

---

## Technical Note: Specification of message exchange with ioNode

**Author:** James Bennett ,Wynne Rees, Selwyn Lloyd  
**To:**  
**Date:** 25<sup>th</sup> June 2003  
**Revised:** 30th June 2003, 02 July 2003, 9<sup>th</sup> July  
**Filename:** IoNodeCommunicationSpecification1.1.doc  
**Title:** Specification of message exchange with ioNode  
**Version:** 1.3 Draft, Confidential

---

## Table of Contents

OVERVIEW .....	2
AUDIENCE.....	3
INTRODUCTION .....	3
THIRD PARTY COMMUNICATION WITH AN IOAGENT .....	6
(1) SCHEMA FOR 3RD PARTY DB CONNECTION .....	6
<b>1.1 Post Queue:</b> .....	6
<b>1.2 Send Response queue</b> .....	7
2. CSV .....	7
3. HTTP POST .....	8
3.1 HTTP headers.....	8
COMMUNICATING WITH AN IONODE HUB.....	9
1.1 ROUTING .....	9
1.2 IDENTIFICATION .....	9
1.3 SERVICE AND ACTION .....	10
1.4 DATA PAYLOADS .....	11
GENERAL OPERATION NOTES.....	11
SOAP IN .....	11
SOAP OUT.....	11
IOHUB .....	11
SENDING MESSAGE BEHAVIOR.....	12
RECEIVING MESSAGE BEHAVIOR.....	12
CONFIGURATION .....	12
GLOSSARY OF TERMS .....	12
REFERENCES .....	13
APPENDIX 1 : CSV METADATA FOR 3RD PARTY COMMUNICATION.....	13
APPENDIX 2 : EBXML.....	13
NAMESPACES .....	13



MESSAGEHEADER.....	14
<i>Example ebXML Message Header</i> .....	14
FIELD DEFINITION.....	14
<b><i>From and To</i></b> .....	14
<b><i>PartyId</i></b> .....	15
<b><i>Role Element</i></b> .....	15
<b><i>ConversationId Element</i></b> .....	15
<b><i>Service Element</i></b> .....	15
<b><i>Action Element</i></b> .....	15
<b><i>MessageData Element</i></b> .....	16
<b><i>DuplicateElimination Element</i></b> .....	17
<b><i>Description Element</i></b> .....	17
EXAMPLE EBXML MESSAGE MANIFEST.....	17
RELIABLE MESSAGING.....	17
<i>Methods of Implementing Reliable Messaging</i> .....	18
MULTI-HOP RELIABLE MESSAGING.....	18
<b><i>AckRequested Sample</i></b> .....	18
<i>Acknowledgment Sample</i> .....	19
<i>Multi-Hop Acknowledgments</i> .....	19
ERRORLIST SAMPLE.....	19
<b><i>Errors allowed</i></b> .....	21
<b>APPENDIX 3 : IMS-LIP</b> .....	<b>22</b>
EXAMPLE OF A LEARNER INFORMATION PACKAGE.....	22

## Overview

This document provides a working specification for the communication interfaces to ioNodes.

ioNode is a middleware architecture provided using a suite of primarily open source server software. An ioNode is a node within a network of other messaging and data transformation nodes. The nodes may be hubs, agents or a group of agents sharing a common platform. ioNodes are currently in development for use within the JISC funded SHELL project to provide an agent and hub layer for data transport and data transformation.

The SHELL project has a number of requirements and aims derived from its partners and its funding body JISC. The key requirement is providing interoperability between disparate Student Record (SR) Systems at the partner colleges. A design was agreed between CETIS, SHELL, Phosphorix and JISC to build an infrastructure for messaging via a central transaction hub or ‘message centre’ (at the University of Plymouth), to employ agents to transform data to XML and to use SOAP to provide transportation and packaging logic.

Vendors were initially asked to provide agents for their systems, Phosphorix began the ioAgent project to provide a common ioAgent available to plug-in to vendor and or proprietary SR systems as required. During design and research it was soon realised that the ioAgent stack could provide for a Hub configuration as well as an agent. Hence the realisation that ioHub and ioAgent were in fact both ioNodes with different configurations.



An ioNode provides messaging [data transaction] services to the collaborating education systems. Initially two versions of ioNode are being developed, they are:

- ioHub: Provides the ‘middleware’ which will be the SHELL projects hub, it is responsible for queuing and forwarding transactional data, such as new records.
- ioAgent: An agent’s main function within the SHELL project is to transform Data from one system and post it to a HUB using a common data specification. The chosen data specification for the SHELL project is IMS-LIP. IMS LIP is an XML schema for representing student record data or ‘Learner Profile’ information.

This document will provide technical information for methods to interface directly with an ioNode. Those intending to send data to the SHELL hub installation will need to provide routing and rules based data as ebXML wrapped in SOAP with an IMS-LIP payload for learner data. Those partners intending to provide data to an agent interface need only provide data and routing information.

It is important to note that XML data that fails validation will not be routed or further processed. Specifically, XML that fails to conform to ebXML Messaging Service 2.0 and SOAP 1.1 will not be routed.

This document provides enough information to developers of third party agents ahead of the hub interface being live and physically available. Until which time it is not possible to provide for URL and IP specific header data.

## Audience

This document is aimed primarily at developers and technical staff of the SHELL partners. The information may also be disclosed to third party vendors wishing to send data directly to an ioNODE.

