DANGEROUS GOODS MANIFEST SUBMISSION

WITH EBXML MESSAGE SERVICE

(XMLDG)

for

The MARINE DEPARTMENT

TECHNICAL IMPLEMENTATION REFERENCES

Version: 1.0

October 2003

COPYRIGHT

LIMITATION OF LIABILITY

The contents of this Specification are provided for the information and use of all users of the Electronic Submission of Dangerous Goods Manifests Service ("the Service"), including vessel owners, agents and masters. All information contained in this Specification is believed to be accurate but neither the copyright holder nor any person or organisation concerned in the preparation or publication of this Specification accepts any liability of any nature whatsoever for any loss suffered either directly or indirectly as a consequence of the use by users of the Service or this Specification or any trading activity flowing therefrom.

For full details of legal requirements related to the Service, it should refer to the section 4 of the Dangerous Goods (Shipping) Regulations.

TABLE OF CONTENT

1. INTRODUCTION

This document provides technical references on the implementation of the XMLDG project. This document also outlines the deployment architecture, document processor workflow, and feature list of Hermes Message Service Handler (MSH).

For the XMLDG schema related document, please reference HKSAR Dangerous Goods Manifest XML Schema Specification (SS), and XMLDG Implementation Instructions (II).

2. DEPLOYMENT ARCHITECTURE

2.1 XMLDG GATEWAY ARCHITECTURE

It is a PC/Server to serve as a doorway to handle all ebXML messages, as well as handle all DG related XML documents sending and receiving.

There are two main pieces of software in the message gateway host.

1. Hermes Message Service Handler (MSH)

   - It is responsible to send and receive ebXML message with tracking facilities.
   - It requires a database (any JDBC compliant database) for its persistent storage.

   - For more technical detail on MSH, please consult the Hermes MSH documents.

2. Data Handler

   - Retrieve data from the database, package in ebXML message format and submit to MSH
   - Obtain an ebXML message from MSH, then extract the data from the XML document and do further processing.

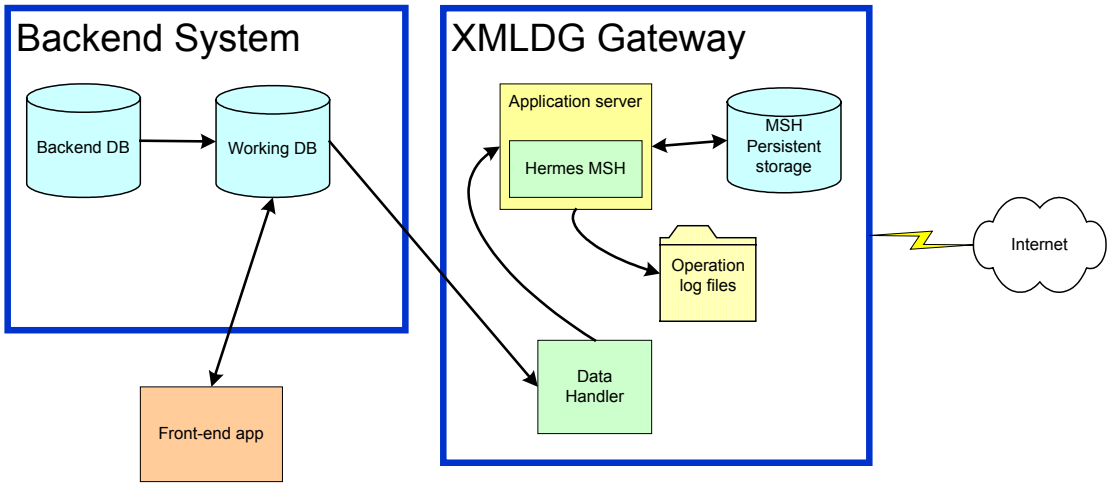


Figure 1: Message gateway architecture overview

2.2 MESSAGE GATEWAY HARDWARE REQUIREMENT

The recommendation of message gateway configuration:

CPU:	800MHz
Memory:	256MB
Hard Disk:	4GB

* If you want to keep message archive for a longer period of time, larger hard disk space is required.

2.3 MESSAGE GATEWAY SOFTWARE REQUIREMENT

The implement of MSH is a java servlet, so the following software should be installed

1. Java SDK Standard Edition 1.4.X
2. Application server with servlet container (e.g. Tomcat server)
3. A JDBC compliant database for persistent storage.

We have successfully test Hermes MSH running properly under Red Hat Linux, Solaris, Windows 2000 & Windows XP

2.4 PROPOSED DATA HANDLER DEPLOYMENT MODE

2.4.1 Data handler architecture

The implementation of a data handler will be varied under different environment; however it should provide a basic function that can send and receive messages. Inside a data handler, there is an interface with XMLDG business vocabulary library. This library provides APIs to manipulate a XMLDG XML document including parsing and composing. Users can also use those APIs to get and set value of each XML element. All XMDG schema syntax checking and business rules validations will be taken care by the library.

Moreover, a data handler should contain error-handling mechanism, so that it can recover any failure to send and receive messages.

Under ebXML message service (ebMS) specification, how a MSH identifies a data handler is based on information specified in `ApplicationContext`. Without this piece of information, MSH cannot deliver a right message to a right data handler. In this project, the `ApplicationContext` should be pre-defined among all parties. An `ApplicationContext` contains four elements.

1. Collaboration Protocol Agreement ID
2. Conversation ID
3. Service
4. Action

The implementation of XMLDG business vocabulary library is a set of java classes. In order to use this library, we recommend using java as a data handler programming language.

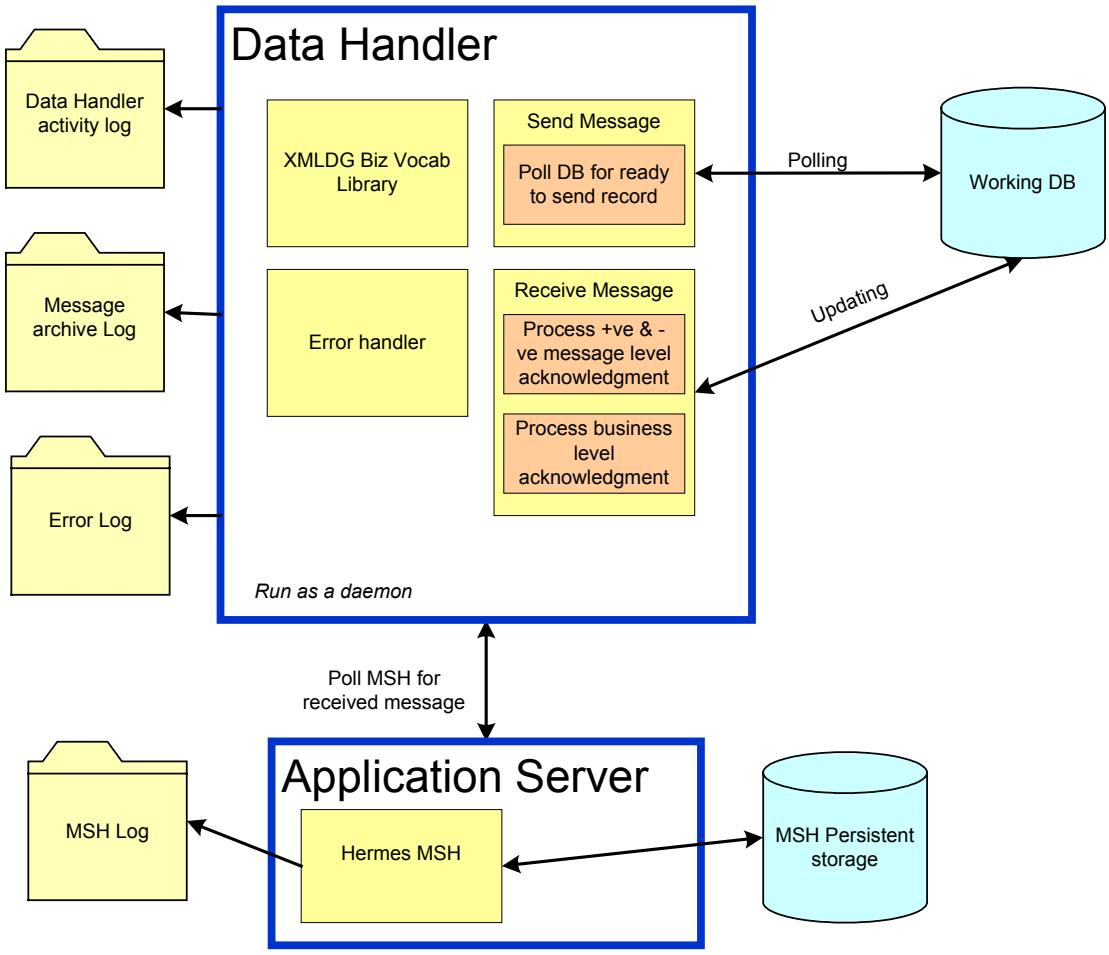


Figure 2: Data handler component

2.4.2 Deployment mode

Assumption:

1. A working database that contains all "ready to send" records.
2. A data handler will interactive with the working database, and MSH only. It has no knowledge on any backend system.

