Public Health Information Network Messaging System

An Overview of PHINMS

Prepared by
U.S. Department of Health & Human Services April 2003

Table Of Contents

Public Health Information Network Messaging System

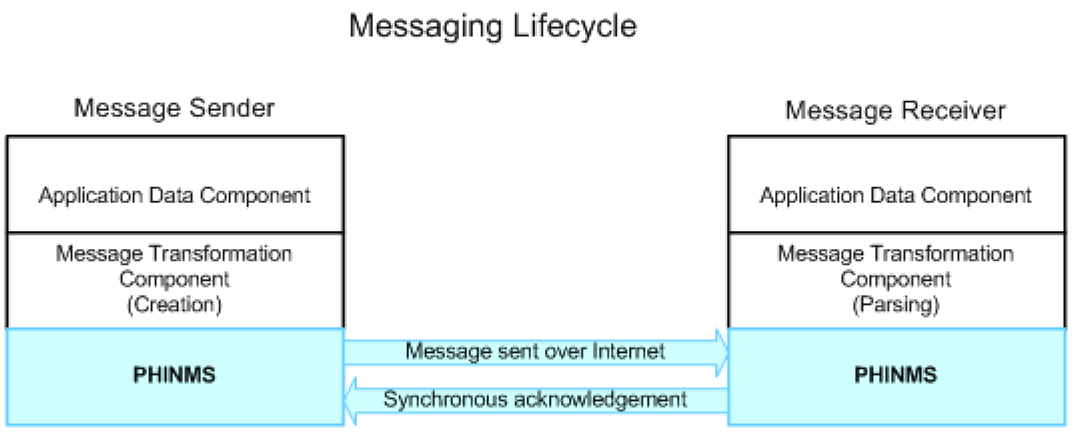
This document provides an overview of the Public Health Information Network Messaging System, PHINMS. It is intended for users, administrators and programmers and it provides general information about how the PHINMS works. For installation and configuration of the PHINMS software see the PHINMS Client Installation Guide and the PHINMS Server Installation Guide.

Functional Description

Developed by the Centers for Disease Control and Prevention, The Public Health Information Network Messaging System, PHINMS, uses the Electronic Business Extensible Markup Language, ebXML, infrastructure to securely transmit public health information over the Internet.

PHINMS is a generic, standards-based, interoperable and extensible message transport system. It is platform-independent and loosely coupled with systems that produce outgoing messages or consume incoming messages.

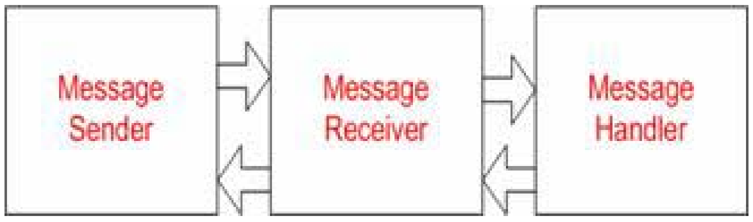
Within NEDSS, the National Electronic Disease Surveillance System, PHINMS functions as a component as shown in the following diagram:


Messaging Lifecycle
Message Sender
Message Receiver
Application Data Component
Application Data Component
Message Transformation Component (Creation)
Message Transformation Component (Parsing)
PHINMS
Message sent over Internet
Synchronous acknowledgement
PHINMS

PHINMS is loosely coupled with the Message Transformation Component. It uses a Transport Queue interface to read and write outgoing and incoming messages. The Transport Queue is implemented as a database table or as a file system directory.

Major Components

PHINMS has three major components: the Message Sender, Message Receiver and the Message Handler.



Message Sender

The Message Sender functions as the client. It is a Java application that runs on a workstation or server. The Message Sender polls the Transport Queue for outgoing data. The Transport Queue can be a database table or a file system directory. When outgoing data is found, the Message Sender packages the data as an ebXML message and sends it to the Message Receiver.

Message Receiver

The Message Receiver functions as a server. It is a servlet that runs on a J2EE compliant application server. When the Message Receiver receives a message, it processes the message envelope, decrypts the message, verifies the signature and then forwards the message payload to the Message Handler or writes the message directly into a worker queue. Afterward the Message Receiver waits for an application status from the Message Handler and when it receives the status it forwards it to the Message Sender.

