ebXML Collaboration-Protocol Profile and Agreement Specification

Version 3.0

Editor’s Draft

Specification URIs:

This Version:

http://docs.oasis-open.org/ebxml-cppa/cppa-v3.0-spec-wd-r01.html

http://docs.oasis-open.org/ebxml-cppa/cppa-v3.0-spec-wd-r01.pdf

Previous Version:

http://docs.oasis-open.org/ebxml-cppa/cppa-v3.0-spec-wd-r01.html

http://docs.oasis-open.org/ebxml-cppa/cppa-v3.0-spec-wd-r01.pdf

Latest Version:

http://docs.oasis-open.org/ ebxml-cppa/ cppa-v3.0-spec-wd-r01.html

http://docs.oasis-open.org/ebxml-cppa/cppa-v3.0-spec-wd-r01.pdf

Latest Approved Version:

http://docs.oasis-open.org/ebxml-cppa/cppa-v3.0-spec-wd-r01.html

http://docs.oasis-open.org/ebxml-cppa/cppa-v3.0-spec-wd-r01.pdf

Technical Committee:

OASIS ebXML Collaboration Protocol Profile and Agreement (CPPA) TC

Chair(s):

Dale Moberg

Editor(s):

Dale Moberg

Marty Sachs

Related work:

This specification replaces or supersedes:

- Collaboration-Protocol Profile and Agreement Specification, Version 2.0

This specification is related to:

- ebXML Message Service Specification [ebMS] or [ebMS3]
- ebXML Business Process Specification Schema [ebBP]
- ebXML Registry Services Specification [ebRS]
- ebXML Registry Information Model [ebRIM]

Declared XML Namespace(s):

http://docs.oasis-open.org/ebxmlcppa/cppa-3.0

Abstract:

The CPP describes the capabilities of an individual Party. A CPA describes the capabilities that two Parties have agreed to use to perform particular business activities that are part of Business Collaborations. These CPAs define the "information technology terms and conditions" that enable Business documents to be electronically interchanged between Parties.

Status:

This document was last revised or approved by the Technical Committee or membership of OASIS on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/ebxml-cppa.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/ebxml-cppa/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/ebxml-cppa/errata.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications,

Table of Contents

Collaboration Protocol Profile and Agreement Specification

1 Introduction

Text following the conventions as outlined in Section 1.1 determine the conformance requirements for this specification. Need to determine if we place conformance in an appendix and outline or state only those RFC2119 language that are normative in nature (i.e. UPPER CASE).

1.1 Terminology

The UPPER CASE key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119]. An appendix is provided that outlines common terms used in this specification with definitions (See Appendix I).

1.2 Normative References

[ISO6523] Structure for the Identification of Organizations and Organization Parts, International Standards Organization (ISO), ISO-6523, 1998 (unofficial) http://www.nic.it/NA/iso6523.txt.

[RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering Task Force (IETF), RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt.

[RFC2141]. URN Syntax, IETF, RFC 2141, May 1997, http://www.ietf.org/rfc/rfc2141.txt.

[RFC2396]. Uniform Resource Identifiers (URI): Generic Syntax, Internet Engineering Task Force (IETF) RFC 2396, August 1998, http://www.ietf.org/rfc/rfc2396.txt.

[SAML2] Security Assertion Markup Language, OASIS, March 2005, http://www.oasis-open.org/specs/index.php#samlv2.0.

[WSDL11] Web Services Description Language (WSDL) 1.1, W3C, World Wide Web Consortium (W3C) Note 15 March 2001, http://www.w3.org/TR/wsdl.

[WSDL20] Web Services Description Language (WSDL) Version 2.0, Part 1: Core Language, Part 2 Adjuncts, W3C, June 2007, http://www.w3.org/TR/wsdl20.

[XLINK] XML Linking Language, W3C, June 2001, http://www.w3.org/TR/xlink/.

