT BY 1
 START WITH 10
 MINVALUE 1
 MAXVALUE 9999999999999999999999999
 NOCYCLE
 NOORDER
 CACHE 20;
```

Note To correctly increment the worker queue records, the

sequence name must be **<Tablename>_record_count**.

Appendix: Example of the Collaboration Protocol Agreement

```
<?xml version="1.0" ?>
<tp:CollaborationProtocolAgreement>
<tp:PartyInfo>
  <tp:PartyId tp:type="DUNS">NEBRASKADUNSNUMBER</tp:PartyId>
  <tp:PartyRef xlink:href="http://www.nebraska.com/about.html" />
  <tp:Transport tp:transportId="N05">
   <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
   <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
   <tp:Endpoint  tp:uri="www.lab.com/soapreceiver/receiver"  tp:type="allPurpose"
/>
   <tp:TransportSecurity>
        <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
        <tp:CertificateRef tp:certId="N03" />
   </tp:TransportSecurity>
  </tp:Transport>
</tp:PartyInfo>
<tp:PartyInfo>
  <tp:PartyId tp:type="DUNS">CDCDUNSNUMBER</tp:PartyId>
  <tp:PartyRef xlink:type="simple" xlink:href="http://www.cdc.gov/about.html"/>
  <tp:Transport tp:transportId="N35">
  <tp:SendingProtocol tp:version="1.1">HTTPS</tp:SendingProtocol>
  <tp:ReceivingProtocol tp:version="1.1">HTTPS</tp:ReceivingProtocol>
  <tp:Endpoint                  tp:uri="icdc-xdv-sdn7.cdc.gov/ebxml/receivefile"
tp:type="allPurpose" />
        <tp:TransportSecurity>
            <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
            <tp:CertificateRef></tp:CertificateRef>
            <tp:authenticationType>netegrity</tp:authenticationType>
            <!-- basic, custom, sdn, clientcert, netegrity -->

            <tp:sdnAuth>
      <tp:sdnPassword>sdnPassword1</tp:sdnPassword>
              <tp:sdnLoginPage>/sdn_ext/sdn4.dll?ValidateInitialLogin</tp:sdnLog
       inPage>
            </tp:sdnAuth>

              <tp:netegrityAuth>
      <tp:sdnPassword>sdnPassword1</tp:sdnPassword>
              <tp:sdnLoginPage>/certphrase/login.fcc</tp:sdnLoginPage>
            </tp:netegrityAuth>

            <tp:clientCertAuth>
                <tp:config>
       d:\\projects\\clebint\\ebxmlvob\\config\\sdnlogin.props</tp:config>
            </tp:clientCertAuth>

            <tp:customAuth>
                <tp:customLoginPage>/login.asp</tp:customLoginPage>
                <tp:publicParams>logine=check</tp:publicParams>
        <tp:secretParams>username=user2&amp;userpwd=passwd2</tp:secretParams>
           </tp:customAuth>

           <tp:basicAuth>
                <tp:indexPage>/session.asp</tp:indexPage>
                <tp:basicAuthUser>user1</tp:basicAuthUser>
                <tp:basicAuthPasswd>passwd1</tp:basicAuthPasswd>
           </tp:basicAuth>

</tp:TransportSecurity>
</tp:Transport>
</tp:PartyInfo>
<tp:Comment    xml:lang="en-us">send/receive    agreement    between    cdc    and
Nebraska</tp:
Comment>
</tp:CollaborationProtocolAgreement>
```

Appendix: Example of the Message Sender Configuration File, Sender.xml

