



## **Hermes Message Service Handler Development Guide**

**Version 0.9.3.1**

**Maintained by**

**Patrick Yee ([kcyee@cecid.hku.hk](mailto:kcyee@cecid.hku.hk))  
Center for E-commerce Infrastructure Development  
The University of Hong Kong**

Copyright © Center for E-commerce Infrastructure Development 2002. All Rights Reserved.

## Table of Content

Table of Content .....	2
1. Introduction .....	3
1.1. Document Purpose.....	3
1.2. Project Background .....	3
2. Hermes Architecture .....	4
2.1. Introduction.....	4
2.2. Delivery Modes.....	5
2.2. Other Protocols .....	7
3. Client application development .....	8
3.1. Request Object.....	8
3.2. Application Context Object .....	9
3.3. Message Listener Interface .....	9
3.4. Client Message Listener Interface .....	9
3.5. MessageListenerImpl Object.....	9
3.6. ClientMessageListenerImpl Object.....	10
3.7. NoMessageListenerImpl Object .....	10
3.8. EbxmlMessage Object .....	10
3.9. Message Header Object.....	11
Appendix A. Fields in Properties Files .....	12
A.1. msh.properties.xml.....	12
A.2. msh_client.properties.xml .....	16
A.3. monitor.properties.xml .....	17
Appendix B. Error Codes .....	18
Appendix C. Sample code of client application .....	20

# 1. Introduction

## 1.1. Document Purpose

The aim of this document is to present the information for development of Hermes Message Service Handler (MSH). This document describes the architecture of our implementation of Message Service Handler (Hermes) is described. It provides a high level understanding on what fundamental services the MSH supports, and how are we going to implement those services. It also describes the key components for building client applications to run on the MSH.

## 1.2. Project Background

Hermes is developed under the Project Phoenix, by the Center for E-Commerce Infrastructure Development, the University of Hong Kong.

Project Phoenix is officially titled "Establishment of ebXML Software Infrastructure in Hong Kong". As its name says, the project aims to establish an e-commerce infrastructure in Hong Kong using the ebXML standard.

The project is primarily funded by the Hong Kong Government's Innovation and Technology Fund, and is also supported by some sponsorship from several industrial partners.

The project commenced in 1<sup>st</sup> January 2002 and shall be completed by the end of year 2003. The project duration is two years.

Hermes is released as an open-source project under Academic Freeware License. The source code repository is being hosted at SourceForge.

