Technical Overview

Version 3.1

September 19, 2008

Content

1 About this document

This document provides an introduction to the software architecture of Ponton X/P, including

- Software modules
- Interfaces
- Supported standards

For further information, please refer to:

www.ebxml.org

www.oasis-open.org

www.ponton-consulting.de (particularly the Products page and the Download area)

2 About XML Messaging…

Although XML is generally accepted as a standard for document markup, and as an interchange format between applications, it is not yet sufficiently developed to allow for message-based interoperability between arbitrary peers. To achieve this, the following conditions must apply:

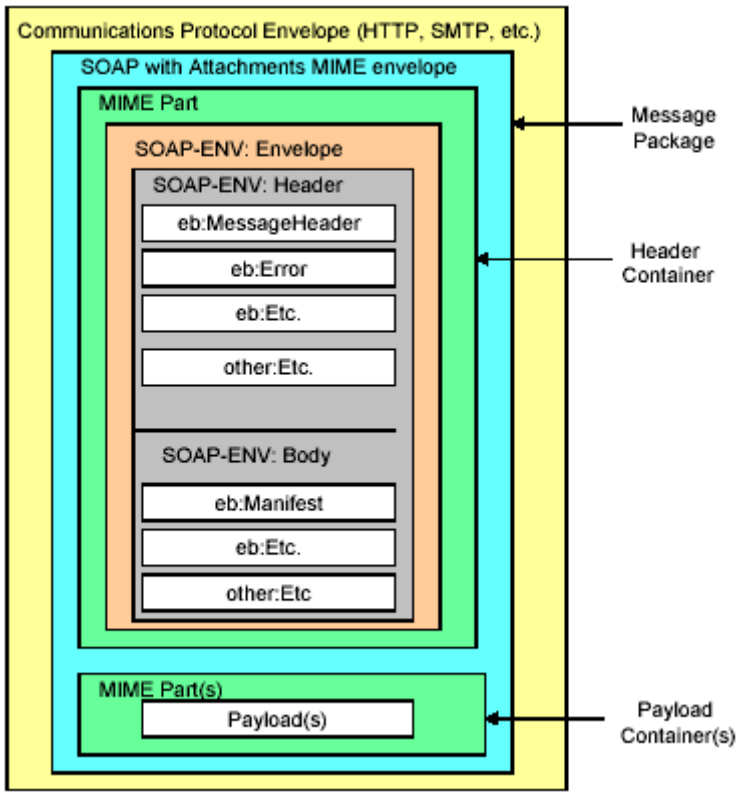
- The applications of two organizations must be able to properly process the documents exchanged. These may be purchase orders, invoices, customs declarations or insurance contracts, to name just a few examples. To achieve a common language here, business partners have to agree on a **commonly understood document type**. In XML, this is defined by the XML Schema. An example for a sector-specific standard format is papiNet (see www.papinet.org).
- It is one thing to agree on a document format, but another matter to ensure proper delivery of this document to a business partner. To achieve this, a common set of metadata elements has been defined under the ebXML standardization (www.ebxml.org). This metadata provides information on the communicating partners, their IDs, the protocol to be used for message transfer, whether the transfer is to be acknowledged, etc. This information is wrapped around the business document. For this reason, the business document is called the **payload** while the metadata is stored in the **envelope**.
- While interoperability requires exact maintenance of ebXML standard requirements between two Message Services, a high degree of flexibility is needed to connect the Message Service to the application software in the back-end of an organization. Here, the interface must be simple and easy to configure.

Ponton X/P implements the ebXML standard for Message Services. This defines the envelope structure and the semantics of its data elements and attributes. Further information on ebXML can found in the specification: http://www.oasis-open.org/committees/ebxml-msg.

To provide practical examples of ebXML Envelope data and business processes, we use papiNet documents and configurations as they are used within the papiNet project. Readers who have in mind to use Ponton X/P for different sectors may adjust the data values accordingly in their configuration of Ponton X/P.

2.1 What does an XML Message look like?

The XML Messages exchanged by Ponton X/P look like this:



The actual business document is stored in the Payload Container at the bottom. This is a separate MIME container. The ebXML envelope is embedded into a SOAP envelope. The main components of the ebXML data can be broken down into:

- The **MessageHeader**, containing basic information on sender, receiver, protocol, and transaction data.
- **Header extensions**, to carry error information or electronic signatures, for example.
- The **Manifest**, which serves as a kind of a "delivery note" in that it describes which application data is attached to the ebXML envelope. The application data includes the payload document, binary signatures, as well as any attachments such as image documents, etc.

To provide some insight into what this looks like in reality, here is an extract of an ebXML message:

```
SOAPAction: "ebXML"

Content-type: multipart/related; boundary="BoundarY"; type="text/xml";
start="<papiNet_envelope_example>"

--BoundarY
Content-ID: < papiNet_envelope_example >
```

```
Content-Type: text/xml

<?xml version = "1.0" encoding = "UTF-8"?>

<soap:Envelope xmlns:soap = "http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi =
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation =
"http://schemas.xmlsoap.org/soap/envelope/http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header
xmlns:eb = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
xsi:schemaLocation = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:MessageHeader eb:version = "2.0" soap:mustUnderstand = "1">
      <eb:From>
            <eb:PartyId eb:type="pn:MessageService">
            http://www.foo-inc.com:8080/papinet/HttpListener</eb:PartyId>
            <eb:PartyId eb:type="pn:OrganisationID">FooInc.NewYork[1]</PartyId>
      </eb:From>
      <eb:To>
            <eb:PartyId eb:type="pn:MessageService">
            http://www.bar-ltd.com:8080/papinet/HttpListener</eb:PartyId>
            <eb:PartyId eb:type="pn:OrganisationID">BarLtd.London[2]</PartyId>
      </eb:To>
      <eb:CPAId>FooInc.NewYork-BarLtd.London</eb:CPAId>
      <eb:ConversationId> FooInc.NewYork-BarLtd.London-PO-20020729-164423</eb:ConversationId>
      <eb:Service type="papinet"> Test</eb:Service>
      <eb:Action>PurchaseOrder</eb:Action>
      <eb:MessageData>
            <eb:MessageId>CERTCODE_HOSTNAME_TIMESTAMP</eb:MessageId>
            <eb:Timestamp>2002-05-14T14:51:00</eb:Timestamp>
      </eb:MessageData>
      <eb:DuplicateElimination/>
    </eb:MessageHeader>
  </soap:Header>

  <soap:Body
    xmlns:eb = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
    xsi:schemaLocation = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
    http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:Manifest eb:version = "2.0">
      <eb:Reference xmlns:xlink = "http://www.w3.org/1999/xlink"
xlink:href = "cid:papiNet_message_fragment.xml" xlink:role="http://regrep.org/gci/purchaseOrder">
        <eb:Schema eb:location="http://papinet.org/.../PurchaseOrder.xsd" eb:version="2.0"/>
```

[1] Check EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registration standards. Value must be globally unique.

```
    </eb:Reference>
    <eb:Reference xmlns:xlink = "http://www.w3.org/1999/xlink" xlink:href = "cid:damagePic01.jpg">
    </eb:Reference>
   </eb:Manifest>
  </soap:Body>
</soap:Envelope>
```

From this example, you can see that this message was sent from *FooInc* to *BarLtd*. Below the MessageHeader structure we find the Manifest embedded in the SOAP Body. The Manifest includes two references:

- To the payload XML document in the MIME container "**cid:papiNet_message_fragment.xml"**
- To a JPEG image, referenced by "**cid:damagePic01.jpg**"

Both of these documents are contained in the subsequent MIME containers of the message. First, the XML payload, which is a papiNet PurchaseOrder:

```
--BoundarY
Content-ID: <papiNet_message_fragment.xml>
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrder xmlns="http://www.papiNet.org/papiNet/PurchaseOrderV2R00"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
PurchaseOrderType="TrialOrder" PurchaseOrderStatusType="New">
  <PurchaseOrderHeader>
    …
  </PurchaseOrderHeader>
  <PurchaseOrderLineItem>
    …
  </PurchaseOrderLineItem>
  <PurchaseOrderSummary>
    …
  </PurchaseOrderSummary>
</PurchaseOrder>
--BoundarY--
```

Finally, the ebXML message contains the image file (encoded in base-64 format):

```
--BoundarY
Content-ID: <DamagePic01.jpg>
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

7VxLc+M2Er7rVzC8JxrPZFM5yE7Jkl1WrTXWWppJzd4QEpKwSxIqEHSs/PoAfONBCpJlT6rch/GI
```

```
6EYDaABfN9hojn57jiPvCbOU0OTSv/jpg+/hJKAhSTaX/pfV7Y+/+r9dDUY/TB8mq2+LG+8mecIR
3WFv+W25upl7/gRF0cN6/fXi8cOHn0Ie+leDwajiuhp43mjFUJLGJJVNzJI1ZTHi4qekadTJFjEU
cMxIykmQem3aglFOAxpd+ner1cJXaHMa4kv/Gqf8Zi3kc7+QXUlfYzabXi05E4MaDVtFFdcSJyFm
Xx5nNVNTUvE84gCTJ5WrXaa0WPZrRWK85CjeqY0b1EITwx5VWJT1wDYoIWmuy1mIE07WRMyjNqg2
E7c0dKrLiwLZ0zRFVUB6NOYcBVvZ7SlOA0Z2sm9NgMBKfVFNmaexj4SJvhqMhjfJE46ovKD+Nw==
--BoundarY--
```

2.2 Encrypting Documents

If payload encryption is turned on, the binary message data is encrypted using 3DES/RSA. The encrypted data is attached to the ebXML message as a PKCS #7 object.

