Technology Dedicated to Business Efficiency

ebMS Implementation Guidelines

eb1.1.0 2010v1

ebMS Implementation Guidelines: eb1.1.0 2010v1

Editor:
David Carver, STAR

Contributors:
Jason Loeffler, Karmak

Russell Shephard, T-Systems

David Carver, STAR

Table of Contents

List of Figures

Preface

I.I. Purpose

The purpose of this document is to provide guidelines and best practices for implementing the ebXML protocol specifications when exchanging ebXML messages containing STAR XML BODs.

This document is broken into sections for easier navigation. The following sections are:

- Preface – Overview of the specifications and background
- Section 1 – Message Handling
- Section 2 – Infrastructure
- Appendices – Examples

I.II. Summary of Changes from EB3.0G001

The only change from eb3.0g001 is the version number.

I.III. Scope

This document covers the implementation of the electronic business Messaging Service (ebMS) 2.0 and electronic business Collaboration Protocol Profile and Agreement (ebCPPA) 2.0 protocols for STAR message exchange.

I.IV. Audience

This document is intended for application developers and application architects developing STAR ebXML interfaces.

I.V. Background

The ebXML specifications contain detailed definitions for the ebXML Message Service protocol, Collaboration Protocol Profile (CPP) and Collaboration Protocol Agreement (CPA). The specifications do not, however, define details for implementing these components. Specifically, they do not explicitly define the formats or acceptable values for many of the elements contained in the ebMS message, CPP, or CPA. Elements such as Party Id, Service and Action, and the CPA Id are defined; however the proper formats and values for these elements are left up to the implementer.

This document will define the STAR guidelines and best practices for implementing the ebXML messaging service and the CPP/CPA. Required and recommended formats and acceptable values will be defined for key ebXML elements. These implementation guidelines will ensure consistent usage of the ebXML specifications for all STAR members and help to ensure interoperability.

I.VI. References

ebXML is an initiative of OASIS (Organization for the Advancement of Structured Information Standards). The OASIS web site, as well as the specifications referenced in this document can be found using the following URLs.

OASIS Web Site:	http://www.oasis.org [http://www.oasis.org/]
ebMS 2.0 Messaging Service:	http://www.ebxml.org/specs/ebMS2.pdf
CPP and CPA Specification v2.0	http://www.ebxml.org/specs/ebcpp-2.0.pdf

Chapter 1. ebMS Messaging

Table of Contents

1.1. ebMS Messaging

1.1.1. ebMS Message Packaging

An ebMS Message is a MIME/Multipart message envelope referred to as a Message Package. The Message Package is structured in compliance with [SOAP] version 1.1 and SOAP Messages with Attachments [SwA] specifications.

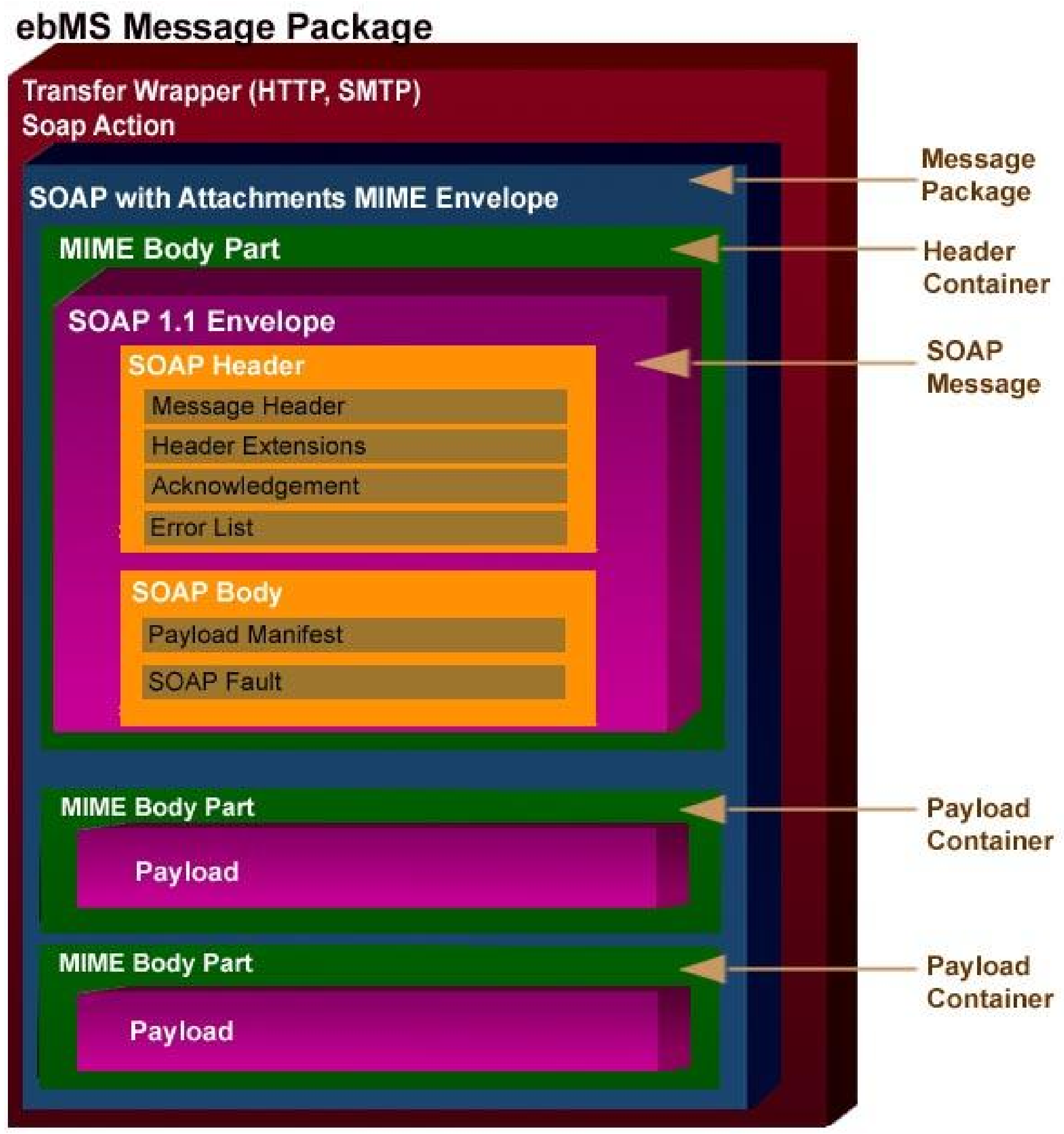
There are two or more logical MIME parts within a Message Package:

1. The Header Container contains one SOAP version 1.1 compliant message that holds the ebMS Header elements. The large majority of ebMS Headers are placed in the SOAP Header, if any Manifest or SOAP Fault elements occur they are placed in the SOAP Body.

2. Payload Containers, which contain application level payloads. For the STAR XML Infrastructure project, these will consist mostly of STAR BODs.

1.1.2. Message Packaging

Figure 1.1. ebMS Message Package



Highlights of this packaging format:

- Based on a widely accepted industry de-facto standard SOAP v1.1
- Transport Metadata such as To/From Parties and MessageIDs can be placed in the SOAP Header
- Messages can be simple SOAP messages or more complex messages with multiple payloads

1.1.3. Payload Containers

ebMS version 2.0 and the STAR Transport require that if any payloads are present they MUST be contained in a Payload Container and MUST be referenced by a Header Manifest element entry. Payloads may be composed of any type of data including and not limited to word processor files, graphics, sound, EDI, XML or any data that can be digitized. In the context of the STAR XML Infrastructure project, the payloads will primarily be STAR BODs.

1.1.4. STAR ebMS Guidelines Message Elements

The ebMS Element Summary table (shown below) identifies the critical ebMS message elements for STAR ebMS Guideline. These elements were identified for the original STAR Message Infrastructure Guidelines and have been updated to account for changes between ebMS version 1.0 and ebMS version 2.0.

Element / Attribute Name **Required** *Required if Present* (Optional) [attribute] default	Infrastructure Sample/Recommendation
Content-Type	text/xml
Charset	UTF-8
?xml	
[*version="1.0"*]	
([encoding])	UTF-8
SOAP-ENV:Header	
MessageHeader	
[SOAP:mustUnderstand="1"]	
[version="2.0"]	
([id])	
From	
PartyID	Logical identifier
[type]	DUNS, string
(Role)	Agreed to by both parties, if CPA is used, value must match CPA
To	
PartyID	Logical identifier
[type]	DUNS, string
Role	Agreed to by both parties, if CPA is used, value must match CPA
CPAId	If no CPA exists,

	FromPartyId-ToPartyId-cpa[-x.x]
ConversationId	Timestamp + unique host identifier
Service [type]	For BOD Payloads: STARBOD
Service	For BOD Payloads: STAR BOD Noun
Action	For BOD Payloads: STAR BOD Verb
MessageData	
MessageId	Service Name concatenated with a period (.) followed by the GUID, followed by an at sign (@) followed by the company name in domain name format. Must conform to [RFC2392]
Timestamp	UTC with no offsets. Represents the time the message was created.
RefToMessageID (Required for Error or Acknowledgement or Pong)	MessageId from earlier message
TimeToLive	
MessageOrder	
AckRequested	
[SOAP:mustUnderstand=”1”]	
[signed]	false or true
[version=”2.0”]	
[SOAP:actor=”urn:oasis:names:tc:ebxml-msg:actor:ToPartytMSH”]	
Acknowledgement	
[SOAP:mustUnderstand=”1”]	
[SOAP:actor=”urn:oasis:names:tc:ebxml-msg:actor:nextMSH”]	If present, must match this value
[version=”2.0”]	
[id]	
Timestamp	UTC with no offsets, represents the time the Acknowledgment was created
RefToMessageID	MessageID of message being acknowledged
ErrorList Element	
[SOAP:mustUnderstand=”1”]	
[highestSeverity=(Error I Warning)]	