```
<Sender>
<installDir>d:\\projects\\evalebxmlint\\ebxmlvob\\</installDir>
<filePoll>true</filePoll>
<dbPoll>false</dbPoll>
<routeNotReadPoll>true</routeNotReadPoll>
<test>false</test>
<passwordFile>config\\passwds</passwordFile>
<maxAttempts>2</maxAttempts>
<filePollMode>loop</filePollMode>
<!-- possible values for filePollMode are { once, loop } -->
<filePollInterval>30</filePollInterval>
<fileDescriptorDir>filedescriptors\\</fileDescriptorDir>
<!-- fileDescriptor XML, nameValue -->
<fileDescriptorFormat>nameValue</fileDescriptorFormat>
<!-- <filePollDir>outgoing\\</filePollDir> -->
<!-- sqlserver, oracle, fositex -->
<dbType>fositex</dbType>
<!-- possible values for dbPollMode are { once, loop } -->
<dbPollMode>loop</dbPollMode>
<!-- possible values for dbPollMode are { once, loop } -->
<dbPollDir>outgoing\\</dbPollDir>
<responseToDb>true</responseToDb>
<incomingDir>incoming\\</incomingDir>
<dbPollInterval>10</dbPollInterval>
<dataReadTimeOut>30</dataReadTimeOut>
<myPartyId>LABCORPDUNSNUMBER</myPartyId>
<routeMap>config\\routemap.xml</routeMap>
<cpaLocation>config\\CPA\\</cpaLocation>
<connectionTimeOut>20</connectionTimeOut>
<trustedCerts>config\\cacerts</trustedCerts>
<trustedCertsPasswd>cacertsPasswd1</trustedCertsPasswd>
<keyStore>config\\dddtest.pfx</keyStore>
<keyStorePasswd>keyStorePasswd1</keyStorePasswd>
<logLevel>messages</logLevel>
<maxLogSize>10000000</maxLogSize>
<logArchive>true</logArchive>
<logDir>logs\\</logDir>
<processedDir>processed\\</processedDir>
<ldapKeyRetrieval>false</ldapKeyRetrieval>
<ldapCache>true</ldapCache>
<ldapCacheTimeoutHours>1</ldapCacheTimeoutHours>
<ldapCachePath>config\\ldapCache</ldapCachePath>

<queueMap>config\\clientqueuemap.xml</queueMap>
<serviceMap>config\\clientservicemap.xml</serviceMap>

<!-- Monitor Parameters -->
<monitorTimerInterval>10000</monitorTimerInterval>

<!-- To setup this command as a service -->
<service>true</service>
<serviceKey>2o3i23</serviceKey>
<serviceSeed>151209139182126100100162</serviceSeed>

<!-- ebXML Transport parameters -->
<syncReply>true</syncReply>
<ackRequested>true</ackRequested>
<signedAck>false</signedAck>

<!-- MS Access database configuration
<jdbcDriver>sun.jdbc.odbc.JdbcOdbcDriver</jdbcDriver>
<databaseUrlPrefix>jdbc:odbc:</databaseUrlPrefix>
```

```
<databaseUrlSuffix>messagequeue</databaseUrlSuffix>
<messageTable>messagequeue</messageTable>
<databaseUser>accessdbUser</databaseUser>
<databasePasswd>accessdbPasswd</databasePasswd>
→
```

<!—SQL Server Database Configuration

```
<jdbcDriver>com.microsoft.jdbc.sqlserver.SQLServerDriver</jdbcDriver>
<databaseUrlPrefix>jdbc:microsoft:sqlserver:</databaseUrlPrefix>
<databaseUrlSuffix>//nedss-sql3:1433;DatabaseName=Phmsg</databaseUrlSuffix>
<messageTable>TransportQ_out</messageTable>
<databaseUser>sqlServerDbUser</databaseUser>
<databasePasswd>sqlServerDbPasswd</databasePasswd>
-->
```

<!-- CSV File - Fositex Driver -->

```
<jdbcDriver>com.inet.csv.CsvDriver</jdbcDriver>
<databaseUrlPrefix>jdbc:csv:</databaseUrlPrefix>
<databaseUrlSuffix>data\\phmsg</databaseUrlSuffix>
<messageTable>messagequeue</messageTable>
```

<!—ORACLE Database Configuration

```
<jdbcDriver>oracle.jdbc.driver.OracleDriver</jdbcDriver>
<databaseUrlPrefix>jdbc:oracle:thin:</databaseUrlPrefix>
<databaseUrlSuffix>@nedss-report:1521:ebxml</databaseUrlSuffix>
<messageTable>TransportQ_out</messageTable>
<databaseUser>oracleServerDbUser</databaseUser>
<databasePasswd>oracleServerDbPasswd</databasePasswd>
-->
```

<!-- <tempFileFolder>data\\</tempFileFolder> -->