## 2. Hermes Architecture

### 2.1. Introduction

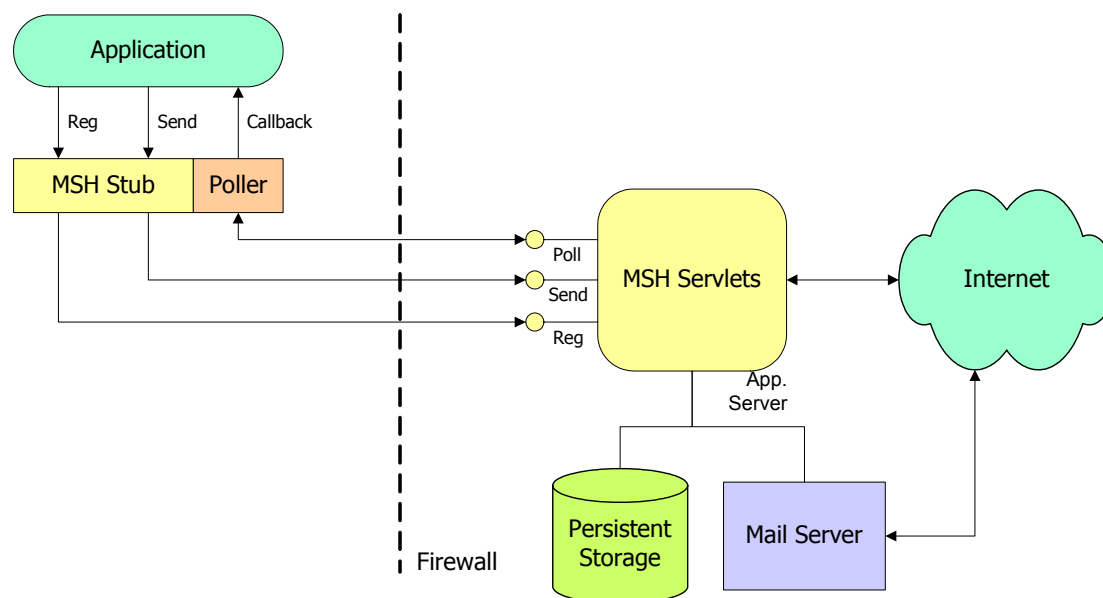


Figure 1: Basic MSH Architecture

Figure 1 shows the basic architecture of our solution. The MSH system is divided into two halves. The main reason for this arrangement is that we have to cope with firewalls. In many cases, the MSH connection endpoints are connected to the Internet directly, which will be generally outside the corporate firewall. However, the actual recipient of the ebXML messages will be some application inside the corporate firewall. So, we will implement the main functionalities of MSH outside the firewall, and make use of a MSH stub to link with the application inside the firewall.

Another concern is that we want to leverage existing products to handle some low level parts. For example, we want to make use of a servlet container, e.g. Jakarta Tomcat, for handling HTTP. As the servlet container and the client application will be running in two separate processes, the communication between the MSH and client application cannot be simple API calls. So, we have to define a proprietary protocol for the client application to talk to the servlet container. In our architecture, the MSH stub is responsible for handling such protocol.

To use the MSH functionalities, the client application has to first of all create a MSH stub object. As a first step, it will register the details of the CPA with the MSH. The MSH needs that information to (1) send out the message correctly, and (2) identify the actual recipient of the incoming ebXML messages. The client application can perform the registration by calling a method in the stub object. The stub object will in turn delegate the call to the MSH Servlets, by

converting the parameters into a SOAP message and sending to the MSH Servlets using HTTP.

The client application can send out ebXML messages by calling a method in the stub object. Some processing will be done in the stub side, for example, packaging, digitally signing and encryption. Then the message will be transferred to the MSH Servlets. Some more processing will be done in the MSH Servlets side, for example, reliable messaging handling. And finally, the message will be sent out to the destination. Actually, the client application will never notice the details of reliable messaging, as those are just some additional messages interchanged between MSHs.

## 2.2. Delivery Modes

To receive incoming messages, our MSH supports four modes of operation. Figure 1 shows the most typical setting. When an incoming message comes, the MSH Servlets will do the necessary processing first. Then the MSH Servlets will identify the recipient and push the messages into the recipient's queue. The MSH Servlets can identify messages for individual recipient client applications by the registration information passed in by the client applications in the first step. The MSH stub object has a built-in polling thread. It polls the MSH Servlets periodically for its messages. When there are messages available, the polling thread will get the message from the MSH Servlets and pass to the client applications through a callback method.