[XML] Extensible Markup Language (XML), 1.0, W3C, August 2006, http://www.w3.org/TR/2006/REC-xml-20060816/.

[XMLC14N] Canonical XML 1.0, W3C, 15 March 2001, http://www.w3.org/TR/2001/REC-xml-c14n-20010315.

[XMLDSIG] XML Signature Syntax and Processing, W3C, 12 February 2002, http://www.w3.org/TR/xmldsig-core/.

[XMLENC] XML Encryption Syntax and Processing, W3C, 4 March 2002, http://www.w3.org/TR/2002/CR-xmlenc-core-20020304/.

[XMLNS] Namespaces in XML, W3C, , 16 August 2006, http://www.w3.org/TR/REC-xml-names/.

[XMLSCHEMA-1] XML Schema Part 1: Structures, W3C, 28 October 2004, http://www.w3.org/TR/xmlschema-1/.

[XMLSCHEMA-2] XML Schema Part 2: Datatypes, W3C, 28 October 2004, http://www.w3.org/TR/xmlschema-2/.

[XPATH] XML Path Language (XPath), 1.0. W3C, 16 November 1999, http://www.w3.org/TR/xpath.

[XPATH2] XPath Language (XPath), 2.0, 23 January 2007, http://www.w3.org/TR/2007/REC-xpath20-20070123/.

[XPOINTER] XML Pointer Language, W3C, 16 August 2002, http://www.w3.org/TR/xptr/.

[XPOINTERFRAME] XPointer Framework. W3C, 10 July 2002, http://www.w3.org/TR/2002/WD-xptr-framework-20020710/.

[XMLNS-SCHEME] Steven DeRose, Eve Maler, and Ron Daniel Jr., editors. XPointer xmlns() Scheme Proposal. W3C, 10 July 2002, http://www.w3.org/TR/2002/WD-xptr-xmlns-20020710/.

1.3 Non-Normative References

[ccOVER] ebXML Core Components Overview, 1.05, UN/CEFACT, 10 May 2001, http://www.ebxml.org/specs/ccOVER.pdf. Note: Core Components Technical Specification (CCTS) 2.1 found at: http://www.untmg.org/index.php?option=com_docman&task=view_category&Itemid=137&subcat=14&catid=65&limitstart=0&limit=5.

[RFC3852]. Cryptographic Message Syntax (CMS), IETF, RFC, July 2004, http://www.ietf.org/rfc/rfc3852.txt. Note: Updates or errata include: CMS Authenticated-Enveloped-Data Content Type, Proposed Standard 5083, http://tools.ietf.org/html/rfc5083; errata: http://www.rfc-editor.org/errata_search.php?rfc=3852; and CMS Multiple Signer Clarification, Proposed Standard 4853, http://tools.ietf.org/html/rfc4853.

[DIGENV] Digital Envelope, RSA Laboratories, Section 2.2.4, http://www.rsa.com/rsalabs/node.asp?id=2184 (overall section: http://www.rsa.com/rsalabs/node.asp?id=2152).

[ebBP] ebXML Business Process Specification Schema, ebXML Business Process (ebBP) v2.0.4, Organization for the Advancement of Structured Information Standards (OASIS), 21 December 2006, http://docs.oasis-open.org/ebxml-bp/2.0.4/.

[ebBPSS] ebXML Business Process Specification Schema, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-bp (See Additional Information).

[ebMS] ebXML Message Service Specification v2, OASIS, 2002, http://www.oasis-open.org/specs/index.php#ebxmlmsgv2.

[ebMS3] ebXML Message Service Specification v3, Part 1: Core Features, OASIS, October 2007, http://www.oasis-open.org/specs/index.php#ebxmlmsgv3.

[ebRIM] ebXML Registry Information Model (RIM) v2, OASIS, 2002, http://www.oasis-open.org/specs/index.php#ebxmlrimv2.0.