Message Handler

The Message Handler functions as a server. It is a servlet that runs on a J2EE compliant application server. The Message Handler and the Message Receiver can reside on the same system. When the Message Handler receives the message payload from the Message Receiver it processes the message payload and then sends a response, which contains the Message Handler's status, back to the Message Receiver.

Message Sender

The Message Sender, the client, is an ebXML compliant Java application that resides on the host that sends the message. The Message Sender performs the following:

- Initialization
- Polling
- Operations and Transformations

Initialization

During initialization the Message Sender does the following:

- Reads its configuration file and prompts the user for a single password.

- Uses the password to decrypt the user's passwords file. The user's password file contains all of the passwords the Message Sender needs to perform authentication. Other configuration files make references to the passwords within this encrypted passwords file.

- Stores the decrypted passwords in memory and uses them during the Message Sender's uptime.

Polling

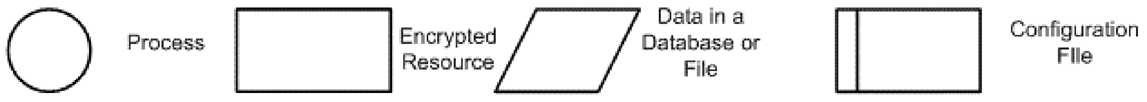
The Message Sender operates in one of two polling modes, which is specified in its configuration file:

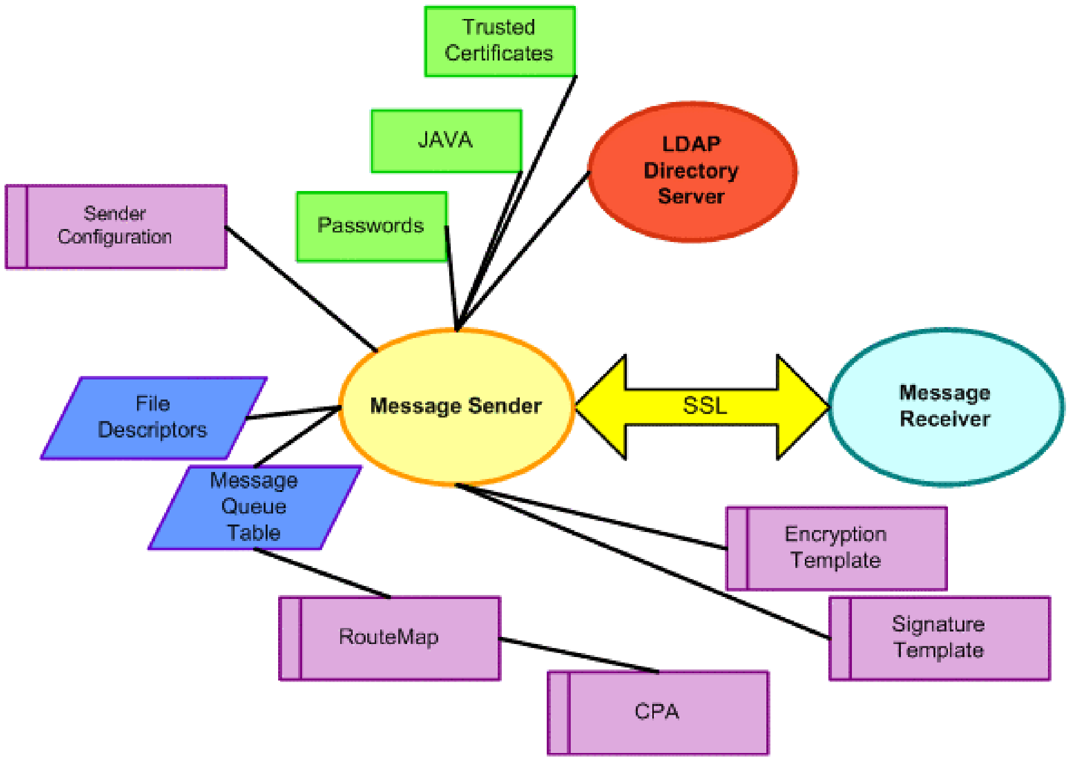
- **Database Polling Mode –** The application that creates messages writes data to a Transport table. The Message Sender polls the Transport Queue table for outgoing messages. Message responses are written back to the Transport Queue table.