The recommended deployment mode is using one data handler (one process) to handle all sending and receiving requests. The data handler will run as a daemon process, and it will continue polling the working database to check whether there are "ready to send" records. If this is the case, the data handler will send those records out, as well as perform some business logics rules.

There will be three kind of received messages.

1. Positive receive acknowledgment

   Receiver MSH receives the message successfully, and a positive receive acknowledgment will send back to the sender.

2. Negative receive acknowledgment

Sender MSH cannot deliver the message to receiver MSH after specified retry period.

3. Business acknowledgment

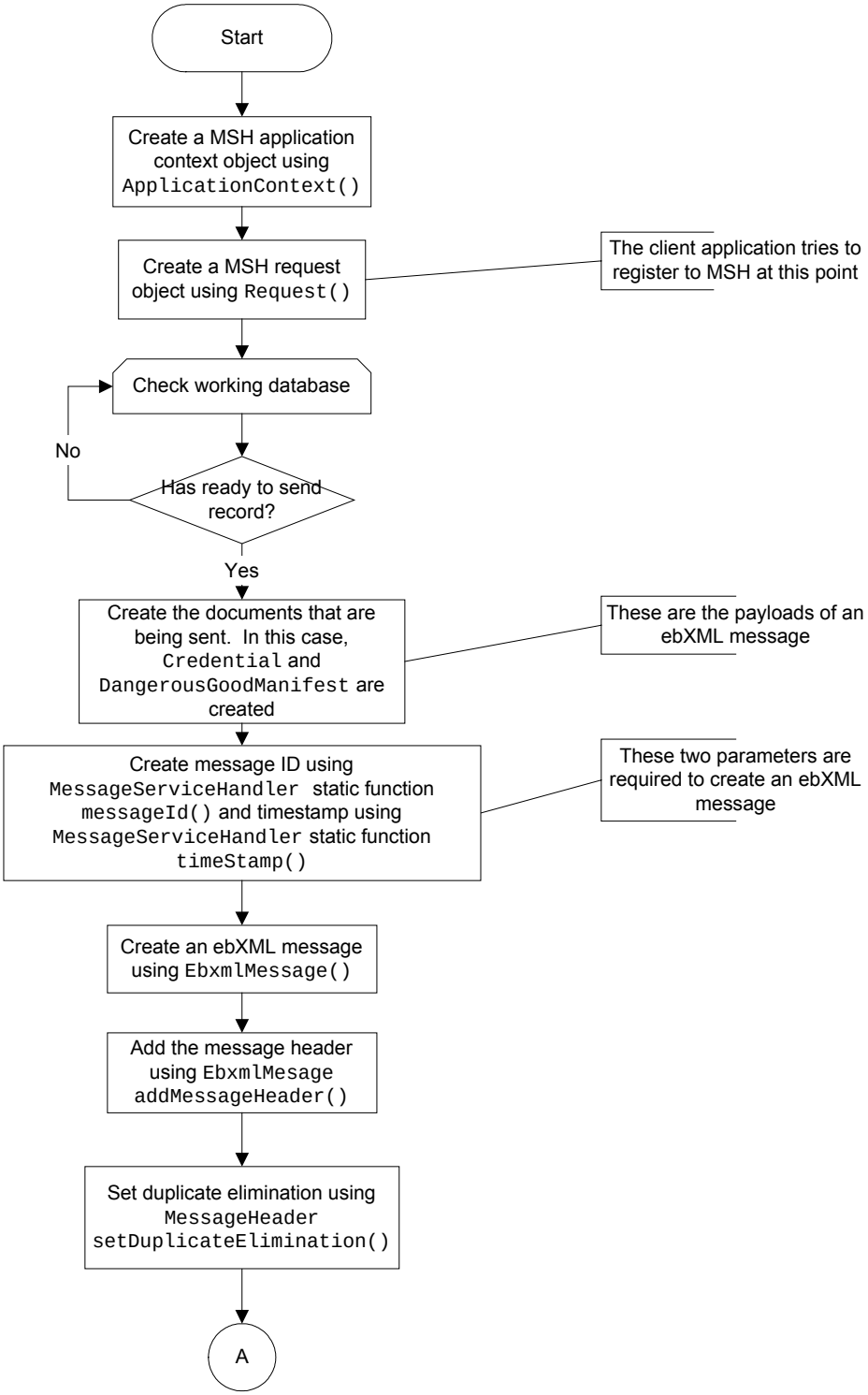
   An acknowledgement sent by the data handler of the receiver end after data handler processed the message based on business logic and process flow.

Since the data handler is run in background, the MSH will invoke it when a message is received. The data handler should have the ability to determine what kind of received message and perform certain kind of action accordingly.