[ebRIM3] ebXML RIM v3, OASIS, 2005, http://www.oasis-open.org/specs/index.php#ebxmlrimv3.0.

[ebRS] ebXML Registry Services (RS) Specification, OASIS, 2002, ,http://www.oasis-open.org/specs/index.php#ebxmlrsv2.0.

[ebRS3] ebXML Registry Services Specification v3, OASIS, 2005, http://www.oasis-open.org/specs/index.php#ebxmlrsv3.0.

[HTTP] Hypertext Transfer Protocol, IETF, RFC 2616, June 1999,(unofficial) http://www.ietf.org/rfc/rfc2616.txt.

[IPSEC] IP Security Document Roadmap, IETF, RFC 2411, November 1998, http://www.ietf.org/rfc/rfc2411.txt.

[ISO9735] United Nations Centre for Trade Facilitation and eBusiness (UN/CEFACT) Syntax, International Standards Organization (ISO) ISO-9735 (9735-4), http://www.unece.org/trade/untdid/download/r1244r1.doc.

[MIME] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, IETF, RFC 1521, September 1993, http://www.ietf.org/rfc/rfc1521.txt.

[RFC1123] Requirements for Internet Hosts -- Application and Support, IETF, RFC 1123, October 1989, http://www.ietf.org/rfc/rfc1123.txt.

[RFC1579] Firewall-Friendly FTP, IETF, RFC 1579, February 1994, http://www.ietf.org/rfc/rfc1579.txt.

[RFC1950] ZLIB Compressed Data Format Specification v3.3, IETF, RFC 1950, May 1996, http://www.ietf.org/rfc/rfc1950.txt.

[RFC2015] MIME Security with Pretty Good Privacy, IETF, RFC 2015, October 1996, http://www.ietf.org/rfc/rfc2015.txt.

[RFC2246] The TLS Protocol, IETF, RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt.

[RFC2251] Lightweight Directory Access Protocol v3, IETF, RFC 2251, December 1997, http://www.ietf.org/rfc/rfc2251.txt.

[RFC2617] HTTP Authentication: Basic and Digest Authentication, IETF, RFC 2617, June 1999, http://www.ietf.org/rfc/rfc2617.txt.

[RFC2822] Internet Message Format, IETF, RFC 2822, April 2001, http://www.ietf.org/rfc/rfc2822.txt.

[RFC3335] MIME-based Secure Peer-to-Peer Business Data Interchange over the Internet, IETF, RFC 3335, September 2002, http://www.ietf.org/rfc/rfc3335.txt. Note: Applicability Statement 1 (AS1).

[RFC3798] Message Disposition Notification, IETF, RFC 3798, May 2004, http://www.ietf.org/rfc/rfc3798.txt

[RFC4130] MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, IETF, RFC 4130, July 2005, http://www.ietf.org/rfc/rfc4130.txt. Note: Applicability Statement 2 (AS2).

[RFC4823]. FTP Transport for Secure Peer-to-Peer Business Data Interchange over the Internet, IETF, RFC 4823, April 2007, http://www.ietf.org/rfc/rfc4823.txt. Note: Applicability Statement 3 (AS3).

[RFC959] File Transfer Protocol (FTP), IETF, RFC 959, October 1985, http://www.ietf.org/rfc/rfc959.txt.

[S/MIME] S/MIME Version 3 Message Specification, Internet Engineering Task Force RFC 2633, June 1999, http://www.ietf.org/rfc/rfc2633.txt.

[SMTP] Simple Mail Transfer Protocol, Internet Engineering Task Force RFC 2821, April 2001, http://www.ietf.org/rfc/rfc2821.txt.

[SOAP11]. Simple Object Access Protocol (SOAP) 1.1, W3C, W3C Note, 8 May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[SOAP12]. SOAP Version 1.2 Part 1: Messaging Framework, W3C, 27 April 2007 http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[SSL] Secure Sockets Layer v3, http://www.openssl.org/docs/ssl/ssl.html.