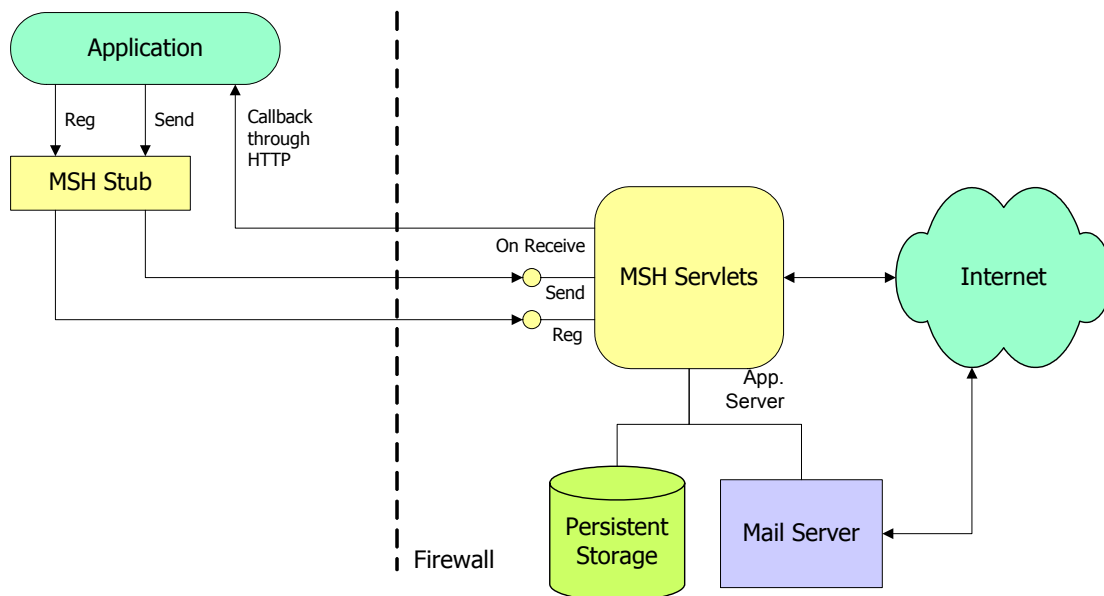


Figure 2: Direct HTTP mode operation

Figure 2 shows another mode of operation. Here, there is no polling thread in the MSH stub object. Instead, the client applications should provide HTTP URLs for callback. Whenever an incoming message comes and after the necessary processing, the MSH Servlets will forward the message to the

provided HTTP URLs. One typical setting for this mode of operation is that: the client application is implemented as another servlet, registering its own URL to the MSH Servlets. Upon receiving the messages, the client application servlet will then execute the backend processing, e.g. saving to database, etc.

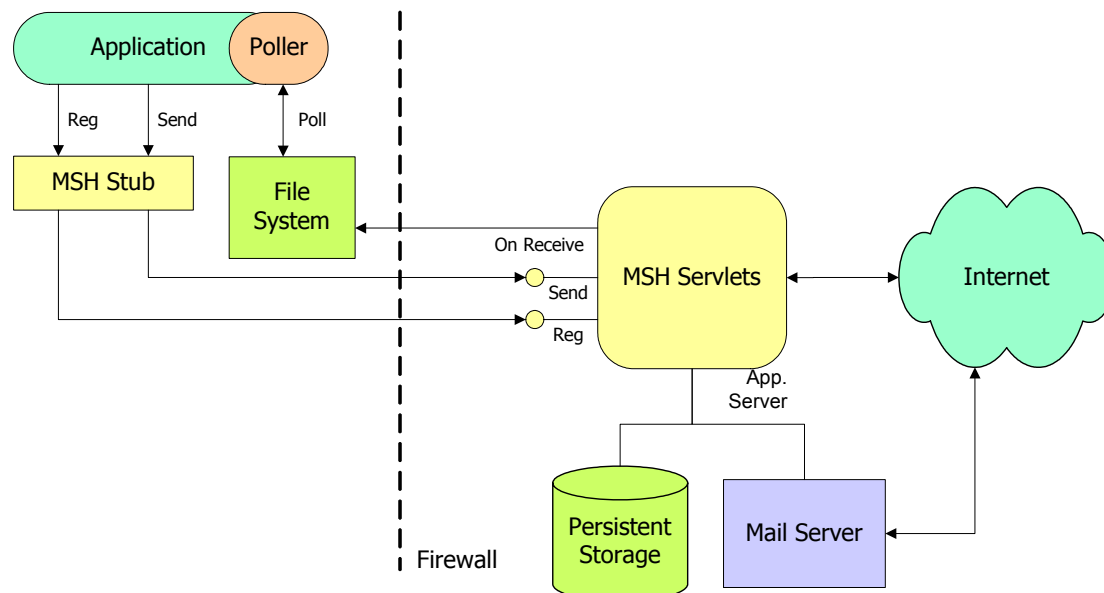


Figure 3: File system mode operation - I

Figure 3 shows the third operation mode. In this case, when an incoming message comes and after the necessary processing, the MSH Servlets will save the message in a file system accessible by the client applications. The client applications should use their own mechanisms to detect and retrieve the message files. Typically, polling will be used, and the client applications should implement their own polling threads.

Note that in this mode of operation, we assumed the MSH server side and the MSH client side are accessing the same file system. The common directory has to be pre-registered to MSH server. There is a property entry in MSH server called TrustedRepository. If the common directory specified by the client application is not listed in the TrustedRepository property, exception will be thrown.

If there are difficulties to setup a common accessible directory in both side, and yet the file system based delivery mode is desirable, there is the fourth choice. Figure 4 shows the setting for another file system based delivery mode. In this mode, the MSH stub will start a thread to poll the MSH server for querying new messages. The MSH stub will write each new message get from the server to the specified directory in MSH stub side. So the effect to the application is similar to the third mode: the new messages are saved to a directory, and the application will be responsible to poll that directory for getting those new messages.