- **File Based Polling Mode** - The application that uses PHINMS creates file descriptors in a files system directory. The Message Sender polls this directory for file descriptors and, when it finds them, sends the corresponding file as an ebXML compliant message and then creates a response descriptor.

Operations and Transformations

When the Message Sender finds new outgoing data, in a Transport Queue database table or a file descriptor, it performs the following operations and transformations in the diagram below:





Route Mapping

A configuration file, called **routeMap**, maps the route to its Collaboration Protocol Agreement, the CPA. The route is specified in a field in the Message Queue database table or as a field in the file descriptor that is associated with an outgoing message.

Collaboration Protocol Agreement Parsing

The CPA is read to determine the Message Receiver's end point, and security attributes, such as the authentication mode.

LDAP Public Key Searching

If the Message Sender has selected the encryption option and if the LDAP attributes of the Message Receiver’s public key certificate are specified, an LDAP search is performed and the Message Receiver's public key is retrieved.

Message Encryption and Signing

If the Message Sender has selected the message signature option, which is a field in the Transport Queue table or a field in the file system directory, the message is signed using the XML Signature standard.

A signature template configuration file, in XML format, is used to generate the signature. Afterward, the message is encrypted using the XML Encryption standard. An encryption template configuration file, in XML format, is used to generate the cipher text.

SSL and Authentication

Using SSL, the Message Sender connects to the Message Receiver’s end point, a URL specified in the CPA, and then performs authentication with the Message Receiver. If the CPA specifies a basic or custom authentication mode, the user name and password parameters are read from the CPA and from an encrypted passwords file.

SOAP Call

The ebXML compliant message, which includes the file payload and message envelope, is sent to the Message Receiver in a SOAP call.

Response Parsing and Writing to Database Table or Descriptor

The response from the Message Receiver is parsed for transport and application status information. This information is written to the Transport Queue table or the acknowledgement descriptor.

Exchanging Messages

Message Receiver

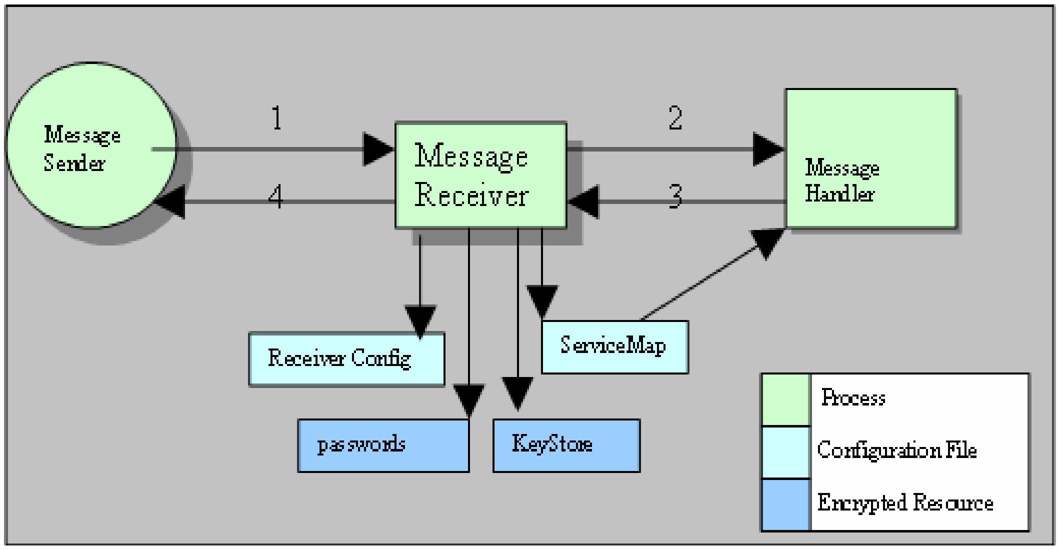
The Message Receiver is a servlet that runs on a J2EE compliant application server, such as Tomcat or Silver Stream. The Message Receiver uses Java XML messaging libraries, JAXM, to implement SOAP messaging.