[version=”2.0”]	
[id]	
Error	
[codeContext=”URI” (default=” http://www.ebxml.org/messageSer-viceErrors ”)	
errorCode=(ValueNotRecognized I NotSupported I Inconsistent I OtherXml I	
DeliveryFailure I TimeToLiveExpired I Secu-rityFailure I MimeProblem I Unknown)	
severity=(Warning I Error)	
location (Xpointer I CID)	
xml:lang=”en-US”	
Id]	
ds:Signature	Must confirm to the [XMLDSIG] specification.
SOAP-ENV:Body	Every direct child of SOAP-ENV:Body has an automatic attribute of SOAP-ENV:mustUnderstand=”1”. (see SOAP 1.1 sect 4.3.1)
Manifest	Required if one or more payloads (i.e. ebXML wrapped BOD) are present
[version=”2.0”]	
[id]	
Reference	
[xlink:href]	URI of the payload object referenced.
[xlink:type=”simple”]	
[id]	
Schema	
[location]	/STAR/Rev1.2/BODs/Stan-dalone/ProcessPartsOrder.xsd
[version]	
Description	
xml:lang=”en-US”	
StatusRequest	
[version=”2.0”]	
[id]	

RefToMessageId	MessageId from earlier message
StatusResponse	
[version=”2.0”]	
RefToMessageId	MessageId of the original message being reported on
TimeStamp	The Timestamp of the original message being reported on
[messageStatus= (UnAuthorized | NotRecognized | Received | Processed | Forwarded]	MessageId of the original message being reported on
[id]	

1.1.5. ebMS MessageHeader Elements

1.1.5.1. From/To PartyId Elements

Within the ebMS MessageHeader, the REQUIRED From and To elements include a PartyId element which identifies the sending or receiving party for the message. The PartyId element value is defined by the PartyId type attribute. Commonly used types are the Uniform Resource Identifier (URI) and the Uniform Resource Name (URN), however a generic string type is also allowed providing that it is understood by both parties. STAR recommends the use of the following PartyId types:

- For OEMs and large institutions: The urn:duns type if a DUNS number is available
- For Automotive Dealers: A string type containing the short manufacturer code followed by the dealer number
- For Non-Automotive Dealers: A string type containing unique identification information.

Examples:

Example of a Volkswagen OEM:

```
<eb:PartyId type=”urn:duns”>006972475</eb:PartyId>
```

Example of a Volkswagen Automotive Dealer:

```
<eb:PartyId type=”string”>VW400110</eb:PartyId>
```

1.1.5.2. CPAId Element

The CPAId element is a REQUIRED ebXML element. It is a string that identifies the parameters governing the exchange of messages between the parties. The recipient of a message MUST be able to resolve the CPAId to an individual set of parameters, taking into account the sender of the message. This does NOT mean that a formal CPA conforming to the ebXML CPA specification must be in place for ebXML messaging to be used. If no formal CPA exists, the CPAId is simply the location of party specific information such as IP addresses and dealer Ids. This is a temporary format until party id formats can be established in a future versions of this publication.

For STAR, the CPAId element shall have the following format:

<CPAId>fromPartyId-ToPartyId-cpa</CPAId>

The CPAId may also optionally contain a version number:

<CPAId>fromPartyId-ToPartyId-cpa-x.x</CPAId>

Examples:

```
<eb:CPAId>VW400110-006942475-cpa</eb:CPAId>

<eb:CPAId>VW400110-006942475-cpa-1.0</eb:CPAId>
```

1.1.5.3. 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 From and To party pair. The Party initiating a conversation determines the value of the ConversationId element that SHALL be reflected in all messages pertaining to that conversation. The value for ConversationId in the STAR XML Infrastructure environment SHOULD be datestamp + timestamp + a unique host identifier; UTC format is used for datestamp. For example, if two salespeople at the same dealership submit an inquiry at the same time (obviously from separate terminals), then the algorithm to generate the ConversationId should be such that this situation would generate two separate ConversationIds.

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.

1.1.5.4. Service and Action Elements

The REQUIRED Service and Action elements identify the service that acts on the message.

For STAR XML Infrastructure, the value of the type attribute of the Service element for messages that contain a STAR BOD payload MUST be “STARBOD” and the value of the Service element MUST be the STAR BOD Noun. The value of the Action element MUST be the STAR BOD Verb.

Exceptions to this include ebMS standalone error messages and asynchronous acknowledgments, whose Service Action values are defined by ebMS version 2.0.

An example of the Service and Action elements for an STAR BOD payload follows:

```
<eb:Service eb:type=”STARBOD”>PartsOrder</eb:Service>

<eb:Action>Process</eb:Action>
```

1.1.5.5. MessageData Element

The REQUIRED MessageData element provides a means of uniquely identifying an ebMS message.

MessageId element

Message IDs MUST be Globally unique and conformant to ebMS specifications which require that the value is conformant to RFC2392.

STAR requires three (3) data elements within the Message ID:

1. Company Name, in domain name format, for example starstandards.org

2. Service Identifier, the name of the service being invoked

3. A Globally Unique Identifier (GUID), as specified in RFC2822 section 3.6.4

The Service Name should be concatenated with a period (.) followed by the GUID, followed by an at sign (@) followed by the company name in domain name format.

Example:

```
<eb:MessageId>PartsOrder.323210:e5c74:7ffc@starstandards.org</eb:MessageId>
```

Timestamp element

The REQUIRED Timestamp is a value representing the creation time of the message header and MUST be in UTC format (Universal Time Code as defined by ISO 8601).