```
<encryptionTemplate>config\\encryptiontemplate.xml</encryptionTemplate>
<signatureTemplate>config\\payloadSignature.xml</signatureTemplate>

<pollList>
 <destination>
  <route>CDC</route>
  <service>Router</service>
  <action>autopoll</action>
  <recipient>nedoh</recipient>
  <pollInterval>10</pollInterval>
 </destination>
</pollList>

<databasePool>
  <database>
    <databaseId>sqlserver1</databaseId>
    <dbType>sqlserver</dbType>
    <poolSize>1</poolSize>
    <jdbcDriver>com.microsoft.jdbc.sqlserver.SQLServerDriver</jdbcDriver>

<databaseUrl>jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=Phmsg</datab
a
seUrl>
    <databaseUser>sqlServerDbUserLocal</databaseUser>
    <databasePasswd>sqlServerDbPasswdLocal</databasePasswd>
  </database>
  <database>
   <databaseId>access1</databaseId>
   <dbType>access</dbType>
   <poolSize>1</poolSize>
   <jdbcDriver>sun.jdbc.odbc.JdbcOdbcDriver</jdbcDriver>
   <databaseUrl>jdbc:odbc:workerqueue</databaseUrl>
   <databaseUser>accessdbUser</databaseUser>
   <databasePasswd>accessdbPasswd</databasePasswd>
  </database>
  <database>
    <databaseId>sqlserver2</databaseId>
    <dbType>sqlserver</dbType>
    <poolSize>1</poolSize>
    <jdbcDriver>com.microsoft.jdbc.sqlserver.SQLServerDriver</jdbcDriver>
```

```
    <databaseUrl>jdbc:microsoft:sqlserver://nedss-
sql3.cdc.gov:1433;DatabaseName=Phms
g</databaseUrl>
    <databaseUser>sqlServerDbUserRemote</databaseUser>
    <databasePasswd>sqlServerDbPasswdRemote</databasePasswd>
  </database>
  <database>
    <databaseId>oracleserver1</databaseId>
    <dbType>oracle</dbType>
    <poolSize>1</poolSize>
    <jdbcDriver>oracle.jdbc.driver.OracleDriver</jdbcDriver>
    <databaseUrl>jdbc:oracle:thin:@nedss-report:1521:ebxml</databaseUrl>
    <databaseUser>oracleServerDbUserRemote</databaseUser>
    <databasePasswd>oracleServerDbPasswdRemote</databasePasswd>
  </database>
</databasePool>

</Sender>
```

Appendix: Server-Side Persistent Cache Schema

Messaging Cache for SQL Server (server-side persistent cache)

```
CREATE TABLE [dbo].[messagingcache] (
	[sequence] [int] IDENTITY (1, 1) NOT NULL ,
	[partyId] [char] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[convId] [char] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[recordId] [char] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[response] [text] COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[timestamp] [char] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[status] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

Messaging Cache for Oracle

```
CREATE TABLE messagingcache (
	sequence NUMBER (20) NOT NULL,
partyId VARCHAR2 (255) NULL,
convId VARCHAR2 (255) NULL,
recordId VARCHAR2 (255) NULL,
response CLOB,
timestamp VARCHAR2 (50) NULL,
status VARCHAR2 (20) NULL);
CREATE SEQUENCE messagingcache_record_count
 INCREMENT BY 1
 START WITH 10
 MINVALUE 1
 MAXVALUE 9999999999999999999999999999
 NOCYCLE
 NOORDER
 CACHE 20;
```

Appendix F: Route-not-Read Schema

Broadcast Generation Script for SQL Server

```
CREATE TABLE [dbo].[broadcast] (
 [name] [char] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
 [addresses] [char] (1000) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
) ON [PRIMARY]
GO
```

Broadcast Generation Script for Oracle

```
CREATE TABLE broadcast (
        name VARCHAR2 (255) NOT NULL,
        addresses VARCHAR2 (255) NOT NULL);
```