Message Handler

The Message Handler is any web resource, such as a servlet, JSP, CGI or ASP that is capable of:

- Parsing a MIME multi-part message which contains the payload and text with data from the ebXML envelope such as the Message Sender's party ID, the message manifest and so on.
- Performing base64 decoding.

The message exchange process is illustrated in the following diagram:



1. SOAP Request

When the Message Receiver receives a request from the Message Sender, the Message Receiver parses the ebXML envelope and obtains the Service and Action attributes. Every message is associated with a Service and Action attribute. This pair of attributes is specified in the Message Sender, the client, as a file descriptor, which corresponds to an outgoing file or a as a database record in a relational database table.

2. Mapping Service Action Attributes

Using the Service Map configuration file, the Message Receiver maps the Service and Action attributes to the Message Handler or to a worker queue where it waits for a

Message Handler to poll for the message. After finding the Message Handler, the Message Receiver determines if the message's payload is encrypted. If it is encrypted, the Message Receiver tries to decrypt the payload and verify the signature.

If the payload is not signed or if the Message Receiver cannot decrypt the payload, it sends the payload to the Message Handler as it is. If the Message Receiver successfully decrypts the payload, it sends the decrypted plain text file to the Message Handler as an attachment.

3. Payload Processing

When the Message Handler receives the payload, it processes it, and then sends a response to the Message Receiver, which contains the application's status and other information.

4. Sending Transport Status

The Message Receiver sends the transport and application status and any errors back to the Message Sender in a synchronous manner, which means, the Message Receiver waits for the Message Handler to finish processing the message payload before it sends a response to the Message Sender.

The Message Sender waits for a response from the Message Receiver before it begins a new message process.

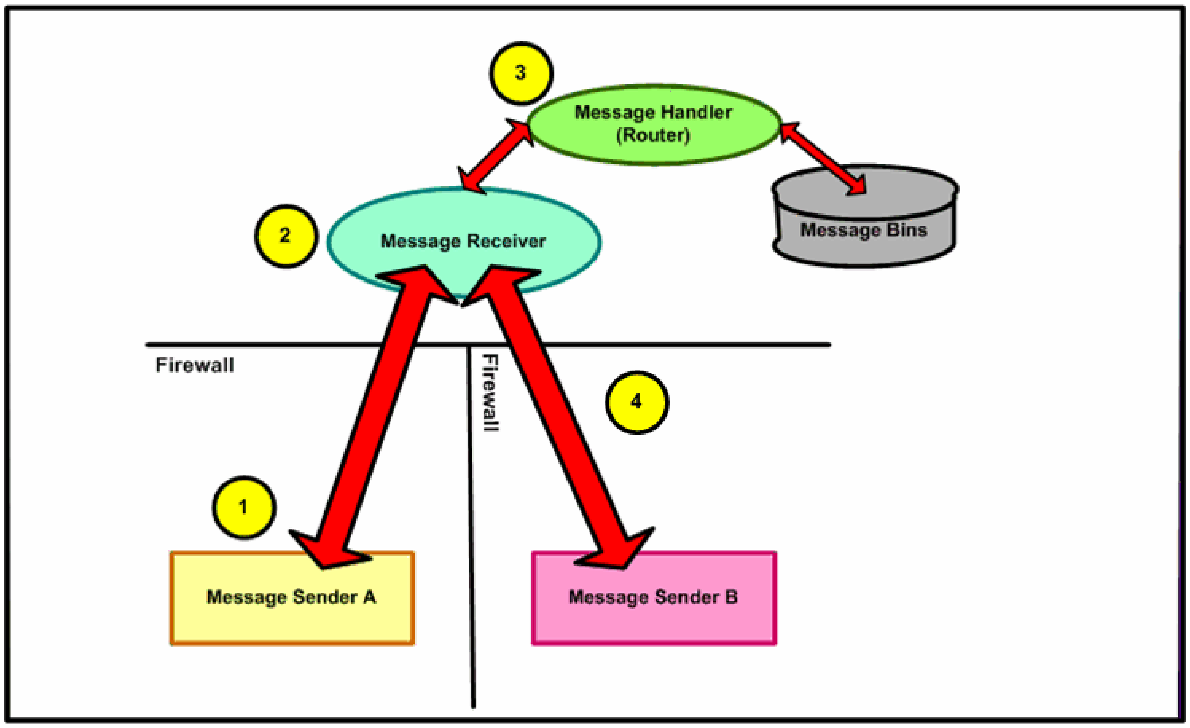
Using a Third Party to Route Messages

When the Message Sender and the recipient, which can also be a Message Sender, are behind separate firewalls, they need an intermediary to communicate. A Router Message Handler acts as this intermediary. It "routes" the message to a temporary Message Bin instead of reading it.