## Introduction

A key aim of the SHELL project is to use the IMS XML data specifications to exchange data between education establishments. In very brief terms, the ioNode provides for the transport of SOAP messages over https to allow for the exchange of IMS XML data. The implementation gives a reliable messaging protocol based upon the ebXML Message Service Specification.

The intention is that the ioNode architecture is as generic as possible and it is built as a web-service for this reason. The benefit of the use of the ioNode component is that it takes responsibility for implementation of a reliable messaging system and data transformation away from the partner systems and allows them to think in terms of very simple interfaces for the getting and sending of data.

The diagrams below shows the basic relationship between ioNodes and the collaborating systems.

Fig. 1: ioNodes and partner systems



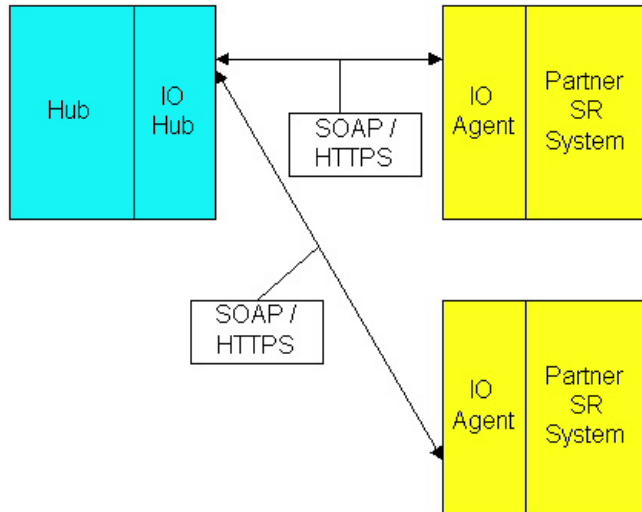
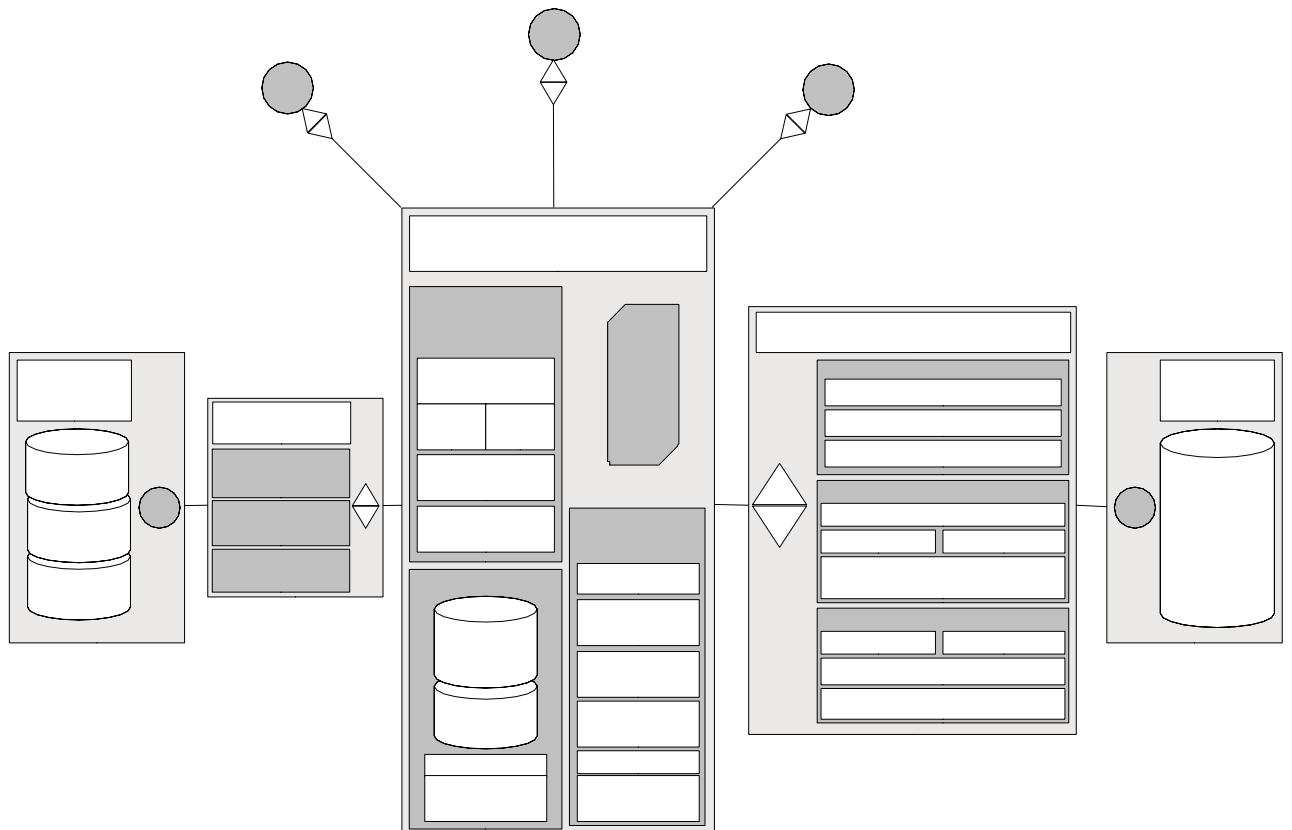


Fig. 2: A diagram showing the modules that collaborate to give the SHELL project hub and ioNode architectures. Please note that here the term Interoperability Agent refers to ioNode components.