3. DATA HANDLER PROGRAMMING

3.1 SHIPPING AGENT

3.1.1 Sending message flow



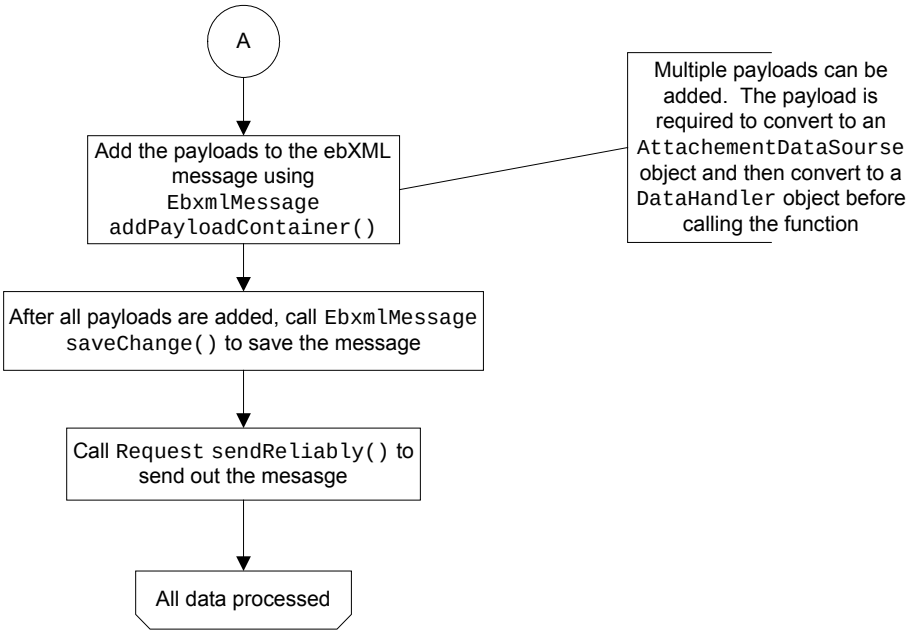


Figure 3: Sending message programming flow for shipping agent

3.1.2 Receiving message flow

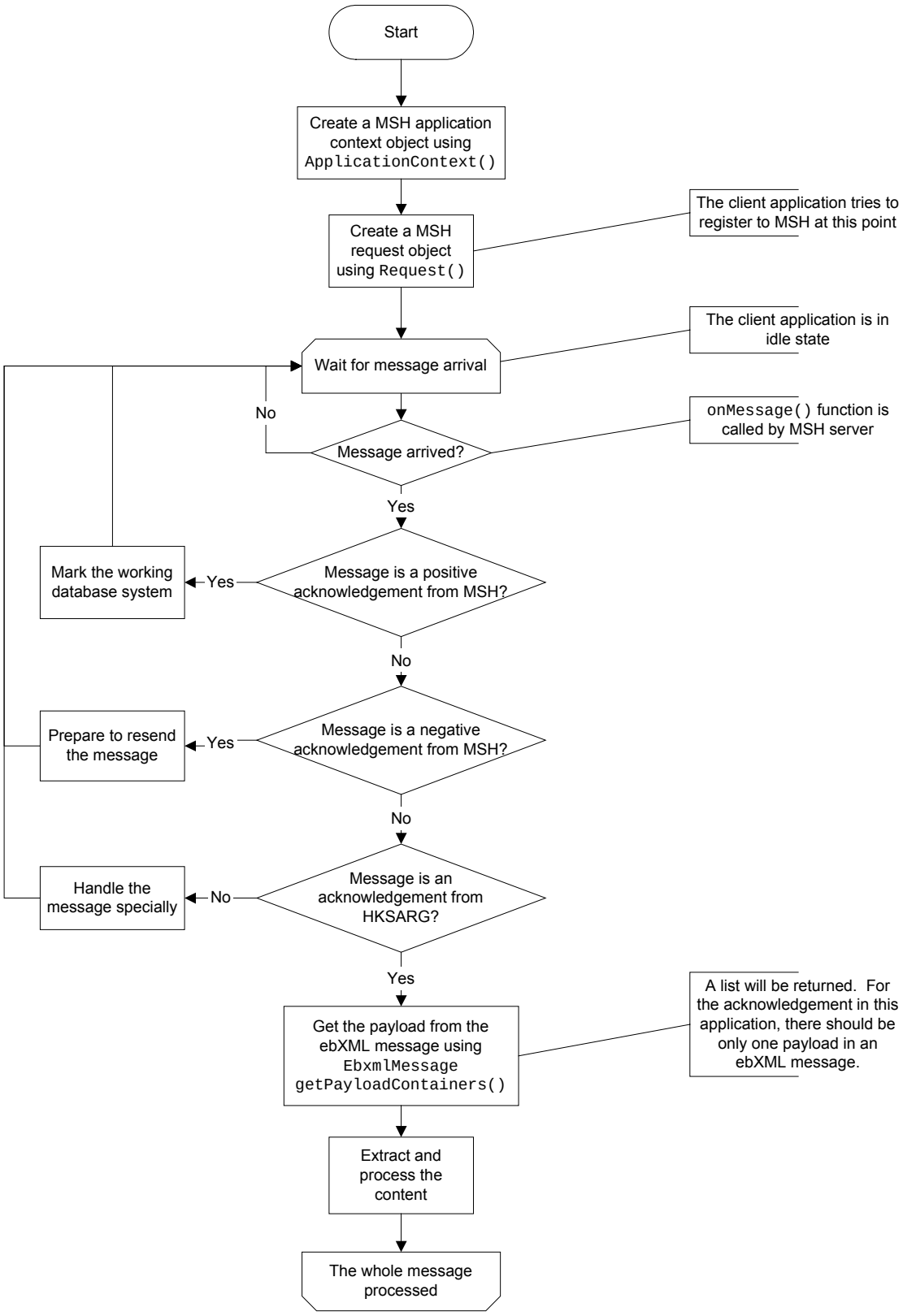
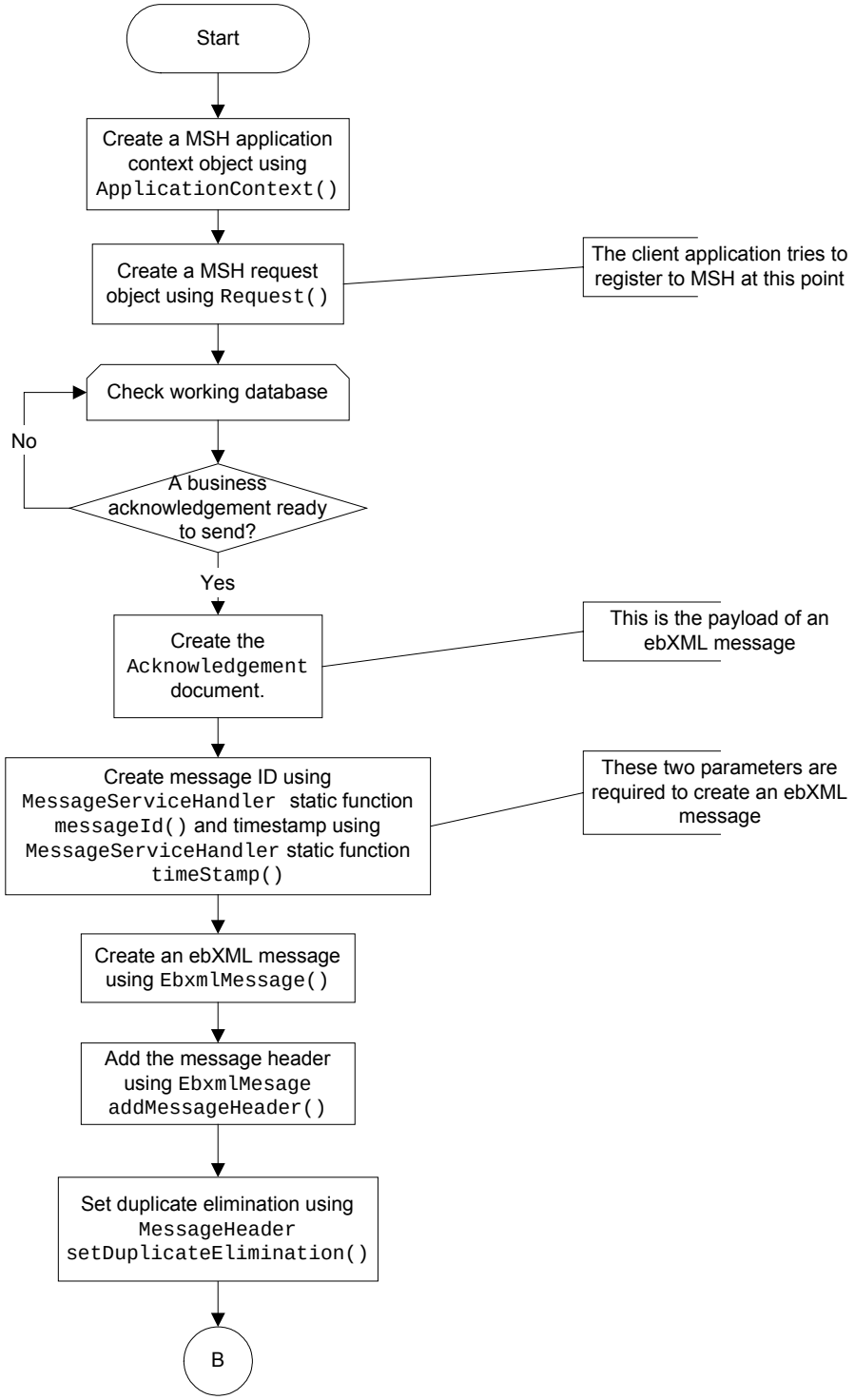