To retrieve the message from the Message Bin, the recipient polls the Message Receiver, which communicates to the Router, which retrieves the message from the Message Bin. This scenario is called "route-not-read."

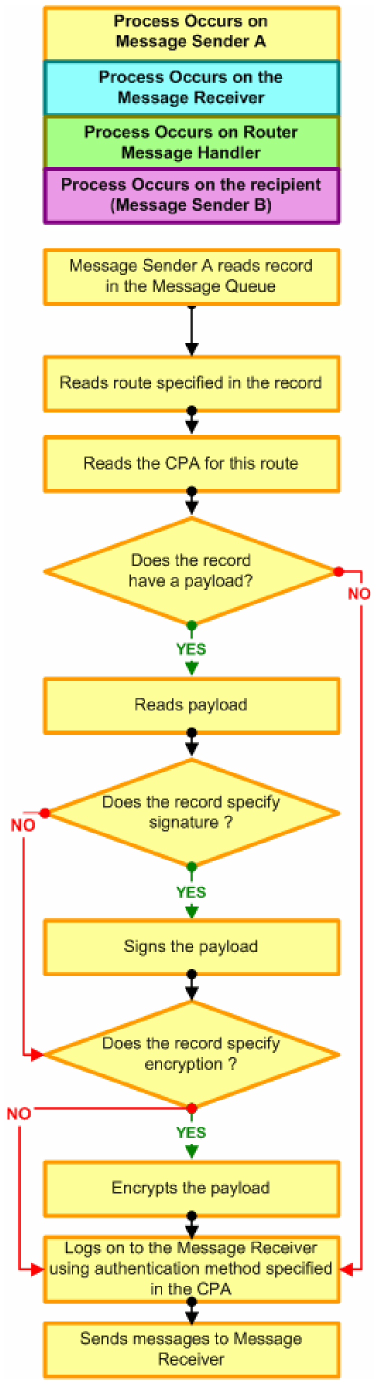
If your Message Receiver and Message Handler reside on the same system, you do not need to route messages through a third party.

The following diagram illustrates the "route-not-read" scenario. Message Sender A is sending a message to Message Sender B. Both of these Message Senders are clients.