[WSDL11Ref] WSDL 1.1 Element Identifiers, W3C, July 2007, http://www.w3.org/TR/2007/NOTE-wsdl11elementidentifiers-20070720/,

[WSPolicy15] WS-Policy v1.5, W3C, September 2007, http://www.w3.org/TR/ws-policy.

[X12] ANSI X12 Standard for Electronic Data Interchange, X12 Standard Release 4050, December 2001, (unofficial to CPPA) http://lists.oasis-open.org/archives/ebxml-cppa/200309/doc00001.doc.

[XAML] Transaction Authority Markup Language, http://xaml.org/.

[ZLIB] Zlib: A Massively Spiffy Yet Delicately Unobtrusive Compression Library, http://www.gzip.org/zlib/ or http://www.zlib.net/.

[UMM-R10] UN/CEFACT Modeling Methodology (UMM), R10, TBD.

[UMM] UMM Foundation Module, 2006, .

1.4 CPPA Updates

Version 3 (v3) of the Collaboration-Protocol Profile and Agreement Specification (CPPA) completes deferred and updated tasks from Version 2 (v2) that ensure compatibility with the new ebMS v3 and ebBP v2.0.4, and also allows greater extensibility for business processes, policies and messaging protocols..http://www.untmg.org/index.php?option=com_docman&task=view_category&Itemid=137&subcat=1&catid=63&limitstart=0&limit=5 The modifications made have been conservative whereby CPPA v2 instances can be easily upgraded to v3 primarily by changing the namespace. In this introductory section, we will summarize the main changes found in the new v3.

Each component specification of ebXML is intended to work with or without the use of other ebXML components. CPPA v2 only partially fullfilled this intent; it primarily provided support for messaging or business process description protocols in other ebXML specifications.

To continue this transition, the CPPA schema has been extended so that associations of business process steps with delivery channel details could be made for other several messaging and business process protocols.

The approach accepted by the TC members has been to make use of XML's concept of substitution groups in refactoring the existing content of the CPPA, while accommodating several major extension points that produce the flexibility needed to handle additional collaboration protocols and other business process descriptions or languages.

The v2 defined elements have been retained with new elements added to accommodate the substitution extension framework while others have been generalized for broader applicability. The technical framework elements and complexTypes will be summarized at the latter portion of this specification after the detailed inventory of CPPA constituents is presented.

An overview of the enhanced extension framework and its application to other collaboration protocols and technologies are also provided. That section includes web services such as use of policy, other messaging protocols like AS2 and web services, a flattened action binding structure, and other features.

It is important to note this specification makes full use of the technical extensibility framework and a modular construct approach to use emerging technologies while supporting other emergingchoices. This structure enables use of technologies such as web services using SOAP, policy, ebMS v3 Core, and also established messaging protocols such as AS2 (or ediint) [RFC4130].

Advancements in the automation of business processes and the relationship to quality of service agreements has also gained importance. This specification serves as a technical quality of service conduit between business process definitions and the concrete messaging protocols and associated services they provide for enabling business activities.

A flattened structure for Action bindings and linkage to ActionContext (ActionContext2) to the business transactions in business process definitions.

In this specification, previous errata approved for v2.0 have been integrated as solutions to identified community requirements (See http://www.oasis-open.org/committees/ebxml-cppa/documents/ebCPP-2_0-Errata.php). In addition, PartyID constructs have been enhanced in support of user communities that leverage ISO standards.

2 Summary of Contents of Document

The exchange of information between two Parties requires each Party to know the other Party's supported Business Transactions realized as activities composed into Business Collaborations found in business process definitions, the other Party's role in the these activities, and the technology details about how the other Party sends and receives Messages. Consistent with a business process definition such as ebXML Business Process [ebBP] or [ebBPSS], a Business Partner is an entity that engages in Business Transactions with another Business Partner(s). In some cases, it is necessary for the two Parties to reach agreement on some of the details.