Figure 4: Receiving message programming flow for shipping agent

3.2 MARINE DEPARTMENT

3.2.1 Sending message flow



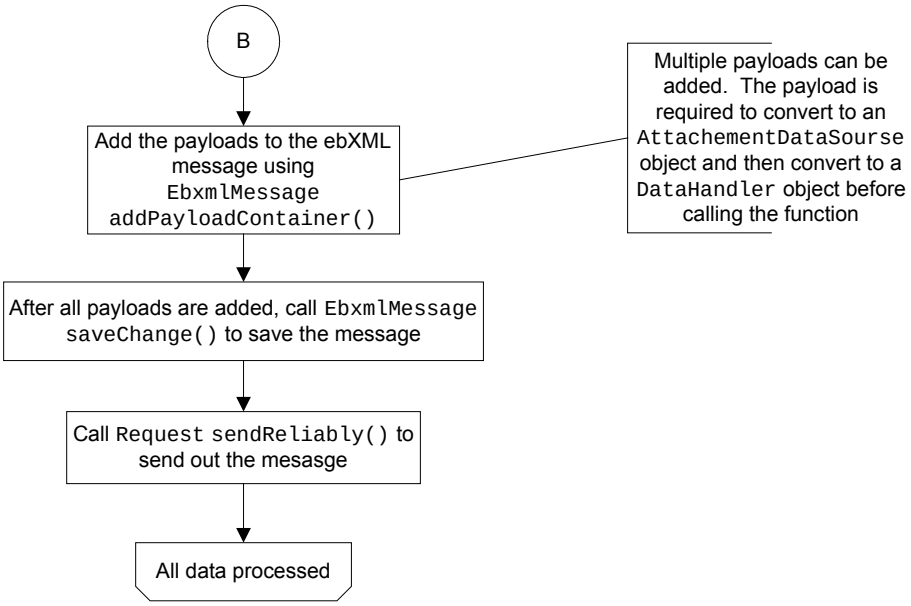


Figure 5: Sending message programming flow for Marine Department

3.2.2 Receiving message flow

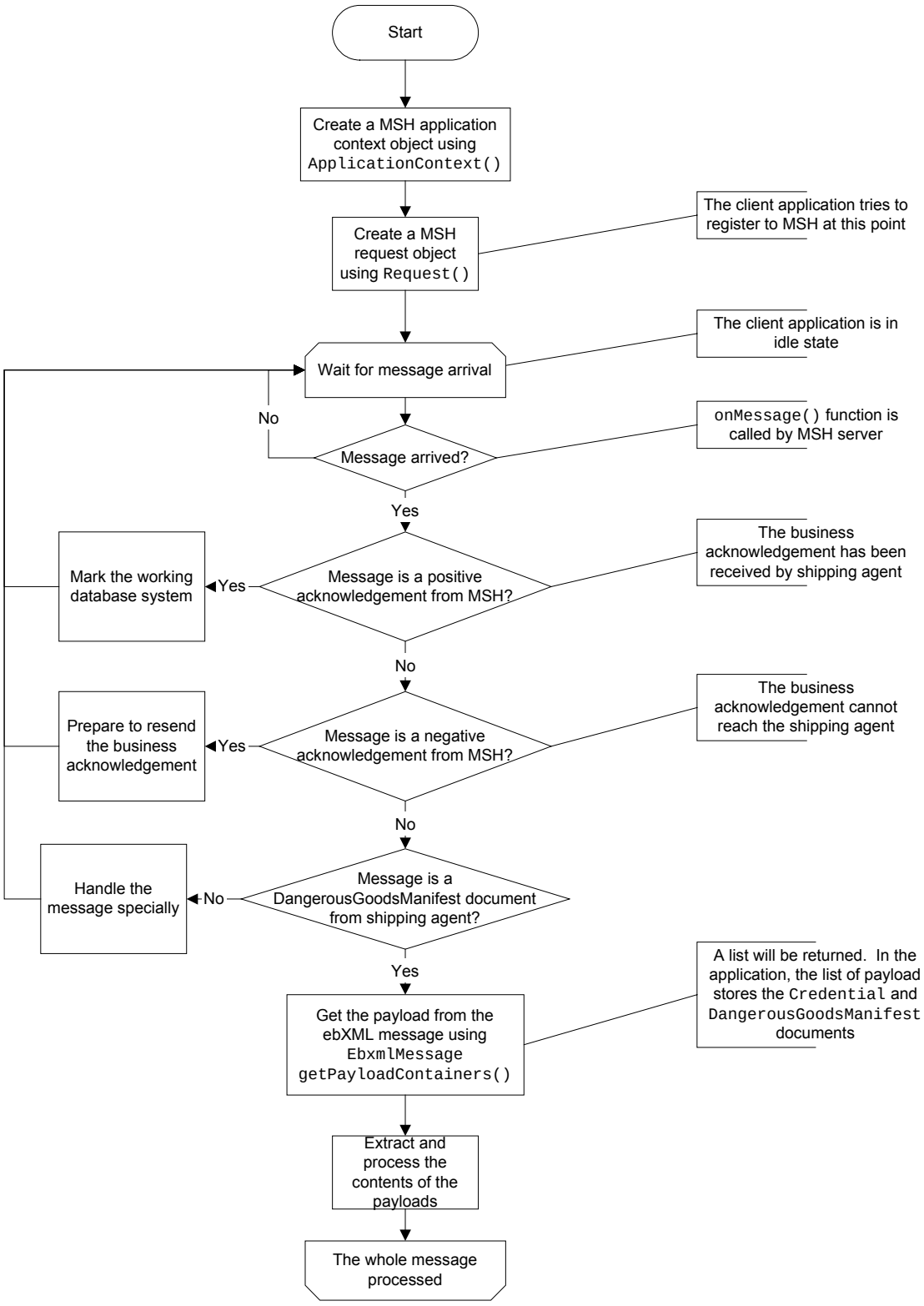


Figure 6: Receiving message programming flow for Marine Department

3.3 MESSAGE LIFE-TIME

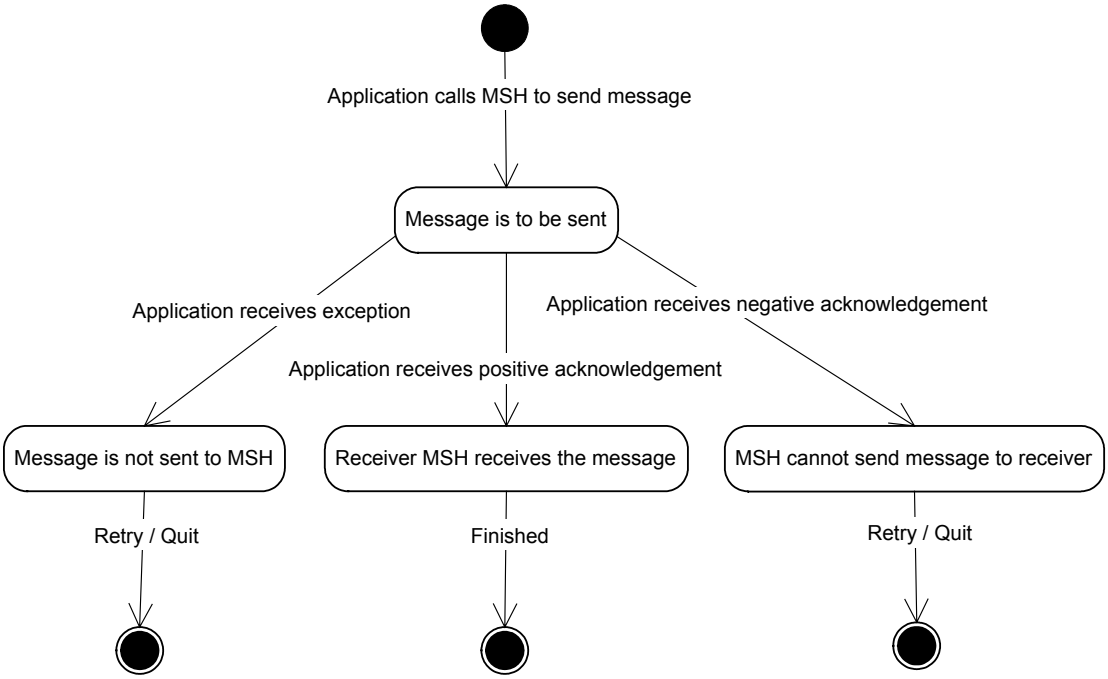


Figure 7: Message life-time state diagram

4. USING XMLDG BUSINESS VOCABULARY

The library covers the three types of documents that are involved in the dangerous goods manifest submission process. These documents are dangerous goods manifest, acknowledgement and credential. In this document, the sequence of object creation is provided and dangerous goods manifest is used as an example. For more details, please refer to XMLDG Vocabulary Library User Guide.

4.1 PACKAGE OVERVIEW

There are four packages in the XMLDG business vocabulary library.

1. hk.hku.cecid.phoenix.dg

   This package contains the classes that used to compose a Dangerous Goods Manifest document, e.g. `DocumentHeader`, `DangerousGoodsManifest`, `Transport` and `Acknowledgement` etc.

2. hk.hku.cecid.phoenix.vocab

   This package contains the exception classes like `ConstraintViolatedException`, `VocabException`.

3. hk.hku.cecid.phoenix.vocab.cct

   This package contains the core component types that are used in the library, e.g. `CodeType`, `TextType` and `IdType` etc.

4. hk.hku.cecid.phoenix.vocab.util

   This package contains all utility classes, e.g. `ElementAppender`, `Namespaces` and `Timestamp` etc.

4.2 DANGEROUS GOODS MANIFEST DOCUMENT CREATION

This is the document for agents to declare the dangerous goods to HKSARG. Each document must contain one document header, transport information and a number of dangerous goods items. In this library, a dangerous goods manifest document is represented using a `DangerousGoodsManifest` object.

4.2.1 Creation Sequence

The sequence of creating a dangerous goods manifest document is shown as follows:

1) Create a new DangerousGoodsManifest document instance using `newInstance()` method.

```
DangerousGoodsManifest dgm =
                    DanagerousGoodsManifest.newInstance();
```

2) Get the default DocumentHeader in the Manifest document using `getDocumentHeader()` method.

```
DocumentHeader header = dgm.getDocumentHeader();
```

3) Set the content of the child elements. DocumentId is used as an example.

```
DocumentId documentId = header.getDocumentId();
DocumentId.setContent("1234@shippingagent.com");
```

4) Create and set Transport in the Manifest document using `getTransport()` and `setTransport()` methods. With the `hk.hku.cecid.phoenix.vocab.appendOptionalChildren` attribute in the properties file `vocab.properties` set to `false`, optional elements are not created by the library. The optional elements should be created and then appended to its parent.

```
Transport transport = dgm.createTransport();
dgm.setTransport(transport);
```

5) Set the content of other child elements. Vessel and its child, Name, are used as an example.

```
Vessel vessel = transport.createVessel();
Name name = vessel.createName();
name.setContent("UNI-ACCORD");
vessel.setName(name);
transport.setVessel(vessel);
```

6) Add the list of DangerousGoodsItem into the Manifest document. Firstly, `createDangerousGoodsItem()` method is called to create the item.

```
DangerousGoodsItem item = dgm.createDangerousGoodsItem();
```

7) Set the content of the child elements. DangerousGoodsItem is used as an example.

```
DangerousGoodsItem item1 =
        dgm.createDangerousGoodsItem();
dgm.addDangerousGoodsItem(item1);
```

8) After all content of the child elements have been set, `addDangerourGoodsItem()` method is called to add the item into the list.

```
Dgm.addDangerousGoodsItem(item);
```

4.2.2 Element Values

When an element is created, its value is created according to the following rules:

- If there is a fixed value constraint in II for the element, the fixed value is set.
- For elements with fixed attribute in II, the fixed attribute and its value are set.
- For elements with fixed enumeration values both in SS and II, the first value of the enumeration is set. The II overrides the SS.
- If the above rules do not apply, empty string is set for element with `String` type; "0" is set for element with `Integer` type; "0.0" is set for element with `Decimal` type; current date-time is set for element with `dateTime` type. The data types are the data types in the SS.

When an element is created, the number of elements is created according to the following rule:

- If it is a mandatory child specified in the SS, the library creates the child with its occurrence according to the SS.
- If it is an optional child, the library looks for the `hk.hku.cecid.phoenix.vocab.appendOptionalChildren` attribute in the properties file `vocab.properties`. If the attribute is set to `true`, the number of children created is always 1, otherwise, the optional child is not created. It is suggested to set the attribute to be `false`. Optional element requires special handling including element creation and appending to its parent.