The internal components of the ioNodes (Interoperability Agent) and of the hub framework are implementation details of the SHELL project. These allow for reliable two-way exchange of SOAP messages (SOAP Messages IN, SOAP Messages OUT) configuration and the support of a publish subscribe mechanism for the hub. The Transformation Layer handles the mappings required to exchange data between IMS



formats and the 'local' partner formats. In addition there are a group of internet related security services (SSL, authentication) and routing services (Agent routing table, Routing rules).

The subject of this document is the specification of the following two interfaces:

1. Interface between the Interoperability Agent and the application, as shown above (ie. the application layer). This interface is described in terms of a set of simple data exchange mechanisms.
2. Interface between the Interoperability Agent and the hub. This interface is described in terms of the SOAPwa / ebXML MS messaging protocol.



## Third Party Communication with an ioAgent

There will be three methods for sending data to an ioAgent from a third party application.

1. Direct exchange with an ioNode database
2. CSV Drop point
3. HTTP POST

These three interfaces are described in the following sections.

### **(1) Schema for 3rd party DB connection**

Schema for 3rd party DB connection table within PostgreSQL.

The process by which the post queue is used is shown in a sequence diagram at the end of this section. Briefly an asynchronous message protocol is supported where a message is posted onto a 'Post Queue', this is read and validated by ioNode and a response is then placed on the 'Send Response Queue'.

#### **1.1 Post Queue:**

Role: To accept data for send to partner systems

Field Name	key	Type	Notes
index	primary key	int	auto generated by sequence or increment
timestamp	/	timestamp	milliseconds set by script
target	/	varchar(80)	Name agreed for SHELL profile
action	/	varchar(20)	Action name agreed within SHELL project
data	/	typea	RAM limitations associated with large data sets. Metadata is as for CSV file format as described in Appendix 1

Notes:

(a) Validation of the message will be done when it is read by the ioNode implementation.

(b) The data metadata is defined in Appendix 1. This metadata includes the authorisation fields required for the transaction.



## 1.2 Send Response queue

Role: Repository for response messages associated with the send messages in 1.1 above.

This supports a simple asynchronous response mechanism that allows the ioNode to validate and report back the condition of the messages posted.

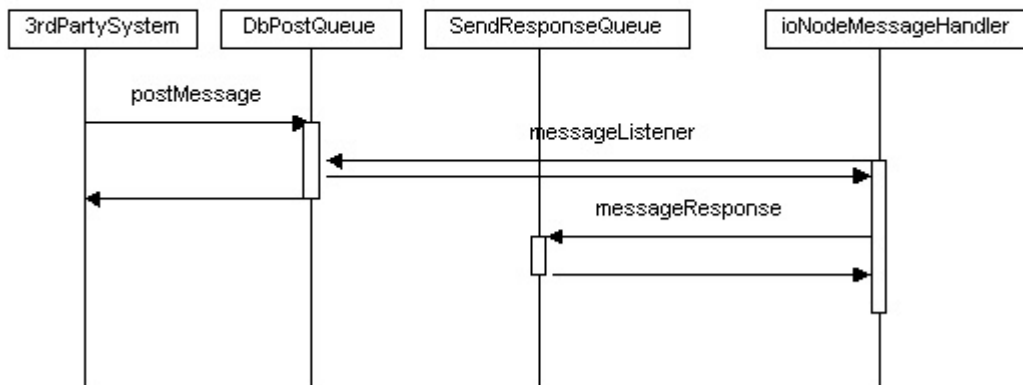
Field Name	key	Type	Notes
index	primary key	int	auto generated by sequence or increment
sendKey		int	reference to pk of send queue
timestamp	/	timestamp	milliseconds set by script
target	/	varchar(80)	Name agreed for SHELL profile
action	/	varchar(20)	Action name agreed within SHELL project

Notes:

(a) Validation of the message will be done when it is read by the ioNode implementation.

Fig. 1: Process for post of messages from 3rd party systems to ioNode

Sequence diagram showing process of posting a message onto the ioNode DB queue



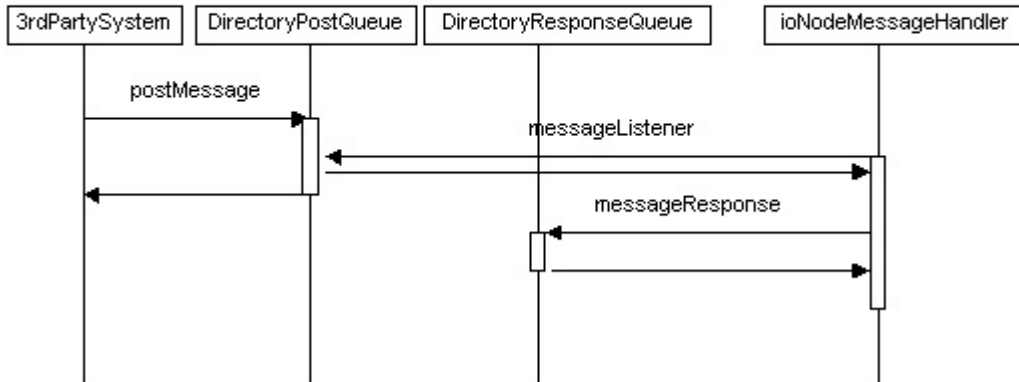
## 2. CSV

A CSV [Comma Separated Values] file containing the relevant record and destination information can be dropped into a directory where it will be picked up and processed into the ioAgent queue. ioNode implements a listener onto this directory. A response directory is also maintained where ioNode reports on the status of this message post. Please refer to Appendix 1 for the CSV file metadata.