The Message-exchange capabilities of a Party MAY be described by a Collaboration-Protocol Profile (CPP). The Message-exchange agreement between two Parties MAY be described by a Collaboration-Protocol Agreement (CPA).

A CPA MAY be created by computing the intersection of the two Partners' CPPs or as a result of the use of a CPA template. Included in the CPP and CPA are details of transport, messaging, security constraints, and bindings to a Business-Process-Specification (or, for short, Process-Specification) document that contains the definition of the interactions between the two Parties while engaging in interactions included in a specified electronic Business Collaboration.

This specification concentrates on binary interactions of business transactions (through requesting and responding business activities) over collaboration protocols. Those business transactions could be logically decomposed from many compound and/or nested Business Transactions and Business Collaborations. A Business Collaboration may be held in a business process specification language, definition or description.

This specification contains the detailed definitions of the Collaboration-Protocol Profile (CPP) and the Collaboration-Protocol Agreement (CPA).

This specification is a component of the suite of ebXML specifications although it may be used or is compatibility with other specifications such as message protocols or business process definitions.

This specification is organized as follows:

- 3: System Overview
- 4: Definition of the CPP and identification of the structure and relevant fields.
- 5: CPA definition.
- 6: Extensions to CPPs and CPAs.

The appendices address notices and historical information, examples and other details about the relationship to other specifications. Those appendices include:

- Examples
- Glossary terms
- Mapping to other specifications

> *NOTE: When these mappings are defined, a conditional obligation can exist where a particular functional mapping is used by a run-time system. For example: "If multiple PartyID elements occur under the same PartyInfo element in the CPA, all of those PartyID elements MUST be included in the Message Header" are a conditional obligation when, for example, a message protocol such as ebMS3 [ebMS3] is used (See Appendix A). Take note of the conditional obligation identified in the functional mappings that exist.*

- Extension descriptions
- Acknowledgements
- Revision history
- Other non-normative guidance

2.1 Document Conventions

In this specification, indented paragraphs beginning with "NOTE:" provide non-normative explanations or suggestions that are not mandated by the specification. Such guidance may however point to required semantics or usage when an optional capability is used (and usage requirements to be implemented). See also boundaries for consistency with the [RFC 2119] definition and this specification (See [RFC2119]).

References to external documents are represented with BLOCK text enclosed in brackets Such as [RFC2396]. The references are listed in the first section (See 1.2 and 1.3).

The UPPER CASE keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

By convention, values of [XML] attributes are generally enclosed in quotation marks, although those quotation marks are not part of the values themselves.

2.1.1 Pseudo-Schema

This specification uses the following syntax to define pseudo schemas for defined elements:

The syntax appears as an XML instance.

BNF-style notations for cardinality of attributes and elements are used: "?" denotes optionality (i.e. zero or one occurrence), "*" denotes zero or more occurrences, "+" one or more occurrences, "(" and ")" are used to form groups, and "|" represents choice.

Cardinality of {*min*,*max*} indicates that the preceding element MUST appear no fewer than *min* times, and no more than *max* times.

Italicized values indicate schema types or references to which the instance value MUST conform.

The characters "[" and "]" are used to call out properties.

Ellipses (i.e., "…") indicate points of extensibility.

XML namespace prefixes from Section 4.2 are used to indicate the namespace of elements being defined or referenced.

```
<!-- sample pseudo-schema -->
<defined_element
       required_attribute_of_type_string=xs:string
       optional_attribute_of_type_int=xs:int?
>
       <required_element/>
       <optional_element/>?
       <one_or_more_of_these_elements/>+
       <!-- zero or more elements chosen from the following set: -->
       ( <choice_1/> | <choice_2/> )*
       ...[any,PMW]
</defined_element>
```

2.2 Versioning of the Specification and Schema

Whenever this specification is modified, it SHALL be given a new version number.