1.1.5.6. Digital Signature

A STAR ebMS Message can be digitally signed to provide security countermeasures. Application of Digital Signature is a Recommendation, in other words, conformant implementations should be capable of processing messages with Digital Signature, but individual implementations may choose not to use Digital Signature features.

1.1.5.7. Manifest Element

The Manifest is a composite element that summarizes message payloads. A Manifest element MUST be present if one or more Payloads exist, and all Payloads MUST be referenced in the Manifest. The purpose of the Manifest is:

- To make it easier to directly extract a particular payload

- To allow an application to determine whether it can process a payload without having to parse it

The structure and content of the Manifest element MUST conform to the ebMS version 2.0 specifications.

1.1.5.8. Acknowledgment Element

The Acknowledgment element is used by the To Party that received a message, to let the From Party that sent the message know the message was received. The *RefToMessageId* in a message containing an Acknowledgment element identifies the message for which the receipt is being generated. The *RefToMessageId* is the MessageId of the original message.

Chapter 2. Implementing the CPA

Table of Contents

2.1. Implementing the CPA

2.1.1. Overview

This section describes the implementation of the CPA for STAR trading partners. This section is not intended to be a complete description of the CPA and its usage. For a detailed description of the CPA, please refer to the specifications document referenced in the preface of this document.

The STAR CPA has been designed for maximum flexibility. The Collaboration Roles have been designed to allow each party to define its own transport, security, and reliable messaging characteristics for each transaction type.. Channel, Document Exchange, and Transport definitions have also been pre-defined for every combination of transport type and message delivery option. The CPA parties need only to refer to the appropriate channel ID and DocExchange ID combination in the DeliveryChannel definition for each collaboration.

There are a number of benefits to having such a flexible design.

1. BODs or other transactions can be added or removed individually, allowing the CPA to reflect only those transactions that are actually supported by both parties

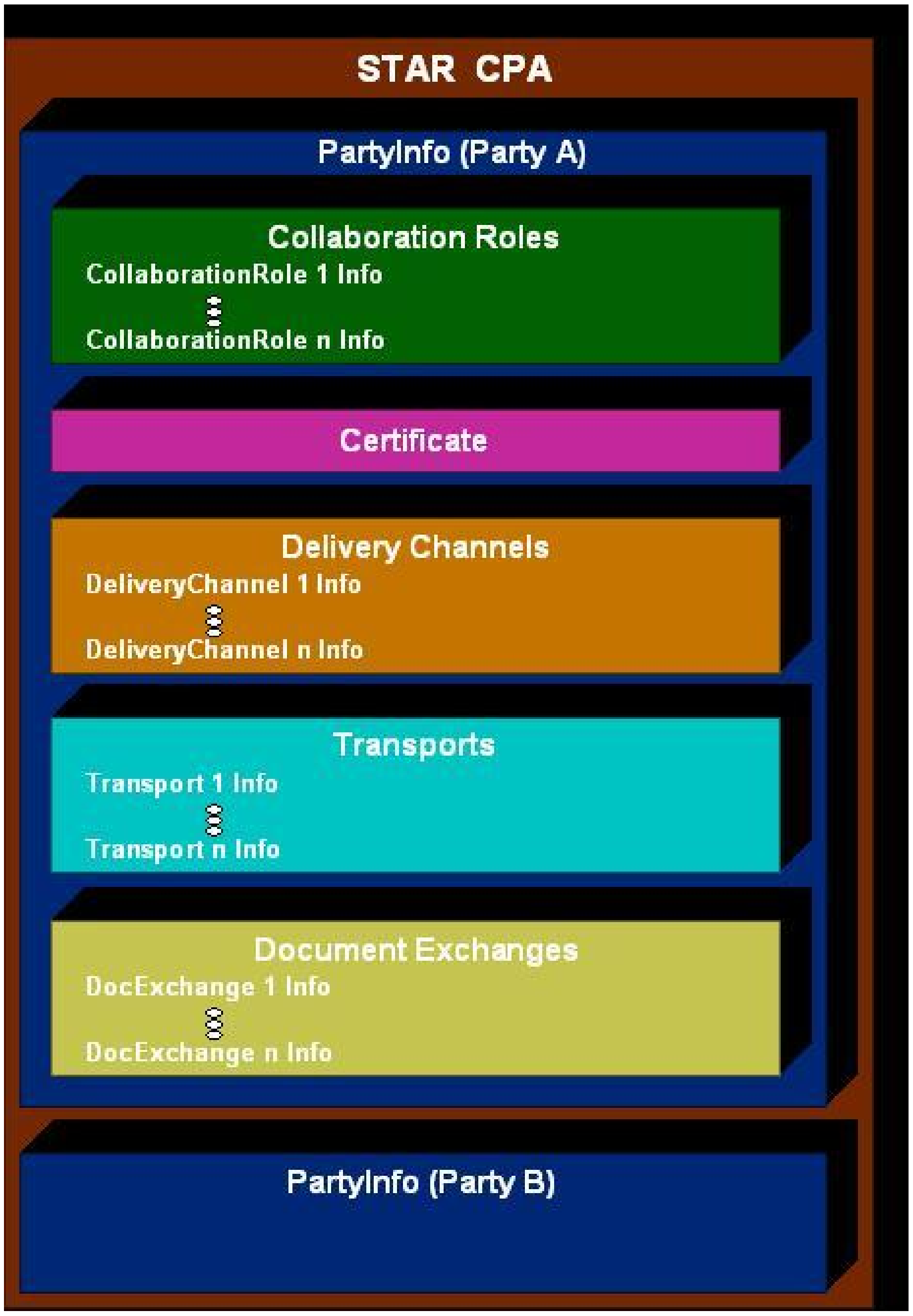
2. Those transactions that require more security or reliable messaging properties can have those properties defined without impacting the other transactions within the CPA

3. Transport definitions can be created for each transaction type. This will allow each BOD to be routed to a different URL, if necessary. This may be needed in large dealer franchises where the Service, Parts, and Accounting applications are running on separate physical machines or on separate application instances.