2.3 Signing Documents

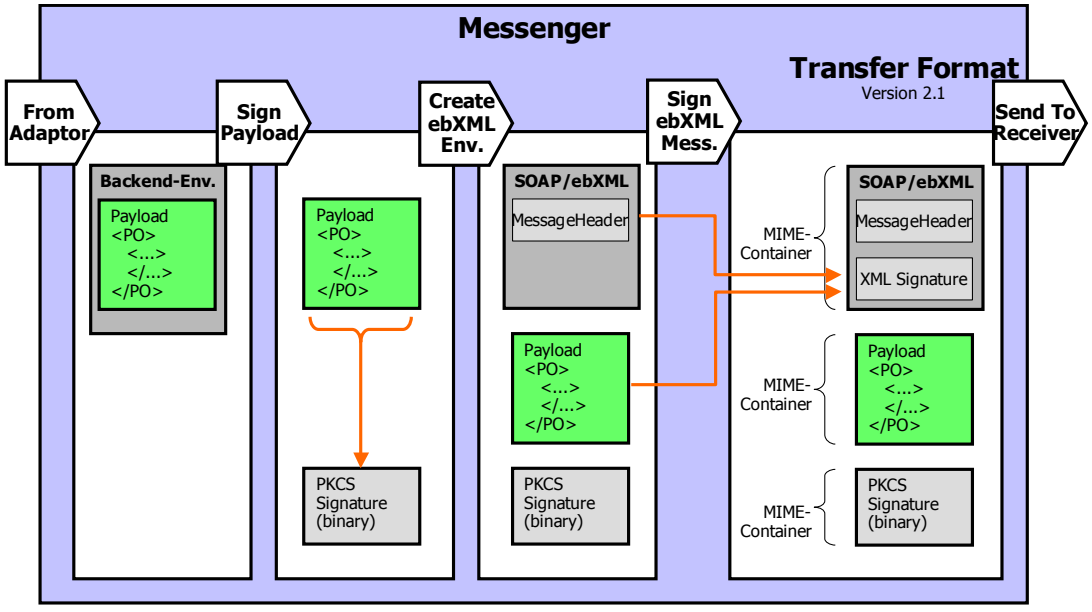
In Ponton X/P, documents are signed in three ways:

1. **XML Signature**. This technique allows selective signing of certain XML substructures, indicated in a list of signed parts. The resulting signature value and accompanying information is represented in XML and can therefore be embedded into the ebXML header. According to the ebXML standard, XML Signatures are applied to the payload data, the ebXML envelope and all attachments. This level of signing is only used between Message Services, i.e., the signature is created by the sending Messenger and verified by the receiver. If the message is consistent and authentic, the signature will be discarded. However, it is still available in the archive if the option for ebXML envelope archiving is activated.
2. **Payload Signing**. The Messenger signs the payload document (e.g. a papiNet PurchaseOrder) as a binary string and attaches the signature value as a further attachment to the ebXML message. This signature is represented as DER-encoded PKCS #1 block as defined in RSA Laboratory's *Public Key Cryptography Standards Note #1*. The data encrypted is the SHA1 digest of the data signed. The receiving Messenger's archiving filter can be configured to store the signature value together with the payload and the certificate in the file system. This way, message authenticity can be verified at any later time. This signature will be represented as a binary signature object and attached next to the payload (role="`http://www.papinet.org/ebXMLAttachment/Roles/BusinessDocumentSignature`"). In this case, the signature applies only to the original payload XML document. This signing method is also required to provide backward compatibility.
3. If SSL is activated, channel authentication applies at the IP level. This means that the sender and receiver can additionally verify that the opposite organization actually is who it purports to be. However, since channel authentication is transient, no signature data object will remain after the transmission is completed. For this reason, only the two signatures mentioned above will prevail as persistent data objects for later verification.

The following figure shows the processing steps done by the Messenger for signing ebXML messages. First, an RSA signature is created from the payload document. The document itself remains unchanged – it is treated as a binary object.

Then, the ebXML envelope is created and the relevant information is stored in its data elements. Both payload and signature are created as MIME attachments. The Manifest is created as a linking element between the ebXML envelope and the remaining MIME containers.

Finally, the XML Signature is applied to both the ebXML envelope and the payload. Should any of these be tampered with after signing, the receiver is able to detect this.



The benefit of the ebXML signature is that even after storing the payload and the envelope into the archive, both can be still be combined to verify the signature. This technique also prevents any re-arrangement of the link between the envelope and the payloads: even if the separate documents should be unchanged, the re-arrangement would be detected.

What exactly gets signed?

By definition, the XML Signature covers at least the Signature Elements plus any additional documents. The ebXML spec suggests that the ebXML Header should also be included in the XML Signature. By including all these parts in one signature, the parts are chained together to avoid possible fraud by replacing e.g. only the payload (example: receiver claims "I did not receive Invoice XYZ in Message ABC – it was only a delivery note").

For this reason, the XML Signature helps to verify full message integrity (Payload plus Envelope), but after the parts have been archived, the payload itself can only be authenticated by verifying the payload signature (both of which are archived).

If the back-end envelope is used, it is wrapped around the payload document and will be stripped-off by the adapter. Instead the payload part of the document will be signed and can therefore be verified by using both the sender's or the receiver's Message Monitor. The archived parts (ebXML envelope, payload, signature, certificate) need to be used for verification.

Since XML processing (importing into DOM format and serializing into XML again) affects the document due to canonicalization, a previous signature (that is not based on the XML Signature standard) would also not be valid any more. For this reason, the XML documents are not touched by the Messenger.

Representation of signed and encrypted documents

If binary signature objects are used as additional attachments, the representation will be as follows:

Signed	Encrypted	Format
No	No	Payload is attached as a MIME attachment in Plain text
Yes	No	Payload is attached as a MIME attachment in Plain text, a second attachment contains the signature data
No	Yes	Encrypted Payload is attached as a PKCS7 binary object. As such it can be deciphered by all applications (Microsoft & non-Microsoft)
Yes	Yes	Encrypted Payload is attached as a PKCS7 binary object, a second attachment contains the signature data

2.4 Synchronous vs. Asynchronous Message Transfer

In Ponton X/P, the distinction between synchronous and asynchronous transfer has only to do with how messages are processed by the adapter and the back-end business applications. The actual messaging process is always synchronous in the sense that the Messenger always generates an ebXML acknowledgement for every incoming message and always receives and an ebXML acknowledgement for every outgoing message. This is not actually synchronous in the literal sense of the term – it would be more correct to refer to this transfer mode as "asynchronous with acknowledgement". In the interest of more usable terminology, however, we refer to this as synchronous.

2.5 "Well Received" vs. "Well Processed"

It is very important to understand what exactly is meant when you receive a certain acknowledgement. We distinguish up to four levels of acknowledgement:

- The **Transport Level Acknowledgement** is some kind of feedback to the Messenger about the success or failure or a transmission. The meaning of this acknowledgement depends on the used transmission protocol. For example in case of SMTP an OK Acknowledgement only tells that the local SMTP server has accepted the email and will be sending it out soon.

  In general the Transport Level Acknowledgement does not indicate in any way that the receiver has actually received nor processed the message.

- The **Technical Acknowledgement** is exchanged between Messengers and confirms the proper receipt and of a message by the receiving messenger. This does not mean that the message was successfully submitted to the receiving application, nor does it mean that the message was successfully processed by the application! The technical acknowledgement is an ebXML acknowledgement that is received by the Messenger and then forwarded to the Adapter.