A sequence diagram for this process is shown below:



Sequence diagram showing process of posting a message onto the ioNode drop directories



### 3. HTTP POST

A HTTP POST of the relevant information to a HTTP receiver which will then build a queue item.

The POST method supported on ioNode version 1.0 allows for the sending of a data payload of the SHELL CSV file format. MIME multipart messages are not supported in v1.0.

The payload of the message will be of the CSV format defined in Appendix 1.

#### 3.1 HTTP headers

The following headers are required for the Post of messages to the ioNode; all of these header fields are mandatory within the ioNode project.

```

POST /axis/services/PartnerMessageService HTTP/1.0
Accept-Language: en
User-Agent: Tomcat/4.1.8
Date: Tue, 15 Nov 2002 08:12:31 GMT
Content-Length: <calculate content length in url encoded bytes and include here>
Host: <host name>
Content-Type: application/shellcsv; charset=utf-8

<field1>=foo&<field2>=bar&<field3>=foobar
    
```



## Communicating with an ioNode Hub

ioNODE makes use of SOAP over https to give a messaging service based on web-service technology. The message service is built using the ebXML Message Service version 2 specification.

The key components of the message exchanges are:

- Message routing
- Identification for authorisation.
- Service/Action to be requested.
- Data to be processed in as a SOAP Attachment

ioNode supports a point-point and publish-subscribe messaging models. In addition a registration process is used to associate the hub with partner systems.

The ebXML MessageHeader element introduces the control fields for the messaging models to be supported. This element contains elements that define routing information, message identification, action identification etc.

It is a requirement of the SHELL project that the data payload should use the IMS LIP specification. This is transported as a SOAP Attachment.

### 1.1 Routing

An example of routing definition as specified in the ebXML MessageHeader SOAP extension element is shown below:

```
<eb:From>
  <eb:PartyId eb:type="uri">plymouth.ac.uk</eb:PartyId>
  <eb:Role>http://plymouth.ac.uk/roles/Originator</eb:Role>
</eb:From>
<eb:To>
  <eb:PartyId eb:type="uri">cornwall.ac.uk</eb:PartyId>
  <eb:Role>http://cornwall.ac.uk/roles/Destination</eb:Role>
</eb:To>
```

### 1.2 Identification

Authorisation and identification of the message includes from and to parties and their respective roles. The identification of the message is contained in the MessageData sub-element of the MessageHeader.



## 1.3 Service and Action

An Action is a sub part of a Service within ebXML.

### 1.3.1 Data services

There are two data related actions for the SHELL Service.

- 'add' Record
- 'update' Record

These two actions cover

- Add new student.
- Add new co-registered student
- Add qualification to student record (update)
- Delete qualification from student record (update)
- Add course module to student record(update)
- Delete course module from student record(update)
- Add qualification result to student record(update)
- Withdraw student from qualification(update)
- Withdraw student from module. (update)

The service name for communication with an ioNode Hub for data exchange is specified as:

"url\_hub /services/data"

An action is specified using one of the following :

- add
- update

The following snippet gives an example of this:

```
<eb:Service>url_hub /services/data</eb:Service>  
<eb:Action>add</eb:Action>
```

### 1.3.2 Subscription services

The subscription service is identified by:

"url\_hub /services/subscription"

The registration process consists of the following actions.

- register
- unregister



## **1.4 Data payloads**

The SOAP payload can technically contain anything but for the purposes of SHELL phase one (ioNode version 1.0) the payload will contain a learner record in IMS-LIP XML format.

The LIP record should be held as a MIME encoded payload associated with and referenced within the SOAP package as an attachment.

## **General Operation Notes**

### ***SOAP In***

A SOAP message would be received through a HTTP port on a web server at a pre-specified URL.

The ioAgent or ioHub would validate the format of the message and then place the received SOAP package into a messaging queue where it will be held for processing.

Once the message is stored the ioAgent or ioHub the receiver will respond with an acknowledgment message sent back to the ioAgent or ioHub that sent the original message. The application (ioHub or ioAgent) sending the original message would be expected to use this reply as confirmation that the message has been accepted by the ioAgent or ioHub.

If the application does not receive an acknowledgment then it will resend the message to the original destination again.

### ***SOAP Out***

The ioAgent or ioHub will poll its outgoing message queue and then create a SOAP envelope with payload from the stored message.

This message will be signed by the ioAgent or ioHub.

The ioAgent or ioHub will then transmit the message to the destination where it will be placed into a processing queue as above.

### ***ioHub***

The Shell HUB will validate and action SOAP envelopes and their IMS-LIP payload according to its authentication method and rules.



## **Sending Message Behavior**

If a MSH [Message Service Handler] is given data by an application needing to be sent reliably, the MSH MUST do the following:

1. Create a message from components received from the application.
2. Insert an AckRequested element
3. Save the message in persistent storage
4. Send the message to the Receiving MSH.
5. Wait for the return of an Acknowledgment Message acknowledging receipt of this specific message and, if it does not arrive before RetryInterval has elapsed, or if a communications protocol error is encountered, then take the appropriate action.

## **Receiving Message Behavior**

If a received message is an Acknowledgment Message:

1. Look for a message in persistent storage with a MessageId the same as the value of RefToMessageId on the received Message.
2. If a message is found in persistent storage then mark the persisted message as delivered.