Messagebin Generation Script for Oracle

```
DROP TABLE <Tablename>;
CREATE TABLE <Tablename> (
        recordId NUMBER (20) NOT NULL,
        messageId VARCHAR2 (255) NULL,
        payloadName VARCHAR2 (255) NULL,
        payloadBinaryContent BLOB,
        payloadTextContent CLOB,
        localFilename VARCHAR2 (255) NULL,
        service VARCHAR2 (255) NOT NULL,
        action VARCHAR2 (255) NOT NULL,
        arguments VARCHAR2 (255) NULL,
        fromPartyId VARCHAR2 (255) NULL,
        messageRecipient VARCHAR2 (255) NULL,
        errorCode VARCHAR2 (255) NULL,
        errorMessage VARCHAR2 (255) NULL,
        processingStatus VARCHAR2 (255) NULL,
        applicationStatus VARCHAR2 (255) NULL,
        encryption VARCHAR2 (10) NOT NULL,
        receivedTime VARCHAR2 (255) NULL,
        lastUpdateTime VARCHAR2 (255) NULL,
```

```
        processId VARCHAR2 (255) NULL);
ALTER TABLE <Tablename>
ADD PRIMARY KEY (recordId);
CREATE SEQUENCE <Tablename>_record_count
 INCREMENT BY 1
 START WITH 10
 MINVALUE 1
 MAXVALUE 9999999999999999999999999999
 NOCYCLE
 NOORDER
 CACHE 20;
```

Messagebin Generation Script for SQL Server

```
CREATE TABLE [dbo].[<Tablename>] (
 [recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
 [messageId] [varchar] (255) NULL,
 [payloadName] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [payloadBinaryContent] [IMAGE] NULL ,
 [payloadTextContent] [TEXT] NULL,
 [localFileName] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
 [service] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
 [action] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
 [arguments] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [fromPartyId] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [messageRecipient] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [errorCode] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [errorMessage] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [processingStatus] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [applicationStatus] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [encryption] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
 [receivedTime] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [lastUpdateTime] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [processId] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
) ON [PRIMARY]
GO
```

Users Generation Script for SQL Server

```
CREATE TABLE [dbo].[users] (
  [name] [char] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
  [description] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
```

Users Generation Script for Oracle

```
CREATE TABLE users (
        name VARCHAR2 (255) NOT NULL,
        description VARCHAR2 (255) NULL);
```

PartyID_User Generation Script for SQL Server

```
CREATE TABLE [dbo].[partyid_user] (
        [partyId] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
        [user] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
        [sdnuser] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
) ON [PRIMARY]
GO
```

PartyID_User generation Script for Oracle

```
CREATE TABLE partyid_user (
        partyId VARCHAR2 (255) NOT NULL,
        user VARCHAR2 (255) NOT NULL),
        sdnuser VARCHAR2 (255) NOT NULL);
```

Appendix: File-Based Message Queue

The message queue can also be implemented using operating system files, without using a relational database table. The file descriptors can be name-value pairs or XML files.

Name-Value Based File Descriptor

The following is a name-value based file descriptor *:*

```
recordId=22
payloadFile=d:\\projects\\clebint\\ebxmlvob\\outgoing\\test.txt
destinationFilename=test.txt
routeInfo=OKLAHOMA
service=Router
action=send
arguments=xyz
messageRecipient=list56
messageCreationTime=time
encryption=no
signature=yes
publicKeyLdapAddress=directory.verisign.com:389
publicKeyLdapBaseDN=o=Centers for Disease Control and Prevention
publicKeyLdapDN=cn=Rajashekar Kailar
acknowledgementFile=d:\\projects\\clebint\\ebxmlvob\\filesend_acks
\\ack_send.props
```

The response from a file-send operation is written to the **acknowledgementFile** specified in the outgoing file descriptor, shown previously, and looks like the following:

```
transportStatus=success
transportError=none
applicationStatus=retrieveSucceeded
applicationError=none
applicationData=TargetTable=payroll
responseLocalFile=1018379449158
responseFileName=test.txt
responseSignature=unsigned
responseMessageOrigin=LABCORP
```

For a detailed description of these fields, refer to the Message Queue table schema, which contains fields with the same name and semantics.