2.1.2. STAR CPA Structure

Figure 2.1, “STAR CPA Structure” below is a high-level depiction of the STAR CPA. It displays the primary CPA elements. Each primary element may have one or more child elements. The STAR implementation of these elements will be described below. A complete sample CPA can be found in Appendix A: Example CPA.

Figure 2.1. STAR CPA Structure



2.1.3. PartyInfo Section

The PartyInfo element is the heart of the CPA. It defines the identification and security information for each party, the collaborations (BODs) supported by each party and all of the transaction characteristics for each collaboration. STAR has defined implementation guidelines for several of the child elements within the PartyInfo section.

2.1.3.1. CPA ID

See section 1.6.2 for the naming conventions for the CPA ID element.

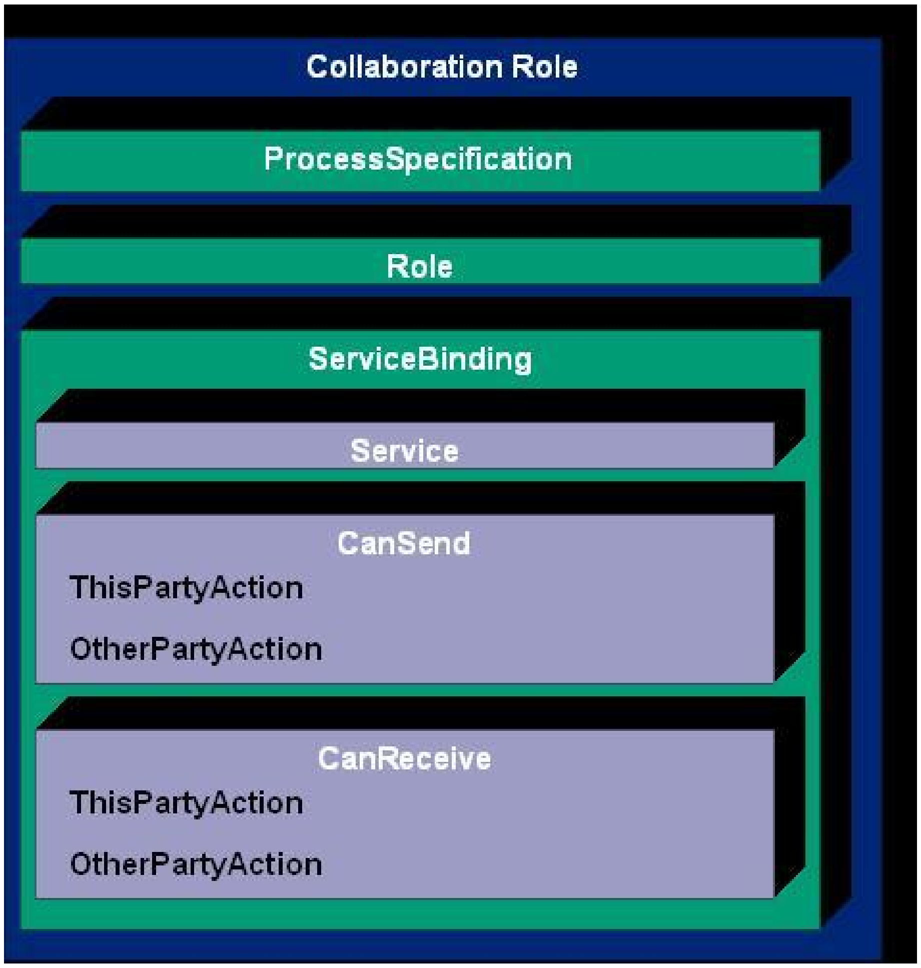
2.1.3.2. Party ID

The naming conventions for the PartyId element are still under development by STAR.

2.1.3.3. Collaboration Roles

The CollaborationRole elements are the heart of the CPA. They are used to define the trading characteristics for each transaction.   In the sample CPA shown in Appendix A, each Collaboration Role element is associated with a STAR BOD using the ProcessSpecification element. The structure of the CollaborationRole element is shown in Figure 2.2, “Collaboration Role Structure”.

Figure 2.2. Collaboration Role Structure



2.1.3.4. Process Specification

The ProcessSpecification element defines the BOD type. The value of the name attribute should match the name attribute in the BPSS document for that BOD.