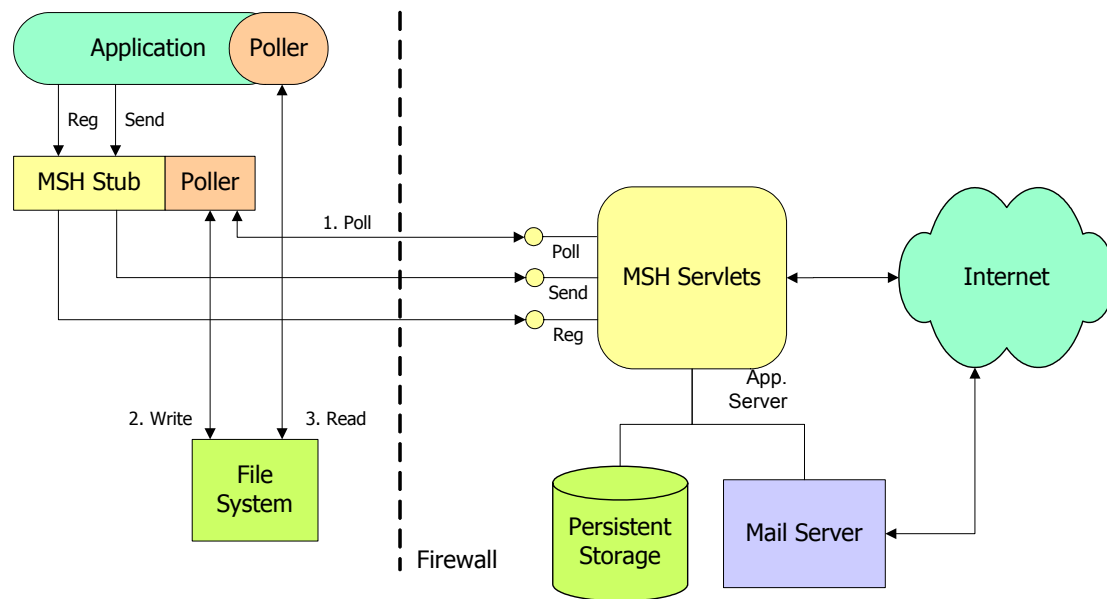


Figure 4: File system mode operation - II

## 2.2. Other Protocols

For SMTP protocol, we may have to implement an additional polling thread in the MSH Servlets. The polling thread will poll the mail server for emails received. When there are messages available, the MSH Servlets will get the emails and do the processing as in the same case of HTTP.

## 3. Client application development

This chapter provides a brief introduction of the major building blocks of client applications. For each section below, we will describe the functionalities and major methods of the main objects. For details for each object, please refer to the API documentation.

### 3.1. Request Object

The Request object is the object controlling the communications between the client application and the MSH application server. The client application registers itself, sends messages through the Request object. Then the Request object in turn passes the information to the MSH application server.

To register the client application to the MSH application server, you can simply create the Request object with the client application's context (see Section 3.2), a message listener object (see Section 3.3) and the outgoing URL to the targeted peer. The Request object will do the registration automatically.

You can call `send()` to send a message out. The outgoing URL should be the same as the one you registered.

You can call `sendReliably()` to send out a message with reliable messaging protocol. The outgoing URL should be the same as the one you registered.

You can call `setSign()` in the Request object to specify the keystore for digital signature applied to all outgoing messages. You can call `setSign()` with `null` as any one of the parameters to denote you don't want digital signature anymore.

Every communication between the Request object and the MSH application server can be authenticated. A password file storing user name and password pairs is located at the MSH application server. The Request is configured with a user name and password. Every time it sends out commands and messages to the MSH application server, the Request object will attach the user name and password. And then the application server will do an authentication. Since plain text will be used to transmit the user name and password, HTTPS encryption is recommended between the MSH application server and Request object.

## 3.2. Application Context Object

The application context object holds the unique key to the client application for the MSH application server. In other words, all client applications should have a unique application context. There are four elements in the context object: CPA ID, Conversation ID, Service and Action. These are the parameters got from the Collaboration Protocol Agreement (CPA) and Business Process Specification (BPS).