4.3 DANGEROUS GOODS MANIFEST DOCUMENT VALIDATION

The library performs validation when :

- element content is set (`setContent()`),
- a child element is added to a parent,
- data is compiled to XML form (`marshal()`)
- information is extracted from XML form (`unmarshal()`)

During validation:

- The library validates the document against the constraints in II followed by those in the SS.
- A newline character is considered as 1 character in a String.
- For a digit which is a negative number, the negative sign is considered as 1 digit in the integral part.

If the XMLDG schema or XMLDG II is violated,
`hk.hku.cecid.phoenix.vocab.VocabException` is thrown.

5. HERMES MSH

5.1 FEATURE LIST AND RELEASE

Hermes MSH specification is based on ebXML Messaging Services (ebMS) Specification V2.0 (http://www.oasis-open.org/committees/ebxml-msg/). In order to enrich Hermes MSH functionality, we have injected several features that are not stated in ebMS specification. Here is a highlight.

Hermes MSH Features	Code Type	Release
Normal Runtime Operation		
Send message		
Retry mechanism	Runtime	0.9.1.0
Duplication elminiation	Runtime	0.9.1.0
Check arrival message		
onMessage(). MSH server will call the onMessage() function implemented by the client application whenever there is an incoming message.	API	0.9.1.0
Positive receive acknowledgement	Runtime	0.9.2.0
Negative receive acknowledgement	Runtime	0.9.1.0
Message Tracking		
API to check message status		
individual message status (sent, retries…)	API	0.9.2.0
number/list of pending messages	API	0.9.2.0
Diagnosis Dump	MSH script	0.9.2.0
MSH Monitoring		
API to check msh status		
Check database availability	API	0.9.1.3
Check Java VM environment	API	0.9.1.3
Check file system	API	0.9.1.3
MSH Maintenance		
Start/Stop	Tomcat script	0.9.1.0
API to manage msh		
Halt service (simply deny response any send/receive request)	API	0.9.2.0
Clean halt service (send a negative acknowledge for any unsend message, and then deny the service)	API	0.9.2.0
Resume service	API	0.9.2.0
Deletion of a queued message	API	0.9.2.0
Archiving by date (filesystem + database) All archived data will be stored in a file system, and the original data will be deleted.	API	0.9.2.0
Backup MSH snapshot (filesystem + database)	API	0.9.2.0
Recover backup snapshot (to a clean environment)	API	0.9.2.0

Current release : 0.9.3.1

5.2 MSH API

For the full MSH API list, please refer to the MSH java doc. Here are some API highlights that will available started from release 0.9.2.0.

5.2.1 Normal runtime operation

Positive receive acknowledgment

API Signature	`void sendReliably(EbxmlMessage msg, boolean needSignedACK)`
Input	The message to be sent and a flag to indicate whether signed ACK is needed from receiving end. Note: whether positive ACK is generated or not is determined in the properties file of MSH.
Output	A special ebXML message will be delivered to application containing the message ID of the sent message.

Negative receive acknowledgment

API Signature	`void sendReliably(EbxmlMessage msg, boolean needSignedACK)`
Input	The message to be sent and a flag to indicate whether signed ACK is needed from receiving end.
Output	A special ebXML message will be delivered to application containing the message ID of the sent message.

5.2.2 Message Tracking

Check individual message status

API Signature	`String getMessageStatus(ArrayList messageIdList)`
Input	A list of message ID to query
Output	An XML string returning the status of the queried message.

Check number/list of pending messages

API Signature	`String [] getPendingMessages()`
Input	N/A
Output	A string array containing the list of pending messages

5.2.3 MSH Monitoring

Check database availability

API Signature	`String checkDatabase()`
Input	N/A
Output	An XML string returning either success or the failed reason.

Check Java VM environment

API Signature	`String reportEnvironment()`
Input	N/A
Output	An XML string returning either success or the failed reason.

Check file system

API Signature	`String checkPersistence()`
Input	N/A
Output	An XML string returning either success or the failed reason.

5.2.4 MSH Maintenance

Halt service

API Signature	`String haltMSH(int level)`
Input	Halt level (NORMAL_HALT, or CLEAN_HALT)
Output	An XML string returning the status of the queried message.

Clean Halt service

API Signature	`String haltMSH(int level)`
Input	Halt level (NORMAL_HALT, or CLEAN_HALT)
Output	An XML string returning the status of the queried message.

Resume service

API Signature	`String resumeMSH()`
Input	N/A
Output	An XML string returning the status of the queried message.

Deletion of pending messages

API Signature	`Map deletePendingMessages(ArrayList messageIdList)`
Input	A list of message ID to delete
Output	An XML string returning the status of the queried message.

Archiving by date

API Signature	`String archiveByDate(Date date)`
Input	Archive reference date. The archive will be made on data on or before the pass-in date.
Output	A Map with message ID as key, and with deletion status as value

Backup MSH snapshot

API Signature	`String backupMSH()`
Input	N/A
Output	An XML string returning the status of the queried message.

Restore MSH snapshot

API Signature	`String restoreMSH()`
Input	N/A
Output	An XML string returning the status of the queried message.

5.3 MSH SCRIPT

For the full technical details, please refer to the MSH Installation Guide. Here is the high level description that will be available started from release 0.9.2.0.

5.3.1 Message Tracking

Diagnosis dump

MSH Script	The script is stored under the diagnosis subdirectory.
Input	`diagnosis.properties.xml` stores the diagnosis dump configuration. Input criteria include time period and message type (application context).
Output	Relevant entries in the database and corresponding message files in message repository are returned.

5.4 MSH TOOLS

Several tools are created based on MSH APIs. For more information, please go to the project website and download the tools.

5.4.1 MSH heartbeat tool

This is a tool that can send a loop back message to a MSH, check the response, and the result will be written to a log. You can execute this tool as a scheduled job (e.g. runs every 1 hours). When the heartbeat fails, it will send out an alert email. Also, this script is NOT necessarily placed under the same machine of MSH.

5.4.2 MSH System Commander

This is a tool that can send a direct command to a Hermes MSH.

Here is a list of commands

Command	Function
checkdb	Check database connection
resetdb	Reset database connection pool
suspend	Suspend MSH
term	Terminate MSH and then send negative ACK for all pending message
resume	Resume MSH
backup	Backup MSH files and database entries
restore	Restore MSH from backup files
archive	Archive-related functions. Options and

	argument 1. archive by date 2. archive by application context 3. delete the archive messages
housekeep	Housekeep startup file (with delete unreferenced startup files options)

5.4.3 Hermes MSH Webmin

It is a web interface for MSH administration. Some of the modules are the same as those provided by System Commander.

Here is a list of modules of Webmin

- Diagnosis Dump
- Message Tracking
- MSH Database Information
- MSH Information
- MSH Maintenance
- Message Archive

5.5 AUTO-GENERATED ELEMENTS IN EBXML MESSAGE HEADER

Every ebXML message requires a message header to specify the properties of the message and the communication channel transferring it. `MessageId` element and `Timestamp` element are elements of ebXML MessageHeader in an ebXML message. Both of the elements are auto-generated by MSH.

5.5.1 `MessageId` element

`MessageId` is a unique identifier for each ebXML message. It is auto-generated by MSH based on the following items:

system time (up to millisecond)

CPAId element

Service element

Action element

host IP address of sender

sequence number (maintained by MSH internally)

5.5.2 `Timestamp` element

The `Timestamp` element contains a value representing the time that the message header was created. It is handled by MSH.

5.6 MSH DATABASE

Hermes MSH requires a database to store the status of communication, to support resilience, reliable messaging, message tracking, etc.

SQL92 standard when designing the MSH database tables, so basically it support all RDBMS, as long as their JDBC drivers are available.