<tns:ProcessSpecification tns:name="PartsOrder" tns:uuid="urn:icann:star.org:bpid:3A4$2.0" tns:version="2.0" xlink:href="http://www.vwoa.com/po_processing/" xlink:type="simple"/>

2.1.3.5. Role

The role names are defined using the Role element. Again, the Role names should match those in the BPSS document for the BOD. Currently STAR has defined the role names “Initiator” and “Responder” for all BOD types.

<<tns:Role tns:name="Initiator" xlink:href="http://www.starstandard.org/processes/3A4.xml#Initiator" xlink:type="simple"/>

The CPA was designed with two CollaborationRole elements per BOD to allow different messaging and transport parameters to be defined depending on whether the party is acting as the Initiator or the Responder for a transaction.

2.1.3.6. Service Binding

The ServiceBinding element defines the Service and Action elements that appear in the ebMS message header

2.1.3.7. Service Element

The Service Element is defined using the BOD Noun value. For example, the ServiceElement for a PartsOrder collaboration role would be “PartsOrder”.

<tns:ServiceBinding>

    <tns:Service tns:type="string">PartsOrder</tns:Service>

2.1.3.8. CanSend and CanReceive Elements

The CanSend and CanReceive elements define the ebXML actions associated with each collaboration. The ebXML action element corresponds to the STAR BOD verb that will be supported by each party. The ThisPartyActionBinding element within the CanSend or CanReceive element is used to indicate the supported action. The OtherPartyActionBinding element is a reference to the ID of the other party’s corresponding CanReceive element.

For example, if Party A is an OEM acting as the “Responder” and we are defining the collaboration for the PartsOrder BOD, the OEM would be capable of sending an “Acknowledge” verb and receiving a “Process” verb. The CanSend element would contain “Acknowledge” in the ThisPartyActionBinding element and the ID of Party 2’s corresponding CanReceive element in the OtherPartyActionBinding element. The CanReceive element would contain “Process” in the ThisPartyActionBinding element and the ID of Party 2’s corresponding CanSend element.

CanSend element for Responder collaboration role

<tns:CanSend>

    <tns:ThisPartyActionBinding tns:action="Acknowledge" tns:id="SendPOAck" tns:packageId="DefaultComposite">

#

#

#

        <tns:OtherPartyActionBinding>Party2_ReceiveAckPO</tns:OtherPartyActionBinding>

CanReceive element for Responder collaboration role

<tns:CanReceive>

        <tns:ThisPartyActionBinding tns:action="Process" tns:id="ReceivePO"
tns:packageId="DefaultPackage">

#

#

#

        <tns:OtherPartyActionBinding>Dealer_SendPO</tns:OtherPartyActionBinding>

2.1.3.9. Business Transaction Characteristics

The attributes contained in the BusinessTransactionCharacteristics element will be negotiated and defined by each party according to their individual requirements.

2.1.3.10. Channel ID

The ChannelId element contains the identifier for the delivery channel that will be used to send or receive the BOD defined for a particular collaboration. The STAR CPA contains 14 delivery channel definitions. Each definition defines a different set of transport and message delivery parameters. Each party will determine the appropriate delivery channel types for each of the BODs that they support and assign the required delivery channel using the ChannelId element.

2.1.3.11. Certificate Info

The certificate section of the STAR CPA is defined using the standard constructs as defined in the ebCPPA 2.0 specifications document.

2.1.3.12. Delivery Channel Definitions

As stated above, the STAR CPA contains 14 pre-defined channel types that can be used to define the delivery parameters for any collaboration activity by inserting the appropriate ID into the ChannelID element.

2.1.3.13. Document Exchange Definitions

The STAR CPA contains 3 pre-defined DocExchange elements, each of which defines a different set of encryption and non-repudiation settings. A DocExchange ID is associated with Delivery Channel type definition.

2.1.3.14. Transport Definitions

The STAR CPA contains a single transport definition for a standard HTTP connection. Additional transport definitions may defined to support SSL or SMTP transports as needed.

Chapter 3. Implementing ebMS Reliable Messaging

Table of Contents

3.1. Implementing ebMS Reliable Messaging

3.1.1. ebMS Delivery Assurance Profiles

This section describes how to implement the Delivery Assurance profiles with ebMS version 2.0 and CPPA version 2.0.

3.1.1.1. Best-Effort

To enable Best-Effort an ebMS message is sent without using any of the ebMS reliable messaging features, in other words, for the sender:

- NO Acknowledgment is requested (AckRequested element is not present)
- NO Duplicate Elimination is requested (DuplicateElimination element is not present)
- NO TimeToLive is specified (TimeToLive element is not present)
- Failed messages are not retried

3.1.1.2. At-Least-Once (Message Acknowledgement)

To enable At-Least-Once an ebMS message sender:

- MUST request an Acknowledgment (AckRequested element is present)
- NO Duplicate Elimination requiested (DuplicateElimination element is not present)
- SHOULD specify TimeToLive (TimeToLive element is present)

- MUST retry failed messages

3.1.1.3. At-Most-Once (NOT recommended by STAR for simplicity)

To enable At-Most-Once an ebMS message sender:

- MUST not request an Acknowledgment (AckRequested element is not present)
- MUST request Duplicate Elimination (DuplicateElimination element is present)
- SHOULD specify TimeToLive (TimeToLive element is present)
- MAY retry failed messages
- Parties MUST agree out-of-band to a value for RetryInterval
- Parties MUST agree out-of-band to a value for NumberOfRetries