  A Technical Acknowledgement guarantees that a message was received and processed by the receivers Message Service Handler.

- The **Business Acknowledgement** is nothing but a payload. It is used in papiNet, for example, to confirm a "well received" status at the application level. The business acknowledgement indicates that the document was received by an ERP system and the data was processed successfully. It does *not* mean that the business process can be successfully continued (e.g. that a Purchase Order can be fulfilled). In case of the EFETnet business processes, all business documents (e.g., Trade Confirmations) are acknowledged by an EFET Acknowledgement, which is also a business document. The semantics, in this case, is that a document was received (i.e., technically acknowledged) and well-processed,such that it was possible to interpret the semantics of the document correctly. Again, in EFETnet terms, it does not mean that the Trade Confirmation will match the counterparty's Trade Confirmation.

- An **acknowledging document** within an ebXML conversation. In business practice, many transactions consist of related documents like Purchase Order and Order Confirmation, or Call-off and Call-off Confirmation, or a TradeConfirmation and the counterparty's TradeConfirmation. As in the case of the Business Acknowledgement, the Messenger is not aware of the semantics of these documents and their relationships.

3 Certificate Management

We distinguish two techniques to handle certificates:

- The lightweight CA is run by Ponton and helps simplify the process of key pair creation, requesting certificates, issuing certificates, and installing certificates for partners.
- Support for third-party certificates allows installation of certificates issued by CAs other than Ponton.

To sign a document, a private/public key pair is required. This can be created through the configuration interface of Ponton X/P. For further information on requesting a certificate, please refer to the *Installation and Configuration Guide*.

The Default CA

Ponton Consulting runs its own certification authority that is directly used by the Messenger. The rationale behind Ponton's lightweight CA is to simplify the certification process as far as possible. While there is usually an assorted number of certificate types, registration processes, and policies for issuing certificates, Ponton's CA does this in a very simple way:

1. The root certificate of the Ponton CA is pre-installed with the Messenger.
2. A Ponton certificate can be requested directly from the Messenger configuration as illustrated in the *Installation and Configuration Guide*.
3. After it has been issued, the certificate can be published – along with the relevant partner profile data – to a Partner Registry. Other partners can then download the required certificate and profile data from the registry in order to set up communication with a new partner.
4. Usually, certificates have a standard lifetime of 3 years, depending on the general policies agreed for the user community. The relevant party is warned about the approaching expiration date two weeks in advance.
5. The Messenger can be configured to download partner profiles from the registry at regular intervals. If this is not possible (e.g. due to firewall restrictions), the certificates will have to be installed manually via the administration interface.

The benefit of Ponton's lightweight certificate management is that kick-starting the integration between two business partners is made as simple as possible, thus helping to overcome a typical hindrance factor for the roll-out of authenticated document exchange across organization boundaries.

Third Party CAs

Users of Ponton X/P may also request third party certificates from any provider. The only requirement is that the certificate complies with the X.509 standard – which is the case for all providers at this time.

Note: When third-party certificates are used, it is advisable to disable the automatic download option.

4 Modules of Ponton X/P

Ponton X/P consists of the following main modules:

1. **Messenger**. This is the core of Ponton X/P. It transforms Messages received from the back-end (user or ERP system) into a standards conforming ebXML message. Several processing steps are performed before the ebXML message is sent to the receiver.
2. **Listener**. There is a Listener integrated in the Messenger, so it is not necessary to run this as a separate module. In case of security requirements, however, it is also possible to set up the Listener in the DMZ to receive messages from the outside and to forward them to the Messenger within the secure zone of an organization. No further processing takes place by the Listener. For details on how to set up different Listener modes, please refer to the Installation and Configuration Guide.
3. **Adapters**. There are many ways to integrate the Messenger with the application software in the back-end. An Adapter helps bridge this gap. To name some of the adapters, Ponton X/D is a Database Adapter that maps XML payload content directly to and from a database. The Hot Folder Adapter frequently scans outbox folders and transfers these documents to the Messenger. Vice-versa, messages received from a business partner will be dropped into an inbox.

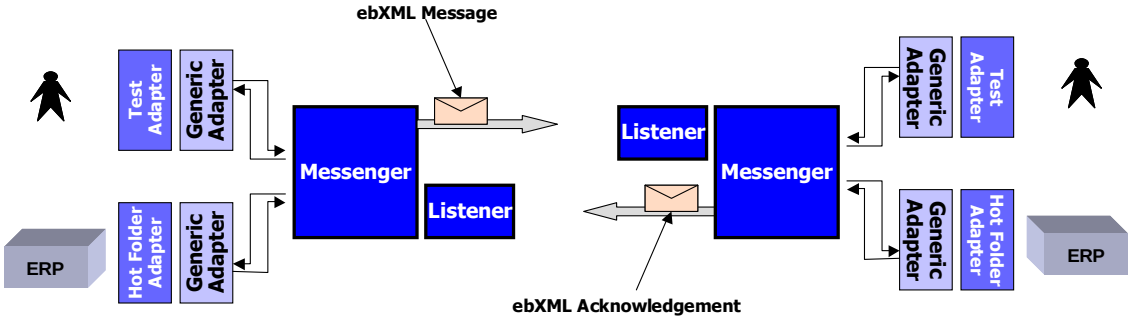


Fig. 1: Message processing with Ponton X/P

4.1 The Messenger

The Messenger consists of a set of pipelines. These are assemblies of filters, which can be configured per pipeline. This configuration determines how documents are processed. The pipeline to be used is set for each partner through the configuration interface.

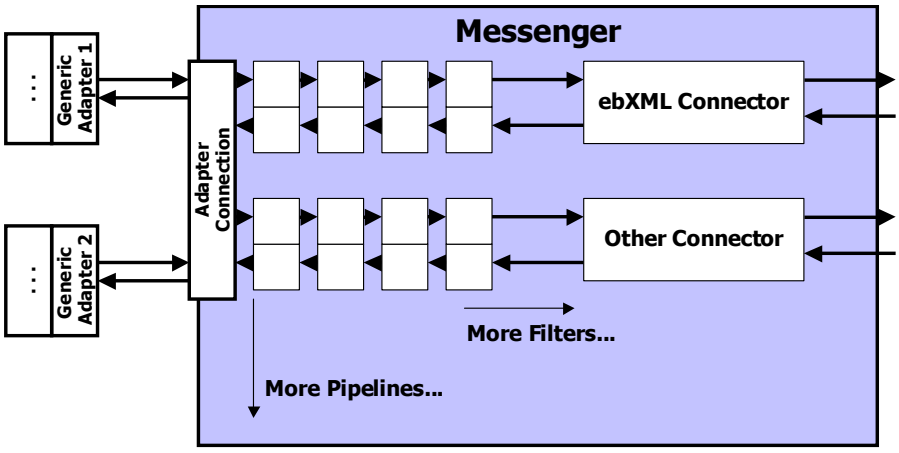


Fig. 2: Definition of pipelines and filters for message processing

Examples for filters are:

- Compression
- Encryption
- Signing
- Archiving
- Routing Rules

Developers may extend the Messenger with additional filters, e.g., a business rule validator, an extended archiving filter, etc.

The *availability* of a filter is configured at the level of the pipeline configuration. *Activation* of a filter, however, can be defined for each individual partner.

The Messenger can also be configured for additional connectors apart from ebXML. To date, the protocols AS1 and AS2 have been implemented (Application Statement 1 & 2 as defined by rfc3335 and rfc4130 of the IETF, http://www.ietf.org/).

4.2 Adapters

The Adapter interface is used to implement the integration between the Messenger and the back-end (business applications). The Messenger supports various types of adapters, as illustrated by the following examples.

- **Test Adapter** – The Test Adapter is provided for test purposes, allowing you to test the Adapter/Messenger connection.
- **Hot Folder Adapter** – The Hot Folder Adapter uses file storage to exchange data with business applications. The adapter allows you to specify folders for incoming and outgoing messages.
- **HTTP Adapter** – The HTTP Adapter provides a means of connecting to the Adapter interface via HTTP, allowing web based access.

- **Database Adapter** – also known as Ponton X/D. This is a separate product, not included in Ponton X/P. Based on a flexible mapping mechanism between databases and XML files, Ponton X/D allows you to integrate the Messenger with your business applications – quickly and effectively.

4.3 Logging and the Log Database

Ponton X/P provides two separates logs.

- Message related processing log:

  This log is written to a database table. It includes only information concerning the success or failure of message processing steps.

  This log is not configurable, so it is not possible to change the logging level or the type of entries stored in the log database.