XML Based File Descriptor

The following is an example of an XML file descriptor:

```
<fileDescriptor>
      <recordId>22</recordId>
      <payloadFile>d:\projects\clebint\ebxmlvob\outgoing\test.txt<
      /payloadFile>
      <payloadContent></payloadContent>
      <destinationFilename>test.txt</destinationFilename>
      <routeInfo>OKLAHOMA</routeInfo>
      <service>Router</service>
      <action>send</action>
      <arguments>xyz</arguments>
      <messageRecipient>list56</messageRecipient>
      <messageCreationTime>time</messageCreationTime>
      <encryption>yes</encryption>
      <signature>yes</signature>
      <messageRecipient>list56</messageRecipient>
      <publicKeyLdapAddress>directory.verisign.com:389</publicKeyL
      dapAddress>
      <publicKeyLdapBaseDN>o=CDC</publicKeyLdapBaseDN>
      <publicKeyLdapDN>cn=Rajashekar Kailar</publicKeyLdapDN>
```

```
        <acknowledgementFile>
        d:\projects\clebint\ebxmlvob\filesend_acks\ack_send.xml
        </acknowledgementFile>
    </fileDescriptor>
```

The response from a file-send operation is written to the **acknowledgementFile** specified in the outgoing file descriptor, shown previously, and looks like the following:

```
        <acknowledgement>
              <transportStatus>success</transportStatus>
              <transportError>none</transportError>
              <applicationStatus>retrieveSucceeded</applicationStatu
              s>
              <applicationError>none</applicationError>
              <applicationData>targetTable=payroll</applicationData>
              <responseLocalFile>1018387200432</responseLocalFile>
              <responseFileName>test.txt</responseFileName>
              <responseSignature>unsigned</responseSignature>
              <responseMessageOrigin>LABCORPDUNSNUMBER</responseMess
              ageOrigin>
        </acknowledgement>
```

For a detailed description of these fields, refer to the Message Queue Table Schema, which contains fields with the same name, and semantics.

Appendix: Transport Level Status and Error Codes

The following status and error codes may be written to the message queues based on the outcome of the message delivery or processing. Applications that use the PHINMS system can read these codes and act on them.

Status Codes

Status	Description
success	Message Send/Receive operation successful
failure	Message Send/Receive operation failure

Error Codes

ErrorCode	Description
SecurityFailure	Error logging into Message Receiver
DeliveryFailure	Failed to deliver message
NotSupported	Format of ebXML message or CPA unsupported
Unknown	Not a standard ebxml error
NoSuchService (*)	Service/Action did not map to a service on the Message receiver
CheckSumFailure (*)	File checksum verification failure at the Message Receiver

(*) Custom error codes - not in ebXML specification.

Appendix: Example receiver.xml File

```
<Receiver>
   <logDir>c:\phmsgServer_2_0\logs\</logDir>
   <logLevel>messages</logLevel>
   <incomingDir>c:\phmsgServer_2_0\incoming\</incomingDir>
   <myPartyId>cdc</myPartyId>
   <cpaLocation>c:\phmsgServer_2_0\config\CPA\</cpaLocation>
   <serviceMap>c:\phmsgServer_2_0\config\servicemap.xml</serviceMap>
   <passwordFile>c:\phmsgServer_2_0\config\receiverpasswds</passwordFil
   e>
   <keyStore>c:\phmsgServer_2_0\config\phmsg2.pfx</keyStore>
   <keyStorePasswd>keyStorePasswd1</keyStorePasswd>
   <trustStore>c:\phmsgServer_2_0\keystores\cacerts</trustStore>
   <trustStorePasswd>cacertsPasswd1</trustStorePasswd>
   <signatureRequired>false</signatureRequired>
   <signingCertsLocation>c:\phmsgServer_2_0\config\signingcerts</signin
   gCertsLocation>
   <key>q490uradf</key>
   <seed>214150145125193</seed>
   <uccTest>false</uccTest>
</Receiver>
```