3.1.1.4. Once-And-Only-Once / Exactly-Once (Guaranteed Delivery)

Implementers MUST provide and use the following features:

- MUST include an AckRequested element
- MUST include a DuplicateElimination element
- MUST include a TimeToLive element and the value of TimeToLive must conform to TimeToLive > Timestamp + ((NumberOfRetries + 1) * RetryInterval)
- Parties MUST agree out-of-band to a value for RetryInterval
- Parties MUST agree out-of-band to a value for NumberOfRetries

3.1.2. ebMS Delivery Assurance Features

3.1.2.1. Message Routing

Message Routing in ebMS can be accomplished through a combination of the underlying Transfer protocol, data elements in the messages themselves and out-of-band agreements as determined by ebXML CP-PA.

Routing at the Transfer level is defined by HTTP URLs. At the message level, parties can key off the ToParty, FromParty, Service, Action and CPAID elements. At the CPPA level, “return addresses” can be defined. Parties may use a combination of these attributes to route messages in a way that makes sense for differing business scenarios or system architectures.

3.1.2.2. Acknowledgment of Receipt

Receipt of an acknowledgment indicates that the original message reached its destination. In other words, an Acknowledgment signifies only that a message has been received securely and intact, it is not a business level Acknowledgment.

ebMS clearly defines the format and content of Acknowledgment messages.

Although ebMS Acknowledgment messages are typically stand-alone messages, this is not required; an Acknowledgment to a message could be returned as part of a synchronous reply, as a stand-alone asynchronous message or as a part of a separate business message exchange.

An ebMS Acknowledgment contains a RefToMessageID, which is the exact MessageID of the original message, i.e. the message that is being acknowledged. This allows the sender of the original message to cross-reference the original message and confirm delivery.

As an option an ebMS Acknowledgment can be signed, which allows the sender to validate that the specific intended party received the message. Optionally, a signed message can contain a digest of the original message, allowing for full Non-repudiation of receipt. In other words, the sender knows who received the message and the sender can prove that the message was received exactly as sent.

3.1.3. ebMS Message Integrity

3.1.3.1. Content Integrity

Content Integrity is provided in ebMS through the use of XML Digital Signature. An original message can be signed, allowing the receiver to validate that the contents of the message have not been altered. In addition, as mentioned above, ebMS Acknowledgements may be signed and may include a digest of the original message, allowing for Non-repudiation of receipt, in other words the sender can prove that the message was received by the intended receiver exactly as sent.

3.1.3.2. TimeToLive

The ebMS TimeToLive element is a UTC (Universal Time Code). TimeToLive is a required message element for Once-And-Only-Once / Exactly-Once message delivery. If a receiver determines that the TimeToLive has expired, it must return an error and not process the message.

3.1.3.3. Message Sequencing

ebMS supports the ability to order multiple messages, guaranteeing that the messages are processed in order.

Message Ordering is set using the SequenceNumber element which is a positive integer that must be unique within a ConversationID. ConversationIDs must be unique within a CPA (Collaborative Partner Agreement) ID. Effectively, two parties establish one or more contexts for messaging in a Partner Agreement, which may also include such Policies as Delivery Assurance levels and security parameters. Within this Policy context, unique Conversations are established and sequences of messages (message 1, message 2, message 3,) can be sent.

A receiver MUST not process a message until all messages in the Conversation with lower sequence numbers have been received and processed.

If a message is received out of sequence, the receiver MUST format and send an error message notifying the sender.

3.1.3.4. ebMS Standardized Error Handling and Monitoring

ebMS defines error handling at the SOAP level (SOAP Faults) and at the ebMS level with an error listing mechanism that provides for both errors and warnings.

In the context of STAR Reliable Messaging, ebMS provides support for Retry, Recovery, TimeOut and Duplicate Detection.

3.1.3.5. Retry

ebMS supports retransmission of unacknowledged messages. As described above, At-Least-Once and Once-And-Only-Once / Exactly-Once require the ability to resend messages. ebMS requires sending implementations to store outbound messages and resend them if an Acknowledgement is not received within an agreed upon TimeOut period. The resent message is intended to be exactly the same as the original message and at the very least it must have the exact same ConversationID and MessageID.

3.1.3.6. Recovery Processes / Message Store

To support Retry and Duplicate Elimination, ebMS requires senders to store outbound messages and requires receivers to store inbound messages in a persistent store. Receivers must maintain inbound messages for a period of time agreed upon in a CPA Policy element known as PersistDuration. ebMS recommends that the persistent stores be durable (maintain information through a system failure) and resilient enough to survive the failure of any single software or hardware component. In the case of system failure, messages must be processed as if the system failure had not occurred.

3.1.3.7. TimeOut

ebMS supports TimeOut through an out-of-band agreement based on CPPA. Parties use CPPA to agree on values for Retries (NumberOfRetries) and RetryInterval. Implementations are expected to follow the policy agreements; if a message is sent and not acknowledge within the RetryInterval the sender will retry the message Retries number of times.

3.1.3.8. Duplicate Detection

ebMS Supports Duplicate Detection through a combination of policy agreements and data elements in individual messages.