It is anticipated that during the review period, errors and inconsistencies in the specification and in the schema may be detected and have to be corrected. When such inconsistencies occur, this specification is the primary reference. All known errors in the specification as well as necessary changes to the schema will be summarized in an errata page found at:

Need updated errata page for post-2.0.

The location for the historical versions of the schemas for the CPPA specification version value 2_0 (or 2.0) and examples is:

http://www.oasis-open.org/committees/ebxml-cppa/schema/

In addition, the OASIS standard version 2.0 of the schema SHALL always be found at the location below. The latter is the namespace URI used for this specification and the corresponding schema is supposed to be directly resolvable from the namespace URI.

http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd

This specification version of the OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee SHALL be version “3.0” and the schema for it SHALL have the targetNamespace:

```
    http://docs.oasis-open.org/ebxmlcppa/cpp-cpa-3.0
```

This URL is to be resolvable into the most current version of the schema for the specification version level 3.0.

The schema for the v3 specification version is to maintain compatibility in general with the version 2.0 level schema. For example, instances of XML documents for the version 2.0 will validate against the v3 one with only minor changes needed. A new namespace applies to this document.

Another change involves attribute prefixes. The need for the latter change reflects a change to the value of the attribute attributeFormDefault from “true” in version 2.0 to “false” in v3. The goal for this specification was to enable as much as practical backward compatibility to version 2.0 instances while enabling a transition to use of enhanced extensibility features and use of emerging technologies.

As was true for version 2.0, the value of the version attribute of the Schema element in a given version of the schema SHALL be equal to the version of the schema. The value to be used SHALL be “3.0”.

Allowing a convention of 2_0 or 2.0 could lead to issues.CORE

2.3 Definitions

Technical Glossary of Terms for this specification are found in the appendices (See Appendix I).

2.4 Audience

One target audience for this specification is the group of implementers of ebXML services and other designers and developers of middleware and application software that is to be used for conducting electronic Business. Another target audience is the enterprise stakeholders who are responsible for creating CPPs and CPAs, or CPA templates.

It is expected that the reader has an understanding of XML and is familiar with the concepts of electronic Business (eBusiness).

2.5 Conformance to this Specification

In this specification, RFC 2119 language in UPPER CASE is used to denote normative requirements for a CPP and/or CPA. Other conformance statements are clearly identified.

3 System Overview

3.1 What This Specification Does

The exchange of information between two Parties requires each Party to know the other Party's supported Business Transactions realized as business activities. These activities are composed and choreographed into Business Collaborations found in business process definitions where the business requirements are held. Each also knows the other Party's role in these activities, and the technology details about how the other Party sends and receives Messages. In some cases, it is necessary for the two Parties to reach agreement on some details.

Unless otherwise specifically identified, the use of the term Business Collaboration herein relates to the collaboration protocols supported by this specification (through definition or extension). The CPP or CPA functionality supports the decomposition of groups of compound activities into binary interactions or atomic units. In a CPP or CPA, the characteristics of those interactions are found in elements such as **CollaborationActivity** and BusinessTransactionCharacteristics found in the ActionContext or ActionContext2 that relate to business functionality found in a referenceable business process definition or description (or a more concrete specification language).

This specification concentrates on two-Party interactions and allows independent configuration of each constituent delivery channel used by requesting and responding business activities. Those interactions typically are based on Business Transactions that are choreographed in Business Collaborations over Business Collaboration Protocols.

This specification allows independent configuration of delivery channels used by two collaborating party, one of which is sending a message that the other is receiving. Using terminology from the ebBP specification readily known in eBusiness, each requesting and responding business activity that produces or consumes a message is configurable. These activities are contained in a Business Transactions, referenced within a choreography described within a Business Collaboration.

Business Transactions can be logically decomposed from many compound and/or nested Business Transactions and Business Collaborations. Most often, Business Collaboration(s) may be held in a business process specification language, definition or description.

