Technical Guide 2.7.00 SP1

Public Health Information Network Messaging System (PHINMS)

Version 2.7.00 SP1

Prepared by:
U.S. Department of Health & Human Services

August 24, 2007

EXECUTIVE SUMMARY

Public health involves many organizations throughout the PHIN (Public Health Information Network), working together to protect and advance the public's health. These organizations need to use the Internet to securely exchange sensitive data between varieties of different public health information systems. The exchange of data, also known as "Messaging" is enabled through messages created using special file formats and a standard vocabulary. The exchange uses a common approach to security and encryption, methods for dealing with a variety of firewalls, and Internet protection schemes. The system provides a standard way for addressing and routing content, a standard and consistent way for information systems to confirm an exchange.

The Centers for Disease Control and Prevention (CDC) PHINMS (Public Health Information Network Messaging System) is the software which makes this work. The system securely sends and receives sensitive data over the Internet to the public health information systems.

The Technical Reference Guide provides advanced instructions for configuring the PHINMS 2.7.00 SP1 application.

REVISION HISTORY

VERSION #	IMPLEMENTER	DATE	EXPLANATION
2.6.00	Wendy Fama	08/01/06	Create 2.6.00 Technical guide.
2.6.00	Wendy Fama	08/01/06	Technical Review.
2.7.00	Travis Mayo	11/01/06	Technical Review.
2.7.00	Wendy Fama	11/13/06	Technical Review.
2.7.00 SP1	Wendy Fama	03/15/07	Updated document pertaining to 2.7.00 SP1.
2.7.00 SP1	Travis Mayo	03/15/07	Updated document pertaining to 2.7.00 SP1.
2.7.00 SP1	Wendy Fama	03/28/07	Added scripts.
2.7.00 SP1	Wendy Fama	04/25/07	Referenced ports listed in Implementation Guide.
2.7.00 SP1	Wendy Fama	05/04/07	Updated text and graphics.
2.7.00 SP1	Rajeev Seenappa	05/04/07	Reveiwed document.
2.7.00 SP1	Wendy Fama	08/24/07	Update Oracle Script.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYM LIST

CDC	Centers for Disease Control and Prevention
CPA	Collaboration Protocol Agreement
ebXML	Electronic Business Extensible Markup Language
ErrorQ	Error Queue
LDAP	Lightweight Directory Access Protocol
PHIN	Public Health Information Network
PHINMS	Public Health Information Network Messaging System
TransportQ	Transport Queue
URL	Uniform Resource Locator
WorkerQ	Worker Queue
XML	Extensible Markup Language

1.0 INTRODUCTION

The Centers for Disease Control and Prevention (CDC) Public Health Information Network Messaging System (PHINMS) Technical Reference Guide assists with manually performing outside the graphical user interface configurations. Ensure the most recent versions are referenced from the PHINMS website.

Note: Navigate to www.cdc.gov/phin/phinms when this manual references the PHINMS website.

1.1 Communiqués

The PHINMS team responds to user's communiqués. Send questions, suggestions, and/or comments concerning PHINMS support or documentation to the PHINMS website using the Contact PHINMS email link located at the top of the home page.

2.0 ADVANCED INSTALLATION

This section covers the PHINMS advanced installation features.

2.1 Upgrading vs. Install

A full install simply creates a PHINMS directory in a user-specified location and configures the PHINMS 2.7 Apache Tomcat service with an automatic startup type. When the upgrade option during the PHINMS Installation is chosen, the following happens:

1. PHINMS copies the installation files to the 2.7.0 folder in the location specified by the user. (Ex. c:\specified_phinms_dir\2.7.0\),
2. PHINMS copies and updates the configuration files from a previous 2.5.01 or 2.6 installation into the new installation directory,
3. PHINMS stops the existing PHINMS 2.x Apache Tomcat service and configures its startup type to be manual,
4. PHINMS creates a new PHINMS 2.7 Apache Tomcat and sets the startup type to automatic, and
5. all previous installation files/directories are preserved for the purpose of restoring to the previous version if necessary.

2.2 Typical vs. Custom Install

The PHINMS application has two types of installation modes, Typical and Custom. The PHINMS Implementation Guide on the PHINMS website http://www.cdc.gov/phin/phinms documents detailed instructions for a typical installation.

The following options pertain to the Custom Installation:

1. Select components to enable.
   - Sender Only - the option enables the Sender service and disables the Receiver. The Receiver can only be enabled by reinstalling PHINMS.
   - Receiver Only - the option to enables the Receiver and disables the Sender. The Sender can only be enabled by reinstalling PHINMS.
   - Sender and Receiver - the option installs both the Sender and Receiver services.
2. Install PHINMS as a windows service.
   - Select **Yes** and the PHINMS application installs the PHINMS 2.7 Apache Tomcat application service into the windows service control panel. This enables the PHINMS application to run even when a user is not logged into the system.
   - Select No and the PHINMS application must be started from a cmd window using the phinms\2.7.0\tomcat-5.0.19\bin\startup.bat file. If the PHINMS application needs to be installed in the windows service control panel after selecting **No**, refer to Section 2.3.

2.3 Manually Install Service

Some situations, such as tuning the Java memory setting, editing the Windows service display name, or simply installing the PHINMS as a Windows service after a custom install may require removing and reinstalling the PHINMS Apache Tomcat service manually by the PHINMS Administrator. If the service does not currently exist as a Windows Service, proceed to step 3.

1. run <install-root>\phinms\2.7.0\tomcat-5.0.19\installservice.bat remove,
2. edit the desired configurations in the installservice.bat file,
   - increase/decrease the numbers according to desired memory usage by Tomcat -- **JvmMs 500m --JvmMx 500m**,
   - change the Windows Service Display Name - Display Name "PHINMS2.7 Apache Tomcat" which can be changed to any value, and

Note: The display name needs to be changed when more than one instance of PHINMS are installed.

3. run <install-root>\phinms\2.7.0\tomcat-5.0.19\installservice.bat install.

2.4 PartyID

A PartyID is used to provide the instance with a unique ID when sending messages to the CDC. In the case of multiple PHINMS instances, the PartyID must be unique for each PHINMS install. Each PHINMS installation must have its own PartyID.

Each and every instance of PHINMS sending messages to the CDC must have a registered PartyID from the CDC. Instances not sending data to the CDC may create an unofficial PartyID. This is used to test and become familiar with the PHINMS application when multiple installations are configured. Once the site becomes comfortable using the application, apply for a registered PartyID when messages are going to be sent to the CDC. If the servers were installed without a registered PartyID, the PHINMS application software must be removed and re-installed using the registered PartyID. The instructions for requesting a PartyID are located in the PHINMS Implementation Guide or on the web site at http://www.cdc.gov/phin/phinms.

2.5 Domain Name

The domain name is an added identifier maintained for future compatibility with PHINMS products. It is not currently used; however, it is recommended to specify the domain name of the organization. If no domain name is available, simply use "test.test".

2.6 Directories

PHINMS 2.6.00 and future versions have a standardized directory structure different from PHINMS 2.5 and earlier. This change is designed to allow backward compatibility for future installations. When upgrading previous versions, PHINMS 2.7.00 SP1 copies the configuration files from the previous instance and uses them to automatically configure 2.7.00 SP1.

The installation folders from previous versions are not removed from the system. This provides a contingency plan if the 2.7.00 SP1 installation is unsuccessful.

2.7 Dual Sender Configuration

Dual Sender Configuration allows the installation of multiple PHINMS instances which can be hosted on the same machine. The PHINMS application is installed and configured in different directories.

1. install PHINMS to the directory for example - C:\Program Files\PhinmsPrd,
2. run C:\program files\phinmsprd\2.7.0\tomcat-5.0.19\installservice.bat remove,
3. edit the installservice.bat batch file and configure the following lines:

Note: Add <_description> to the line to uniquely identify the service name.

- set SERVICE_NAME=PHINMS2.7.00_Tomcat5**_PRD**,
- --DisplayName PHINMS 2.7.00 Apache Tomcat**_PRD**, and

4. run C:\program files\phinmsprd\2.7.0\tomcat-5.0.19\installservice.bat install.

The configuration files will automatically be installed into the 2.7.0 subfolder under directories C:\Program Files\PhinmsPrd\2.7.0 and C:\Program Files\PhinmsStg\2.7.0.

Enable the PHINMS application to run as separate instances on same server by editing the following files:

Console.cfg – Location: (..\2.7.0\phinmsstg\tomcat-5.0.19\bin\console.cfg)
HOST_LOCATION=http\://localhost\:5088 (Change port to 5086.)

Note: Requires one line to be changed.

Server.xml – Location: (..\2.7.0\phinmsstg\tomcat-5.0.19\conf\server.xml)

<Server port="5087" shutdown="SHUTDOWN" debug="0"> (Change port to 5085.)

<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector port="5088" (Change port to 5086.)

Note: Requires two lines to be changed.

2.8 Recommended PHINMS Ports

The recommended PHINMS Ports for the Sender and/or Receiver are located in the PHINMS Implementation Guide.

3.0 ADVANCED DATABASE INFORMATION

Section 4 explains procedures for creating a database, cache, and tables for the Transport Queue (TransportQ) and Worker Queue (WorkerQ). A database contains tables which store incoming and outgoing messages. The messaging cache is an index of incoming messages. Table 1 lists the fields used in PHINMS tables.

FIELD NAME	DESCRIPTION	QUEUE
recordId	Unique ID of the record in the table created by the database.	Both
messageId	Application level message identifier.	Both
payloadName	File name of the payload, specified by the Message Sender.	WorkerQ
payloadFile	File name of the payload file of an outgoing message relative to a local directory such as myinputs.txt.	TransportQ
payloadBinaryContent	Image field written to by the Receiver servlet.	WorkerQ
payloadContent	Used only when the payloadFile field is not specified. Populates the contents of a file within the table.	TransportQ
payloadTextContent	Text field populated if textPayload=true in the servicemap entry.	WorkerQ
destinationFilename	The name of the payload file when it is stored on the Receiver/handler.	TransportQ
localFilename	File written to disk instead of a database when payloadToDisk =true.	WorkerQ
routeInfo	Points to the routemap table which points to the message route. Maps to a CPA, a configuration file which maps to the uniform resource locator (URL) of the Message Receiver.	TransportQ
service	Electronic Business Extensible Markup Language (ebXML) service name.	Both
action	ebXML action.	Both
arguments	Arguments specified by the Message Sender.	Both
fromPartyId	PartyID of the Message Sender.	WorkerQ
messageCreationTime	Time when record was created, in UTC format.	TransportQ
messageRecipient	Recipient's ID specified by the Sender in the TransportQ_out.	Both
errorCode	Error code.	WorkerQ
errorMessage	Error message.	WorkerQ
processingStatus	Initial value of the status of record created queued.	Both
applicationStatus	Status of the application.	Both
encryption	The value is Yes if payload is encrypted and No if it is not.	Both
receivedTime	Time when payload was received, in UTC format.	WorkerQ
lastUpdateTime	Time when record was last updated, in UTC format.	WorkerQ
processId	Identifies the process processing the record.	WorkerQ
signature	If Yes, extensible markup language (XML) signature is applied to the payload.	TransportQ
publicKeyLdapAddress	LDAP address of the LDAP directory server.	TransportQ
publicKeyLdapBaseDN	LDAP Base Distinguished Name of the public key such as o=.	TransportQ
publicKeyLdapDN	LDAP Distinguished Name of the public key such as cn=.	TransportQ
certificateURL	URL of a recipeint's public key certificate.	TransportQ
transportStatus	Transport level status.	TransportQ
transportErrorCode	Error code describing the transport failure.	TransportQ
applicationErrorCode	The error code returned by the service/action in a synchronous manner.	TransportQ
applicationResponse	The synchronous response returned by the service/action.	TransportQ
messageSentTime	Time when the message was sent, in UTC format.	TransportQ
messageReceivedTime	Time when the message was received, in UTC format.	TransportQ
responseMessageId	Message ID of the response message in the rout-not-read scenario.	TransportQ
responseArguments	Used in the Route-not-Read scenario to convey arguments being sent by	TransportQ

FIELD NAME	DESCRIPTION	QUEUE
	a Message Sender to a receiving client.	
responseLocalFile	The response to a poll type request which may contain a payload file in the Route-not-Read scenario	TransportQ
responseFilename	The response file name in the Route-not-Read scenario.	TransportQ
responseContent	Used when the sender.xml configuration file in the Message Sender specifies the response payload should be written into a database field instead of to a disk.	TransportQ
responseMessageOrigin	The PartyID of the party originating the message in the Route-not-Read scenario.	TransportQ
responseMessageSignature	The PartyID of the party signing the message in the Route-not-Read scenario.	TransportQ
priority	An integer indicating the request's priority.	TransportQ

Table 1. PHINMS Table Fields

3.1 Create MSSQL Database

The scripts for creating MSSQL and/or Oracle databases are located in Appendix A.