1. *Message Sender A Sends a Message to the Message Receiver*

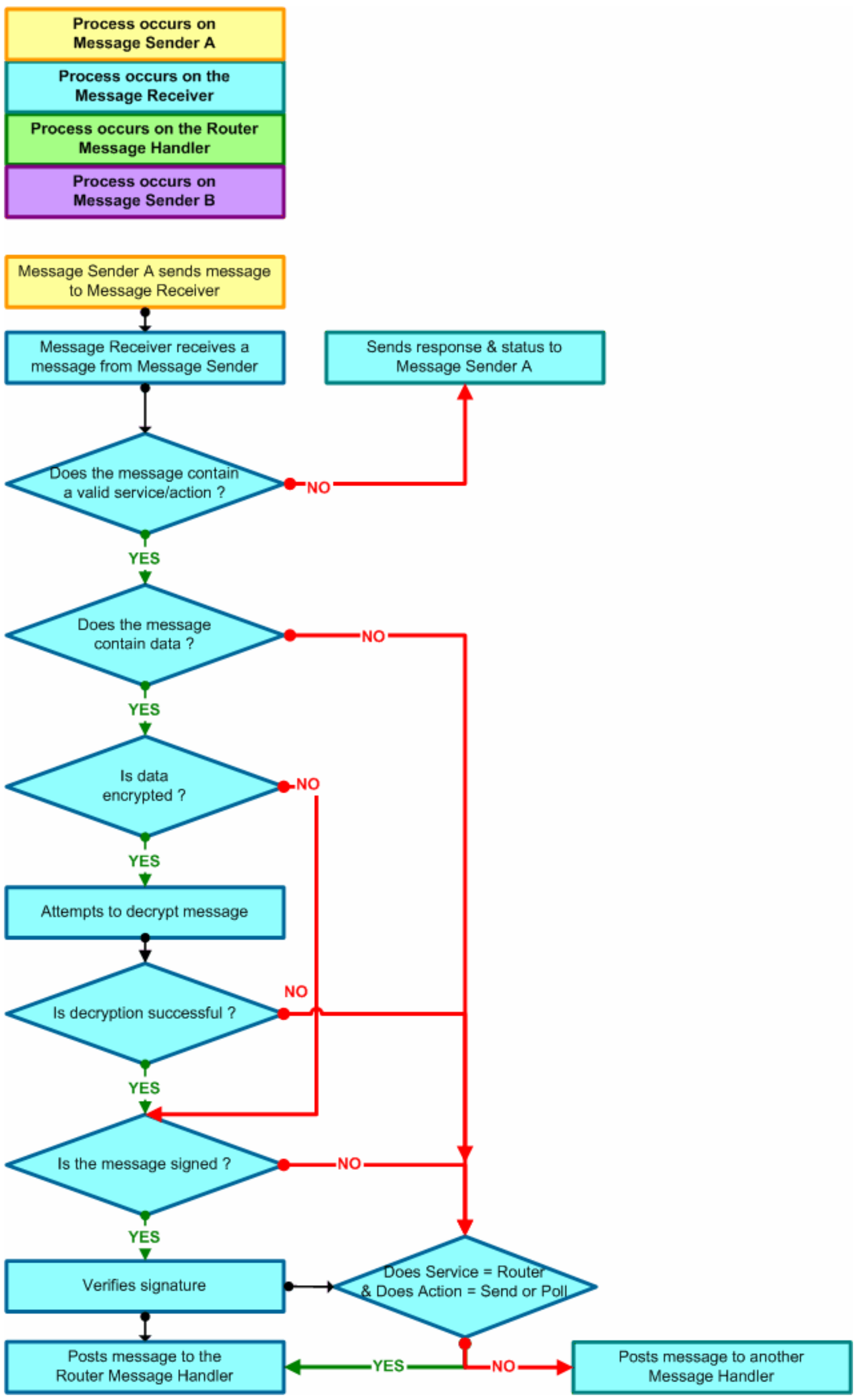
The following flow chart describes the first portion of the "route-not-read"scenario. Message Sender A sends the message to the Message Receiver.



Process Occurs on Message Sender A
Process Occurs on the Message Receiver
Process Occurs on Router Message Handler
Process Occurs on the recipient (Message Sender B)
Message Sender A reads record in the Message Queue
Reads route specified in the record
Reads the CPA for this route
Does the record have a payload?
NO
YES
Reads payload
Does the record specify signature ?
NO
YES
Signs the payload
Does the record specify encryption ?
NO
YES
Encrypts the payload
Logs on to the Message Receiver using authentication method specified in the CPA
Sends messages to Message Receiver

2. *Message Receiver to the Router Message Handler*

The following flow chart describes the second portion of the "route-not-read"scenario in which the Message Receiver sends the message to the Router Message Handler.