Appendix: Example Receiver Password File

```
<?xml version="1.0"?>
<passwordFile>
      <trustStorePass>changeit</trustStorePass>
      <keyStorePass>passphrase</keyStorePass>
      <dbUser>phmsg</dbUser>
      <dbPasswd>phmsg123</dbPasswd>
      <mysqldbUser>root</mysqldbUser>
      <mysqldbPasswd>sql</mysqldbPasswd>
</passwordFile>
```

Appendix K: Example Routerconfig.xml File

```
- <Router>
      - <!--
       SETTINGS FOR MYSQL DB ON LINUX
       <jdbcDriver>org.gjt.mm.mysql.Driver</jdbcDriver>
       <databaseUrl>jdbc:mysql://158.111.1.164:3306/test</databaseUrl>
       <databaseUser>myqsldbUser</databaseUser>
       <databasePasswd>myqsldbPasswd</databasePasswd>
    -->
      - <!--
       SETTINGS FOR SQL SERVER DB ON NEDSS-SQL3
       values can be sqlServer, oracle
    -->
    <dbType>sqlServer</dbType>
      <jdbcDriver>com.microsoft.jdbc.sqlserver.SQLServerDriver</jdbcDri
      ver>
    <databaseUrl>jdbc:microsoft:sqlserver://nedss-
      sql3.cdc.gov:1433;DatabaseName=Phmsg</databaseUrl>
    <databaseUser>sqlServerDbUserRemote</databaseUser>
    <databasePasswd>sqlServerDbPasswdRemote</databasePasswd>
  - <!--
   END SETTINGS FOR SQL SERVER
    -->
      - <!--
       SETTINGS FOR ORACLE ON NEDSS-REPORT
       <dbType>oracle</dbType>
       <jdbcDriver>oracle.jdbc.driver.OracleDriver</jdbcDriver>
       <databaseUrl>jdbc:oracle:thin:@nedss-report:1521:ebxml</databaseUrl>
       <databaseUser>oracleServerDbUserRemote</databaseUser>
```

```
        <databasePasswd>oracleServerDbPasswdRemote</databasePasswd>

    -->
    <sdnAuth>false</sdnAuth>

       <passwordFile>d:\tomcat4\ebxml\config\receiverpasswds</passwor
       dFile>
    <payloadDir>d:\tomcat4\ebxml\payloaddir\</payloadDir>
    <logLevel>info</logLevel>
    <logDir>d:\tomcat4\ebxml\logs\</logDir>
    <maxLogSize>10000000</maxLogSize>
    <logArchive>true</logArchive>
    <maxMessageBufferSize>1000000</maxMessageBufferSize>
    <deleteOnPickup>true</deleteOnPickup>
    <key>2o3i23</key>
    <seed>151209139182126100100162</seed>
  </Router>
```