3.2 Create MSSQL Tables

Appendix A contains various scripts for creating tables used with MSSQL and/or Oracle.

3.3 Transport Codes

A transport status code is sent back to the TransportQ when a message is delivered or processed. If an error occurs during the delivery of a message an error code is sent back to the TransportQ. Table 2 describes the transport status codes and Table 3 describes the transport error codes.

TRANSPORT STATUS CODE	DESCRIPTION
Success	Message send or receive operation succeeded.
Failure	Message send or receive operation failed.

Table 2. Transport Status Codes

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 failed to map a service on the Message Receiver.
CheckSumFailure*	File checksum verification failure at the Message Receiver.

Table 3. Transport Error Codes

Note: The asterisk (*) symbol indicates a custom error code meaning the code is not in the ebXML specifications.

4.0 FILE SYSTEM-BASED TRANSPORT QUEUES

File System-Based TransportQ is a folder-based option used to send and receive messages. This option is used as a substitute for database sending and receiving configurations.

4.1 XML File Descriptor

An example XML File Descriptor is shown below.

```
<fileDescriptor>
<recordId>22</recordId>
<payloadFile>D:\phinms\shared\outgoing\test.txt</payloadFile>
<payloadContent></payloadContent>
<destinationFilename>test.txt</destinationFilename>
<routeInfo>CDCStaging</routeInfo>
<service>Router</service>
<action>send</action>
<arguments>XXDOHelr</arguments>
<messageRecipient>XXDOH</messageRecipient>
<messageCreationTime>time</messageCreationTime>
<encryption>yes</encryption>
<signature>yes</signature>
<publicKeyLdapAddress>directory.verisign.com:389</publicKeyLdapAddress>
<publicKeyLdapBaseDN>o=Centers for Disease Control and Prevention </publicKeyLdapBaseDN>
<publicKeyLdapDN>cn=cdc phinms</publicKeyLdapDN>
<acknowledgementFile>D:\phinms\shared\acknowledgments\ack_send.xml
</acknowledgementFile>
</fileDescriptor>
```

4.2 XML File Descriptor Response

An example XML File Descriptor response is shown below.

```
<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>Poller's_PartyID</responseMessageOrigin>
</acknowledgement>
```

4.3 File-Based TransportQ

When the TransportQ 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.

4.4 Name-Value Based File Descriptor

An example name-value based file-descriptor is shown below.

recordId=22
payloadFile=d:\\phinms\\outgoing\\test.txt
destinationFilename=test.txt
routeInfo=CDCStaging
service=Router
action=send
arguments=XXDOHelr
messageRecipient=XXDOH

4.5 Sending File Response

An example of a response (written to the acknowledgement file specified in the outgoing file descriptor) from a file send operation is shown below.

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

5.0 JDBC DRIVER SYNTAX INFORMATION

5.1 SQL

JDBC Driver: com.microsoft.jdbc.sqlserver.SQLServerDriver

Database URL PreFix: jdbc:microsoft:sqlserver

Database URL Suffix: //(Computer Name):(Port);DatabaseName=(Name of Database)

5.2 HSQL

JDBC Drive: org.hsqldb.jdbcDriver

Database URL PreFix: jdbc:hsqldb:hsql:

Database URL Suffix: //(Computer Name):(Port)/(Name of Database)

5.3 MS Access

JDBC Driver: sun.jdbc.odbc.JdbcOdbcDriver

Database URL PreFix: jdbc:odbc:

Database URL Suffix: PhinmsgAccessDSN270

5.4 MySQL

JDBC Driver: com.mysql.jdbc.Driver

Database URL PreFix: jdbc:mysql:

Database URL Suffix: //(Computer Name):(Port)/(Name of Database)

5.5 Oracle

JDBC Driver: oracle.jdbc.driver.OracleDriver

Database URL PreFix: jdbc:oracle:thin:

Database URL Suffix: @(Computer Name):(Port):(Name of Database)

Note: Remove the parathenses "**( )**" from the URL Suffix when the **Computer Name**, **Port**, and **Name of Database** are inserted. Ensure all the other characters are not removed.

6.0 ADVANCED MESSAGE SENDER CONFIGURATION

6.1 Sender Configuration Fields

The figures in Section 6.1 show all screen shots and fields used to configure the PHINMS Sender. An explanation of the field will be noted beside the field identified.

6.1.1 General

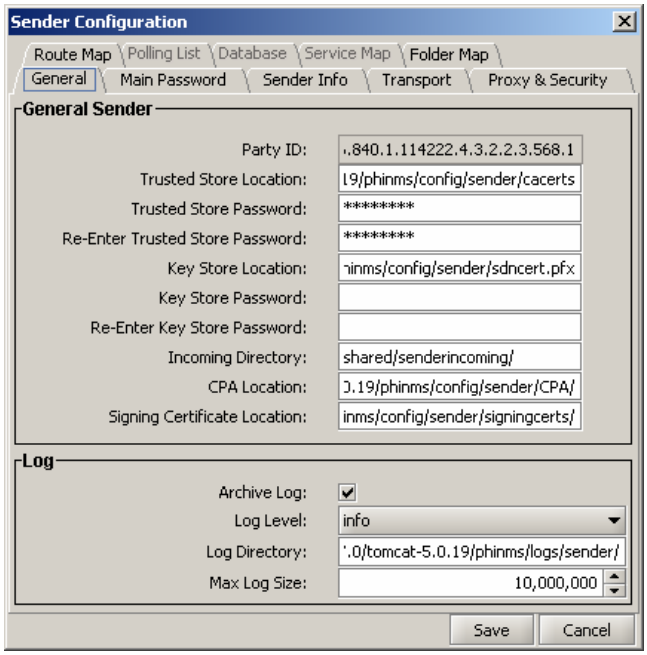

Figure 6.1. General - Sender Configuration

- Party ID - the Sender's Organization ID.
- Trusted Store Location - the relative path of the Message Sender's trusted store.
- Trusted Store Password - the password for the Trusted Store file.
- Re-Enter Trusted Store Password - re-enter the Trusted Store password.
- Key Store Location - the relative path name of the Message Sender's Key Store including the certificate name and extension.
- Key Store Password - password assigned to the Sender's Key Store.
- Re-Enter Key Store Password - re-enter the Sender's Key Store password.
- Incoming Directory - the relative path name where incoming Route-not-Read files are stored.
- CPA Location - the relative path name to the directory storing the CPA.
- Signing Certificate Location - the relative path which signed certificates are stored.

- Archive Log - archives log files when they reach their size limit.
- Log Level - the amount of detail written to the log file.
- Log Directory - the relative path where PHINMS stores the Sender log files.
- Max Log Size - indicates the maximum size of a log file.

6.1.2 Main Password

The Main Password for the Sender Configuration is used to encrypt/protect all the username/password configured through the console. The encrypted file is located at phinms/config/sender/senderpasswds.

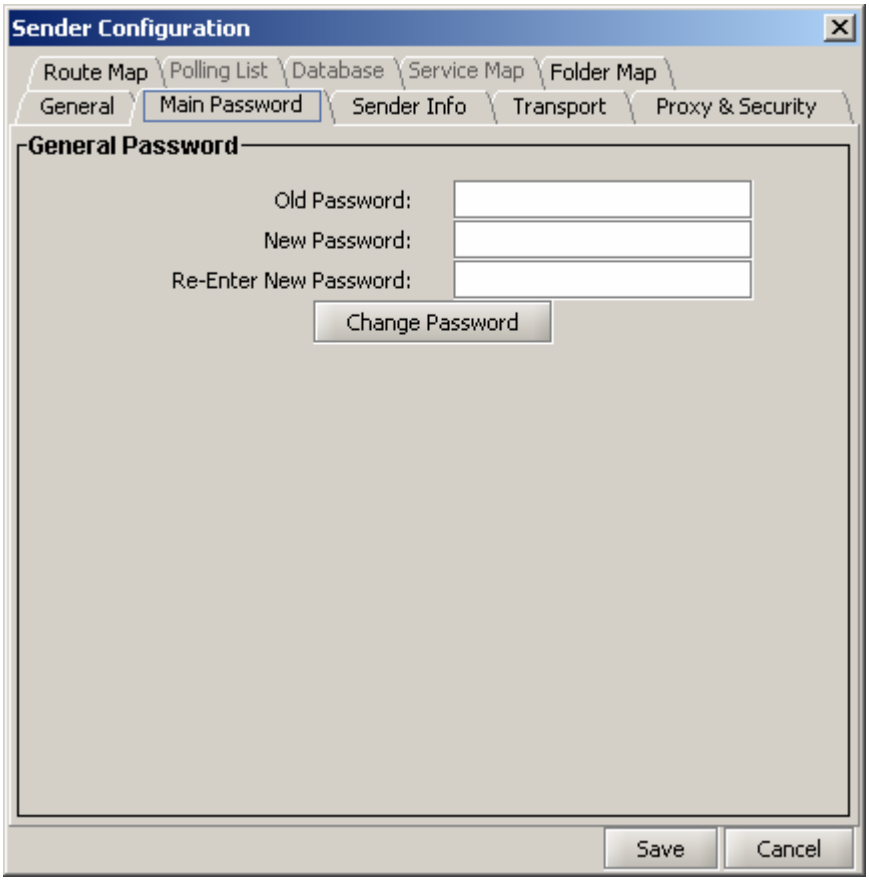

Figure 6.2. Main Password - Sender Configuration

- Old Password - enter old password assigned.
- New Password - enter a new password.
- Re-Enter New Password - confirm new password entered.

6.1.3 Sender Info

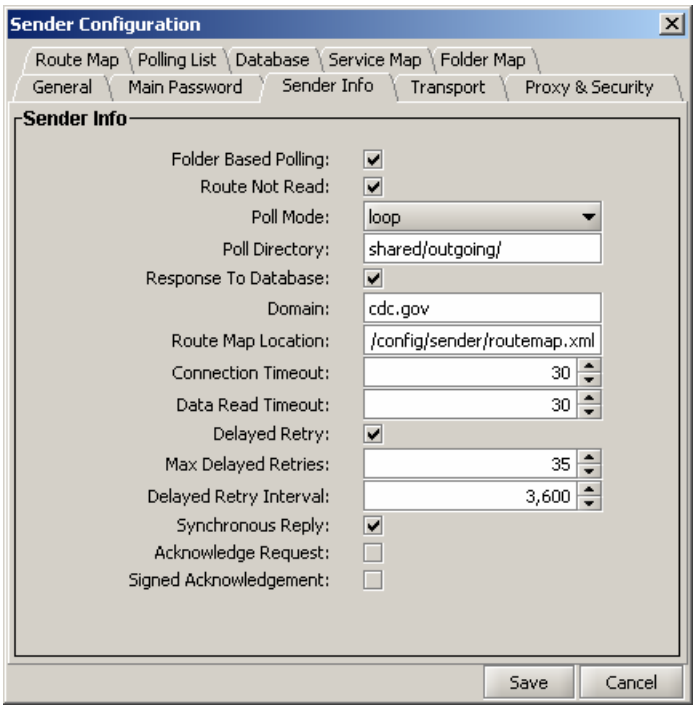

Figure 6.3. Sender Info - Sender Configuration

- Folder Based Polling - messages are sent from a folder instead of a database.
- Route-not-Read - automatically polls an intermediary Receiver for incoming messages.

Note: When Route-not-Read is checked, the Polling List, Database, and Service Map tabs are activated. Refer to Sections 6.1.18, 6.1.19, and 6.1.20.

- Poll Mode - select loop and the outgoing directory is continually polled, select once and it is polled one time.
- Poll Directory - the relative path where outgoing payload files are stored.
- Response To Database - if checked, the response file is written to the TransportQ, if not checked the response file is written to disk.
- Domain - the domain name where PHINMS is installed.
- Route Map Location - the relative path name of the Route Map.
- Connection Timeout - the number of milliseconds the Message Sender waits before timing out the attempt to connect to the SSL.
- Data Read Timeout - the number of seconds the Message Sender waits to receive a response from the Message Receiver before the Message Sender times out.