Parties agree via CPPA whether or not to use DuplicateElimination. To support duplicate elimination receivers are required to durably store MessageIDs. Individual messages that require DuplicateElimination must contain the DuplicateElimination element.

If a receiver determines a message is a duplicate, it must not forward the message for processing and it must return to the sender a copy of the original acknowledgment that was sent concerning the original message.

3.1.4. ebMS CPA Configuration and Examples

A CPA (Collaborative Partner Agreement) is an XML document which represents an agreement between exactly two parties who will exchange ebXML messages. This agreement defines the names of the parties involved and the messaging characteristics they will be using. ebXML message exchanges can have various reliability and security modes and can be synchronous or asynchronous. CPA’s are uniquely identified in messages using an element named CPAID.

The CPAID attribute of the CollaborationProtocolAgreement element identifies the CPA to the messaging server. It can have any form but the recommendation is to concatenate party names in alphabetical order separated by a dash. For example "ABMotorCo-DEMotorCo". For example:

```
<tp:CollaborationProtocolAgreement

    Xmlns:tp=http://www.oasis-open.org/committees/ebXML-cppa/schema/cpp-cpa-2_0.xsd

    xsi:schemaLocation=”http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-

    tp:cpaid=” ABMotorCo-DEMotorCo” tp:version=”2_0a”>
```

The CPA provides two elements for configuring reliable messaging agreements. The ReliableMessaging tag and the MessagingCharacteristics tag.

The ReliableMessaging tag insures message handlers will create and manage AckRequest and Acknowlegment tags in the ebMS message. Without this tag, by default "Best-Effort" or no reliability is used in the message exchange. Thus optional tp:ReliableMessaging element under tp:ebXMLSenderBinding and tp:ebXMLReceiverBinding will be required in the implementation.

The ReliableMessaging Tag in the CPA can be configured as follows:

```
<tp:ReliableMessaging>

   <tp:Retries>5</tp:Retries>

   <tp:RetryInterval>PT2H</tp:RetryInterval>

    <tp:MessageOrderSemantics>NotGuaranteed</tp:MessageOrderSemantics>

</tp:ReliableMessaging>
```

The MessagingCharacteristics in the CPA can be configured as follows:

```
<tp:MessagingCharacteristics

    tp:syncReplyMode="none"

    tp:ackRequested="always"

    tp:ackSignatureRequested="none"
```

```
        tp:duplicateElimination="always"/>

    </tp:DeliveryChannel>
```

3.1.4.1. ebMS Once-And-Only-Once Sending Message Behavior

If an MSH is given data by an application needing to be sent reliably to the recipient, the MSH MUST do the following:

1. Create a message from components received from the application

    The message MUST have a globally unique MessageID.

1. Insert an AckRequested element

2. Save the message in persistent storage

3. Send the message to the recipient

4. Wait for the return of an Acknowledgment Message acknowledging receipt of this specific message and, if it does not arrive before RetryInterval has elapsed, the message must be resent until an acknowledgment is received or the NumberOfRetries has been reached. If a communications protocol error is encountered, then take appropriate error handling action.

Here is the sample of Reliable Messaging elements in ebMS message.

```
…

<eb:MessageHeader>

        …

        <eb:MessageData>

        <eb:MessageId>PartsOrder.323210:e5c74:7ffc@sender.com</eb:MessageId>

        <eb:Timestamp>2003-10-31T12:22:30</eb:Timestamp>

        <eb:TimeToLive>2003-11-01T12:22:30</eb:TimeToLive>

        </eb:MessageData>

</eb:MessageHeader>

<eb:AckRequested

        SOAP-ENV:mustUnderstand="1" eb:signed="true" eb:version="2.0"/>

<eb:DuplicateElimination/>

…
```

3.1.4.2. ebMS Once-And-Only-Once Receiving Message Behavior

If an AckRequested element is present in the received message then the receiver should generate an Acknowledgment Message is only performed when DuplicateElimination tag is present in the incoming message.

Here is a sample Acknowledgement message that is sent back to the party that sent the AckRequested.

```
<eb:Acknowledgment SOAP:mustUnderstand="1" eb:version="2.0">

<eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>

<eb:RefToMessageId>PartsOrder.323210:e5c74:7ffc@sender.com</eb:RefToMessageId>

<eb:From>

    <eb:PartyId>uri:www.example.com</eb:PartyId>

</eb:From>

</eb:Acknowledgment>
```

Chapter 4. Implementing ebMS Message-Level Security

Table of Contents

4.1. Implementing ebMS Message-Level Security

4.1.1. Digitally Signing a STAR ebMS Message

It is optional for a specific STAR ebMS message exchange to use Digital Signature, but if a Digital Signature is applied to a message the signature MUST be in full compliance with [XMLDSIG] and [ebMS version 2.0].

ebMS version 2.0 is very specific about how to apply Digital Signatures. Though multiple signatures are allowed, only the first signature is defined. The first signature is a signature over the SOAP Envelope (excluding the Signature elements themselves) and over all Attachments. ebMS requires specific algorithms for canonicalization and transformation of the SOAP Envelope. In other words, the sender creates a digital signature over the SOAP Envelope and all payloads.

A receiver MAY make use of ebXML CPA to associate a Digital Certificate with a sender.