## 3.3. Message Listener Interface

For sending messages out, you will need Request object. If you want to receive messages, you need to implement a Message Listener and pass the object (implementation) to Request object. Or, you can make use of the Message Listener implementation provided by us. To implement a Message Listener, you need to implement two methods:

`getClientURL()` should return a file based URL. In this case, the MSH application server will try to save all received messages to the directory pointed by the URL. Please note that the file URL is with respect to the MSH application server's logical file system. And for security reason, the exact path of the directory should be specified in the MSH application server's trusted repository property. We assume the client application can access that directory for polling of incoming messages.

`onMessage()` is the method for the MSH application server to invoke when there is a received message. The MSH application server will invoke this method when your `getClientURL()` method returns `null` or a non-file protocol. The client application should then parse the received messages for further processing inside the `onMessage()` method.

## 3.4. Client Message Listener Interface

Basically this interface is the same as the Message Listener Interface described above. The only difference is that the file based URL returned in `getClientURL()` will be with respect to the file system in MSH stub (client) side. Therefore, no trusted repository constraints will be enforced. Please refer to Section 2.2 for the internal mechanism of these 2 different delivery modes.

## 3.5. MessageListenerImpl Object

This is an object implementing Message Listener interface. Therefore you can use it to pass to Request object to denote your preferred message-receiving mode. The constructor of this object takes an URL. You can pass in a non-file

URL. In this case, MSH application server will post all received messages directly to that specified URL.

If you pass in a file based URL, the MSH application server will try to save all received messages to the directory pointed by the URL. Please note that the file URL is with respect to the MSH application server's logical file system. And for security reason, the exact path of the directory should be specified in the MSH application server's trusted repository property. We assume the client application can access that directory for polling of incoming messages. This mechanism is called Trusted Repository.

### 3.6. ClientMessageListenerImpl Object

This is another object implementing Message Listener interface. Again, you can use it to pass to Request object to denote your preferred message-receiving mode. The constructor of this object takes a file based URL. The URL will be with respect to the file system in MSH stub (client) side).

### 3.7. NoMessageListenerImpl Object

This is another object implementing Message Listener interface. In this case, all messages received in MSH application server will be stored in the server side. No polling will be issued from MSH stub. The client application should use APIs to query and download available messages. Those methods are implemented in Request object.

### 3.8. EbxmlMessage Object

The `EbxmlMessage` object represents a message being sent or received. It basically can be divided into 2 parts: message header and payloads.

You can construct a brand new `EbxmlMessage` object by calling the default constructor. Then you can add the message header by calling `addMessageHeader()` method. You may want to specify all parameters needed in a single call to the `addMessageHeader()` method, or just call `addMessageHeader()` with no argument. Anyway, `addMessageHeader()` will return a Message Header object, to which you can further set the parameters. However, due to limitation in the SOAP library we are using, you are recommended to set the parameter once and only once.

Two static methods from `MessageServiceHandler` object can help to automatically generate the message ID and timestamp for the message.

To add payload to the message, you should call `addPayloadContainer()` method of the `EbxmlMessage` object. To specify the content of the payload, a

data handler object should be created first, please refer to the documentation of Java Activation Framework for more information about data handler.

### **3.9. Message Header Object**

There are a number of fields in the Message Header object to be set. Some of which are mandatory, like From Party, To Party, CPA ID, Conversation ID, Service, Action, Message ID and Timestamp. Some of others are optional, like From Role, To Role, Description, Duplicate Elimination, etc.

## Appendix A. Fields in Properties Files

### A.1. msh.properties.xml

This properties file is used by the MSH server:

Field	Description
MSH/Log/UseLogger	The logging parameter for the servlet. Choose whether Log4J or JDK logging package is used for logging. Valid values are "LOG4J" and "JDK".
MSH/Log/LogPath	The logging parameter for the servlet. The directory for storing the log file.
MSH/Log/LogFile	The logging parameter for the servlet. The log file name.
MSH/Log/LogLevel	The logging parameter for the servlet. The log level of the logger. Valid values range from 0 to 4. "0" means most log, and "4" means no log.
MSH/Log/MaxFileSize	The maximum log file size in bytes. If the maximum size is reached, a new log file will be created and the logger will "roll over" to use the new log file. If this property is missing, or the value is smaller than 0, the roll over is disabled.
MSH/Config/URL	The URL for the external systems to locate the MSH application server. You should fill in the host name, port number and context path of the MSH application server.
MSH/Config/AuthenticationFile	A password file storing the user name and password for authentication. If this property is missing, no authentication will be done on MSH application server.
MSH/Config/PositiveAcknowledgment	Optional property controlling whether a positive acknowledgment message will be generated if MSH successfully sends a message.
MSH/Config/AugmentedErrorMessage	Optional property controlling whether an error message being sent back to the sender application is augmented with the original message as a MIME payload attachment.
MSH/Config/ContentTransferEncoding	Optional property controlling the content transfer encoding of the payloads while sending out messages in HTTP
MSH/Proxy/Host	The host name of HTTP proxy server for sending out messages to another MSH through HTTP. If you can connect to Internet