  There is a cleanup functionality built into the Messenger that removes message related logs from the database whenever message references are removed. This can be done manually or automatically using the "Maximum Age" setting.

- Detailed technical log:

  The technical log is written to log files. The number of log entries can be configured using the log levels. Please note that the "DEBUG" log level reduces performance noticeably and should only be used for evaluation and error tracking purposes.

5 ebXML Header Details

The ebXML envelope only exists between two instances of Ponton X/P. As illustrated in the example further down, the envelope uses both SOAP and ebXML Schemata (namespaces are prefixed with "soap" and "eb").

5.1 Example for ebXML Envelope

Please note that the ebXML Header and the ebXML Payload are shown in plaintext here. In real transmissions they should be base64 encoded to avoid any problems with text encoding and signature issues.

```
SOAPAction: "ebXML"

Content-type: multipart/related; boundary="BoundarY"; type="text/xml";
start="<papiNet_envelope_example>"

--BoundarY
Content-ID: < papiNet_envelope_example >
Content-Type: text/xml

<?xml version = "1.0" encoding = "UTF-8"?>

<soap:Envelope xmlns:soap = "http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi =
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation =
"http://schemas.xmlsoap.org/soap/envelope/http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header
xmlns:eb = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
xsi:schemaLocation = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:MessageHeader eb:version = "2.0" soap:mustUnderstand = "1">
      <eb:From>
<eb:PartyId eb:type="DunsNumber">                                    <!--dummy value -->
     45678901234</eb:PartyId>                                        <!--dummy value -->
      </eb:From>
      <eb:To>
<eb:PartyId eb:type="DunsNumber">                                    <!--dummy value -->
    234567890</eb:PartyId>                                           <!--dummy value -->
      </eb:To>
      <eb:CPAId>SCA.Fine-Springer.Ahrensburg</eb:CPAId>
      <eb:ConversationId>SCA.Fine-Springer.Ahrensburg-PO-20020729-164423</eb:ConversationId>
      <eb:Service type="papinet"> Test</eb:Service>
      <eb:Action>PurchaseOrder</eb:Action>
      <eb:MessageData>
```

```
        <eb:MessageId>NO_12345@sca.com</eb:MessageId>
       <!--from TransferID if exists plus domain name of Sender Messenger -->
        <eb:Timestamp>2002-05-14T14:51:00</eb:Timestamp>
      </eb:MessageData>
      <eb:DuplicateElimination/>
    </eb:MessageHeader>
  </soap:Header>

  <soap:Body
     xmlns:eb = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
     xsi:schemaLocation = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
     http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:Manifest eb:version = "2.0">
      <eb:Reference xmlns:xlink = "http://www.w3.org/1999/xlink"
xlink:href = "cid:papiNet_message_fragment.xml" xlink:role="http://regrep.org/gci/purchaseOrder">
       <!--Dummy value -->
        <eb:Schema eb:location="http://papinet.org/.../PurchaseOrder.xsd" eb:version="2.0"/>
      </eb:Reference>
      <eb:Reference xmlns:xlink = "http://www.w3.org/1999/xlink" xlink:href = "cid:damagePic01.jpg">
      </eb:Reference>
    </eb:Manifest>
  </soap:Body>
</soap:Envelope>
--BoundarY--

--BoundarY
Content-ID: <papiNet_message_fragment.xml>
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrder xmlns="http://www.papiNet.org/papiNet/PurchaseOrderV2R00"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
PurchaseOrderType="TrialOrder" PurchaseOrderStatusType="New">
  <PurchaseOrderHeader>
    ...
  </PurchaseOrderHeader>
  <PurchaseOrderLineItem>
    ...
  </PurchaseOrderLineItem>
  <PurchaseOrderSummary>
    ...
  </PurchaseOrderSummary>
</PurchaseOrder>
--BoundarY--

--BoundarY
Content-ID: <DamagePic01.jpg>
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
```

```
Osdrhfgu hdagiuhdg iushdrguihstig hsdthubs dt
Ofthouizthbioutho uistfrhb s rthjitfrho srjth
Stj hosrjth osrjtgordsthjo isrtho idrjth idrt
Osdrhfgu hdagiuhdg iushdrguihstig hsdthubs dt
Ofthouizthbioutho uistfrhb s rthjitfrho srjth
Stj hosrjth osrjtgordsthjo isrtho idrjth idrt
Osdrhfgu hdagiuhdg iushdrguihstig hsdthubs dt
Ofthouizthbioutho uistfrhb s rthjitfrho srjth
Stj hosrjth osrjtgordsthjo isrtho idrjth idrt
Osdrhfgu hdagiuhdg iushdrguihstig hsdthubs dt
Ofthouizthbioutho uistfrhb s rthjitfrho srjth
Stj hosrjth osrjtgordsthjo isrtho idrjth idrt
Osdrhfgu hdagiuhdg iushdrguihstig hsdthubs dt
Ofthouizthbioutho uistfrhb s rthjitfrho srjth
Stj hosrjth osrjtgordsthjo isrtho idrjth idrt
--BoundarY--
```

5.2 ebXML MessageHeader

- XML Schema
- Example document
- Meaning of elements and attributes

→ MessageID comes from TransferID in Backend Envelope

The format should be <localUniqueID> @ <local Domain Name of Sending Message Servcie>

e.g.: No_12345678@company.com

If it is not supplied in the Backend Envelope, one will be generated by the Messenger. It is composed of "MID-" + current timestamp + domain of the Messenger email from address.

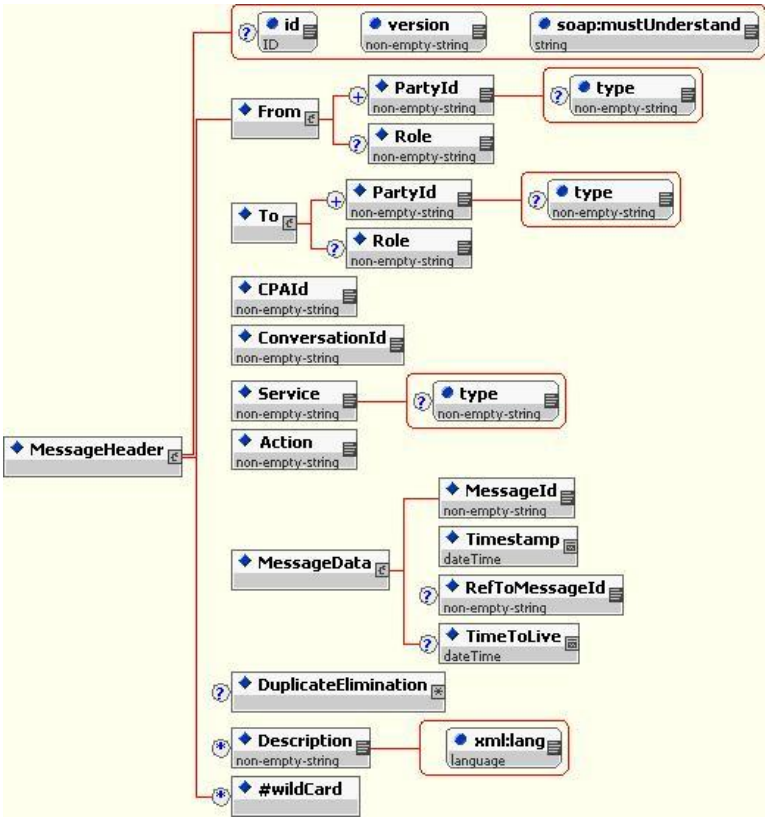


Fig. 3: Schema for ebXML Message Header

Table 1: Settings for Ponton X/P for ebXML Message Headers

Element/Attr	Required for papiNet	Comment
Id	N	
Version	Y	Fixed to: '2.0'
SoapMustUnderstand	Y	Fixed to '1'
PartyID	Y	See further down in text
Type	Y	See further down in text
Role	N	
CPAId	Y	Fixed value, not interpreted, See further down in text
ConversationID	Y	See further down in text
Service	Y	Either "Production" or "Test"
Action	Y	Equals document type
MessageID	Y	Equals TransferID
TimeStamp	Y	Will be local time stamp of sending Messenger's computer
RefToMessageId	N	

Element/Attr	Required for papiNet	Comment
TimeToLive	Y	This Element is optional as defined by ebXML, but any Messaging Service has to be able to deal with it. For the interoperability guide it is suggested that this element should be at least "recommended" to use, since it guarantees a more reliable messaging. The messenger will always set this Element.
DuplicateElimination	Y	Always
Description	N	
Wildcard	N	

Key Information in the MessageHeader

- PartyID Element
- CPAID Element
- Service Element
- Action Element
- ConversationID Element

CPAID – What is it?