The CPPA realizes concretely the technical contract around those two-Party interactions through actions that capture business messages, business and/or messaging signals or request-responses over Business Collaboration protocols, using various messaging transport mechanisms for these Business Transactions.

Business Transactions Activities are typically choreographed in a business process definition and can be composed as Business Collaboration activities. For ease of construction and flexibility for reuse, those activities are concretely defined in atomic units of work in binary interactions in the CPP or CPA. The actual choreographic definitions remain in the business process that applies (and therefore where the business requirements for business quality of service are held).

The characteristics of those two-Party interactions are found in elements such as ***BusinessTransactionCharacteristics*** referenced in a CPP or CPA or found in the action binding constructs. Those characteristics may relate to business functionality found in a referenceable business process definition or description (or a more concrete specification language).

The way each Party can exchange information, in the context of collaboration protocols that support activities in Business Collaborations, can be described by a Collaboration-Protocol Profile *(*CPP*)*. The agreement between the Parties can be expressed as *a* Collaboration-Protocol Agreement *(*CPA*)*.

In a CPP or CPA, Business activities become actionable in actions, context, and characteristics of the collaboration protocol(s) and bindings that are realized from the business process definition and its associated activities. The synergy between a CPP or CPA and the business process definition(s) used also enables the messaging protocols defined as bindings in this specification. For example, conversation identifiers link to message agents in the messaging protocol. These are associated with constructs in this specification and relate to or provide initial hooks for capturing expectations that exist in the business process definition.

A Party MAY describe itself in a single CPP or multiple CPPs. Those CPPs could describe, for example, different Business Collaborations that it supports, its operations in different worldwide regions, or diverse parts of its organization.

To enable Parties to find other Parties that are suitable Business Partners, CPPs MAY be stored in a repository and MAY be discovered by using the capabilities of a registry such as the ebXML Registry[ebRS3]. .

Interactions between two Parties MAY be found in a Process-Specification document. A Process-Specification MAY conform to the ebXML Business Process Specification Schema [ebBPSS] or [ebBP], or several other business process definitions. The CPP and CPA include references to a Process-Specification document. The Process-Specification document MAY be stored in a repository such as the ebXML Registry. See also Business Collaboration descriptions in 4.3.3.1.

Figure 1 illustrates the relationships between a CPP and two Process-Specification documents, A1 and A2, in an ebXML Registry. On the left is a CPP, A, which includes information about two parts of an enterprise that are represented as different Parties. On the right are shown two Process-Specification documents. Each of the PartyInfo elements in the CPP contains a reference to one of the Process-Specification documents. This identifies the Business Collaborations that the Party can perform. The constraints and parameters around this relationship is initially described earlier in this Section.

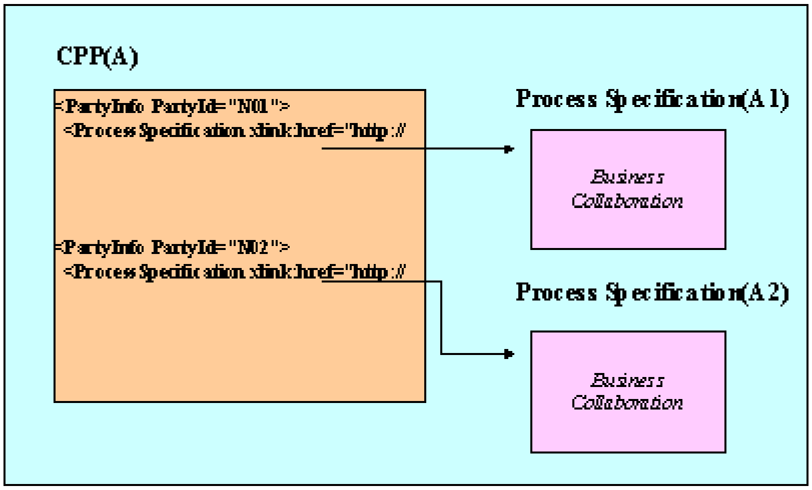

CPP(A)
<PartyInfo PartyId="N01">
<ProcessSpecification xlink:href="http://
<PartyInfo PartyId="N02">
<ProcessSpecification xlink:href="http://
Process Specification(A1)
Business Collaboration
Process Specification(A2)
Business Collaboration