If a received message is a Service Request Message:

1. Validate the message
2. Apply local rules
3. If there is an AckRequested then build an acknowledgment message and send.
4. Process message.

If the Receiving MSH is NOT the To Party MSH then see section Multi-Hop Reliable Messaging for the behaviour of the AckRequested element.

## **Configuration**

The HUB will be designed with intelligence to be stored in configuration data.

The Hub will examine payloads, where necessary, to apply business rules against them.

## **Glossary of Terms**

ioHub – The central messaging software hub through which messages are routed to arrive at their destination. The ioHub also applies limited business logic.

ioAgent – The endpoint nodes of the ioNode structure. ioAgents receive and send messages to and from the ioHub. ioAgents communicate with the institutions applications via a plug-in interface.

ioNode – Generic name for any component endpoint in the io architecture.

SOAP – Simple Object Access Protocol, A way of describing a standardised container of information in a human readable form (text) that can be transmitted over an independent transport mechanism.

ebXML – Electronic Business Extensible Markup Language

MSH – Message Service Handler, The endpoint that interprets messages.



UTC - Coordinated Universal Time

## References

- [1] <http://www.jisc.ac.uk/>
- [2] <http://www.imsproject.org>
- [3] ebXML Message Services Specification v2.0 [http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS\\_v2\\_0.pdf](http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf)
- [4] SHELL Project <http://www.educationaldevelopment.net/shellproject/default.htm>

## Appendix 1 : CSV Metadata for 3rd party communication

ioNode CSV Field List

The comma separated field list will follow the following format:

See spreadsheet ioNode\_CSV\_Field\_Definition.(x).xls

## Appendix 2 : ebXML

The message formatting is based upon the ebXML Message Service Specification version 2.0.

The following sections introduce the ebXML SOAP extension elements used in the ioNodes and clarify their usage in this project.

### Namespaces

The following namespace declarations are REQUIRED for the ioNode project. These are given as namespace/value pairs in the following list:

#### Envelope namespaces:

[Note use of SOAP as prefix in place of SOAP-ENV in ebXML MS v1.0]

name = "SOAP" within xmlns;

namespace = "http://schemas.xmlsoap.org/soap/envelope/";

name ="xsi" within xmlns;

namespace="http://www.w3.org/2000/10/XMLSchema-instance";

name="schemaLocation" within xsi ;

namespace="http://schemas.xmlsoap.org/soap/envelope/ http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd";

#### MessageHeader namespace

name = "eb" within xmlns;



namespace = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2\_0.xsd";

## **MessageHeader**

The MessageHeader element contains the principal control elements for implementing the ebXML messaging protocols.

It is noted that within ioNode it is REQUIRED that the SOAP 'mustUnderstand' MessageHeader attribute is true. This gives a behaviour that the contents of the element must be understood.

### **Example ebXML Message Header**

```
<eb:MessageHeader eb:id="..." eb:version="2.0" SOAP:mustUnderstand="1">
  <eb:From>
    <eb:PartyId eb:type="uri">plymouth.ac.uk</eb:PartyId>
    <eb:Role>http://plymouth.ac.uk/roles/Originator</eb:Role>
  </eb:From>
  <eb:To>
    <eb:PartyId eb:type="uri">cornwall.ac.uk</eb:PartyId>
    <eb:Role>http://cornwall.ac.uk/roles/Destination</eb:Role>
  </eb:To>
  <eb:CPAId>http://plymouth.ac.uk/cpa/123456</eb:CPAId>
  <eb:ConversationId>987654321</eb:ConversationId>
  <eb:Service eb:type="shellservices">NewStudent</eb:Service>
  <eb:Action>AddStudent</eb:Action>
  <eb:MessageData>
    <eb:MessageId>UUID-2</eb:MessageId>
    <eb:Timestamp>2003-07-25T12:19:05</eb:Timestamp>
    <eb:RefToMessageId>UUID-1</eb:RefToMessageId>
  </eb:MessageData>
  <eb:DuplicateElimination/>
</eb:MessageHeader>
```

## **Field Definition**

### **From and To**

The REQUIRED From element identifies the Party that originated the message.

The REQUIRED To element identifies the Party that is the intended recipient of the message. Both To and From contain logical identifiers, we have chosen to identify institutions by their internet domain.

The From and the To elements each contains:

- PartyId elements – occurs one or more times
- Role element – occurs zero or one times.



## PartyId

The PartyId element has a single attribute, type and the content is a string value. The type attribute indicates the domain of names to which the string in the content of the PartyId element belongs. The value of the type attribute MUST be mutually agreed and understood by each of the Parties. It is RECOMMENDED that the value of the type attribute be a URI.

If the PartyId type attribute is not present, the content of the PartyId element MUST be a URI [RFC2396], otherwise the Receiving Message Service Handler SHOULD report an error with errorCode set to Inconsistent and severity set to Error.

## Role Element

The Role element identifies the authorized role (fromAuthorizedRole or toAuthorizedRole) of the Party sending (when present as a child of the From element) and/or receiving (when present as a child of the To element) the message. The value of the Role element is a non-empty string.

Note: Role is better defined as a URI – e.g. <http://plymouth.ac.uk/roles/sender>.

## ConversationId Element

The REQUIRED ConversationId element is a string identifying the set of related messages that make up a conversation between two Parties. It MUST be unique within the context of the specified CPAId. The Party initiating a conversation determines the value of the ConversationId element that SHALL be reflected in all messages pertaining to that conversation.