Following the ebXML CPPA standard, a CPA (collaboration partner agreement) holds information on the technical contract points, protocols, communication modes etc. to be used when the partners exchange messages to transfer documents of a certain type.

Example: Two partners may agree on the following (in prose):

> "If partner "Cust" sends a purchase order (Action: PO) to partner "Supp" this will be compressed and based on schema format *papiNet PurchaseOrderR2V10* . "Supp" may send an order confirmation (Action: OC). This has to be effected within 48 hours after receiving a purchase order. The order confirmation is signed and compressed. Both actions PO and OC are combined under the service Ordering. Supp may send an invoice of type papiNet InvoiceV2R10. It has to be signed, compressed and encrypted." Etc. etc.

Actually, a CPA corresponds to the partner configurations entered by business partners on both sides of the relationship.

However, CPAs are **not supported** by Ponton X/P, for this reason, a dummy value is used here.

Service and Action Elements

An <Action> is an elementary ebXML transaction, i.e., the transfer of a document (e.g. a purchase order or an order confirmation). What is actually called an <Action> depends on those who standardise the vocabulary, e.g., for sector-specific standards like papiNet. The semantics of <Actions> values is therefore out of scope for a Message Service.

A <**Service**> denotes a combination of <Actions>. The classic example is the pair of a purchase order and the corresponding order confirmation. However, one may also bundle dozens of related document types under one service – or even only one single <Action>. This, again, depends on the decision of the standardisers and is therefore out of scope as well for the Message Service.

The ebXML specification defines <Service> as an URI-based reference to a Web Service.

Example:

```
…
<eb:Service eb:type="BankingStandard">AccountManagement</eb:Service>
<eb:Action>Withdrawal</eb:Action>
…
```

ConversationID

This is end-to-end information since it is exchanged between the back-end application of the involved partners. Standardisers may restrict the usage of a ConversationID, e.g., by requiring that a conversation corresponds 1:1 to a <Service> as defined above.

The Conversation is taken from the Backend Envelope if one is defined. If there is none defined, the Messenger will create a new one using the following scheme:

"CID-" + current timestamp + domain of the Messenger email from address.

Ex. "CID-1052483642995@ponton-consulting.de"

Actor type

No Actor types are defined within Ponton X/P.

Roles

No roles types are defined within Ponton X/P.

5.3 ebXML Manifest Format

A different kind of Schema Reference can be found in the **Manifest**: Here the Reference element uses a /Schema/@location attribute to refer to the Schema (or DTD) used for the payload attachment.

```
<eb:Reference xmlns:xlink = "http://www.w3.org/1999/xlink"
    xlink:href = "cid:papiNet_message_fragment.xml">
    <eb:Schema eb:location="http://papinet.org/.../PurchaseOrder.xsd" eb:version="2.0"/>
</eb:Reference>
```

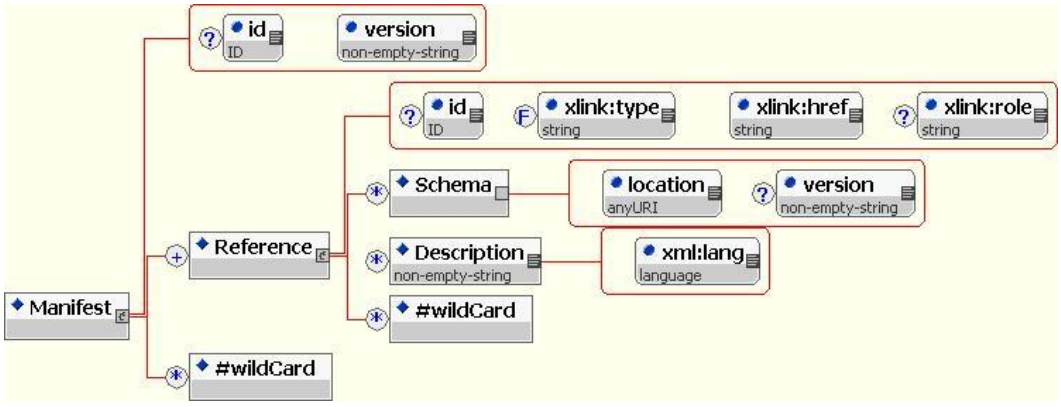


Fig. 4: Schema for ebXML Manifest

Table 2: Settings for Ponton X/P for papiNet ebXML Manifests

Element/Attr	Required for papiNet	Comment
Id	N	
Version	Y	Fixed to '2.0'
Reference/id	N	
type	N	Type of the href xlink, fixed to "simple"
Href	Y	Is a MIME container ID (referring to the attachment)
role	Y	Either "http://papinet.org/papiNet/ebXMLAttachment/Roles/BusinessDocument", or "http://papinet.org/papiNet/ebXMLAttachment/Roles/BusinessDocumentSignature", or the attribute is left out. The role attribute is more application-oriented and characterises the attachment itself: - "http://papinet.org/papiNet/ebXMLAttachment/Roles/BusinessDocument" – this is the papiNet document - http://papinet.org/papiNet/ebXMLAttachment/Roles/BusinessDocumentSignature"
Location	Y	Is always URI to papiNet Schema location, e.g. : It will have to follow the papiNet naming schema for XML Schemata, e.g.,

		http://www.papinet.org/Schema/V2R0/PurchaseOrder XXX This will be updated according to the final version of the Interoperability Guidelines (→ Charleston meeting)
Version	Y	Following the example above, this is "2.0"
Description	Optional	Just informational
Wildcard	N	

Since reference roles and schema locations are application-specific data values, this example is papiNet-specific. The naming of reference roles or schema locations is a matter of project-specific standardization .

In the case of the EFET 3.1 standard, for example, the following values are used:
Schema location: http://www.efet.org/Schemas/V3R0/TradeConfirmations.xsd .
Reference roles: http://www.efet.org/ebXMLAttachment/Roles/BusinessDocument
http://www.efet.org/ebXMLAttachment/Roles/BusinessDocumentSignature

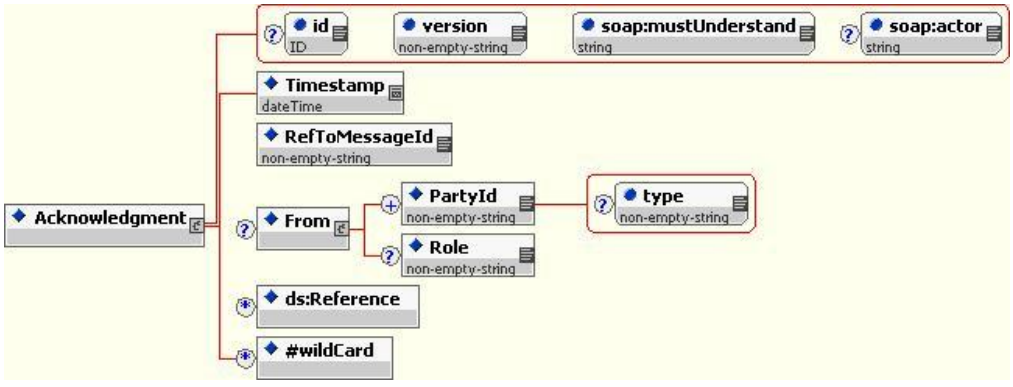
Schema locations and reference roles can be configured as required by a given integration project.

5.4 ebXML Reliable Messaging

Reliable Messaging means that each transfer of a payload document is acknowledged by a dedicated XML structure, as specified by the ebXML Message Service standard.

ebXML Acknowledgements

The ebXML specification defines the following structure for Acknowledgements:



An Acknowledgement structure will be used in addition to the ebXML MessageHeader as a SOAP Extension Element.

For the papiNet and EFETnet projects, for example, the following restrictions apply:

Table 3: Settings for Ponton X/P for Acknowledgements

Element/Attr	Required for papiNet / EFETnet	Comment
Id	N	
Version	Y	'2.0 '
MustUnderstand	Y	Must be '1'
Actor	N	
TimeStamp	Y	Timestamp of computer where Messenger is running
RefToMessageID	Y	MessageID of acknowledged message
From	N	Since the From structure is already present in the MessageHeader, it can be dropped for Acknowledgement.
Reference	N	
Wildcard	N	

Here, we are talking about the **technical acknowledgement** that is transferred between the two messengers (the Business Acknowledgement is treated as a payload).