- Delayed Retry - select the check box to automatically retry sending failed records at intervals specified by the value in the Max Delayed Retries field.
- Max Delayed Retries - the maximum number of times the Message Sender will automatically retry sending failed records.
- Delayed Retry Interval - the number of seconds before failed records are re-queued to be resent.
- Synchronous Reply - select when the Message Sender wants to wait for a reply from the Message Receiver for the previous message before the Message Sender attempts to send the next message.
- Acknowledge Request - select and the Message Receiver sends the Message Sender an acknowledgement after the Message Receiver receives the message.
- Signed Acknowledgement - the digitally signed acknowledgement which comes from the Message Receiver to the Message Sender.

6.1.4 Transport

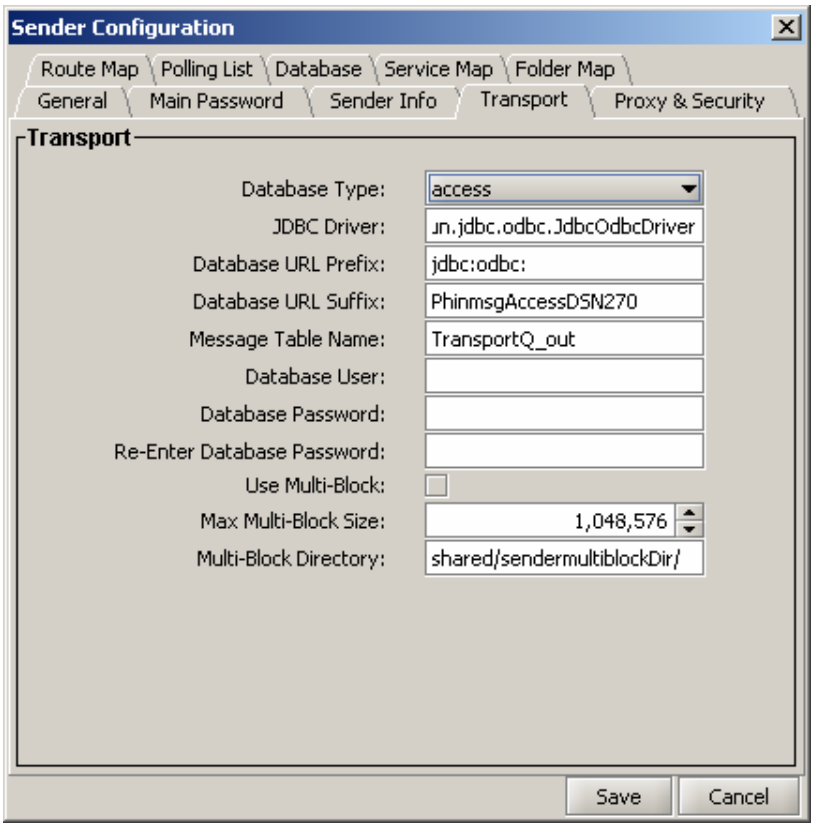

Figure 6.4. Transport - Sender Configuration

- Database Type - the type of database which connects the database pool.
- JDBC Driver - the JDBC driver name which transports queue database table uses.

- Database URL Prefix - contains the database driver portion of the database connection URL.
- Database URL Suffix - contains the server name portion of the database connection URL.
- Message Table Name - the TransportQ table name.
- Database User - the database user id.
- Database Password - the database user password.
- Re-Enter Database Password - re-enter the database user password.
- Use Multi-Block - enables chunking.
- Max Multi-Block Size - the chunk size.
- Multi-Block Directory - the location of stored message chunks.

6.1.5 Proxy & Security

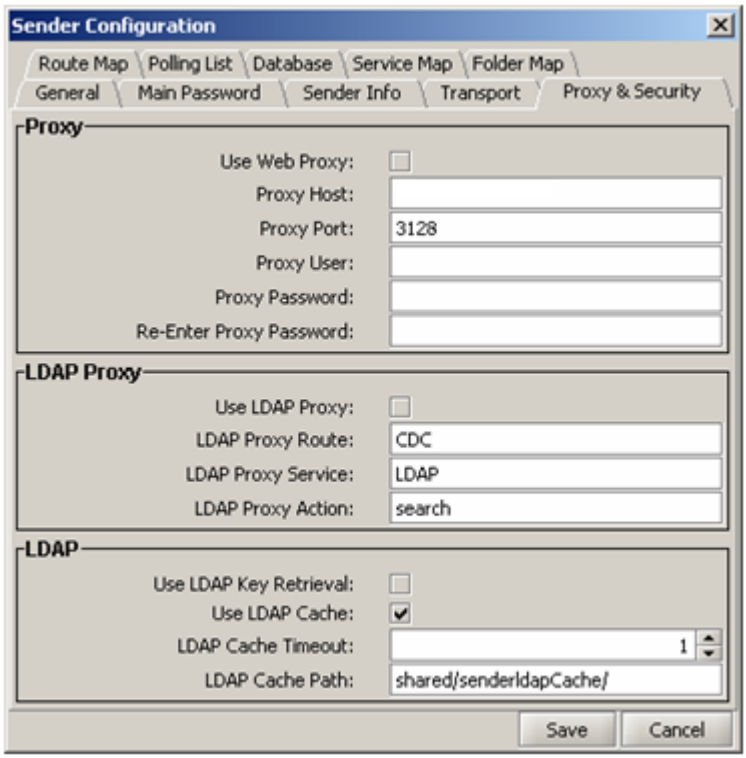

Figure 6.5. Proxy & Security - Sender Configuration

- Use Web Proxy - select when the PHINMS client wants to send HTTP requests through a proxy server.
- Proxy Host - the host name or the IP address of the proxy server.
- Proxy Port - the port number used by the proxy.

- Proxy User - the name of the proxy user.
- Proxy Password - the password for the proxy user.
- Re-Enter Proxy Password - re-enter proxy user password.
- Use LDAP Proxy - select when PHINMS clients are unable to send LDAP lookup requests to directory.verisign.com over port 389.
- LDAP Proxy Route - the route used when the proxy server sends the LDAP request.
- LDAP Proxy Service - the ebxml service proxying the LDAP request.
- LDAP Proxy Action - the ebxml action performed when proxying the LDAP request.
- Use LDAP Key Retrieval - select and the LDAP search is used to retrieve the public key of the recipient to encrypt the message.
- Use LDAP Cache - when selected, the public keys retrieved from the LDAP search are stored.
- LDAP Cache Timeout - defines how many hours to wait before refreshing the LDAP cache.
- LDAP Cache Path - the location of the LDAP cache.

6.1.6 Route Map

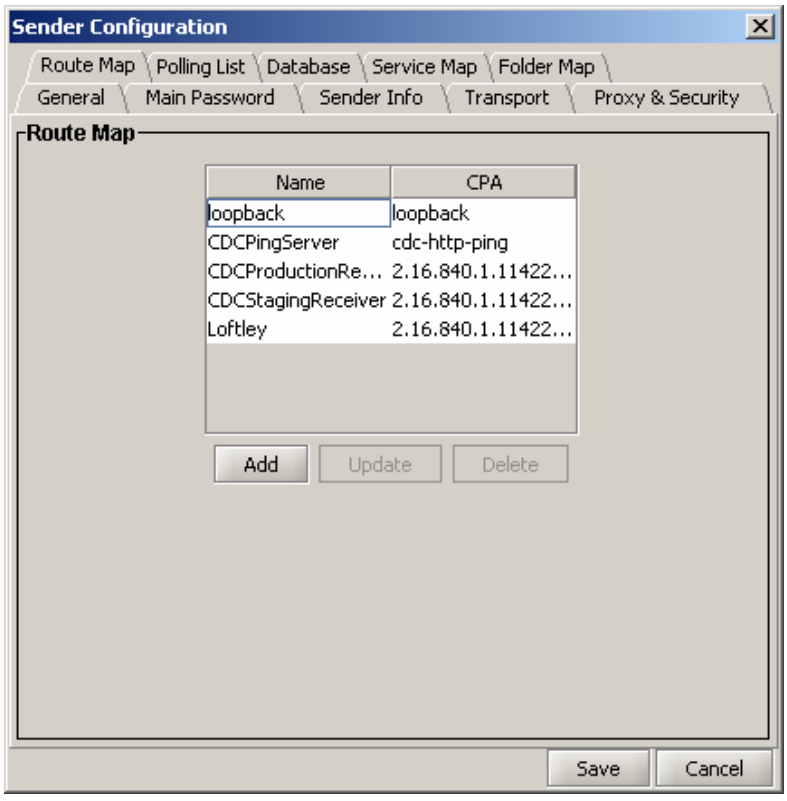

Figure 6.6. Route Map - Sender Configuration

- Add - used to add a new Route Map. Figure 6.7 displays when **ADD** is selected.
- Update - used to update an existing Route Map.
- Delete - used to delete an existing Route Map.

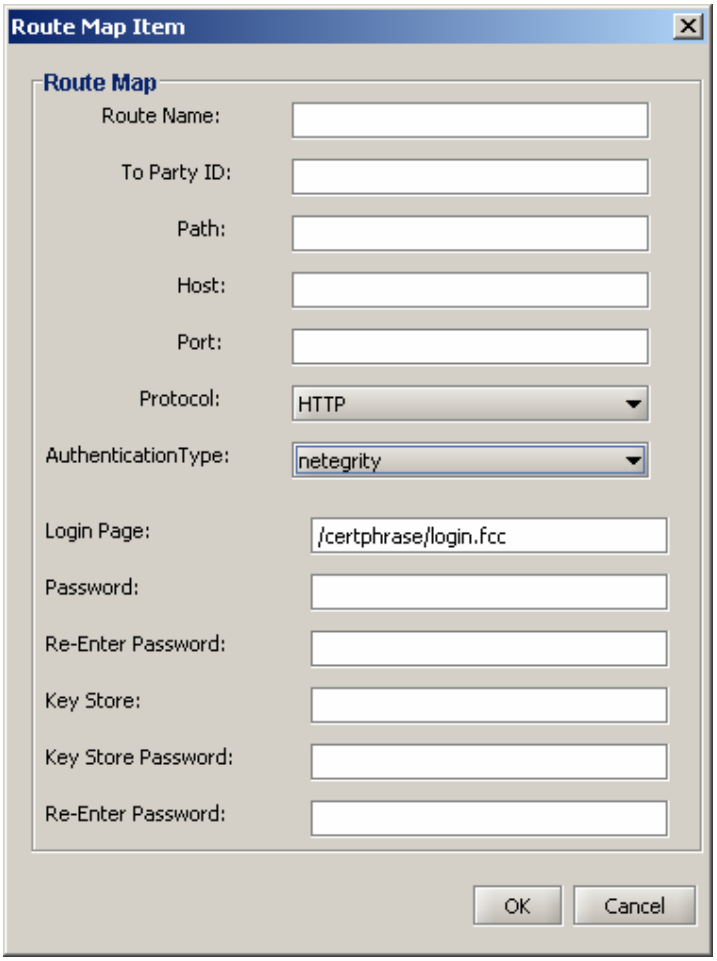

Figure 6.7. Route Map - Sender Configuration

- Route Name - used to define a particular route or distinction intended to send a message to the Receiver.
- To Party ID - the Receiver's PartyID.
- Path - the path to the Receiver's servlet (example: receiver/receivefile).
- Host - the name of the proxy server.
- Port - the Receiver's proxy server port.
- Protocol - either HTTP or HTTPS.
- Authentication Type - select what kind of security measures are used to verify identity.

- Login Page - the URL of the login page.
- Password - the challenge phrase created when enrolling for a CDC SDN Digital Certificate.
- Re-Enter Password - re-enter the challenge phrase.
- Key Store - the full path name of the Message Sender's Key Store including the certificate name and extension.
- Key Store Password - password for the user's Key Store.
- Re-Enter Password - re-enter password name.

6.1.7 Polling List

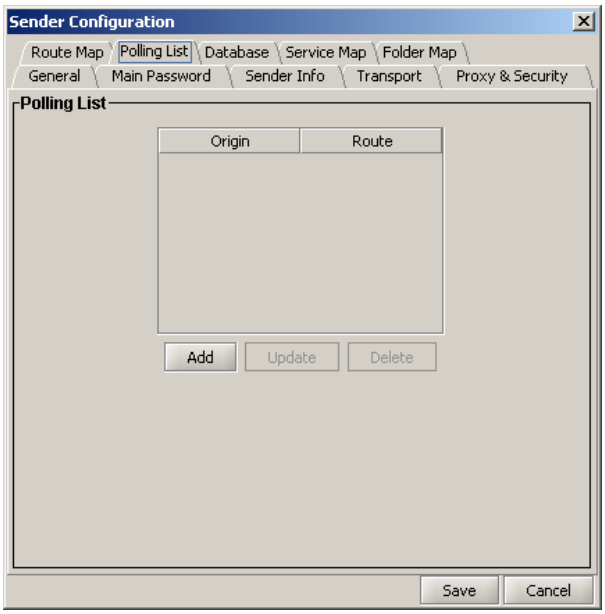

Figure 6.8. Polling List - Sender Configuration

- Add - used to add a new Polling List. Figure 6.9 displays when the **ADD** tab is selected.
- Update - used to update an existing Polling List.
- Delete - used to delete an existing Polling List.