	without using proxy server, this field should be commented.
MSH/Proxy/Port	The port number of HTTP proxy server for sending out messages to another MSH through HTTP. If you can connect to Internet without using proxy server, this field should be commented.
MSH/Mail/SMTP/Host	The host name of the SMTP server for sending out messages to another MSH through SMTP.
MSH/Mail/SMTP/User	The user name to connect to the SMTP server. Use only for SMTP authentication. If missing, ordinary SMTP without authentication is assumed.
MSH/Mail/SMTP/Password	The password to connect to the SMTP server. Use only for SMTP authentication. If missing, ordinary SMTP without authentication is assumed.
MSH/Mail/Debug	Controls whether debug message will be output for mail protocol handling or not.
MSH/Mail/Poll/Protocol	The protocol used by MSH server to connect to the mail server for incoming messages. If you do not need to support SMTP protocol for incoming messages, this field should be commented.
MSH/Mail/Poll/Host	The host name used by MSH server to connect to the mail server for incoming messages. If you do not need to support SMTP protocol for incoming messages, this field should be commented.
MSH/Mail/Poll/Port	The port number used by MSH server to connect to the mail server for incoming messages. If you do not need to support SMTP protocol for incoming messages, this field should be commented.
MSH/Mail/Poll/Folder	The folder name on the mail server for retrieving messages, used by MSH server to connect to the mail server for incoming messages. If you do not need to support SMTP protocol for incoming messages, this field should be commented.
MSH/Mail/Poll/User	The user name used by MSH server to connect to the mail server for incoming messages. If you do not need to support SMTP protocol for incoming messages, this field should be commented.
MSH/Mail/Poll/Password	The password used by MSH server to connect to the mail server for incoming messages. If you do not need to support SMTP protocol for incoming messages, this

	field should be commented.
MSH/Mail/Poll /MonitorInterval	The time interval for the MSH server to connect to the mail server for incoming messages. If you do not need to support SMTP protocol for incoming messages, this field should be commented.
MSH/Mail/Poll /ForceChangeSubType	A boolean value to instruct MSH whether to rebuild the content type of the incoming MIME message. This is useful when some mail servers mistakenly change the MIME message content type upon receiving the message.
MSH/Mail/SMIME /Encryption/KeyStore /Path	RESERVED. For future S/MIME development.
MSH/Mail/SMIME /Encryption/KeyStore /File	RESERVED. For future S/MIME development.
MSH/Mail/SMIME /Encryption/KeyStore /Password	RESERVED. For future S/MIME development.
MSH/Mail/SMIME /Decryption/KeyStore /Path	RESERVED. For future S/MIME development.
MSH/Mail/SMIME /Decryption/KeyStore /File	RESERVED. For future S/MIME development.
MSH/Mail/SMIME /Decryption/KeyStore /Alias	RESERVED. For future S/MIME development.
MSH/Mail/SMIME /Decryption/KeyStore /Password	RESERVED. For future S/MIME development.
MSH/DigitalSignature /TrustedAnchor /KeyStore/Path	The directory holding the keystore that holding trusted certificates. This is used for referencing a keystore for trust anchor and certificate path verification.
MSH/DigitalSignature /TrustedAnchor /KeyStore/File	The file name of the keystore that holding trusted certificates. This is used for referencing a keystore for trust anchor and certificate path verification.
MSH/DigitalSignature /TrustedAnchor /KeyStore/Password	The password of the keystore that holding trusted certificates. This is used for referencing a keystore for trust anchor and certificate path verification.
MSH/DigitalSignature /AckSign/KeyStore /Path	The directory holding the keystore for signing acknowledgments.
MSH/DigitalSignature /AckSign/KeyStore /File	The file name of the keystore for signing acknowledgments.
MSH/DigitalSignature	The signing algorithm used for signing