The Messenger will **always** request an Ack from the receiver. If this is not transmitted, an error will be raised. An Acknowledgment request is expressed by the following SOAPHeader extension:

```
<eb:AckRequested SOAP:mustUnderstand="1" eb:version="2.0"
eb:signed="false"/>
```

From the partner configuration the Messenger can obtain information if the Ack is to be signed or not. If so, the attribute **eb:signed** has to be set to "true".

If the opposite side (in case of third party software) does not request an Ack, the Messenger will not raise an error if communication is asynchronous.

Errors are reported back from the receiver of the business document to the sender by embedding an ebXML Error List right after the Acknowledgement module (see example further down).

If there are no errors in the message received and an *Acknowledgment Message* is being sent on its own, not as a message containing payload data, then the *Service* and *Action* must be set as follows:

- The Service element must be set to: urn:oasis:names:tc:ebxml-msg:service
- The Action element must be set to: Acknowledgment

The ebXML Acknowledgement will look like this:

> SOAPAction: "ebXML"
> Content-type: multipart/related; boundary="ebXMLBoundary"

```
--ebXMLBoundary
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
      xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
      http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
  <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
      xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
      http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:MessageHeader version="2.0" SOAP:mustUnderstand="1"
      xmlns=eb:"http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
      xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
      http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
      <eb:From>
            <eb:PartyId eb:type="pn:MessageService">                          <!--dummy value -->
            http://papinet.asv.de:80/papinet/HttpListener</eb:PartyId>        <!--dummy value -->
            <eb:PartyId eb:type="pn:OrganisationID">Springer.Ahrensburg3</PartyId>    <!--dummy value -->
      </eb:From>
      <eb:To>
            <eb:PartyId eb:type="pn:MessageService">                          <!--dummy value -->
            http://papinet.sca.com.de:80/papinet/HttpListener</eb:PartyId>    <!--dummy value -->
            <eb:PartyId eb:type="pn:OrganisationID">SCA.Fine4</PartyId>       <!--dummy value -->
      </eb:To>
      <eb:CPAId>SCA.Fine-Springer.Ahrensburg</eb:CPAId>
      <eb:ConversationId>SCA.Fine-Springer.Ahrensburg-PO-20020729-434343434</eb:ConversationId>
      <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
      <eb:Action>Acknowledgement</eb:Action>
      <eb:MessageData>
      <eb:MessageId>20010215-111212-28572@example.com</eb:MessageId>
      <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
    </eb:MessageData>
  </eb:MessageHeader>
  <eb:Acknowledgment SOAP:mustUnderstand="1" eb:version="2.0">
     <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>
     <eb:RefToMessageId>323210:e52151ec74:7ffc@xtacy</eb:RefToMessageId>
  </eb:Acknowledgment>
```

[3] Check EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registration standards. Value must be globally unique.

[4] Check EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registration standards. Value must be globally unique.

```
<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-US">Validation of signature failed<eb:Description>
  </eb:Error>
  <eb:Error eb:errorCode="CertificateNotFound" eb:severity="Error"
        eb:location="URI_of_ds:Signature">
    <eb:Description xml:lang="en-US">Certificate not found<eb:Description>
  </eb:Error>
  <eb:Error ...> ... </eb:Error>
 </eb:ErrorList>

 </SOAP:Header>
 <SOAP:Body/>
</SOAP:Envelope>
--ebXMLBoundary—
```

An acknowledgement is always requested by the following ebXML structure:

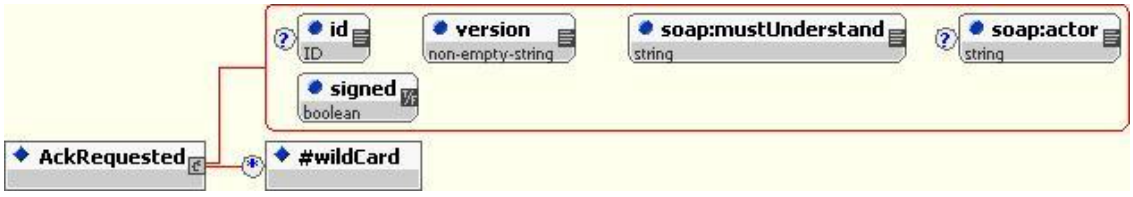


Table 4: Settings by Ponton X/P for AckRequested

Element/Attr	Required for papiNet / EFETnet	Comment
Id	N	
Version	Y	'2.0'
MustUnderstand	Y	Must be '1'
Actor	N	
Signed	Y	This is set to 'N', i.e., signed acknowledgements are not *required*. However, if the acknowledgment is signed, this is not considered as an error. The signature will be verified. If a verification error occurs, this will raise an error.
wildCard	N	

5.5 Embedding Electronic Signatures

There are two possibilities to embed signatures into a Ponton X/P message:

1. If users prefer to exchange signature and certificate data as a binary data object, the signature will be attached as a secondary MIME attachment next to the payload

content. In this case, signatures are represented as DER-encoded PKCS #1 block as defined in RSA Laboratory's *Public Key Cryptography Standards Note #1*.

2. If users prefer to represent the signature in XML Signature format, this is embedded into the ebXML Envelope.

Signatures as a MIME attachment

Example:

```
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
--ebXMLBoundary
sCFmwSP5slksK/1/yckDdFwZK+mfGJb4i8Zb4vjhzCiUfoUhDFTlA8ZJRNw1Ta/rTxFAPwNyvdE3
YiPPhiW7Xd4HkBjYB3puutG3SmCssdEG341mVH2cKmZK+Apl9S5mmoPyVX4MTFMbELyvBSg2zjsf
fpPKE17UoVAZeHwW2Yg=
--ebXMLBoundary
```

Signatures in XML Signature format

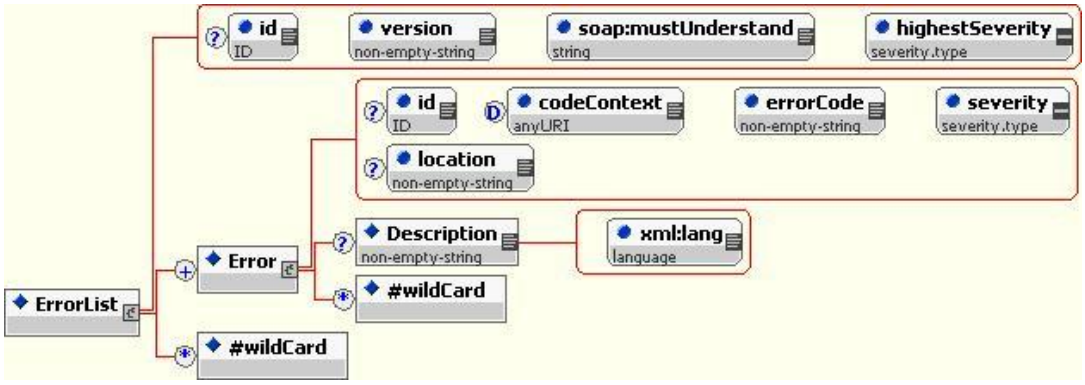
Example:

```
SOAPAction: "ebXML"
Content-type: multipart/related; boundary="ebXMLBoundary"

--ebXMLBoundary
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
  <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">

    <eb:MessageHeader ...>
        ... other stuff here ...
    </eb:MessageHeader>

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    ... (compare current solution)
      <ds:KeyInfo>
      ...
        <ds:X509Data>
          <ds:X509IssuerSerial>
```

```
            <ds:X509IssuerName>
              CN=Michael\20Merz,OU=Ponton\20Consulting,O=Ponton,L=Hamburg,ST=DE,C=DE
            </ds:X509IssuerName>
            <ds:X509SerialNumber>1030532663</ds:X509SerialNumber>
          </ds:X509IssuerSerial>
          <ds:X509SubjectName>
            CN=Michael\20Merz,OU=Ponton\20Consulting,O=Ponton,L=Hamburg,ST=DE,C=DE
          </ds:X509SubjectName>
          <ds:X509Certificate>
MIIDFTCCAtMCBD1srjcwCwYHKoZIzjgEAwUAMHAxCzAJBgNVBAYTAkRFMQswCQYDVQQIEwJERTEQ
MA4GA1UEBxMHSGFtYnVyZzEPMA0GA1UEChMGUG9udG9uMRowGAYDVQQLExFQb250b24gQ29uc3Vs
       ...
          </ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </ds:Signature>

</SOAP:Header>
 <SOAP:Body/>
</SOAP:Envelope>
--ebXMLBoundary—
```