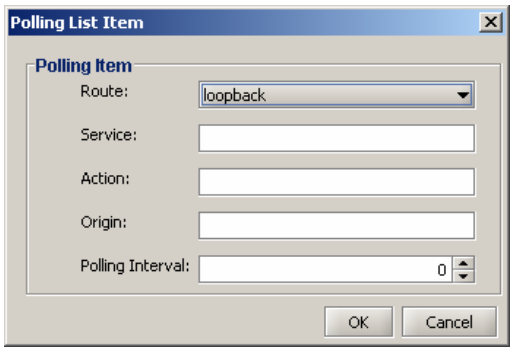

Figure 6.9. Polling List Item - Sender Configuration

- Route - the route name of the polling destination as it appears in the route map.
- Service - the name of the service used at the polling destination.
- Action - the name of the action used at the polling destination.
- Origin - the name of the Message poller.
- Polling Interval - the number of seconds between polls.

6.1.8 Database

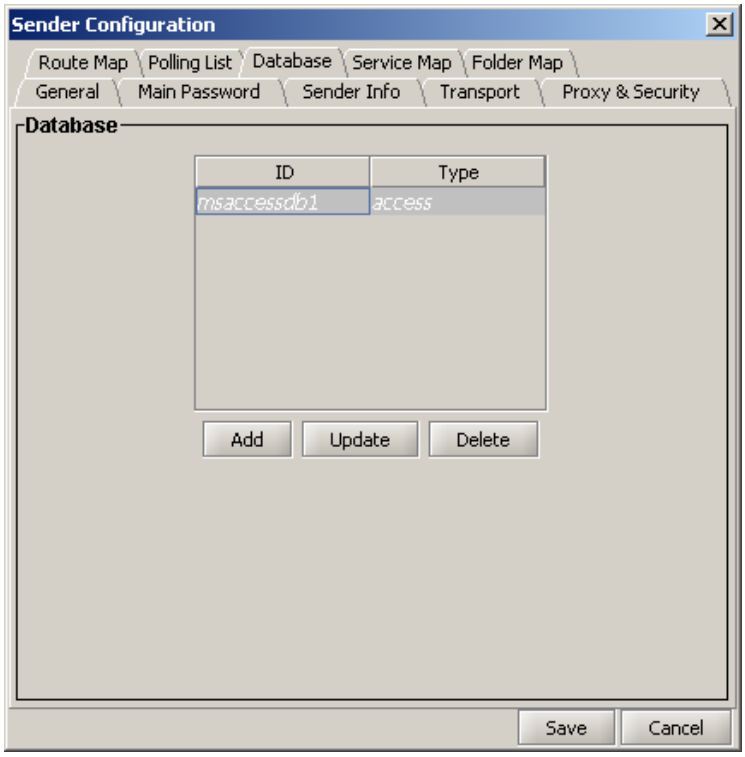

Figure 6.10. Database - Sender Configuration

- Add - used to add a new Database connection. Figure 6.11 is displayed when the **ADD** tab is selected.
- Update - used to update an existing Database connection containing the WorkerQ.
- Delete - used to remove an existing Database connection.

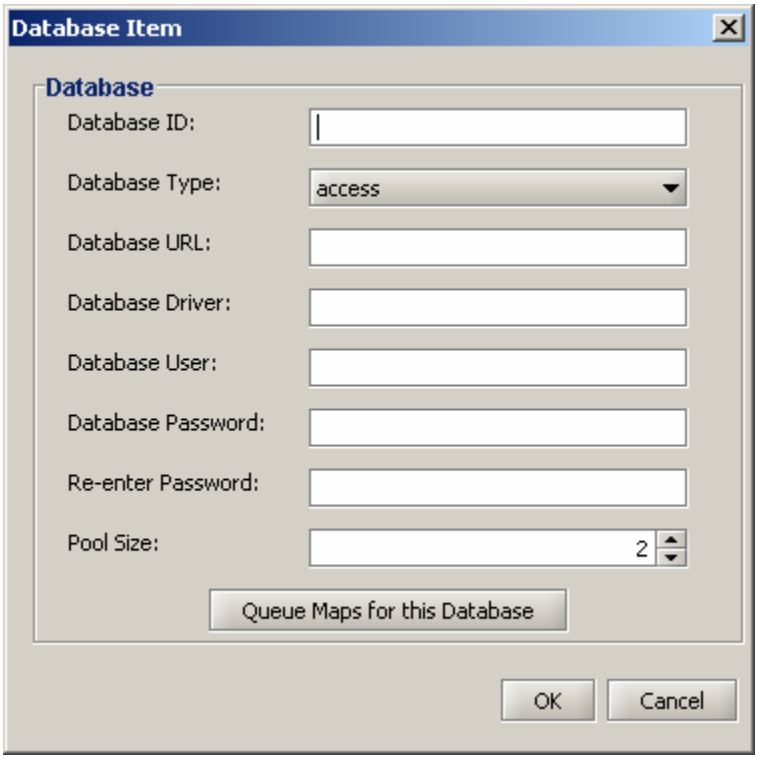

Figure 6.11. Database Item - Sender Configuration

- Database ID - the unique user defined name of the database connection pool.
- Database Type - the type of database to which the database pool connects.
- Database URL - the entire database connection URL.
- Database Driver - the type of JDBC driver.
- Database User - the database user name.
- Database Password - the database user password.
- Re-enter Password - re-enter the database user password.
- Pool Size - the number of database connections to open.
- Queue Maps for this Database - Figure 6.12 displays when the **Queue Maps for this Database** tab is selected.

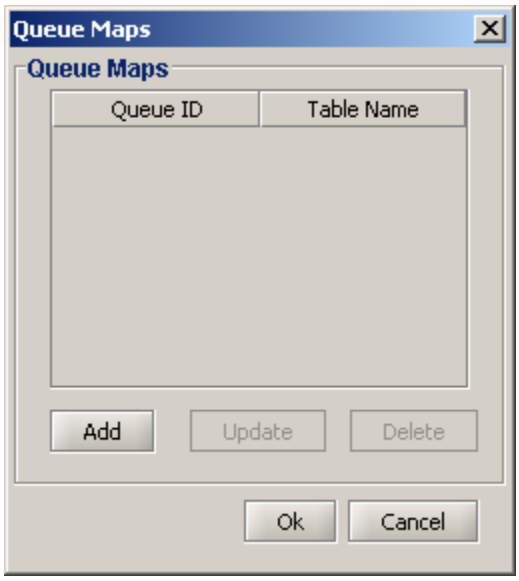

Figure 6.12. Queue Maps - Sender Configuration

- Add - used to add a new Queue Map. Figure 6.13 displays when the **Add** tab is selected.
- Update - used to update an existing Queue Map.
- Delete - used to delete an existing Queue Map.

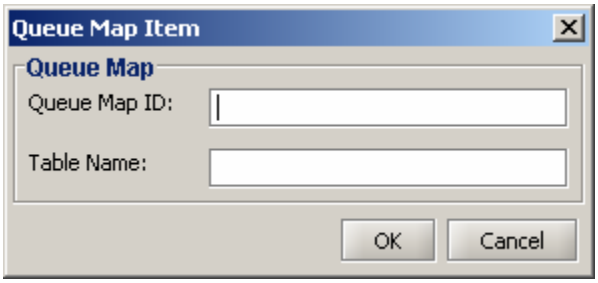

Figure 6.13. Queue Map Item - Sender Configuration

- Queue Map ID - the unique ID created by the user referenced by the service map entry.
- Table Name - the database table name to which the service/action pair is linked.

6.1.9 Service Map

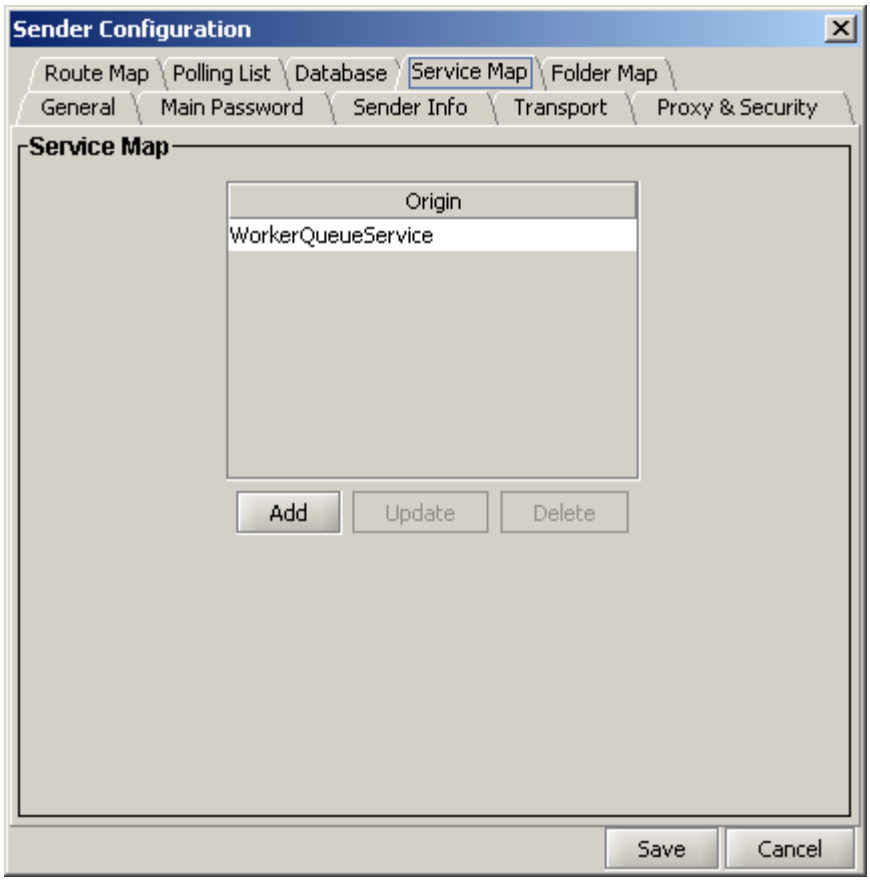

Figure 6.14. Service Map - Sender Configuration

- Add - used to add a new Service Map. Figure 6.15 is displayed when the **ADD** tab is selected.
- Update - used to update an existing Service Map.
- Delete - used to delete an existing Service Map.

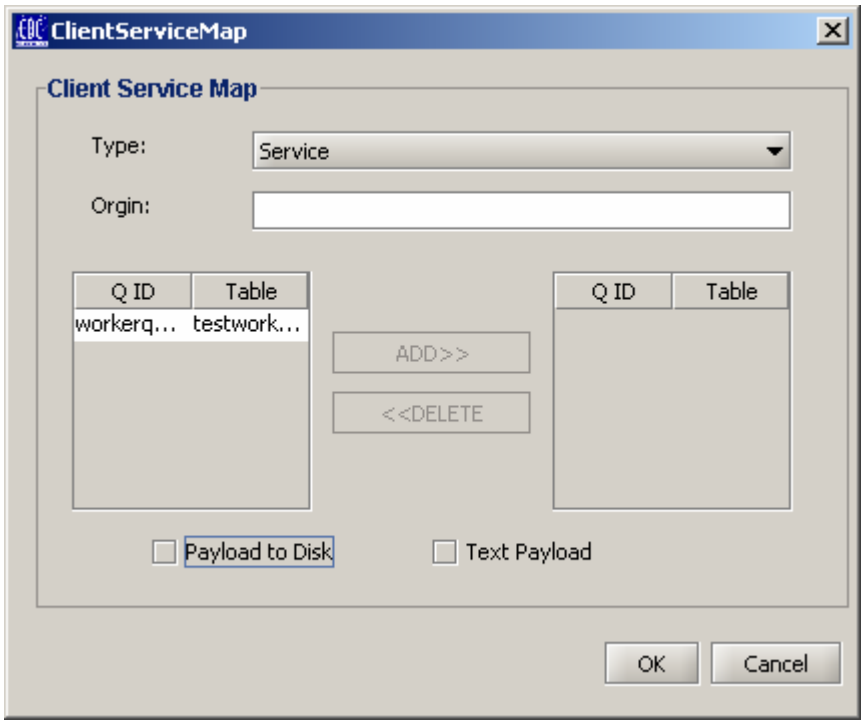

Figure 6.15. Client Service Map - Sender Configuration

- Type - the type of service entry: Service or ErrorQueue.
- Origin - argument specified in the message by Sender.
- Add - select to add to Client Service Map.
- Delete - select to delete from Client Service Map.
- Payload to Disk - select when the incoming payload needs to be written to disk and provide the location of the data to be stored , if not selected, payload will be written to the database.
- Text payload - select when the incoming payload is text, if not selected, the payload will be binary.