For Hermes MSH v0.9.3.1, there are 9 tables and the scripts are shown below.

```
CREATE TABLE MSHConfig (

    c_cpaid VARCHAR(128) not null,

    c_conversationid VARCHAR(128) not null,

    c_service VARCHAR(128) not null,

    c_action VARCHAR(64) not null,
```

```
    c_tomshurl VARCHAR(255) not null,

    c_messagelistener VARCHAR(255) not null,

    c_transporttype VARCHAR(10) not null,

    c_retries INTEGER,

    c_retryinterval VARCHAR(32) not null,

    c_syncreply INTEGER,

    c_messageorder CHAR,

    c_persistduration VARCHAR(32) not null,

    c_enabled CHAR,

    primary key (c_cpaid, c_conversationid, c_service,
c_action)

);



CREATE TABLE MessageStore (

    c_messageid VARCHAR(255) not null,

    c_cpaid VARCHAR(128) not null,

    c_conversationid VARCHAR(128) not null,

    c_service VARCHAR(128) not null,

    c_action VARCHAR(64) not null,

    c_filename VARCHAR(255) not null,

    c_state INTEGER,

    c_sequenceNumber INTEGER,

    c_archived CHAR,

    primary key (c_messageid)

);



CREATE TABLE MessageInfo (

    c_messageid VARCHAR(255) not null,

    c_numberofpayload INTEGER,

    c_timestamp VARCHAR(32) not null,
```

```
    c_rowtimestamp VARCHAR(32) not null,
    primary key (c_messageid)
);

CREATE TABLE RefToMessage (
    c_messageid VARCHAR(255) not null,
    c_reftomessageid VARCHAR(255) not null,
    primary key (c_messageid)
);

CREATE TABLE SendingState (
    c_messageid VARCHAR(255) not null,
    c_currentRetry INTEGER,
    c_nextretrytime VARCHAR(32) not null,
    primary key (c_messageid)
);

CREATE TABLE SentMessage (
    c_time VARCHAR(32) not null,
    c_messageid VARCHAR(255) not null,
    c_status VARCHAR(255),
    c_frompartyid VARCHAR(128) not null,
    c_topartyid VARCHAR(128) not null,
    c_cpaid VARCHAR(128) not null,
    c_conversationid VARCHAR(128) not null,
    c_service VARCHAR(128) not null,
    c_action VARCHAR(64) not null,
    c_ackrequested CHAR,
    c_dupElimination CHAR
```

```
);


CREATE TABLE ReceivedMessage (
    c_time VARCHAR(32) not null,
    c_remoteaddress VARCHAR(16) not null,
    c_remotehost VARCHAR(32) not null,
    c_messageid VARCHAR(255) not null,
    c_status VARCHAR(255),
    c_frompartyid VARCHAR(128) not null,
    c_topartyid VARCHAR(128) not null,
    c_cpaid VARCHAR(128) not null,
    c_conversationid VARCHAR(128) not null,
    c_service VARCHAR(128) not null,
    c_action VARCHAR(64) not null,
    c_ackrequested CHAR,
    c_dupElimination CHAR
);


CREATE TABLE MSHLog (
    c_time VARCHAR(32) not null,
    c_action VARCHAR(64) not null
);
```

6. XMLDG FAQ

6.1 ABOUT XMLDG

Q: What is the XMLDG system?

A: XMLDG service, implemented in accordance with OASIS ebXML Message Service (ebMS) V2 Standard, is provided by Marine Department (MD) of HKSARG to facilitate the system-to-system submission mode of Dangerous Goods Manifest in XML format electronically via Internet.

Q: What are the benefits of using XMLDG for Dangerous Goods Manifest submission?

A: The XMLDG service enables the submission end to directly generate from their backend computer systems the Dangerous Goods Manifests and send them to MD through the Internet. This can help to increase the efficiency by eliminating the manual data input efforts and eliminating the human typo.

Q: How to join XMLDG?

A: Interested parties who wish to join or obtain further information on XMLDG may contact:

The Dangerous Goods & Project Section, Marine Department
Tel. 28523085
Fax. 28158596
Email: pfdg@mardep.gov.hk

6.2 XML SCHEMA & IMPLEMENTATION DOCUMENTS

Q: What is the purpose of Dangerous Goods Manifest XML Schema Specification (SS)?

A: SS provides the XML schema definitions for the business documents to be exchanged in the process of Dangerous Goods Manifest submission.

Q: What are the business documents being exchanged in the project?

A: Currently, there are three types of documents involved in the XMLDG submission process, namely Dangerous Goods Manifest (DGM), Acknowledgement and Credential. DGM is the business document for agents to declare dangerous goods

to HKSARG. Acknowledgement is the business document for HKSARG to acknowledge the DGM submission. Credential contains user identifier and password for HKSARG to authenticate the sender of the DGM.

Q: Does the SS match with International Maritime Organization's (IMO) recommendation?

A: DGM specified in the SS is defined and specified according to the IMO's recommendation.

Q: What is the purpose of Implementation Instructions (II)?

A: II is a document that provides the instructions for shipping agents to implement their systems to use the XMLDG service provided by HKSARG through Marine Department. Interpretation and restrictions of the schema, networking configuration and ebXML messaging are covered in the document.

Q: What is the purpose of Technical Implementation Reference (TIR)?

A: TIR is a document that provides technical references on the implementation of the XMLDG project. It includes the deployment architecture, processing workflow and usage of XMLDG Vocabulary Library.

Q: What is XMLDG Vocabulary Library?

A: The XMLDG Vocabulary Library provides an XML binding framework for the XMLDG project. It maps each element in the XML schema specified in SS and II to a Java class. Each Java class provides getter and setter methods to manipulate, validate and parse the XML documents.

6.3 MESSAGE EXCHANGE

Q: What is ebXML Messaging Service(ebMS)?

A: ebMS provides the mechanism for packaging and transporting business documents in a standardized, secure and reliable manner. ebMS defines the header and the envelope for packaging one or multiple business documents, usually in XML format, into an ebXML message. The header and the envelope define the behaviour, such as reliable and secure messaging features, for the ebXML

Message Service Handler (MSH) software used by two parties to send and receive the message. The Version 2.0 of the ebMS is adopted in the XMLDG project.

For technical details about ebXML message services, please refer to the OASIS ebXML Message Services Specification Version 2 (http://www.ebxml.org/specs/ebMS2.pdf).

Q: What is Message Service Handler (MSH)?

A: MSH is a piece of software used to send and receive ebXML messages. In the XMLDG project, the MSH should be compliance with the OASIS ebMS V2 standard. Shipping Agents can choose any MSH software product that is complicance with the OASIS ebMS V2 standard. An interoperability test on the MSH product between Shipping Agents and MD before production run. Hermes MSH is one of such products that passed the interoperability test with MD.

Q: What is Hermes Message Service Handler (MSH)?

A: Hermes is a MSH implementation. It is in compliance with the OASIS ebMS V2 standard. It is implemented by the Center for E-Commerce Infrastructure Development at the University of Hong Kong. It is released as an open-source project under the Academic Free License. For more details of Hermes MSH, please refer to http://www.freebxml.org/msh.htm.

Q: What are software and hardware requirements of Hermes MSH?

A: Hermes MSH requires Java Runtime Environment (JRE) 1.4 or above. The MSH needs a database to store the status of communication, to support resilience, reliable messaging, message tracking, etc. Hermes MSH runs on a servlet container.

Q: Which platforms are tested using Hermes MSH?

A: OS: Linux, Solaris, Windows 2000, and Windows XP
Database: Oracle, MS SQL Server, PostgreSQL, JDataStore, and MySQL
Application Server: Jakarta Tomcat, WebSphere, WebLogic, JBoss, and JRun

Q: What is the cost of Hermes MSH?

A: Hermes is an open-source project released under the Academic Free License. It can be downloaded at http://www.freebxml.org/msh.htm.

Q: Does Hermes MSH provide any administration or monitoring tool?

A: Supporting tools are provided for administrating and monitoring Hermes MSH. Hermes MSH Heartbeat Tool checks if Hermes MSH is alive. Hermes MSH System Commander sends administrative commands to MSH, such as suspension, backup, archiving and checking of database connection. Hermes MSH Webmin is a web interface for Hermes MSH administration. Some of the modules are the same as those provided by System Commander, like MSH maintenance, MSH information and message archive. Those tools can be found under XMLDG project website.

Q: How to drive Hermes MSH to send and receive messages?

A: A Java data handler should be implemented to send and receive messages to and from Hermes MSH through Java API calls.

Q: What is version of Hermes MSH currently using in XMLDG system?

A: Up to the time of compiling this document, XMLDG is using Hermes MSH version 0.9.3.1.

6.4 NETWORKING

Q: What is the network-setting requirement between Marine Department and Shipping Agent?

A: The messages are exchanged through the Internet using HTTP over IPSec protocol. Shipping agents using the XMLDG service should set up an IPSec VPN server to establish a gateway-to-gateway tunnel with the IPSec server in Marine Department.

6.5 DEVELOPMENT

Q: How to integrate Hermes MSH with my backend system?

A: It is assumed that there is a working database that contains all "ready to send" records. A data handler should be implemented to interact with the working database and Hermes MSH. It has no direct interaction with the backend system.

The recommended deployment mode is using one data handler, i.e. one process, to handle all sending and receiving requests. The data handler will run as a daemon process and it will continue polling the working database to check whether there are "ready to send" records. Hermes MSH invokes the data handler when a message is received. The data handler then performs some business logics.

Q: What resources are required on the implementation?

A: In general, there are two program modules that need to be implemented.

One is data extraction module that will extract DG data from backend system to a working database based on some business logics. These logics can be varied from different shipping agents.

The second one is a data handler that responsible to send and receive ebXML message, and process XML document data content. Currently, a data handler is acting as a Hermes MSH Client, and it is only avaible in Java platform. There are some sample codes in the project website.

Q: Is it possible to test the XMLDG applications with Marine Department before production run?

A: Yes. There is a testing environment established in Marine Department. Interested parties can contact the Marine Department for detailed arrangements.

Q: Apart from technical aspects, what are the other critical success factors to watch out during implementation?