The ConversationId enables the recipient of a message to identify the instance of an application or process that generated or handled earlier messages within a conversation. It remains constant for all messages within a conversation.

The value used for a ConversationId is implementation dependent.

More information is required to be given on how to form a conversationID for the SHELL implementation.

## Service Element

The REQUIRED Service element identifies the service that acts on the message and it is specified by the designer of the service.

Note: In the context of an ebXML business process model, an action equates to the lowest possible role based activity in the Business Process (requesting or responding role) and a service is a set of related actions for an authorized role within a party.

## Action Element

The REQUIRED Action element identifies a process within a Service that processes the Message. Action SHALL be unique within the Service in which it is defined. The value of the Action element is specified by the designer of the service.



If the value of either the Service or Action element are unrecognized by the Receiving Message Service Handler, then it MUST report the error with an errorCode of NotRecognized and a severity of Error.

## **MessageData Element**

The REQUIRED MessageData element provides a means of uniquely identifying an ebXML Message.

It contains the following:

- MessageId element
- Timestamp element
- RefToMessageId element
- TimeToLive element

The following fragment demonstrates the structure of the MessageData element:

```
<eb:MessageData>  
<eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>  
<eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>  
<eb:RefToMessageId>20001209-133003-28571@example.com</eb:RefToMessageId>  
</eb:MessageData>
```

### **• MessageId Element**

The REQUIRED element MessageId is a globally unique identifier for each message conforming to MessageId [RFC2822].

Note: In the Message-Id and Content-Id MIME headers, values are always surrounded by angle brackets. However references in mid: or cid: scheme URI's and the MessageId and RefToMessageId elements MUST NOT include these delimiters.

### **• Timestamp Element**

The REQUIRED Timestamp is a value representing the time that the message header was created conforming to a dateTime [XMLSchema] and MUST be expressed as UTC [Coordinated Universal Time]. Indicating UTC in the Timestamp element by including the 'Z' identifier is optional.

### **• RefToMessageId Element**

The RefToMessageId element has a cardinality of zero or one. When present, it MUST contain the MessageId value of an earlier ebXML Message to which this message relates. If there is no earlier related message, the element MUST NOT be present.

For Error messages, the RefToMessageId element is REQUIRED and its value MUST be the MessageId value of the message in error.

### **• TimeToLive Element**



If the TimeToLive element is present, it MUST be used to indicate the time, expressed as UTC, by which a message should be delivered to the To Party MSH. It MUST conform to an XML Schema dateTime.

In this context, the TimeToLive has expired if the time of the internal clock, adjusted for UTC, of the Receiving MSH is greater than the value of TimeToLive for the message.

If the To Party's MSH receives a message where TimeToLive has expired, it SHALL send a message to the From Party MSH, reporting that the TimeToLive of the message has expired. This message SHALL be comprised of an ErrorList containing an error with the errorCode attribute set to TimeToLiveExpired and the severity attribute set to Error.

## DuplicateElimination Element

The DuplicateElimination element, if present, identifies a request by the sender for the receiving MSH to check for duplicate messages.

Valid values for DuplicateElimination:

- DuplicateElimination present – duplicate messages SHOULD be eliminated.
- DuplicateElimination not present – this results in a delivery behavior of Best-Effort.

## Description Element

The Description element may be present zero or more times. Its purpose is to provide a human readable description of the purpose or intent of the message. The language of the description is defined by a required xml:lang attribute. The xml:lang attribute MUST comply with the rules for identifying languages specified in XML [XMLSchema]. Each occurrence SHOULD have a different value for xml:lang.

## Example ebXML Message Manifest

```
<eb:Manifest eb:id="Manifest" eb:version="2.0">
<eb:Reference eb:id="studentrecord-01" link:href="cid:payload-1"
xlink:role="http://regrep.org/gci/purchaseOrder">
<eb:Schema eb:location="http://schemalocation.com/ims-lip.xsd" eb:version="1.1"/>
<eb:Description xml:lang="en-GB">New Student record</eb:Description>
</eb:Reference>
</eb:Manifest>
```

## Reliable Messaging

Reliable Messaging defines an interoperable protocol such that two Message Service Handlers (MSH) can reliably exchange messages, using acknowledgment, retry and duplicate detection and elimination mechanisms, resulting in the To Party receiving the message Once-And-Only-Once. The protocol is flexible, allowing for both store-and-forward and end-to-end reliable messaging.



Reliability is achieved by a Receiving MSH responding to a message with an Acknowledgment Message. An Acknowledgment Message is any ebXML message containing an Acknowledgment element. Failure to receive an Acknowledgment Message by a Sending MSH MAY trigger successive retries until such time as an Acknowledgment Message is received or the predetermined number of retries has been exceeded at which time the From Party MUST be notified of the probable delivery failure.

Whenever an identical message may be received more than once, some method of duplicate detection and elimination is indicated, suggested is the use of a persistent message store.

After a system interruption or failure, a MSH MUST ensure that messages in persistent storage are processed as if the system failure or interruption had not occurred. How this is done is an implementation decision.

In order to support the filtering of duplicate messages, a Receiving MSH MUST save the MessageId in persistent storage. It is also RECOMMENDED the following be kept in persistent storage:

- The complete message, at least until the information in the message has been passed to the application or other process needing to process it,
- The time the message was received, so the information can be used to generate the response to a Message Status Request, the complete response message.