Appendix: Example Catalina.bat File

The following is an example of the catalina.bat file. If you have chosen the default PHINMS installation, the catalina.bat file will be in your Tomcat directory.

```
@echo off
if "%OS%" == "Windows_NT" setlocal
rem ---------------------------------------------------------------------------
rem Start/Stop Script for the CATALINA Server
rem
rem Environment Variable Prequisites
rem
rem   CATALINA_HOME   May point at your Catalina "build" directory.
rem
rem   CATALINA_BASE   (Optional) Base directory for resolving dynamic portions
rem                   of a Catalina installation.  If not present, resolves to
rem                   the same directory that CATALINA_HOME points to.
rem
rem   CATALINA_OPTS   (Optional) Java runtime options used when the "start",
rem                   "stop", or "run" command is executed.
rem
rem   CATALINA_TMPDIR (Optional) Directory path location of temporary directory
rem                   the JVM should use (java.io.tmpdir).  Defaults to
rem                   %CATALINA_BASE%\temp.
rem
rem   JAVA_HOME       Must point at your Java Development Kit installation.
rem
rem   JAVA_OPTS       (Optional) Java runtime options used when the "start",
rem                   "stop", or "run" command is executed.
```

```
rem
rem   JSSE_HOME      (Optional) May point at your Java Secure Sockets Extension
rem                  (JSSE) installation, whose JAR files will be added to the
rem                  system class path used to start Tomcat.
rem
rem   JPDA_TRANSPORT  (Optional) JPDA transport used when the "jpda start"
rem                  command is executed. The default is "dt_shmem".
rem
rem   JPDA_ADDRESS    (Optional) Java runtime options used when the "jpda start"
rem                  command is executed. The default is "jdbconn".
rem
rem $Id: catalina.bat,v 1.7 2003/11/05 11:40:18 remm Exp $
rem ---------------------------------------------------------------------------

set JAVA_OPTS=-Xms30m -Xmx500m
set JAVA_HOME=..\..\jdk\win32\
rem Guess CATALINA_HOME if not defined
if not "%CATALINA_HOME%" == "" goto gotHome
set CATALINA_HOME=.
if exist "%CATALINA_HOME%\bin\catalina.bat" goto okHome
set CATALINA_HOME=..
:gotHome
if exist "%CATALINA_HOME%\bin\catalina.bat" goto okHome
echo The CATALINA_HOME environment variable is not defined correctly
echo This environment variable is needed to run this program
goto end
:okHome

rem Get standard environment variables
if                exist                "%CATALINA_HOME%\bin\setenv.bat"                call
"%CATALINA_HOME%\bin\setenv.bat"

rem Get standard Java environment variables
if exist "%CATALINA_HOME%\bin\setclasspath.bat" goto okSetclasspath
echo Cannot find %CATALINA_HOME%\bin\setclasspath.bat
echo This file is needed to run this program
goto end
:okSetclasspath
set BASEDIR=%CATALINA_HOME%
call "%CATALINA_HOME%\bin\setclasspath.bat"

rem Add on extra jar files to CLASSPATH
if "%JSSE_HOME%" == "" goto noJsse
set
CLASSPATH=%CLASSPATH%;%JSSE_HOME%\lib\jcert.jar;%JSSE_HOME%\lib\jnet.j
ar;%JSSE_HOME%\lib\jsse.jar
:noJsse
set CLASSPATH=%CLASSPATH%;%CATALINA_HOME%\bin\bootstrap.jar

if not "%CATALINA_BASE%" == "" goto gotBase
set CATALINA_BASE=%CATALINA_HOME%
```

```
:gotBase

if not "%CATALINA_TMPDIR%" == "" goto gotTmpdir
set CATALINA_TMPDIR=%CATALINA_BASE%\temp
:gotTmpdir

rem ----- Execute The Requested Command ---------------------------------------

echo Using CATALINA_BASE:   %CATALINA_BASE%
echo Using CATALINA_HOME:   %CATALINA_HOME%
echo Using CATALINA_TMPDIR: %CATALINA_TMPDIR%
echo Using JAVA_HOME:       %JAVA_HOME%

set _EXECJAVA=%_RUNJAVA%
set MAINCLASS=org.apache.catalina.startup.Bootstrap
set ACTION=start