6.1.10 Folder Map

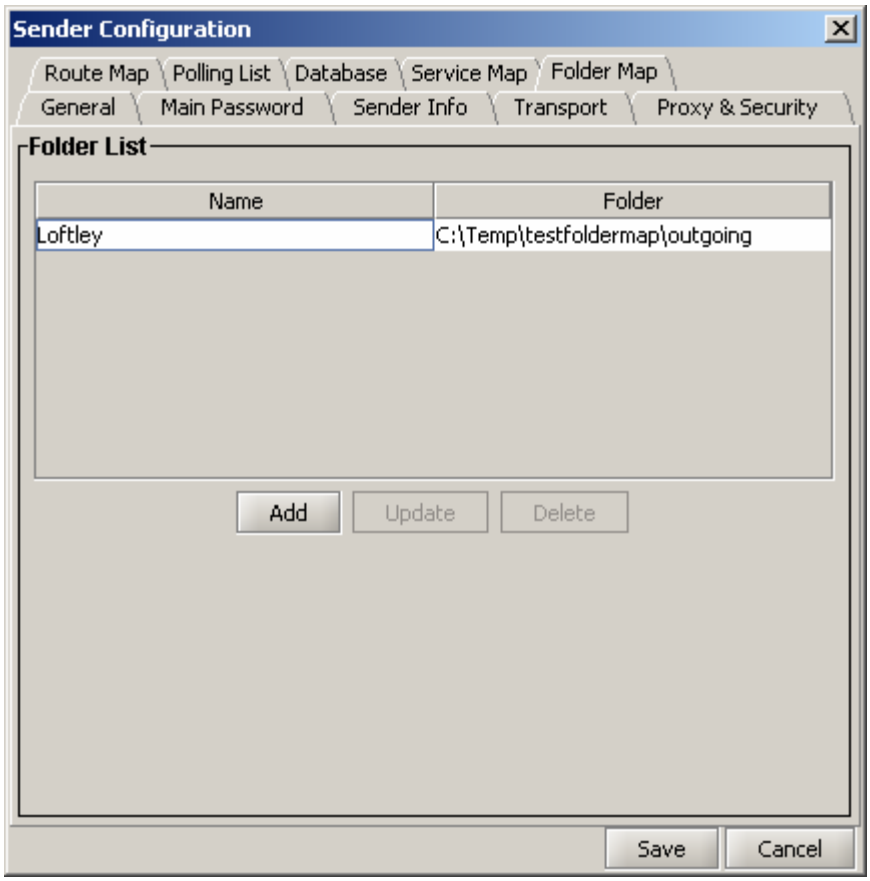

Figure 6.16. Folder List - Sender Configuration

- Add - used to add a new Folder List. Figure 6.17 is displayed when the **ADD** tab is selected.
- Update - used to update an existing Folder List.
- Delete - used to delete an existing Folder List.

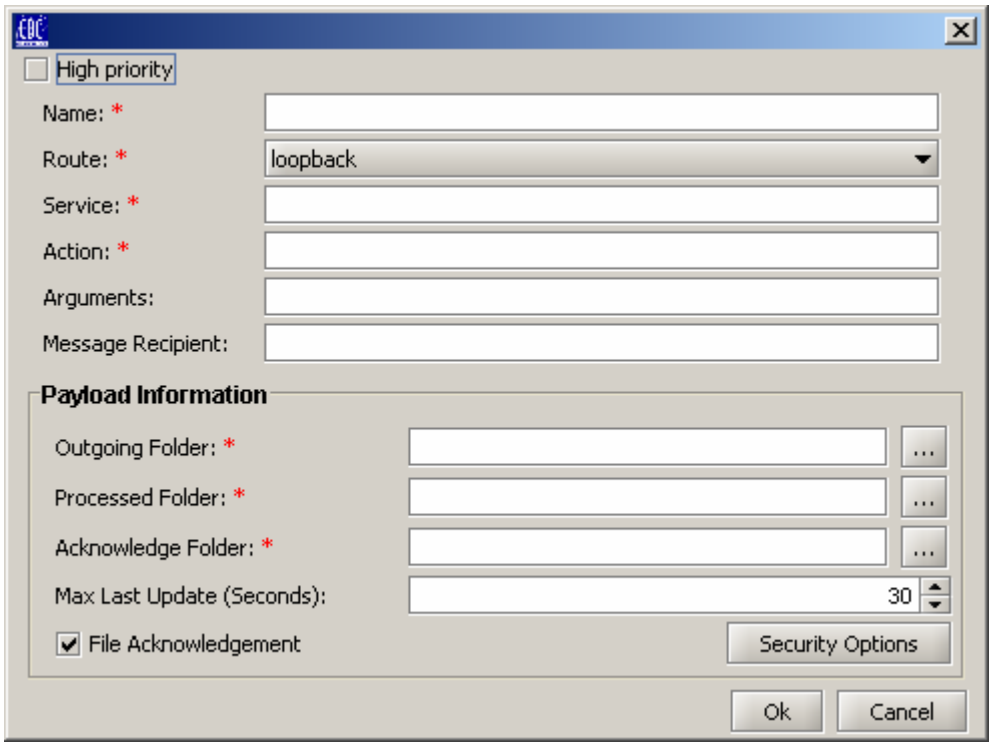

Figure 6.17. Folder List Properties - Sender Configuration

- High Priority - check when the message needs to be prioritized to be sent immediately.
- Name - the assigned name created for the folder polling creation.
- Route - the route to send the message.
- Service - the name of the service used at the polling destination.
- Action - the name of the action used at the polling destination.
- Arguments - additional information used to send to the message handler.
- Message Recipient - identifies the Message Receiver's PartyID to send the message when using Route-not-Read.
- Outgoing Folder - used to store messages to be sent.
- Processed Folder - regional file which messages have been processed and are stored.
- Acknowledge Folder - stores the message receipt from the Receiver.
- Max Last Update (Seconds) - the amount of time the system will wait for the file to be written before processing the message.
- File Acknowledgement - check to activate the Acknowledge folder.
- Security Options - Figure 6.18 displays when the **Security Options** tab is selected.

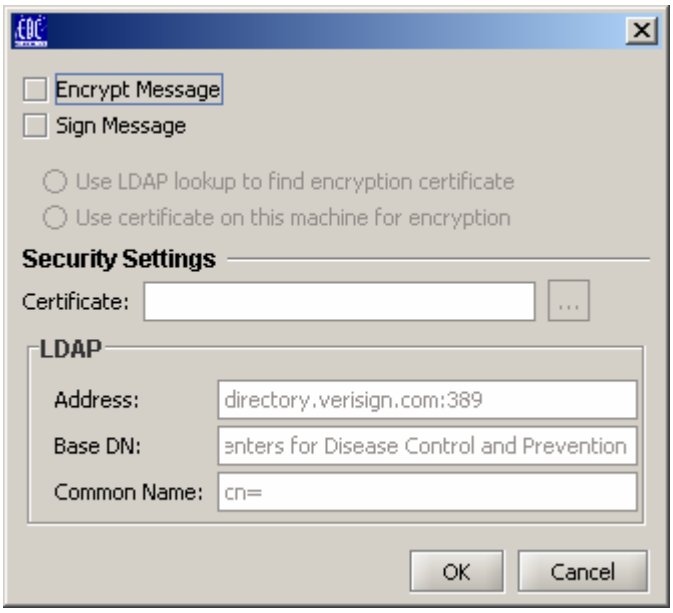

Figure 6.18. Security Options - Sender Configuration

- Encrypt Message - select to encrypt message.
- Sign Message - select to assign a XML signature to a message.
- Use LDAP lookup to find encryption certificate - select to have LDAP will locate the certificate.
- Use certificate on this machine for encryption - select to use a stored certificate.
- Certificate - URL of the message recipient's public key certificate.
- Address - the location of the entity holding the public key.
- Base DN - the recipient’s organization’s name.
- Common Name - the recipient’s common name used to query the LDAP server and download the Receiver’s public key used to encrpty the message.

7.0 ADVANCED MESSAGE RECEIVER CONFIGURATION

7.1 Receiver Configuration Fields

The figures in Section 7.1 show all screen shots and fields used to configure the PHINMS Receiver. An explanation of the field will be noted beside the field identified.

7.1.1 General

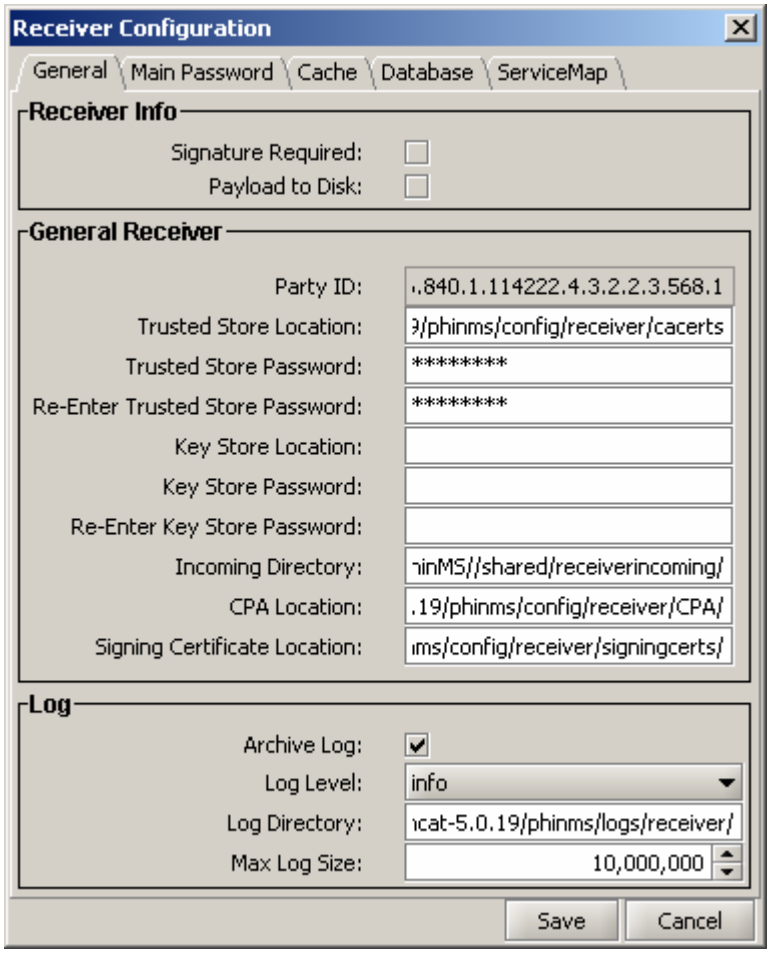

Figure 7.1. General - Receiver Configuration

- Party ID - the Receiver's Organization ID.
- Trusted Store Location - the full path of the Message Receiver's trusted store.
- Trusted Store Password - the password for the Trusted Store file.
- Re-Enter Trusted Store Password - re-enter the Trusted Store password.
- Key Store Location - the full path name of the Message Sender's Key Store including the certificate name and prefix.

- Key Store Password - password assigned to the Receiver's Key Store.
- Re-enter Key Store Password - re-enter the Receiver's Key Store password.
- Incoming Directory - the full name where incoming files are stored.
- CPA Location - the full path name to the directory storing the CPA.
- Signing Certificate Location - the directory in which signed certificates are stored.
- Archive Log - archive log files when they reach their size limit.
- Log Level - the amount of detail written to the log file.
- Log Directory - the full path where PHINMS stores the Receiver log files.
- Max Log Size - indicates the maximum size of a log file.