### **Methods of Implementing Reliable Messaging**

Support for Reliable Messaging is implemented in one of the following ways:

- using the ebXML Reliable Messaging protocol,
- using ebXML SOAP structures together with commercial software products that are designed to provide reliable delivery of messages using alternative protocols,
- user application support for some features, especially duplicate elimination, or
- some mixture of the above options on a per-feature basis.

### ***Multi-hop Reliable Messaging***

Multi-hop (hop-to-hop) Reliable Messaging is accomplished using the AckRequested element and an Acknowledgment Message containing an Acknowledgment element each with a SOAP actor of Next MSH between the Sending MSH and the Receiving MSH. This MAY be used in store-and-forward multi-hop situations.

The use of the duplicate elimination is not required for Intermediate nodes. Since duplicate elimination by an intermediate MSH can interfere with End-to-End Reliable Messaging Retries, the intermediate MSH MUST know it is an intermediate and MUST NOT perform duplicate elimination tasks.

At this time, the values of Retry and RetryInterval between Intermediate MSHs remains implementation specific.

### **AckRequested Sample**

An example of the AckRequested element targeted at the NextMSH is given below:

```
<eb:AckRequested SOAP:mustUnderstand="1" eb:version="2.0" eb:signed="false"
SOAP:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"/>
```



In the preceding example, an Acknowledgment Message is requested of the next ebXML MSH node in the message. The Acknowledgment element generated MUST be targeted at the next ebXML MSH node along the reverse message path (the Sending MSH) using the SOAP actor with a value of NextMSH.

Any Intermediary receiving an AckRequested with SOAP actor of NextMSH MUST remove the AckRequested element before forwarding to the next MSH. Any Intermediary MAY insert a single AckRequested element into the SOAP Header with a SOAP actor of NextMSH. There SHALL NOT be two AckRequested elements targeted at the next MSH.

When the SyncReply element is present, an AckRequested element with SOAP actor of NextMSH MUST NOT be present. If the SyncReply element is not present, the Intermediary MAY return the Intermediate Acknowledgment Message synchronously with a synchronous transport protocol. If these two elements are received in the same message, the Receiving MSH SHOULD report an error with errorCode set to Inconsistent and severity set to Error.

### Acknowledgment Sample

An example of the Acknowledgment element targeted at the NextMSH is given below:

```
<eb:Acknowledgment SOAP:mustUnderstand="1" eb:version="2.0"
SOAP:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH">
  <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>
  <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
  <eb:From> <eb:PartyId>uri:www.example.com</eb:PartyId> </eb:From>
</eb:Acknowledgment>
```

### Multi-Hop Acknowledgments

There MAY be two AckRequested elements on the same message. An Acknowledgement MUST be sent for each AckRequested using an identical SOAP actor attribute as the AckRequested element.

If the Receiving MSH is the To Party MSH, then see section 6.5.2. If the Receiving MSH is the To Party MSH and there is an AckRequested element targeting the Next MSH (the To Party MSH is acting in both roles), then perform both procedures for generating Acknowledgment Messages. This MAY require sending two Acknowledgment elements, possibly on the same message, one targeted for the Next MSH and one targeted for the To Party MSH.

There MAY be multiple Acknowledgements elements, on the same message or on different messages, returning from either the Next MSH or from the To Party MSH. A MSH supporting Multi-hop MUST differentiate, based upon the actor, which Acknowledgment is being returned and act accordingly.

If this is an Acknowledgment Message then:

1. Look for a message in persistent storage with a MessageId the same as the value of RefToMessageId on the received Message.
2. If a message is found in persistent storage then mark the persisted message as delivered.

If an AckRequested element is present (not an Acknowledgment Message) then generate an Acknowledgment Message in response (this may be as part of another message).

The Receiving MSH MUST NOT send an Acknowledgment Message until the message has been persisted or delivered to the Next MSH.

### ErrorList Sample



An example of an ErrorList element is given below.

```
<eb:ErrorList eb:id="3490sdo", eb:highestSeverity="error" eb:version="2.0"
SOAP:mustUnderstand="1">
  <eb:Error eb:errorCode="SecurityFailure" eb:severity="Error" eb:location="URI_of_ds:Signature">
  <eb:Description xml:lang="en-GB">Validation of signature failed</eb:Description>
  </eb:Error>
  <eb:Error ...>
  ...
  </eb:Error>
</eb:ErrorList>
```



## Errors allowed.

### XML Errors :

- ValueNotRecognized - Element content or attribute value not recognized. - Although the document is well formed and valid, the element / attribute contains a value that could not be recognized and therefore could not be used by the ebXML Message Service.
- NotSupported - Element or attribute not supported - Although the document is well formed and valid, a module is present consistent with the rules and constraints contained in this specification, but is not supported by the ebXML Message Service processing the message.
- Inconsistent - Element content or attribute value inconsistent with other elements or attributes. - Although the document is well formed and valid, according to the rules and constraints contained in this specification the content of an element or attribute is inconsistent with the content of other elements or their attributes.
- OtherXml - Other error in an element content or attribute value. - Although the document is well formed and valid, the element content or attribute value contains values that do not conform to the rules and constraints contained in this specification and is not covered by other error codes. The content of the Error element should be used to indicate the nature of the problem.