A: Shipping companies shall take the opportunities to re-design the related business processes in order to derive the full benefits and efficiency of the new system-to-system service delivery mode. Change management activities coupled with comprehensive training will certainly increase user buy-in.

6.6 SUPPORT

Q: What is the support arrangement?

A: The Marine Department will engage the necessary ongoing technical support services for the Hermes MSH software suite to ensure effective operations of the production service provision. However, any application development or maintenance efforts for interfacing with and integrating backend systems will be borne by the participating companies concerned.

APPENDIX A: SHIPPING AGENT DATA HANDLER SAMPLE CODE

```
import hk.hku.cecid.phoenix.vocab.*;
import hk.hku.cecid.phoenix.vocab.util.*;
import hk.hku.cecid.phoenix.vocab.cct.*;

import hk.hku.cecid.phoenix.dg.DangerousGoodsManifest;
import hk.hku.cecid.phoenix.dg.DangerousGoodsItem;
import hk.hku.cecid.phoenix.dg.DocumentHeader;
import hk.hku.cecid.phoenix.dg.Transport;
import hk.hku.cecid.phoenix.db.Acknowledgement;

import hk.hku.cecid.phoenix.message.handler.*;
import hk.hku.cecid.phoenix.message.packaging.*;

.
.
.

public class DGDataHandler implement MessageListener {

    public DGDataHandler() {}

    /**
      * Handles the received messages.
      * This is for implementing the MessageListener interface.
      */
    public void onMessage(EbxmlMessage ebxmlMessage) {
        //
        // check the type of received messages and
        // process the message depends of the type.
        //
        .
        .
        .
    }

    /**
     * Function create a MSH Request object
     */
    public Request createMSHRequest() {
        Request mshReq = null;
        try {
            //
            // create a MSH ApplicationContext object
            //
            ApplicationContext ac = new ApplicationContext(cpaID,
                    conversationID, service, action);

            //
            // create the Request object
            //
            mshReq = new Request(ac, new URL(toMSHUrl), this,
                    transportType);
        }
        catch (RequestException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
```

```
        }

        return mshReq;
    }

    /**
     * Function a Dangerous Goods Manifest document
     */
    public DangerousGoodsManifest newDGManifestDocument() {
        try {
            //
            // create the XMLDG document
            //
            DangerousGoodsManifest dgm =
              DangerousGoodsManifest.newInstance();

            //
            // add the document header
            //
            DocumentHeader header = dgm.getDocumentHeader();

            DocumentId documentId = header.getDocumentId();
            documentId.setContent("1234@shippingagent.com");
            .
            .
            .


            //
            // add transport infomation
            //

            //Transport is an optional element in SS
            // As the appendOptionalChildren=false
            // in vocab.properties, we need to create
            // the optional element first, and append it
            // to its parent.
            //
            Transport transport = dgm.createTransport();
            dgm.setTransport(transport);

            .
            .
            .
            //
            //sample optional elements: Vessel, Name
            //
            Vessel vessel = transport.createVessel();
            transport.setVessel(vessel);

            Name name = vessel.createName();
            name.setContent("UNI-ACCORD");
            vessel.setName(name);
            .
            .
            .


            //
            // add dangerous goods item (another optional element)
            //
```

```
        //
        //DangerousGoodsItem (optional)
        //
        DangerousGoodsItem item1 = dgm.createDangerousGoodsItem();
        dgm.addDangerousGoodsItem(item1);

        //
        //DangerousGoods (mandatory)
        //
        DangerousGoods dg1 = item1.getDangerousGoods();

        //
        //DangerousGoods' children (all are optional)
        //
        UNDGId id = dg1.createUNDGId();
        id.setContent("1549");
        dg1.setUNDGId (id);
        .
        .
        .

        //
        //Other children of DangerousGoodsItem
        // (all others are optional)
        //
        GrossMassMeasure grossMassMeasure =
        item1.createGrossMassMeasure()
        ;
        grossMassMeasure.setContent("460");
        item1.setGrossMassMeasure(grossMassMeasure);
        .
        .
        .
    }
    catch (ConstraintViolatedException e) {
        //
        // Constraints in II violated
        //
        e.printStackTrace();
    }
    catch (SchemaNotFoundException e) {
        //
        // Schema files cannot be found
        //
        e.printStackTrace();
    }
    catch (ConstraintsNotFoundException e) {
        //
        // Constraints file cannot be found
        //
        e.printStackTrace();
    }
    catch (VocabException e) {
        //
        // Constraints in schema violated
        //
        e.printStackTrace();
    }

    return dgm;
```

```
    }

    /**
      * Function package a Dangerous Goods Manifest document to
      * an ebXML Message
      */
    public EbxmlMessage packageData(DangerousGoodsManifest
            manifest) {
        EbxmlMessage ebxmlMesg = null;
        try {
            ebxmlMesg = new EbxmlMessage();
            //
            // call MSH API to generate a message ID
            //
            String messageID = Utility.generateMessageId
                (new Date(), toPartyId, cpaId, service, action);
            //
            // call MSH API to generate a time stamp
            //
            String timestamp = Utility.toUTCString(new Date());
            //
            // add message header to EbxmlMessage
            //
            ebxmlMesg.addMessageHeader(fromPartyID, toPartID, cpaID,
                    converstationID, service, action, messageID,
                    timestamp);
            //
            // marshal the document into a file
            //
            File payloadFile = new File(fileName);
            manifest.marshal(new FileWriter(payloadFile));
            //
            // convert the payloadFile to AttachementDataSource
            // and add to the EbxmlMessage
            //
            AttachmentDataSource attachment =
                    new AttachementDataSource(payloadFile,
                    "text/xml");
            DataHandler dataHandler = new DataHandler(attachment);
            ebxmlMesg.addPayloadContainer(dataHandler, payloadID,
        null);
            //
            // save the message
            //
            ebxmlMesg.saveChanges();
        }
        catch (SOAPException e) {
            e.printStackTrace();
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        catch (MarshalException e) {
            e.printStackTrace();
        }

        return ebxmlMesg;
    }
```

```
/**
 * Function extract the payload from EbxmlMessage and return an
 * Acknowledgement object for further handling
 */
public Acknowledgement extractData(EbxmlMessage msg) {
    Acknowledgement acknowledgement = null;
    try {
        Iterator iter = ebxmlMessage.getPayloadContainers();
        while (iter.hasNext()) {
            PayloadContainer payload =
                (PayloadContainer) iter.next();
            AttachmentPart part = payload.getAttachmentPart();
            DataHandler dataHandler = part.getDataHandler();
            if (dataHandler.getContentType()
            .startsWith("text/xml")) {
                StreamSource source =
                        (StreamSource) dataHandler.getContent();
                //
                // create temp file for transform
                //
                File payloadFile = new File(fileName);
                StreamResult result = new StreamResult
                        (new FileWriter(payloadFile));
                Transformer transformer =
                TransformerFactory.newInstance().
                newTransformer()
                ;
                transformer.transform(source, result);

                //
                // create an Acknowledgement from a file
                //
                acknowledgement = Acknowledgement.unmarshal
                        (new FileReader(payloadFile));
                processAcknowledgement(acknowledgement);
            }
        }
    }
    catch (IOException e) {}
    catch (SOAPException e) {
        e.printStackTrace();
    }
    catch (TransformerConfigurationException e) {
        e.printStackTrace();
    }
    catch (TransformerException e) {
        e.printStackTrace();
    }
    catch (UnmarshalException e) {
        e.printStackTrace();
    }
    catch (IOException e) {
        e.printStackTrace();
    }

    return acknowledgement;
}

/**
 * Function process an Acknowledgement
```

```
 */
public void processAcknowledgement(Acknowledgement
    acknowledgement) {
    try {
        //
        // get the document header
        //
        DocumentHeader header = ack.getDocumentHeader();

        DocumentId documentId = header.getDocumentId();
        String documentIdContent = documentId.getContent();
        .
        .
        .

        //
        // get the processing results (optional element).
        // resultCount is the count of processing results.
        //
        List resultList = ack.getProcessingResult();
        int resultCount = resultList.size();
        for (int i=0; i<resultCount; i++) {
            //
            //Code is an optional element
            //invoke getContent() only if Code is not null
            //
            Code code = result.getCode();
            if (code != null)
                String codeContent = code.getContent();

            //
            //Text is an optional element
            //
            Text text = result.getText();
            if (text!=null)
                String testContent = test.getContent();

            //
            // process the results
            //
            .
            .
            .
        }
    }
    catch (ConstraintViolatedException e) {
        //
        // Constraints in II violated
        //
        e.printStackTrace();
    }
    catch (SchemaNotFoundException e) {
        //
        // Schema files cannot be found
        //
        e.printStackTrace();
    }
    catch (ConstraintsNotFoundException e) {
        //
        // Constraints file cannot be found
        //
```

```
            e.printStackTrace();
        }
        catch (VocabException e) {
            //
            // Constraints in schema violated
            //
            e.printStackTrace();
        }
    }
}
```