7.1.2 Main Password

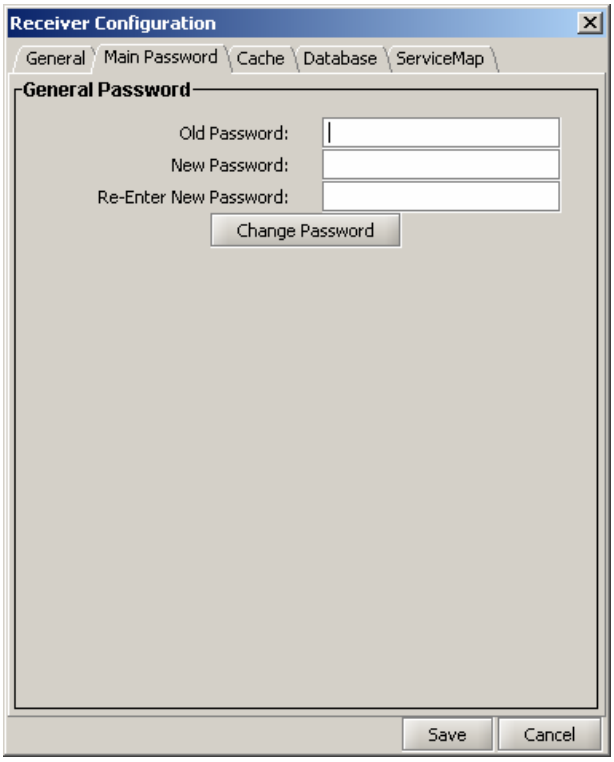

Figure 7.2. Main Password - Receiver Configuration

- Old Password - enter old password assigned
- New Password - enter a new password
- Re-Enter New Password - confirm new password entered

7.1.3 Cache

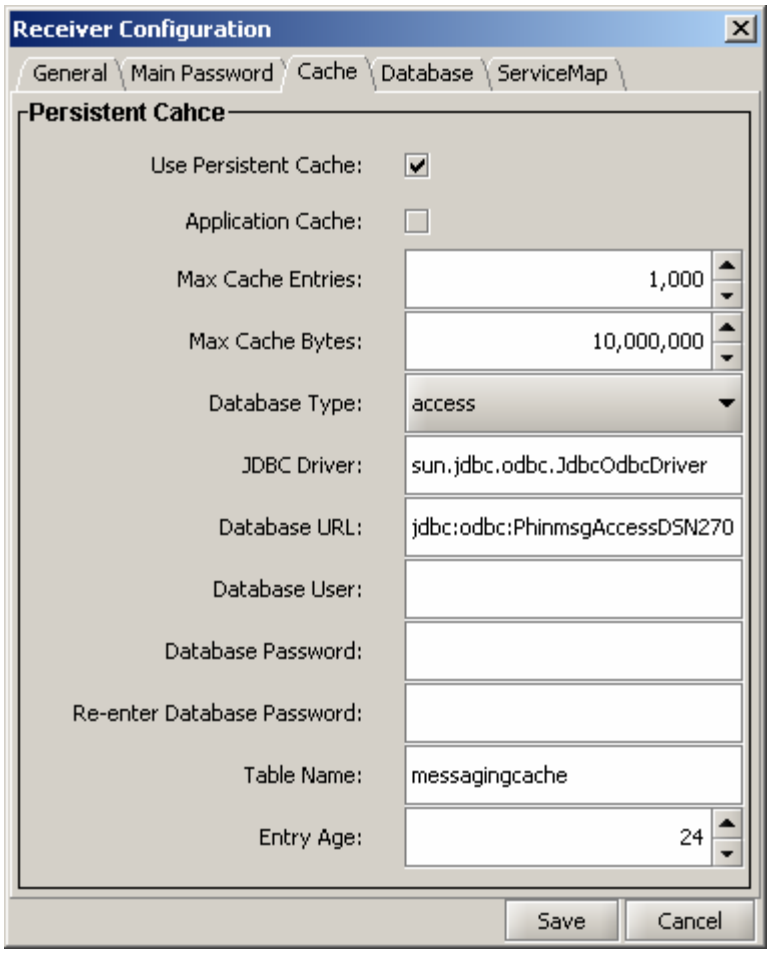

Figure 7.3. Persistent Cache - Receiver Configuration

- Use Persistent Cache - select to cache RecordID and message information to a database which prevents duplicate messages from being received.
- Application Cache - select to add the Message ID with the Record ID and PartyID as an additional field to determine if the message is already sent to its destination service.
- Max Cache Entries - the maximum number of entries which can be cached in persistent cache.
- Max Cache Bytes - the maximum number of bytes which can be cached in persistent cache.
- Database Type - the type of database used for persistent cache.
- JDBC Driver - the JDBC driver to use with the database.
- Database URL - the entire database connection URL.

- Database User - the database user name.
- Database Password - the database user password.
- Re-enter Database Password - re-enter the database user password.
- Table name - the database table name used for the message cache.
- Entry Age - the maximum time in hours when entries are cached.

7.1.4 Database

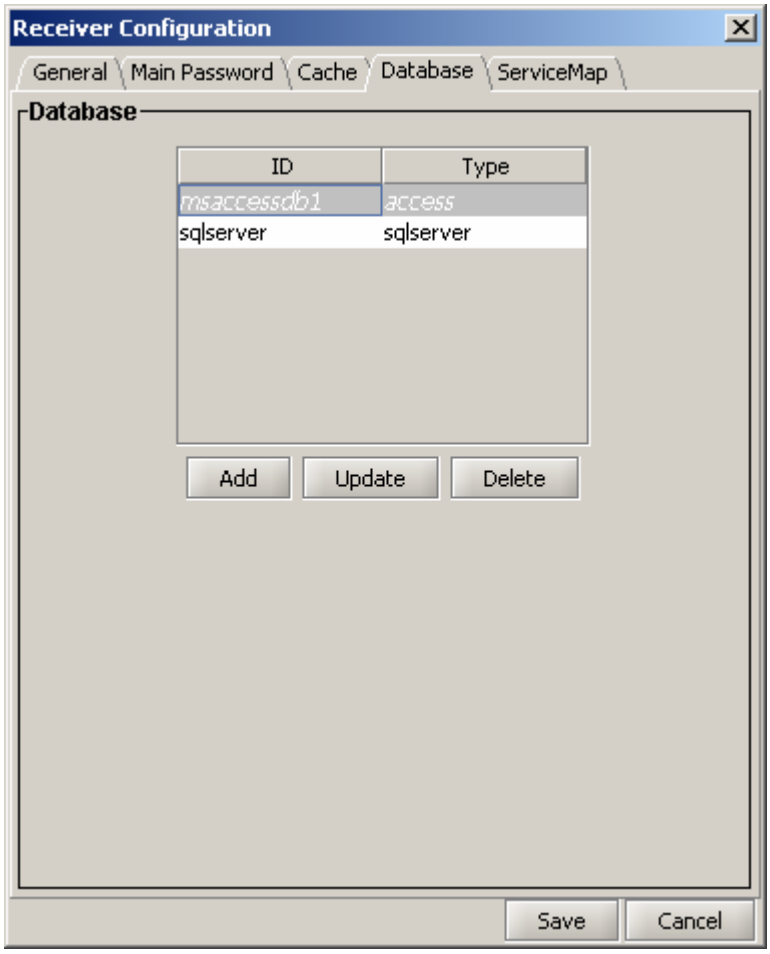

Figure 7.4. Database - Receiver Configuration

- Add - used to add a new Database. Figure 7.5 is displayed when the **ADD** tab is selected.
- Update - used to update an existing database connection.
- Delete - used to remove an existing database connection.

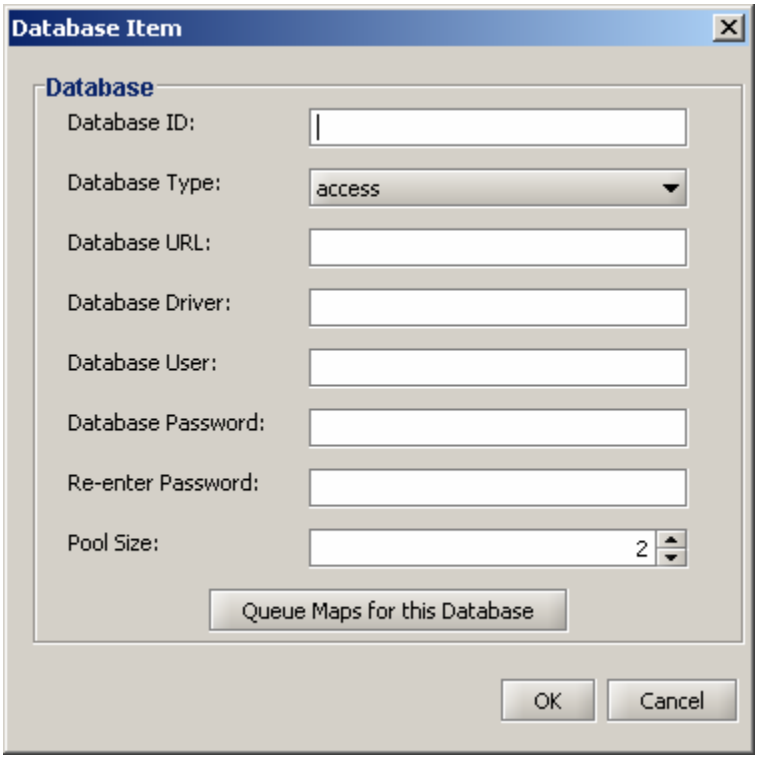

Figure 7.5. Database Item - Receiver Configuration

- Database ID - the unique user defined name of the database connection pool.
- Database Type - the type of database to which the database pool connects.
- Database URL - the entire database connection URL.
- Database Driver - the type of JDBC driver.
- Database User - the database user name.
- Database Password - the database user password.
- Re-enter Password - re-enter the database user password.
- Pool Size - the number of database connections to open.
- Queue Maps for this Database - Figure 7.6 displays when the **Queue Maps for this Database** tab is selected.

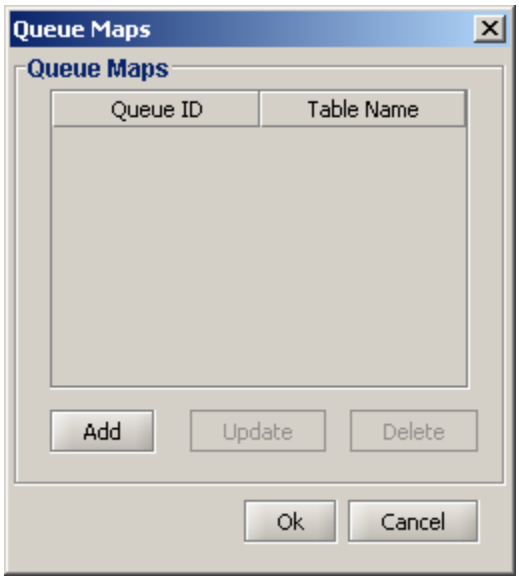

Figure 7.6. Queue Maps - Receiver Configuration

- Add - used to add a new Queue Map. Figure 7.7 displays when the **Add** tab is selected.
- Update - used to update an existing Queue Map.
- Delete - used to delete an existing Queue Map.

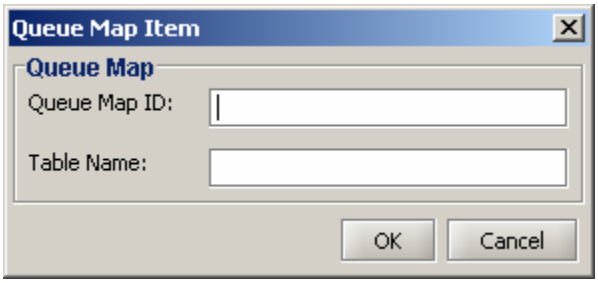

Figure 7.7. Queue Map Item - Receiver Configuration

- Queue Map ID - the unique ID created by the user referenced by the service map entry.
- Table Name - the database table name to which the service/action pair is linked.