### Non XML Errors:

- DeliveryFailure - Message Delivery Failure - A message has been received that either probably or definitely could not be sent to its next destination. Note: if severity is set to Warning then there is a small probability that the message was delivered.
- TimeToLiveExpired - Message Time To Live Expired - A message has been received that arrived after the time specified in the TimeToLive element of the MessageHeader element.
- SecurityFailure - Message Security Checks Failed - Validation of signatures or checks on the authenticity or authority of the sender of the message have failed.
- MimeProblem - URI resolve error - If an xlink:href attribute contains a URI, not a content id (URI scheme "cid"), and the URI cannot be resolved, then it is an implementation decision whether to report the error.
- Unknown - Unknown Error - Indicates that an error has occurred not covered explicitly by any of the other errors. The content of the Error element should be used to indicate the nature of the problem.



## Appendix 3 : IMS-LIP

### *Example of a Learner Information Package*

This section is still undergoing additions. So far, no changes to existing information have been made. Missing currently are values for the start date of the modules, the delivery institution and the awarding body. It is still being decided as to the best way of mapping this data, either referentially or if possible through explicit definition in the module definitions. The reasoning behind not wanting referential information is due to the nature of this information to change over time.

#### **Use Case 1A – INITIAL REGISTRATION - FE**

A learner is registered on an FE course at a Partner Institution. The qualification is one which is broken down to modular level. At registration the learner selects three modules – Computer Art, Computer Aided Design and Spreadsheets.

```
<learnerinformation>
  <comment>A basic example of creating a LIP record.</comment>
  <contenttype>
    <referential>
      <sourcedid>
        <source>SHELL</source>
        <id>cornwall.ac.uk</id>
      </sourcedid>
    </referential>
  </contenttype>

  <identification>
    <contenttype>
      <referential>
        <indexid>cornwall.ac.uk.211653</indexid>
        <!-- could be built from the URI for this source and the
local 'studentID' IE cornwall.ac.uk.211653-->
      </referential>
    </contenttype>
    <name>
      <typename>
        <tysource sourcetype="imsdefault"/>
        <tyvalue>Preferred</tyvalue>
      </typename>
      <contenttype>
        <referential>
          <indexid>unknown</indexid>
        </referential>
      </contenttype>
      <partname>
        <typename>
          <tysource sourcetype="imsdefault"/>
          <tyvalue>First</tyvalue>
        </typename>
        <text>Ian</text>
      </partname>
      <partname>
        <typename>
          <tysource sourcetype="imsdefault"/>
          <tyvalue>Last</tyvalue>
        </typename>
        <text>Jones</text>
      </partname>
    </name>
    <address>
      <typename>
        <tysource sourcetype="imsdefault"/>
```



```

    <tyvalue>Private</tyvalue>
  </typename>
</contenttype>
  <referential>
    <indexid>cornwall.ac.uk.211653</indexid>
  </referential>
</contenttype>
<street>
  <nonfieldedstreetaddress>6 Smith Place, Truro, TR16 6HB
</nonfieldedstreetaddress>
  <streetnumber>6</streetnumber>
  <streetname>Smith</streetname>
  <streettype>Place</streettype>
</street>
<city>Truro</city>
<country>UK</country>
<postcode>TR16 6HB</postcode>
<timezone>GMT</timezone>
</address>
</identification>
<activity>
  <typename>
    <tysource sourcetype="imsdefault"/>
    <tyvalue>Education</tyvalue>
  </typename>
  <contenttype>
    <referential>
      <indexid>TACNNCL1</indexid>
    </referential>
  </contenttype>
  <definition>
    <typename>
      <tysource sourcetype="imsdefault"/>
      <tyvalue>Course</tyvalue>
    </typename>
    <contenttype>
      <referential>
        <indexid>Computer</indexid>
      </referential>
    </contenttype>
    <definition>
      <typename>
        <tysource sourcetype="imsdefault"/>
        <tyvalue>Curriculum</tyvalue>
      </typename>
      <contenttype>
        <referential>
          <indexid>Year1</indexid>
        </referential>
      </contenttype>
      <definition>
        <typename>
          <tysource sourcetype="imsdefault"/>
          <tyvalue>Module</tyvalue>
        </typename>
        <contenttype>
          <referential>
            <indexid>CL1CA1</indexid>
          </referential>
        </contenttype>
        <definitionfield>
          <fieldlabel>
            <typename>
              <tyvalue>Module</tyvalue>
            </typename>
          </fieldlabel>
          <fielddata>Computer Art</fielddata>
        </definitionfield>
      </definition>
    </definition>
  </definition>

```



```

<definition>
  <typename>
    <tysource sourcetype="imsdefault"/>
    <tyvalue>Module</tyvalue>
  </typename>
  <contenttype>
    <referential>
      <indexid>CL1CA8</indexid>
    </referential>
  </contenttype>
  <definitionfield>
    <fieldlabel>
      <typename>
        <tyvalue>Module</tyvalue>
      </typename>
    </fieldlabel>
    <fielddata>Computer Aided Design</fielddata>
  </definitionfield>
</definition>
<definition>
  <typename>
    <tysource sourcetype="imsdefault"/>
    <tyvalue>Module</tyvalue>
  </typename>
  <contenttype>
    <referential>
      <indexid>CL1CA7</indexid>
    </referential>
  </contenttype>
  <definitionfield>
    <fieldlabel>
      <typename>
        <tyvalue>Module</tyvalue>
      </typename>
    </fieldlabel>
    <fielddata>Spreadsheets</fielddata>
  </definitionfield>
</definition>
</definition>
</activity>
</learnerinformation>

```