APPENDIX B: MARINE DEPARTMENT DATA HANDLER SAMPLE CODE

```
import hk.hku.cecid.phoenix.vocab.*;
import hk.hku.cecid.phoenix.vocab.util.*;
import hk.hku.cecid.phoenix.vocab.cct.*;

import hk.hku.cecid.phoenix.dg.DangerousGoodsManifest;
import hk.hku.cecid.phoenix.dg.Credential;
import hk.hku.cecid.phoenix.dg.DocumentHeader;
import hk.hku.cecid.phoenix.dg.Acknowledgement;
import hk.hku.cecid.phoenix.dg.ProcessingResult;

import hk.hku.cecid.phoenix.message.handler.*;
import hk.hku.cecid.phoenix.message.packaging.*;

.
.
.

public class DGDataHandler implement MessageListener {

    public DGDataHandler() {}

    /**
     * Handles the received messages.
     * This is for implementing the MessageListener interface.
     */
    public void onMessage(EbxmlMessage ebxmlMessage) {
        //
        // check the type of received messages and process
        // the message depends of the type.
        //
        .
        .
        .
    }

    /**
     * Function create a MSH Request object
     */
    public Request createMSHRequest() {
        Request mshReq = null;
        try {
            //
            // create a MSH ApplicationContext object
            //
            ApplicationContext ac = new ApplicationContext(cpaID,
                    conversationID, service, action);

            //
            // create the Request object
            //
            mshReq = new Request(ac, new URL(toMSHUrl), this,
                    transportType);
        }
        catch (RequestException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
```

```
            e.printStackTrace();
        }

        return mshReq;
    }

    /**
     * Function create an Acknowledgement document
     */
    public Acknowledgement newAcknowledgement() {
        try {
            //
            // create the Acknowledgement document
            //
            Acknowledgement ack = Acknowledgement.newInstance();

            //
            // add the document header
            //
            DocumentHeader header = ack.getDocumentHeader();

            DocumentId documentId = header.getDocumentId();
            documentId.setContent("1234@shippingagent.com");
            .
            .
            .

            //
            // add the processing results (optional element)
            // note : resultCount is the count of processing results
            //
            for (int i=0; i<resultCount; i++) {
                //
                // ProcessingResult is an optional element
                //  As the appendOptionalChildren=false
                //  in vocab.properties, we need to create
                //  the optional element first, and append it
                //  to its parent.
                //
                ProcessingResult result =
                ack.createProcessingResult();

                // Similar to ProcessingResult,
                // Code is an optional element that
                // requires special handling
                // (i.e. createCode(), setCode())
                //
                Code code = result.createCode();
                code.setContent("201");
                result.setCode();

                //
                // Similar to ProcessingResult, and Code
                // Text is an optional element
                //
                Text text = result.createText();
                test.setContent("Declaring party unknown or invalid");
                result.setText(text);

                ack.addProcessingResult(result);
            }
```

```
        }
        catch (ConstraintViolatedException e) {
            //
            // Constraints in II violated
            //
            e.printStackTrace();
        }
        catch (SchemaNotFoundException e) {
            //
            // Schema files cannot be found
            //
            e.printStackTrace();
        }
        catch (ConstraintsNotFoundException e) {
            //
            // Constraints file cannot be found
            //
            e.printStackTrace();
        }
        catch (VocabException e) {
            //
            // Constraints in schema violated
            //
            e.printStackTrace();
        }

        return ack;
    }

    /**
     * Function package an Acknowledgement document to
     * an ebXML Message
     */
    public EbxmlMessage packageData(Acknowledgement ack) {
        EbxmlMessage ebxmlMesg = null;
        try {
            ebxmlMesg = new EbxmlMessage();
            //
            // call MSH API to generate a message ID
            //
            String messageID = Utility.generateMessageId
                (new Date(), toPartyId, cpaId, service, action);
            //
            // call MSH API to generate a time stamp
            //
            String timestamp = Utility.toUTCString(new Date());
            //
            // add message header to EbxmlMessage
            //
            ebxmlMesg.addMessageHeader(fromPartyID, toPartID, cpaID,
                    converstationID, service, action, messageID,
                    timestamp);
            //
            // marshal the document into a file
            //
            File payloadFile = new File(fileName);
            ack.marshal(new FileWriter(payloadFile));
            //
            // convert the payloadFile to AttachementDataSource
            // and add to the EbxmlMessage
            //
```

```
            AttachmentDataSource attachment =
                    new AttachementDataSource(payloadFile,
                    "text/xml");
            DataHandler dataHandler = new DataHandler(attachment);
            ebxmlMesg.addPayloadContainer(dataHandler, payloadID,
                    null);
            //
            // save the message
            //
            ebxmlMesg.saveChanges();
        }
        catch (SOAPException e) {
            e.printStackTrace();
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        catch (MarshalException e) {
            e.printStackTrace();
        }

        return ebxmlMesg;
    }

    /**
     * Function extract the payload from EbxmlMessage
     * Assumption : the first payload is a Credential
     *              document and the others are
     *              DangerousGoodsManifest document
     * This function returns a list containing
     *     the credential (as the first element)
     *     and the DangerousGoodsManifest document(s)
     *     for further handling
     */
    public List extractData(EbxmlMessage msg) {
        ArrayList list = new ArrayList();
        Credential credential = null;
        DangerousGoodsManifest dgm = null;
        try {
            Iterator iter = ebxmlMessage.getPayloadContainers();
            int payloadCount = 0;
            while (iter.hasNext()) {
                PayloadContainer payload = (PayloadContainer)
                        iter.next();
                AttachmentPart part = payload.getAttachmentPart();
                DataHandler dataHandler = part.getDataHandler();
                if (dataHandler.getContentType().startsWith
                        ("text/xml")) {
                    StreamSource source =
                            (StreamSource) dataHandler.getContent();
                    //
                    // create temp file for transform
                    //
                    File payloadFile = new File(fileName);
                    StreamResult result = new StreamResult
                            (new FileWriter(payloadFile));
                    Transformer transformer = TransformerFactory
                            .newInstance().newTransformer();
```

```
                    transformer.transform(source, result);

                    if (payloadCount == 0) {
                        //
                        // create a Credential from a file
                        //
                        credential = Credential.unmarshal
                                (new FileReader(payloadFile));
                        list.add(credential);
                    }
                    else {
                        //
                        // create a DangerousGoodsManifest from a
                        // file
                        //
                        dgm = DangerousGoodsManifest.unmarshal
                                (new FileReader(payloadFile));
                        list.add(dgm);
                    }
                    payloadCount++;
                }
            }
            //
            //process the manifest ( processManifest(dgm) )
            //
            .
            .
            .
        }
        catch (IOException e) {}
        catch (SOAPException e) {
            e.printStackTrace();
        }
        catch (TransformerConfigurationException e) {
            e.printStackTrace();
        }
        catch (TransformerException e) {
            e.printStackTrace();
        }
        catch (UnmarshalException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }

        return list;
    }

    /**
     * Function to process the content of a DangerousGoodsManifest
     */
    public void processManifest(DangerousGoodsManifest manifest) {
        try{
            //
            // process the header data
            //
            DocumentHeader header = manifest.getDocumentHeader();
            String documentID = header.getDocumentId.getContent();
            .
            .
```

```
.
//
// process transport infomation
//
Transport transport = dgm.getTransport();

//
//Tranport is an optional element
//go to its children only if it is not null
//
if (transport!=null){
    .
    .
    .
    Vessel vessel = transport.getVessel();
    //
    //Vessel is an optional element
    //
    if (vessel!=null){
        Name name = vessel.getName();
        //
        //Name is an optional element
        //
        if (name!=null)
        String nameContent = name.getContent();
    }
    .
    .
    .

}


//
// process the dangerous goods item
//
List goodsItem = manifest.getDangerousGoodsItem();
for (int i=0; i<goodsItem.size(); i++) {
    //
    //DangerousGoodsItem
    //
    DangerousGoodsItem item = (DangerousGoodsItem)
        goodsItem.get(i);

    //
    //DangerousGoods (mandatory)
    //
    DangerousGoods dg = item.getDangerousGoods();

    //
    //DangerousGoods' children (all are optional)
    //
    UNDG undgId= dg.getUNDGId();
    if (undgId!=null)
        String undgID = undgId.getContent();
    .
    .
    .
    //
    //Other children of DangerousGoodsItem
    // (all others are optional)
```

```
                //
                GrossMassMeasure gmm = item.getGrossMassMeasure();
                if (gmm!=null)
                    String gmmContent = gmm.getContent();
                .
                .
                .
                //
                // process the manifest submission
                //
                .
                .
                .
            }
        }
        catch (ConstraintViolatedException e) {
            //
            // Constraints in II violated
            //
            e.printStackTrace();
        }
        catch (SchemaNotFoundException e) {
            //
            // Schema files cannot be found
            //
            e.printStackTrace();
        }
        catch (ConstraintsNotFoundException e) {
            //
            // Constraints file cannot be found
            //
            e.printStackTrace();
        }
        catch (VocabException e) {
            //
            // Constraints in schema violated
            //
            e.printStackTrace();
        }
    }
}
```