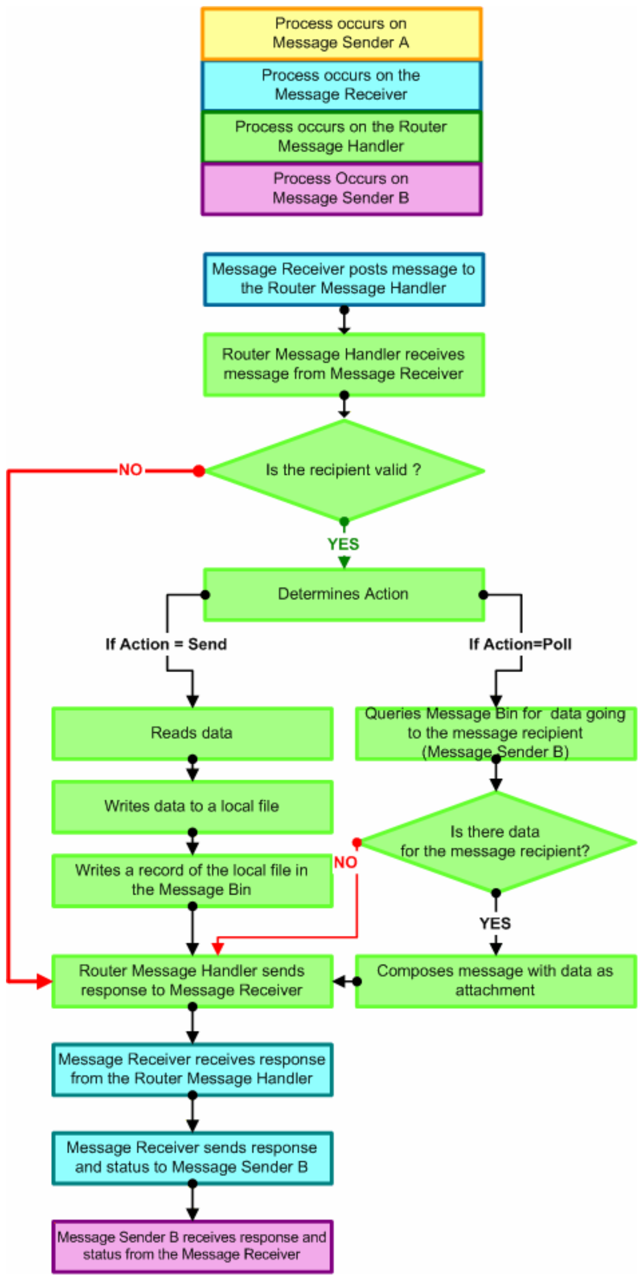


Process occurs on Message Sender A
Process occurs on the Message Receiver
Process occurs on the Router Message Handler
Process occurs on Message Sender B
Message Sender A sends message to Message Receiver
Message Receiver receives a message from Message Sender
Sends response & status to Message Sender A
Does the message contain a valid service/action ?
NO
YES
Does the message contain data ?
NO
YES
Is data encrypted ?
NO
YES
Attempts to decrypt message
Is decryption successful ?
NO
YES
Is the message signed ?
NO
YES
Verifies signature
Does Service = Router & Does Action = Send or Poll
Posts message to the Router Message Handler
YES
NO
Posts message to another Message Handler

3. *Storing and Retrieving Messages from the Message Bin*

The following flow chart illustrates the two processes in the third portion of the "route-not-read"scenario.

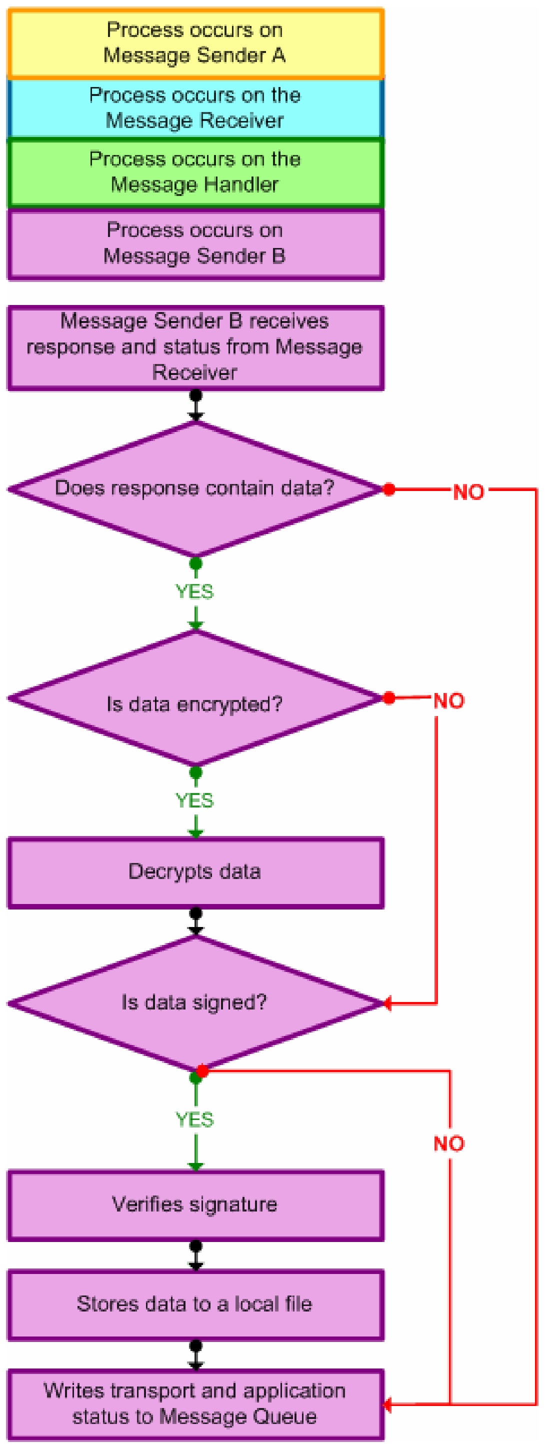
On the left side of the flow chart (If Action=Send), the Router Message Handler receives a request from the Message Receiver to store a message in the Message Bin. The right side (If Action=Poll) illustrates the Router Message Handler retrieving the same message from the Message Bin.