/AckSign/KeyStore /Algorithm	acknowledgments. If not specified, “dsa-sha1” is assumed. Currently only the values “dsa-sha1” or “rsa-sha1” are valid.
MSH/DigitalSignature /AckSign/KeyStore /Alias	The alias of the key used for signing acknowledgments.
MSH/DigitalSignature /AckSign/KeyStore /Password	The password of the keystore for signing acknowledgments.
MSH/Persistent /Database/Driver	The JDBC driver class name used for connecting to the database.
MSH/Persistent /Database/User	The user to connect to the database.
MSH/Persistent /Database/Password	The password to connect to the database.
MSH/Persistent /Database/URL	The JDBC URL to connect to the database.
MSH/Persistent /Database /TransactionIsolationLevel	The transaction isolation level of database. Valid settings are “READ_COMMITTED”, “READ_UNCOMMITTED”, “REPEATABLE_READ” and “SERIALIZABLE”.
MSH/Persistent /Database /InitialConnections	The initial pool size of the database pool.
MSH/Persistent /Database /MaxConnections	The maximum pool size of the database pool.
MSH/Persistent /Database /MaximumWait	The maximum waiting time for MSH to wait for an available database connection.
MSH/Persistent /Database /MaximumIdle	The maximum time that a database connection can be idled. The MSH would reconnect the database connection that has been idled more than the specified period.
MSH/Persistent /MessageRepository	The repository directory for storing ebXML messages persistently for tracking and resilience.
MSH/Persistent /Database/MaxWait	The maximum period of time in milliseconds for MSH to wait for an available database connection from the connection pool.
MSH/Persistent /MaxFiles	The maximum number of files to be stored in a single sub-directory in the repository directory.
MSH/Persistent /BackupFile	The file name to place the backup file when a MSH backup command is issued.
MSH/Persistent /ArchiveDirectory	The directory name where the archived data are placed when a MSH archive command is issued.
MSH/MessageListener /TrustedRepository	A semi-colon delimited string containing the path that the MSH application server can

	save the received messages.
MSH/MessageListener /ObjectStore	The repository directory for storing the state of the MSH server. The MSH server can restore its last state by loading these objects.

## A.2. msh\_client.properties.xml

This properties file is used by the MSH stub:

Field	Description
Request/Log/UseLogger	The logging parameter for the MSH Stub. Choose whether Log4J or JDK logging package is used for logging. Valid values are "LOG4J" and "JDK".
Request/Log/LogPath	The logging parameter for the MSH Stub. The directory for storing the log file.
Request/Log/LogFile	The logging parameter for the MSH Stub. The log file name.
Request/Log/LogLevel	The logging parameter for the MSH Stub. The log level of the logger. Valid values range from 0 to 4. "0" means most log, and "4" means no log.
Request/Log /MaxFileSize	The maximum log file size in bytes. If the maximum size is reached, a new log file will be created and the logger will "roll over" to use the new log file. If this property is missing, or the value is smaller than 0, the roll over is disabled.
Request/Config/URL	The URL for the client applications to locate MSH application server. You should fill in the host name, port number and context path of the MSH application server.
Request/Config /MonitorInterval	The time interval for the client applications to connect to MSH server at Tomcat for downloading new messages.
Request/Config /UserName	The user name to attach to every command message sending to the MSH server for authentication.
Request/Config /Password	The password to attach to every command message sending to the MSH server for authentication.
Request/Config /MaxNumPayload	Optional parameter controlling the maximum number of payload allowed. If not specified, no bound on maximum is assumed.
Request/Config /MaxPayloadSize	Optional parameter controlling the maximum size of payload allowed in bytes. If not specified, no bound on maximum is assumed.

Request/Proxy/Host	The host name of HTTP proxy server for sending out commands/messages to the MSH server. If you can connect to the MSH server without using proxy server, this field should be commented.
Request/Proxy/Port	The port number of HTTP proxy server for sending out commands/messages to the MSH server. If you can connect to the MSH server without using proxy server, this field should be commented.
Request /MessageListener /MessageRepository	The temporary directory at client application side for storing messages. The MSH can be configured to forward all received messages directly to the machine running client applications, this is the directory to store the forwarded messages.