5.6 EbXML Error Codes

The error list will be embedded into the SOAP Header as an extension following the ebXML specification. The schema is as follows:



For papiNet, the following subset applies:

Table 5: Settings for Ponton X/P for ebXML Error Codes

Element/Attr	Required for papiNet / EFETnet	Comment
Id	N	

Version	Y	Fixed to: '2.0'
MustUnderstand	Y	Fixed to '1'
HighestSeverity	Y	
Error/id	N	
CodeContext	Y	Either *'urn:oasis:names:tc:ebxml-msg:service:errors'* or *'papinet'*
ErrorCode	Y	For content see further down
Location	Y	If referring to a document / element within payload, e.g., when a parsing error occurred
Description	Optional	
Lang	Y	Always 'EN'
WildCard	N	

Not all ErrorCode of the Messenger can be mapped to the standard ebXML messages (which also applies to 3rd party Message Services). For this reason, we compiled a list of the superset of error codes from both worlds ebXML and Ponton X/P. Here, all ebXML error messages are qualified by a dedicated ebXML error code context. The same applies to Messenger-specific codes. They may also be combined in an <ErrorList>.

ebXML Error Codes	Severity	Messenger Error Codes	Location	Code Context	Comments
Inconsistent	Error	--	[XML Element]	EB	Applies to CPAID, PartyID/PartyIDType, ServiceType,
ValueNotRecognized	Error	--	[XML Element]	EB	Applies to PartyID, CPAID, ActionType
TimeToLiveExpired	Error	--	"Remote Messenger"	EB	
MimeProblem	Error	--	"Remote Messenger"	EB	Mime attachment could not be accessed
NotSupported	Error	--	Depends	EB	An ebXML feature that is not supported by the Messenger
OtherXml	Error	--	[XML Element]	EB	May be provided by the back/end application
DeliveryFalilure	Error / Warning	"200", "Message was rejected by receiver!"	"Remote Messenger" or "Remote Listener"	EB	Reported by receiving Messenger to sender If Error: Message cannot be delivered to Adapter If Warning: Unknown if Message can be delivered (e.g. because of connection timeout to Adapter).
SecurityFailure	Error	"224", "ERROR while decrypting message." "225", "Could not verify signature."	"Remote Messenger"	EB	Signature could not be verified or message could not be decrypted

ebXML Error Codes	Severity	Messenger Error Codes	Location	Code Context	Comments
Unknown	Error	"221", "Could not validate the message"	"Remote Messenger"	PX	XML Validation problem. Schema not found or message is not valid
		"210", "Could not unzip the message."		PX	
	Error	"212", "Could not save the message to file"		PX	
	Error	"221", "Could not archive the message"		PX	
	Error	"216", "Error while sending notification to the adapter."		PX	

Error Code Contexts

EB: urn:oasis:names:tc:ebxml-msg:service:errors.

PX: urn:ponton:pontonxp:errors

If applicable, an Error element contains a "location" attribute that points to the error-prone XML element in the ebXML header or in the payload.

In this case, the attribute value will be an XPointer to that location.

Automatic Certificate Renewal

To minimise the time gap between expiration of the old certificate and availability of the new one, certificates can be provided by the sender as an extension of the ebXML envelope.

If there is a certificate present, the receiver will validate it (against the locally stored root certificate of the CA) and then validate the sender's signature. The certificate will be stored on disk if it is newer than the previously stored one.

If there is no certificate received, the receiver will look-up the certificate using the local partners.xml entry for the sender. If the corresponding certificate can be found, this will be used. These certificates are either frequently downloaded from the Ponton certificate server or manually installed in case of 3rd party certificates.

If a certificate is renewed on the receiver side and a sender uses the now invalid old certificate (for encryption of the session key) this will result into a verification error.

This error will be reported by the receiving Messenger to the sender. As an attachment to this report, the sender receives the new certificate and is able to update the certificate in the local store.

This only works if the new certificate has been issued by the same (or another well-known) CA. If this is the case, the certificate will be stored and can be used for the next message transfer. However, if the issuing CA is new, the sender first has to manually install the root certificate of that CA.

5.7 Exchanging Test Messages

Please also check the Generic Adapter API documentation to see how to issue a Ping request and how to process a Pong result.

Ping & Pong Messages

This feature is considered important to facilitate connection set-up between two communication partners using Message Services from different vendors. The Ping Message has the following structure:

```
SOAPAction: "ebXML"
Content-type: multipart/related; boundary="ebXMLBoundary"

--ebXMLBoundary
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
    http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
 <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
    xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
    http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
  <eb:MessageHeader version="2.0" SOAP:mustUnderstand="1"
    xmlns=eb:"http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
    xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
    http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
   <eb:From>
        <eb:PartyId eb:type="pn:MessageService">                    <!--dummy value -->
        http://papinet.asv.de:80/papinet/HttpListener</eb:PartyId>  <!--dummy value -->
        <eb:PartyId eb:type="pn:OrganisationID">Springer.Ahrensburg[5]</PartyId> <!--dummy value -->
   </eb:From>
```

[5] Check EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registration standards. Value must be globally unique.

```
    <eb:To>
            <eb:PartyId eb:type="pn:MessageService">                    <!--dummy value -->
            http://papinet.sca.com.de:80/papinet/HttpListener</eb:PartyId>    <!--dummy value -->
            <eb:PartyId eb:type="pn:OrganisationID">SCA.Fine[6]</PartyId>    <!--dummy value -->
    </eb:To>
    <eb:CPAId>SCA.Fine-Springer.Ahrensburg</eb:CPAId>
    <eb:ConversationId>SCA.Fine-Springer.Ahrensburg-PO-20020729-12345678</eb:ConversationId>
    <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
    <eb:Action>Ping</eb:Action>
    <eb:MessageData>
    <eb:MessageId>20010215-111212-28572@example.com</eb:MessageId>
    <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
   </eb:MessageData>
  </eb:MessageHeader>
 </SOAP:Header>
 <SOAP:Body/>
</SOAP:Envelope>
--ebXMLBoundary—
```

The Messenger will send this message using the settings that were defined for the communication with the receiver. Data will be logged in the Log DB as usual. There will be no payload used.

The receiving Messenger logs reception and replies the following message:

```
SOAPAction: "ebXML"
Content-type: multipart/related; boundary="ebXMLBoundary"

--ebXMLBoundary
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
      xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
      http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
  <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
      xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
      http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:MessageHeader version="2.0" SOAP:mustUnderstand="1"
      xmlns=eb:"http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
      xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
      http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
      <eb:From>
```

[6] Check EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registration standards. Value must be globally unique.

```
            <eb:PartyId eb:type="pn:MessageService">                    <!--dummy value -->
            http://papinet.asv.de:80/papinet/HttpListener</eb:PartyId>  <!--dummy value -->
            <eb:PartyId eb:type="pn:OrganisationID">Springer.Ahrensburg[7]</PartyId> <!--dummy value -->
    </eb:From>
    <eb:To>
            <eb:PartyId eb:type="pn:MessageService">                    <!—dummy value -->
            http://papinet.sca.com.de:80/papinet/HttpListener</eb:PartyId>  <!--dummy value -->
            <eb:PartyId eb:type="pn:OrganisationID">SCA.Fine[8]</PartyId>   <!--dummy value -->
    </eb:To>
    <eb:CPAId>SCA.Fine-Springer.Ahrensburg</eb:CPAId>
    <eb:ConversationId>SCA.Fine-Springer.Ahrensburg-PO-20020729-87654321</eb:ConversationId>
    <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
    <eb:Action>Pong</eb:Action>
    <eb:MessageData>
    <eb:MessageId>20010215-111212-28572@example.com</eb:MessageId>
    <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
   </eb:MessageData>
  </eb:MessageHeader>
 </SOAP:Header>
 <SOAP:Body/>
</SOAP:Envelope>
--ebXMLBoundary—
```

5.8 Unsupported ebXML Features

The following ebXML feature is not supported by Ponton X/P:

- ebXML Multi-hop support

[7] Check EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registration standards. Value must be globally unique.

[8] Check EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registration standards. Value must be globally unique.

6 Mapping between Envelope Formats

This section addresses users of older versions of Ponton X/P within the papiNet project. Before release 2.1 of Ponton X/P, a so-called "back-end envelope" was used to wrap routing information around the actual payload XML section.