Process occurs on Message Sender A
Process occurs on the Message Receiver
Process occurs on the Router Message Handler
Process Occurs on Message Sender B
Message Receiver posts message to the Router Message Handler
Router Message Handler receives message from Message Receiver
Is the recipient valid ?
NO
YES
Determines Action
If Action = Send
If Action=Poll
Reads data
Writes data to a local file
Writes a record of the local file in the Message Bin
Queries Message Bin for data going to the message recipient (Message Sender B)
Is there data for the message recipient?
NO
YES
Router Message Handler sends response to Message Receiver
Composes message with data as attachment
Message Receiver receives response from the Router Message Handler
Message Receiver sends response and status to Message Sender B
Message Sender B receives response and status from the Message Receiver

4. Message Sender B Receives the Message

The following flow chart illustrates the fourth and last portion of the "route-not-read"scenario. Message Sender B receives the message.


Process occurs on Message Sender A
Process occurs on the Message Receiver
Process occurs on the Message Handler
Process occurs on Message Sender B
Message Sender B receives response and status from Message Receiver
Does response contain data?
NO
YES
Is data encrypted?
NO
YES
Decrypts data
Is data signed?
YES
NO
Verifies signature
Stores data to a local file
Writes transport and application status to Message Queue

Message Receiver Handling Modes

In The Messaging System 2.0 the Message Receiver supports asynchronous and synchronous message handling modes. You can implement either or both handling modes.

Synchronous Handling Mode

Synchronous message handlers are usually servlets. However, they can also be implemented as other web programs such as ASP and CGI scripts. In synchronous handling mode the Message Receiver performs the following operations:

1. Receives an incoming message from the Message Sender.
2. Parses the incoming message envelope.
3. Maps the Service and Action attributes on the message envelope to a message handler's URL.
4. Posts the message payload to the message handler's URL.
5. Sends the transport and application status to the Message Sender that sent the incoming message.

Asynchronous Handling Mode

In the asynchronous handling mode the Message Receiver performs the following operations:

1. Receives an incoming message from the Message Sender.
2. Parses the incoming message envelope.
3. Maps the Service and Action attributes to one or more worker queues.
4. Drops the payload and meta data into the worker queue.
5. Sends a transport status back to the Message Sender.

   Afterward an asynchronous message handler application polls its worker queue to receive the incoming message.

Combination Handling Modes

The Message Receiver can simultaneously accommodate the asynchronous and synchronous handling modes. To use a combination of both handling modes the Message Receiver does the following:

1. Receives an incoming message from the Message Sender.
2. Parses the incoming message envelope, decrypts the payload and then verifies the signature.
3. Looks in the **servicemap.xml** file for an entry that corresponds to the Service and Action attributes in the envelope of the incoming message.
4. Determines whether the entry type is **servlet** or **workerqueue**.
5. If the entry type is **servlet** the Message Receiver does the following:

- Sends the payload to a message handler.
- Waits for a response from the message.
- Sends a synchronous response to the Message Sender, which includes the transport and application status.

6. If the entry type is **workerqueue**, the Message Receiver writes the payload to a set of worker queues, which are defined in the **servicemap.xml** file, and then it sends a synchronous response to the Message Sender, which contains only the transport status.

Index