set SECURITY_POLICY_FILE=
set DEBUG_OPTS=
set JPDA=
if not ""%1"" == ""jpda"" goto noJpda
set JPDA=jpda
if not "%JPDA_TRANSPORT%" == "" goto gotJpdaTransport
set JPDA_TRANSPORT=dt_shmem
:gotJpdaTransport
if not "%JPDA_ADDRESS%" == "" goto gotJpdaAddress
set JPDA_ADDRESS=jdbconn
:gotJpdaAddress
shift
:noJpda

if ""%1"" == ""debug"" goto doDebug
if ""%1"" == ""run"" goto doRun
if ""%1"" == ""start"" goto doStart
if ""%1"" == ""stop"" goto doStop

echo Usage:  catalina ( commands ... )
echo commands:
echo   debug             Start Catalina in a debugger
echo   debug -security   Debug Catalina with a security manager
echo   jpda start        Start Catalina under JPDA debugger
echo   run               Start Catalina in the current window
echo   run -security     Start in the current window with security manager
echo   start             Start Catalina in a separate window
echo   start -security   Start in a separate window with security manager
echo   stop              Stop Catalina
goto end

:doDebug
shift
set _EXECJAVA=%_RUNJDB%
```

```
set DEBUG_OPTS=-sourcepath "%CATALINA_HOME%\..\..\jakarta-tomcat-catalina\catalina\src\share"
if not ""%1"" == ""-security"" goto execCmd
shift
echo Using Security Manager
set SECURITY_POLICY_FILE=%CATALINA_BASE%\conf\catalina.policy
goto execCmd

:doRun
shift
if not ""%1"" == ""-security"" goto execCmd
shift
echo Using Security Manager
set SECURITY_POLICY_FILE=%CATALINA_BASE%\conf\catalina.policy
goto execCmd

:doStart
shift
if not "%OS%" == "Windows_NT" goto noTitle
set _EXECJAVA=start "Tomcat" %_RUNJAVA%
goto gotTitle
:noTitle
set _EXECJAVA=start %_RUNJAVA%
:gotTitle
if not ""%1"" == ""-security"" goto execCmd
shift
echo Using Security Manager
set SECURITY_POLICY_FILE=%CATALINA_BASE%\conf\catalina.policy
goto execCmd

:doStop
shift
set ACTION=stop
goto execCmd

:execCmd
rem Get remaining unshifted command line arguments and save them in the
set CMD_LINE_ARGS=
:setArgs
if ""%1""=="""" goto doneSetArgs
set CMD_LINE_ARGS=%CMD_LINE_ARGS% %1
shift
goto setArgs
:doneSetArgs

rem Execute Java with the applicable properties
if not "%JPDA%" == "" goto doJpda
if not "%SECURITY_POLICY_FILE%" == "" goto doSecurity

%_EXECJAVA% %JAVA_OPTS% %CATALINA_OPTS% %DEBUG_OPTS% -Djava.endorsed.dirs="%JAVA_ENDORSED_DIRS%" -classpath "%CLASSPATH%" -
```

```
Dcatalina.base="%CATALINA_BASE%" -Dcatalina.home="%CATALINA_HOME%" -
Djava.io.tmpdir="%CATALINA_TMPDIR%" %MAINCLASS% %CMD_LINE_ARGS%
%ACTION%
goto end
:doSecurity
%_EXECJAVA% %JAVA_OPTS% %CATALINA_OPTS% %DEBUG_OPTS% -
Djava.endorsed.dirs="%JAVA_ENDORSED_DIRS%" -classpath "%CLASSPATH%" -
Djava.security.manager -Djava.security.policy=="%SECURITY_POLICY_FILE%" -
Dcatalina.base="%CATALINA_BASE%" -Dcatalina.home="%CATALINA_HOME%" -
Djava.io.tmpdir="%CATALINA_TMPDIR%" %MAINCLASS% %CMD_LINE_ARGS%
%ACTION%
goto end
:doJpda
if not "%SECURITY_POLICY_FILE%" == "" goto doSecurityJpda
%_EXECJAVA% %JAVA_OPTS% %CATALINA_OPTS% -Xdebug -
Xrunjdwp:transport=%JPDA_TRANSPORT%,address=%JPDA_ADDRESS%,server=y,s
uspend=n %DEBUG_OPTS% -Djava.endorsed.dirs="%JAVA_ENDORSED_DIRS%" -
classpath "%CLASSPATH%" -Dcatalina.base="%CATALINA_BASE%" -
Dcatalina.home="%CATALINA_HOME%" -Djava.io.tmpdir="%CATALINA_TMPDIR%"
%MAINCLASS% %CMD_LINE_ARGS% %ACTION%
goto end
:doSecurityJpda
%_EXECJAVA% %JAVA_OPTS% %CATALINA_OPTS% -
Xrunjdwp:transport=%JPDA_TRANSPORT%,address=%JPDA_ADDRESS%,server=y,s
uspend=n %DEBUG_OPTS% -Djava.endorsed.dirs="%JAVA_ENDORSED_DIRS%" -
classpath "%CLASSPATH%" -Djava.security.manager -
Djava.security.policy=="%SECURITY_POLICY_FILE%" -
Dcatalina.base="%CATALINA_BASE%" -Dcatalina.home="%CATALINA_HOME%" -
Djava.io.tmpdir="%CATALINA_TMPDIR%" %MAINCLASS% %CMD_LINE_ARGS%
%ACTION%
goto end

:end
```