Mapping of party IDs

The ERP systems used by different parties may have different IDs for business partners, which may also differ from the IDs assigned papiNet-wide. For this reason the party IDs are mapped as described in the following example:

In this example, three different codes are distinguished to identify the same party:

1. Local partner code in the sender's configuration to identify the receiver (for example, "BURDA" to identify Burda on UPM Kymmene's side). This code is known to the sender's ERP system.
2. Global partner code, which is a unique identifier within papiNet or worldwide, for example a DUNS number: "987654321".
3. Local partner code for the receiver in the receiver's configuration, for example: "Burda/PP/YY/ZZ".

These examples are arbitrarily chosen, of course.

The Payload document has only a DOCTYPE reference in case of DTD versions earlier than 2.0. For papiNet version 2.0 or later, only a Schema Reference is used. This Reference will also be copied to the ebXML-Manifest.

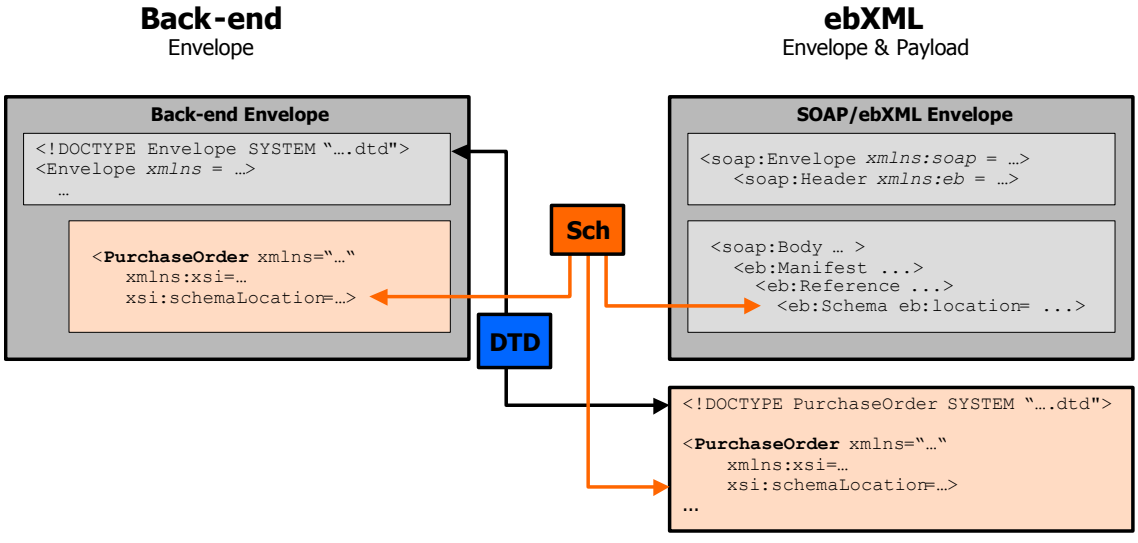


Fig. 5: Overview of used DOCTYPE / Schema references

Where	Data Elements	Example
Sender		UPM Kymmene's ERP system
Sender's Backend-Envelope	<SenderOrganisation> <ReceiverOrganisation>	"UPMKYMMENE" "BURDA"
Sender's Messenger		Uses "BURDA" as a key to look up the PartnerIDs for the receiver and the organisational PartyID in partners.xml Uses "UPMKYMMENE" as a key to look-up the PartnerIDs for the receiver and the organisational PartyID in partners.xml
ebXML Envelope	<PartyID> (1-N entries)	This is just an example, used here in expectation of a final specification by the papiNet CWG. <From> <PartyID type="DunsNumber"> 123456789</PartyID> </From> <To> <PartyID type="DunsNumber"> 987654321</PartyID> </To> Note that there may be multiple PartyID entries for both the sender and receiver.
Receiver's Messenger		Uses partners.xml to reversely map from PartnerID to local ID
Receiver's Backend-Envelope	<SenderOrganisation> <ReceiverOrganisation>	"Upm/AA/XX/BB" "Burda/PP/YY/ZZ"
Receiver		Burda Procurement & Planning SAP System

7 Wire Formats

Since parts of the MIME attachments are binary (e.g. a PKCS signature value or the compressed & signed payload), attachments are transformed into printable characters to avoid any transport problems.

7.1 HTTP Wire Format

Content-Transfer-Encoding is set to 8bit to reduce the size of the messages. The Messenger will also accept base64 encoding when receiving messages.

For display reasons a base64 encoded message is shown here.

```
(HttpHeaders)

SOAPAction: "ebXML"
Content-type: multipart/related; boundary="BoundarY"
--BoundarY
Content-Type: text/xml
Content-Transfer-Encoding: base64
Content-ID: <ebxml-header>

MIAGCSqGSIb3DQEHA6CAMIACAQAxggEJMIIBBQIBADBuMGQxCzAJBgNVBAYTAkVVMRAwDgYDVQQK
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
naOWtuCi4TS7OL/FDGQsmytG0F1e3325PdzHnPntqJ3tigAAAAAAAAAAAAA=

--BoundarY
Content-ID: <ebxml-payload>
Content-Type: text/xml
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHA6CAMIACAQAxggEJMIIBBQIBADBuMGQxCzAJBgNVBAYTAkVVMRAwDgYDVQQK
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
naOWtuCi4TS7OL/FDGQsmytG0F1e3325PdzHnPntqJ3tigAAAAAAAAAAAAA=

--BoundarY
Content-ID: <image1.jpg>
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHA6CAMIACAQAxggEJMIIBBQIBADBuMGQxCzAJBgNVBAYTAkVVMRAwDgYDVQQK
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
```

```
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
naOWtuCi4TS7OL/FDGQsmytG0F1e3325PdzHnPntqJ3tigAAAAAAAAAAAA=

--BoundarY
Content-ID: <ebxml-payload-signature>
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHAqCAMIACAQExCzAJBgUrDgMCGgUAMIAGCSqGSIb3DQEHAQAAoIIGqDCCAvow
JdI8jDmnacf7Az+xmqUJW5b3KXt59/W8fpiGLDeFCrzk0/yMKDh9Xhk8Jn1S9z82xc1UTkZprQ3K
gpNyuBjKVlrqSZTOzsj9QlsS86iIbh+oowhvu8IcHJto2uCT7FkK8nrZtWP9xT3dLkwNZT3TAAAA
AAAA

--BoundarY
```

7.2 SMTP Wire Format

For SMTP the content has to be formatted as multipart mime. The Email body is not used. Content-Transfer-Encoding is always base64 to avoid any character encoding problems or signing issues.

```
From: ebXMLhandler@example.com
To: ebXMLhandler@example2.com
Date: Thu, 08 Feb 2001 19:32:11 CST
MIME-Version: 1.0
SOAPAction: "ebXML"
Content-type: multipart/related; boundary="BoundarY"; type="text/xml";
start="<ebxml-header>"

This Email Body is ignored

--BoundarY
Content-ID: <ebxml-header>
Content-Type: text/xml
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHA6CAMIACAQAxggEJMIIBBQIBADBuMGQxCzAJBgNVBAYTAkVVMRAwDgYDVQQK
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
naOWtuCi4TS7OL/FDGQsmytG0F1e3325PdzHnPntqJ3tigAAAAAAAAAAAA=

--BoundarY
Content-ID: <ebxml-payload>
```

```
Content-Type: text/xml
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHA6CAMIACAQAxggEJMIIBBQIBADBuMGQxCzAJBgNVBAYTAkVVMRAwDgYDVQQK
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
naOWtuCi4TS7OL/FDGQsmytG0F1e3325PdzHnPntqJ3tigAAAAAAAAAAAAAA=

--BoundarY
Content-ID: <image1.jpg>
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHA6CAMIACAQAxggEJMIIBBQIBADBuMGQxCzAJBgNVBAYTAkVVMRAwDgYDVQQK
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
EwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVUIENl
BeQchqYWAlXWkzs+kY6hFRykInNLw69kwvGkt6Q0I5rYO91IzfO6G/tdlXLKkJE56hi8vnmnHf0B
naOWtuCi4TS7OL/FDGQsmytG0F1e3325PdzHnPntqJ3tigAAAAAAAAAAAAAA=

--BoundarY
Content-ID: <ebxml-payload-signature>
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHAqCAMIACAQExCzAJBgUrDgMCGgUAMIAGCSqGSIb3DQEHAQAAoIIGqDCCAvow
JdI8jDmnacf7Az+xmqUJW5b3KXt59/W8fpiGLDeFCrzk0/yMKDh9Xhk8Jn1S9z82xc1UTkZprQ3K
gpNyuBjKVlrqSZTOzsj9QlsS86iIbh+oowhvu8IcHJto2uCT7FkK8nrZtWP9xT3dLkwNZT3TAAAA
AAAA

--BoundarY
```