7.1.5 Service Map

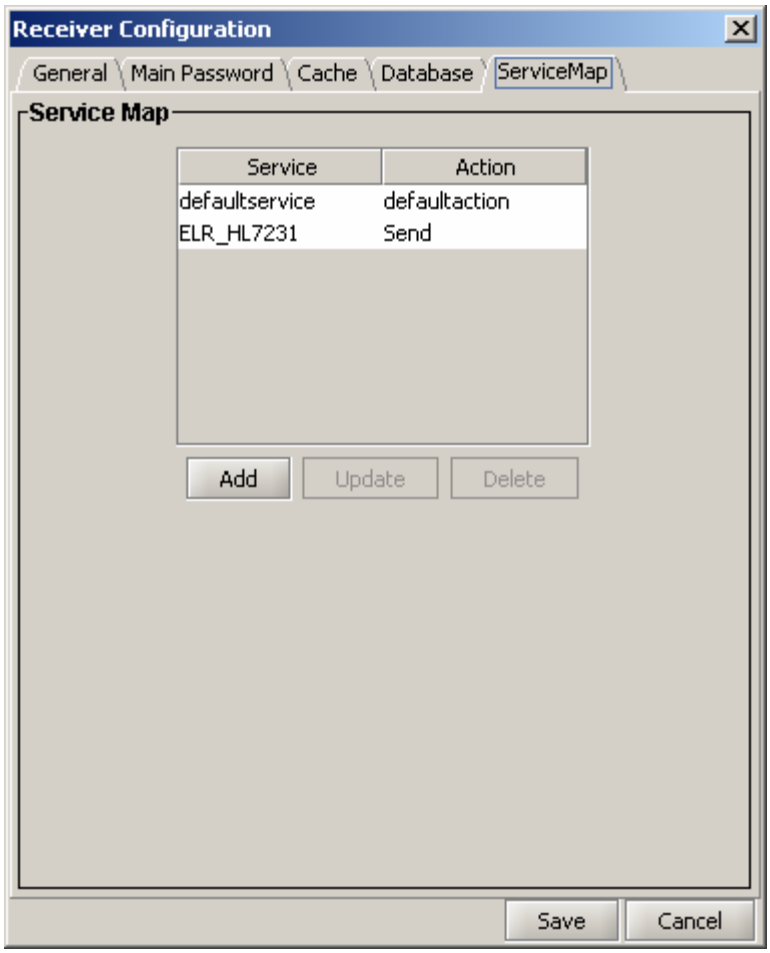

Figure 7.8. Service Map - Receiver Configuration

- Add - used to add a new Service Map. Figure 7.9 is displayed when the **ADD** tab is selected.
- Update - used to update an existing Service Map
- Delete - used to delete an existing Service Map

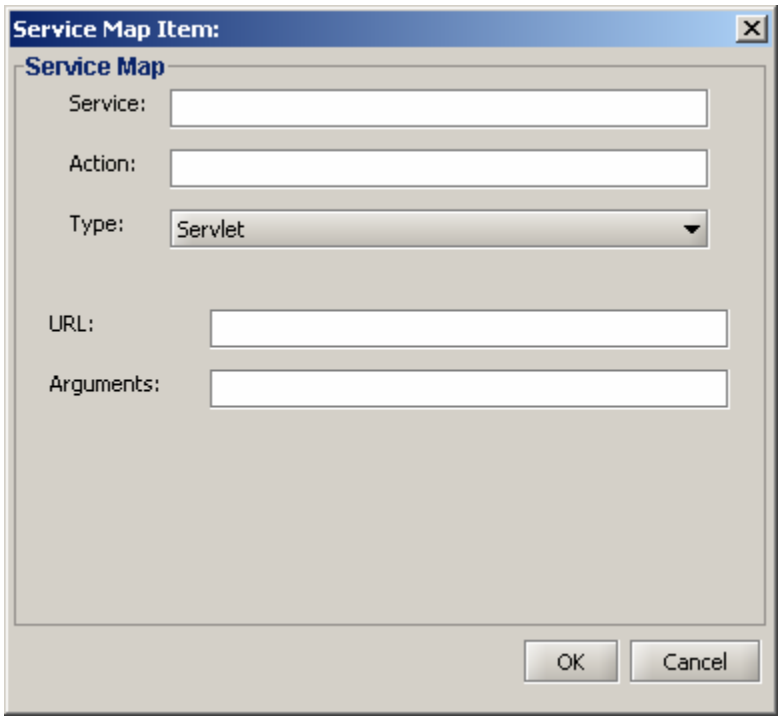

Figure 7.9. Service Map Item - Receiver Configuration

- Service - the name of the service invoked by the Message Receiver when receiving a message
- Action - the action or method within the service invoked by the Message Receiver when receiving a message
- Type - choose WorkerQ, Servlet, or Error Queue service based on the Message Sender's and Message Receiver's agreement
- URL - the URL which maps to the message handler
- Arguments - additional information used to send to the message handler

8.0 ADVANCED CONSOLE INFORMATION

8.1 Tools Menu

PHINMS has added a new Tools Menu for enhanced configuration control. The new Tools Menu allows the user to complete the following:

- import and export CPA files,
- view Sender and Receiver Logs,
- import and export Configuration files,
- import Trusted Certs,
- import JDBC Jar Files, and
- change Login Password.

8.2 Transport Status and Error Codes

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

STATUS	DESCRIPTION
Success	Message Send/Receive operation successful.
Failure	Message Send/Receive operation failure.

Table 4. Status 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.

Table 5. Error Codes

Note: (*) Custom error codes not in ebXML specification.

APPENDIX A
TABLE SCRIPTS

The table scripts identified in the following sections are examples for a database administrator to use to create tables for Senders and Receivers. The PHINMS account permissions needed to create tables are as follows:

- read,
- write,
- insert, and
- update.

A.1 MSSQL SCRIPTS

Section A.1 lists the scripts used to create MSSQL databases.

A.1.1 TransportQ Table - Sender

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

```
    [priority] [int] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

A.1.2 WorkerQ Table - Sender

```
CREATE TABLE [dbo].[message_inq] (
    [recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
    [messageId] [varchar] (255) NULL ,
    [payloadName] [varchar] (255) NULL ,
    [payloadBinaryContent] [image] NULL ,
    [payloadTextContent] [text] NULL ,
    [localFileName] [varchar] (255) NOT NULL ,
    [service] [varchar] (255) NOT NULL ,
    [action] [varchar] (255) NOT NULL ,
    [arguments] [varchar] (255) NULL ,
    [fromPartyId] [varchar] (255) NULL ,
    [messageRecipient] [varchar] (255) NULL ,
    [errorCode] [varchar] (255) NULL ,
    [errorMessage] [varchar] (255) NULL ,
    [processingStatus] [varchar] (255) NULL ,
    [applicationStatus] [varchar] (255) NULL ,
    [encryption] [varchar] (10) NOT NULL ,
    [receivedTime] [varchar] (255) NULL ,
    [lastUpdateTime] [varchar] (255) NULL ,
    [processId] [varchar] (255) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

A.1.3 ErrorQ Table - Sender

```
CREATE TABLE [dbo].[PHINMS_errq] (
    [recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
    [messageId] [varchar] (255) NULL ,
    [payloadName] [varchar] (255) NULL ,
    [payloadBinaryContent] [image] NULL ,
    [payloadTextContent] [text] NULL ,
    [localFileName] [varchar] (255) NOT NULL ,
    [service] [varchar] (255) NOT NULL ,
    [action] [varchar] (255) NOT NULL ,
    [arguments] [varchar] (255) NULL ,
    [fromPartyId] [varchar] (255) NULL ,
    [messageRecipient] [varchar] (255) NULL ,
    [errorCode] [varchar] (255) NULL ,
    [errorMessage] [varchar] (255) NULL ,
    [processingStatus] [varchar] (255) NULL ,
    [applicationStatus] [varchar] (255) NULL ,
    [encryption] [varchar] (10) NOT NULL ,
    [receivedTime] [varchar] (255) NULL ,
    [lastUpdateTime] [varchar] (255) NULL ,
    [processId] [varchar] (255) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

A.1.4 Messaging Cache Table - Sender

```
CREATE TABLE [dbo].[messagingcache] (
```

```
    [sequence] [int] IDENTITY (1, 1) NOT NULL ,
    [partyId] [char] (50) NULL ,
    [convId] [char] (50) NULL ,
    [recordId] [char] (50) NULL ,
    [response] [text] NULL ,
    [timestamp] [char] (20) NULL ,
    [status] [char] (10) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

A.1.5 Messaging Queue Table - Receiver

```
CREATE TABLE [dbo].[message_inq] (
    [recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
    [messageId] [varchar] (255) NULL ,
    [payloadName] [varchar] (255) NULL ,
    [payloadBinaryContent] [image] NULL ,
    [payloadTextContent] [text] NULL ,
    [localFileName] [varchar] (255) NOT NULL ,
    [service] [varchar] (255) NOT NULL ,
    [action] [varchar] (255) NOT NULL ,
    [arguments] [varchar] (255) NULL ,
    [fromPartyId] [varchar] (255) NULL ,
    [messageRecipient] [varchar] (255) NULL ,
    [errorCode] [varchar] (255) NULL ,
    [errorMessage] [varchar] (255) NULL ,
    [processingStatus] [varchar] (255) NULL ,
    [applicationStatus] [varchar] (255) NULL ,
    [encryption] [varchar] (10) NOT NULL ,
    [receivedTime] [varchar] (255) NULL ,
    [lastUpdateTime] [varchar] (255) NULL ,
    [processId] [varchar] (255) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

A.1.6 TransportQ Table - Receiver

```
CREATE TABLE [dbo].[message_outq] (
    [recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
    [messageId] [varchar] (255) NULL ,
    [payloadName] [varchar] (255) NULL ,
    [payloadBinaryContent] [image] NULL ,
    [payloadTextContent] [text] NULL ,
    [localFileName] [varchar] (255) NOT NULL ,
    [service] [varchar] (255) NOT NULL ,
    [action] [varchar] (255) NOT NULL ,
    [arguments] [varchar] (255) NULL ,
    [fromPartyId] [varchar] (255) NULL ,
    [messageRecipient] [varchar] (255) NULL ,
    [errorCode] [varchar] (255) NULL ,
    [errorMessage] [varchar] (255) NULL ,
    [processingStatus] [varchar] (255) NULL ,
    [applicationStatus] [varchar] (255) NULL ,
    [encryption] [varchar] (10) NOT NULL ,
    [receivedTime] [varchar] (255) NULL ,
    [lastUpdateTime] [varchar] (255) NULL ,
```

```
    [processId] [varchar] (255) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

A.1.7 ErrorQ Table - Receiver

```
CREATE TABLE [dbo].[message_errq] (
    [recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
    [messageId] [varchar] (255) NULL ,
    [payloadName] [varchar] (255) NULL ,
    [payloadBinaryContent] [image] NULL ,
    [payloadTextContent] [text] NULL ,
    [localFileName] [varchar] (255) NOT NULL ,
    [service] [varchar] (255) NOT NULL ,
    [action] [varchar] (255) NOT NULL ,
    [arguments] [varchar] (255) NULL ,
    [fromPartyId] [varchar] (255) NULL ,
    [messageRecipient] [varchar] (255) NULL ,
    [errorCode] [varchar] (255) NULL ,
    [errorMessage] [varchar] (255) NULL ,
    [processingStatus] [varchar] (255) NULL ,
    [applicationStatus] [varchar] (255) NULL ,
    [encryption] [varchar] (10) NOT NULL ,
    [receivedTime] [varchar] (255) NULL ,
    [lastUpdateTime] [varchar] (255) NULL ,
    [processId] [varchar] (255) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

A.2 ORACLE SCRIPTS

A.2.1 TransportQ Table - Sender

```
CREATE TABLE TransportQ_out (
    recordId number(19,0) NOT NULL ,
    messageId char (255) NULL ,
    payloadFile char (255) NULL ,
    payloadContent BLOB NULL ,
    destinationFilename char (255) NULL ,
    routeInfo char (255) NOT NULL ,
    service char (255) NOT NULL ,
    action char (255) NOT NULL ,
    arguments char (255) NULL ,
    messageRecipient char (255) NULL ,
    messageCreationTime char (255) NULL ,
    encryption char (10) NOT NULL ,
    signature char (10) NOT NULL ,
    publicKeyLdapAddress char (255) NULL ,
    publicKeyLdapBaseDN char (255) NULL ,
    publicKeyLdapDN char (255) NULL ,
    certificateURL char (255) NULL ,
    processingStatus char (255) NULL ,
    transportStatus char (255) NULL ,
    transportErrorCode char (255) NULL ,
    applicationStatus char (255) NULL ,
```

```
    applicationErrorCode char (255) NULL ,
    applicationResponse char (255) NULL ,
    messageSentTime char (255) NULL ,
    messageReceivedTime char (255) NULL ,
    responseMessageId char (255) NULL ,
    responseArguments char (255) NULL ,
    responseLocalFile char (255) NULL ,
    responseFilename char (255) NULL ,
    responseContent BLOB NULL ,
    responseMessageOrigin char (255) NULL ,
    responseMessageSignature char (255) NULL ,
    priority number(10,0) NULL
) ;