### A.3. monitor.properties.xml

This properties file is used by the Monitor application:

Field	Description
Request/Monitor /DefaultConfig /ToMSHURL	The default configuration parameters for the sample application: MSH Monitor. This represents the To MSH URL field.
Request/Monitor /DefaultConfig/CPAID	The default configuration parameters for the sample application: MSH Monitor. This represents the CPA ID field.
Request/Monitor /DefaultConfig /ConversationID	The default configuration parameters for the sample application: MSH Monitor. This represents the Conversation ID field.
Request/Monitor /DefaultConfig /Service	The default configuration parameters for the sample application: MSH Monitor. This represents the Service field.
Request/Monitor /DefaultConfig /Action	The default configuration parameters for the sample application: MSH Monitor. This represents the Action field.

## Appendix B. Error Codes

Code	Description
10001	Initialization error
10002	Unknown error
10003	Property not properly set
10004	Data error
10005	State error
10006	File IO error
10007	File not found
10008	HTTP POST request failed
10009	Servlet IO error
10010	Authentication failed
10011	Registration failed
10012	Unknown application context
10013	Error reading/writing ZIP stream
10101	Invalid SMTP server
10102	Cannot compose mail message
10103	Cannot send mail message
10104	Invalid POP/IMAP folder
10105	Invalid POP/IMAP server
10201	Invalid keytore
10202	Cannot encrypt message
10203	Cannot decrypt message
10204	Cannot sign message
10205	Verification of signature failed
10301	Cannot close DB connection
10302	Invalid parameter
10303	Cannot get DB connection
10304	DB connection are freed more than allocated
10305	Cannot load JDBC driver
10306	Cannot create DB connection
10307	Existing DB schema incorrect
10308	Data inconsistency
10309	Cannot create DB table
10310	Cannot query record from DB
10311	Cannot write record to DB
10312	Database backup error
10313	Database restore error
10314	Cannot commit DB changes
10315	Cannot rollback DB changes
10316	Cannot find message in DB
10401	Cannot unlock a non-existent object
10402	Cannot rollback
10501	Default MessageFactory cannot be instantiated
10502	Cannot serialize SOAP message

---

10503	Cannot internalize SOAP object
10504	Cannot save change to SOAP message
10505	Cannot send SOAP message
10506	Cannot initialize SOAP connection
10507	Invalid HTTP SOAP connection
10508	Cannot create SOAP object
10509	Cannot process SOAP message

## Appendix C. Sample code of client application

```
import java.net.*;
import java.util.*;
import javax.activation.*;
import hk.hku.cecid.phoenix.message.handler.*;
import hk.hku.cecid.phoenix.message.packaging.*;

public class LoopBack implements MessageListener {

    public static void main(String [] args) {
        LoopBack sample = new LoopBack();
        sample.run();
    }

    public void run() {
        try {
            System.out.println("Packaging...");

            String cpaID = "CPA_2002";
            String conversationID = "Item_No_128";
            String service = "http://www.cecid.hku.hk/ebxml/service";
            String action = "Order";

            ApplicationContext ac = new ApplicationContext(
                cpaID, conversationID, service, action);

            String transportType = "HTTP";
            String toMshUrl = "http://localhost:8980/msh";

            Request mshReq = new Request(
                ac, new URL(toMshUrl), this, transportType);

            AttachmentDataSource ads = new AttachmentDataSource(
                "po.xml", "text/xml");
            DataHandler dataHandler = new DataHandler(ads);

            EbxmlMessage message = new EbxmlMessage();

            MessageHeader header = message.addMessageHeader();
            header.addFromPartyId("fromPartyId", "Sample");
            header.addToPartyId("toPartyId", "Sample");
            header.setCpaId(cpaID);
            header.setConversationId(conversationID);
            header.setService(service);
            header.setAction(action);
            header.setTimestamp(MessageServiceHandler.timeStamp(
                new Date()));

            String messageId = MessageServiceHandler.messageId(
                new Date(), message);
            header.setMessageId(messageId);

            message.addPayloadContainer(
                dataHandler, "contentId", "description");

            message.saveChanges();
        }
    }
}
```

```
        System.out.println("Try to send...");

        mshReq.send(message);

        System.out.println("Message sent!");
        System.out.println("Sleep for 10 seconds then quit...");

        Thread.sleep(10000);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

public URL getClientUrl() {
    return null;
}

public void onMessage(EbxmlMessage ebxmlMessage) {
    System.out.println("Message received!");
}
}
```