CREATE SEQUENCE TransportQ_out_recordId
    START WITH 1
    INCREMENT BY 1;

CREATE TRIGGER TransportQ_out_IDENTITY
before insert on TransportQ_out
for each row
begin
select TransportQ_out_recordId.nextval into :new.recordId from dual;
end;
/
```

A.2.2 WorkerQ Table - Sender

```
CREATE TABLE message_inq (
      recordId number(19,0) NOT NULL ,
      messageId varchar2 (255) NULL ,
      payloadName varchar2 (255) NULL ,
      payloadBinaryContent BLOB NULL ,
      payloadTextContent CLOB NULL ,
      localFileName varchar2 (255) NOT 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 message_inq
ADD PRIMARY KEY (recordId);

CREATE SEQUENCE message_inq_record_count
INCREMENT BY 1
START WITH 1
```

```
MINVALUE 1
MAXVALUE 999999999999999999999999999
NOCYCLE
NOORDER
CACHE 20;

CREATE TRIGGER message_inq_IDENTITY
before insert on message_inq
for each row
begin
select message_inq_record_count.nextval into :new.recordId from dual;
end;
```

A.2.3 ErrorQ Table - Sender

```
CREATE TABLE message_inq (
    recordId number(19,0) NOT NULL ,
    messageId varchar2 (255) NULL ,
    payloadName varchar2 (255) NULL ,
    payloadBinaryContent BLOB NULL ,
    payloadTextContent CLOB NULL ,
    localFileName varchar2 (255) NOT 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
) ;

CREATE SEQUENCE message_inq_recordID
    START WITH 1
    INCREMENT BY 1;

CREATE TRIGGER message_inq_IDENTITY
before insert on message_inq
for each row
begin
select message_inq_recordID.nextval into :new.recordId from dual;
end;
/
```

A.2.4 Messaging Cache Table - Sender

```
CREATE TABLE messagingcache  (
    sequence number(10,0) NOT NULL ,
    partyId char (50) NULL ,
    convId char (50) NULL ,
    recordId char (50) NULL ,
```

```
    response CLOB NULL ,
    timestamp char (20) NULL ,
    status char (10) NULL
) ;

CREATE SEQUENCE messagingcache_sequence
    START WITH 1
    INCREMENT BY 1;

CREATE TRIGGER messagingcache_IDENTITY
before insert on messagingcache
for each row
begin
select messagingcache_sequence.nextval into :new.sequence from dual;
end;
/
```

A.2.5 Messaging Queue Table - Receiver

```
CREATE TABLE message_inq (
    recordId number(19,0) NOT NULL ,
    messageId varchar2 (255) NULL ,
    payloadName varchar2 (255) NULL ,
    payloadBinaryContent BLOB NULL ,
    payloadTextContent CLOB NULL ,
    localFileName varchar2 (255) NOT 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
) ;

CREATE SEQUENCE message_inq_recordID
    START WITH 1
    INCREMENT BY 1;

CREATE TRIGGER message_inq_IDENTITY
before insert on message_inq
for each row
begin
select message_inq_recordID.nextval into :new.recordId from dual;
end;
/
```

A.2.6 TransportQ Table - Receiver

```
CREATE TABLE message_outq (
```

```
    recordId number(19,0) NOT NULL ,
    messageId varchar2 (255) NULL ,
    payloadName varchar2 (255) NULL ,
    payloadBinaryContent BLOB NULL ,
    payloadTextContent CLOB NULL ,
    localFileName varchar2 (255) NOT 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
) ;

CREATE SEQUENCE message_outq_recordID
    START WITH 1
    INCREMENT BY 1;


CREATE TRIGGER message_outq_IDENTITY
before insert on message_outq
for each row
begin
select message_outq_recordID.nextval into :new.recordId from dual;
end;
/
```

A.2.7 ErrorQ Table - Receiver

```
CREATE TABLE message_errq (
    recordId number(19,0) NOT NULL ,
    messageId varchar2 (255) NULL ,
    payloadName varchar2 (255) NULL ,
    payloadBinaryContent BLOB NULL ,
    payloadTextContent CLOB NULL ,
    localFileName varchar2 (255) NOT 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
) ;

CREATE SEQUENCE message_errq_recordID
    START WITH 1
    INCREMENT BY 1;

CREATE TRIGGER message_errq_IDENTITY
before insert on message_errq
for each row
begin
select message_errq_recordID.nextval into :new.recordId from dual;
end;
/
```

A.2.8 Route-not-Read Table - Sender

```
CREATE TABLE broadcast (
    name char (100) NULL ,
    addresses char (1000) NULL
);

CREATE TABLE messagebins (
    recordId number(19,0) NOT NULL ,
    messageId varchar2 (255) NULL ,
    payloadName varchar2 (255) NULL ,
    payloadBinaryContent BLOB NULL ,
    payloadTextContent CLOB NULL ,
    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
);

CREATE SEQUENCE messagebins_recordId
    START WITH 1
    INCREMENT BY 1;

CREATE TRIGGER messagebins_IDENTITY
before insert on messagebins
for each row
begin
select messagebins_recordId.nextval into :new.recordId from dual;
end;
/
```

```
CREATE TABLE partyid_user (
    partyId char (255) NULL ,
    "user" char (255) NULL ,
    sdnuser char (255) NULL
);

CREATE TABLE users (
    name char (100) NULL ,
    description char (255) NULL
);
```

A.3 MYSQL SCRIPTS

A.3.1 TransportQ Table - Sender

```
CREATE TABLE TransportQ_out (
    recordId bigint NOT NULL AUTO_INCREMENT,
    messageId char (255) NULL ,
    payloadFile char (255) NULL ,
    payloadContent LONGBLOB NULL ,
    destinationFilename char (255) NULL ,
    routeInfo char (255) NOT NULL ,
    service char (255) NOT NULL ,
    action char (255) NOT NULL ,
    arguments char (255) NULL ,
    messageRecipient char (255) NULL ,
    messageCreationTime char (255) NULL ,
    encryption char (10) NOT NULL ,
    signature char (10) NOT NULL ,
    publicKeyLdapAddress char (255) NULL ,
    publicKeyLdapBaseDN char (255) NULL ,
    publicKeyLdapDN char (255) NULL ,
    certificateURL char (255) NULL ,
    processingStatus char (255) NULL ,
    transportStatus char (255) NULL ,
    transportErrorCode char (255) NULL ,
    applicationStatus char (255) NULL ,
    applicationErrorCode char (255) NULL ,
    applicationResponse char (255) NULL ,
    messageSentTime char (255) NULL ,
    messageReceivedTime char (255) NULL ,
    responseMessageId char (255) NULL ,
    responseArguments char (255) NULL ,
    responseLocalFile char (255) NULL ,
    responseFilename char (255) NULL ,
    responseContent LONGBLOB NULL ,
    responseMessageOrigin char (255) NULL ,
    responseMessageSignature char (255) NULL ,
    priority int NULL,
    PRIMARY KEY (recordId)
);
```

A.3.2 WorkerQ Table - Sender

```
CREATE TABLE message_inq (
    recordId bigint NOT NULL AUTO_INCREMENT,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGBLOB NULL ,
    payloadTextContent LONGTEXT NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL,
    PRIMARY KEY (recordId)
);
```

A.3.3 ErrorQ Table - Sender

```
CREATE TABLE PHINMS_errq (
    recordId bigint NOT NULL AUTO_INCREMENT,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGBLOB NULL ,
    payloadTextContent LONGTEXT NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL,
    PRIMARY KEY (recordId)
);
```

A.3.4 Messaging Cache Table - Sender

```
CREATE TABLE messagingcache (
    sequence int NOT NULL AUTO_INCREMENT,
    partyId char (50) NULL ,
    convId char (50) NULL ,
    recordId char (50) NULL ,
```

```
    response LONGTEXT NULL ,
    timestamp char (20) NULL ,
    status char (10) NULL,
    PRIMARY KEY (sequence)
);
```

A.3.5 Messaging Queue Table - Receiver

```
CREATE TABLE message_outq (
    recordId bigint NOT NULL AUTO_INCREMENT,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGBLOB NULL ,
    payloadTextContent LONGTEXT NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL,
    PRIMARY KEY (recordId)
);
```

A.3.6 TransportQ - Receiver

```
CREATE TABLE message_outq (
    recordId bigint NOT NULL AUTO_INCREMENT,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGBLOB NULL ,
    payloadTextContent LONGTEXT NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL,
    PRIMARY KEY (recordId)
);
```

A.3.7 Message ErrorQ - Receiver

```
CREATE TABLE message_errq (
    recordId bigint NOT NULL AUTO_INCREMENT,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGBLOB NULL ,
    payloadTextContent LONGTEXT NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL,
    PRIMARY KEY (recordId)
);
```

A.3.8 Route-not-Read Table - Sender

```
CREATE TABLE [dbo].[broadcast] (
    [name] [char] (100) NULL ,
    [addresses] [char] (1000) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[messagebins] (
    [recordId] [bigint] IDENTITY (1, 1) NOT NULL ,
    [messageId] [varchar] (255) NULL ,
    [payloadName] [varchar] (255) NULL ,
    [payloadBinaryContent] [image] NULL ,
    [payloadTextContent] [text] NULL ,
    [localFileName] [varchar] (255) NULL ,
    [service] [varchar] (255) NOT NULL ,
    [action] [varchar] (255) NOT NULL ,
    [arguments] [varchar] (255) NULL ,
    [fromPartyId] [varchar] (255) NULL ,
    [messageRecipient] [varchar] (255) NULL ,
    [errorCode] [varchar] (255) NULL ,
    [errorMessage] [varchar] (255) NULL ,
    [processingStatus] [varchar] (255) NULL ,
    [applicationStatus] [varchar] (255) NULL ,
    [encryption] [varchar] (10) NOT NULL ,
    [receivedTime] [varchar] (255) NULL ,
    [lastUpdateTime] [varchar] (255) NULL ,
    [processId] [varchar] (255) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

```
CREATE TABLE [dbo].[partyid_user] (
    [partyId] [char] (255) NULL ,
    [user] [char] (255) NULL ,
    [sdnuser] [char] (255) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[users] (
    [name] [char] (100) NULL ,
    [description] [char] (255) NULL
) ON [PRIMARY]
GO
```

A.4 HSQL SCRIPTS

A.4.1 TransportQ Table - Sender

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

A.4.2 WorkerQ Table - Sender

```
CREATE TABLE message_inq (
    recordId bigint NOT NULL IDENTITY,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGVARBINARY NULL ,
    payloadTextContent LONGVARCHAR NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL
)
```

A.4.3 ErrorQ Table - Sender

```
CREATE TABLE PHINMS_errq (
    recordId bigint NOT NULL IDENTITY,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGVARBINARY NULL ,
    payloadTextContent LONGVARCHAR NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL
)
```

A.4.4 Messaging Cache Table - Sender

```
CREATE TABLE messagingcache (
    sequence int NOT NULL IDENTITY,
    partyId char (50) NULL ,
    convId char (50) NULL ,
    recordId char (50) NULL ,
    response LONGVARCHAR NULL ,
    timestamp char (20) NULL ,
```

```
    status char (10) NULL
)
```

A.4.5 Messaging Queue Table - Receiver

A.4.6 TransportQ Table - Receiver

```
CREATE TABLE message_outq (
    recordId bigint NOT NULL IDENTITY,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGVARBINARY NULL ,
    payloadTextContent LONGVARCHAR NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL
)
```

A.4.7 ErrorQ Table - Receiver

```
CREATE TABLE message_errq (
    recordId bigint NOT NULL IDENTITY,
    messageId varchar (255) NULL ,
    payloadName varchar (255) NULL ,
    payloadBinaryContent LONGVARBINARY NULL ,
    payloadTextContent LONGVARCHAR NULL ,
    localFileName varchar (255) NOT NULL ,
    service varchar (255) NOT NULL ,
    action varchar (255) NOT NULL ,
    arguments varchar (255) NULL ,
    fromPartyId varchar (255) NULL ,
    messageRecipient varchar (255) NULL ,
    errorCode varchar (255) NULL ,
    errorMessage varchar (255) NULL ,
    processingStatus varchar (255) NULL ,
    applicationStatus varchar (255) NULL ,
    encryption varchar (10) NOT NULL ,
    receivedTime varchar (255) NULL ,
    lastUpdateTime varchar (255) NULL ,
    processId varchar (255) NULL
)
```