Figure 1 Relationship between Process-Specification and Collaboration Protocol Profile

This specification defines the markup language vocabulary for creating electronic CPPs and CPAs. CPPs and CPAs are [XML] documents. The appendices provide mapping to defined message protocols and business process definitions.

The CPP describes the capabilities of an individual Party. A CPA describes the capabilities that two Parties have agreed to use to perform particular Business Collaborations that are decomposed into binary interactions.

These CPAs define the "information technology terms and conditions" that enable Business documents to be electronically interchanged between Parties. The information content of a CPA is similar to the information-technology specifications sometimes included in Electronic Data Interchange (EDI) Trading Partner Agreements (TPAs). However, these CPAs are not paper documents. Rather, they are electronic documents that can be processed by computers at the Parties' sites in order to set up and then execute the desired Business information exchanges. The terms and conditions of a formal Business agreement are outside the scope of this specification.

An enterprise MAY choose to represent itself as multiple Parties. For example, it might represent central office and manufacturing supply procurement organizations as separate Parties. The enterprise MAY then construct a CPP that includes all of its units that are represented as separate Parties and each by a separate PartyInfo element.

This specification is concerned with software that conducts business on behalf of Parties by exchanging Messages such as using [ebMS]. In particular, it is concerned with client and server software programs that engage in Business Transactions by sending and receiving Messages. Those Messages convey Business Documents and/or business signals in their payload. Business signals may be user defined or specified in a Process-Specification. Under the terms of a CPA:

- As software counterparts, a client initiates a connection with a server.

- As business counterparts, a Requester initiates a Business Transaction with a Responder.

- As messaging counterparts, a Sender sends a Message to a Receiver.

There is no fixed relationship between counterparts of different types which allows greater flexibility as the roles of the Parties involved changes. For example, consider a purchasing collaboration. Client software representing the buying party might connect to server software representing the selling party, and then make a purchase request by sending a Message containing a purchase order over that connection. If the CPA specifies a synchronous business response, the server might then respond by sending a Message containing an acceptance notice back to the client over the same connection. Alternatively, if the CPA specifies an asynchronous business response, client software representing the selling party might later respond by connecting to server software representing the buying party and then sending a Message containing an acceptance notice.

In general, the Parties to a CPA can have both client and server characteristics. A client requests services and a server provides services to that Party. In some applications, one Party only requests and another only provides services. These applications can resemble traditional client-server applications. In other applications were a peer-peer relationship is established, each Party MAY request services of the other.

3.2 Design Objectives

PMW: From former Section 3; edit as needed.

The objective of this specification is to enable interoperability between two Parties even though they may procure application and/or run-time support software from different vendors or sources. The CPP defines a Party's Message-exchange capabilities and the Business Collaborations realized in Business activities using Business Transactions that it supports.

A CPA defines the way two Parties will interact in performing the chosen activities within Business Collaborations. Both Parties use identical copies of the CPA to configure their run-time systems. This assures that they are compatibly configured to exchange Messages, regardless of where they have obtained their run-time systems. The configuration process may be automated by means of a suitable tool that reads the CPA and performs the configuration process.

It is an objective of this specification that a CPA be capable of being composed by intersecting the respective CPPs of the Parties involved or be derived from a CPA template. The resulting CPA will contain only those elements that are in common, or compatible, between the two Parties. Variable quantities, such as number of retries of errors, can then be negotiated between the two Parties. The design of the CPP and CPA schemata facilitates composition and/or negotiation processes. However, the composition and negotiation processes themselves are outside the scope of this specification.

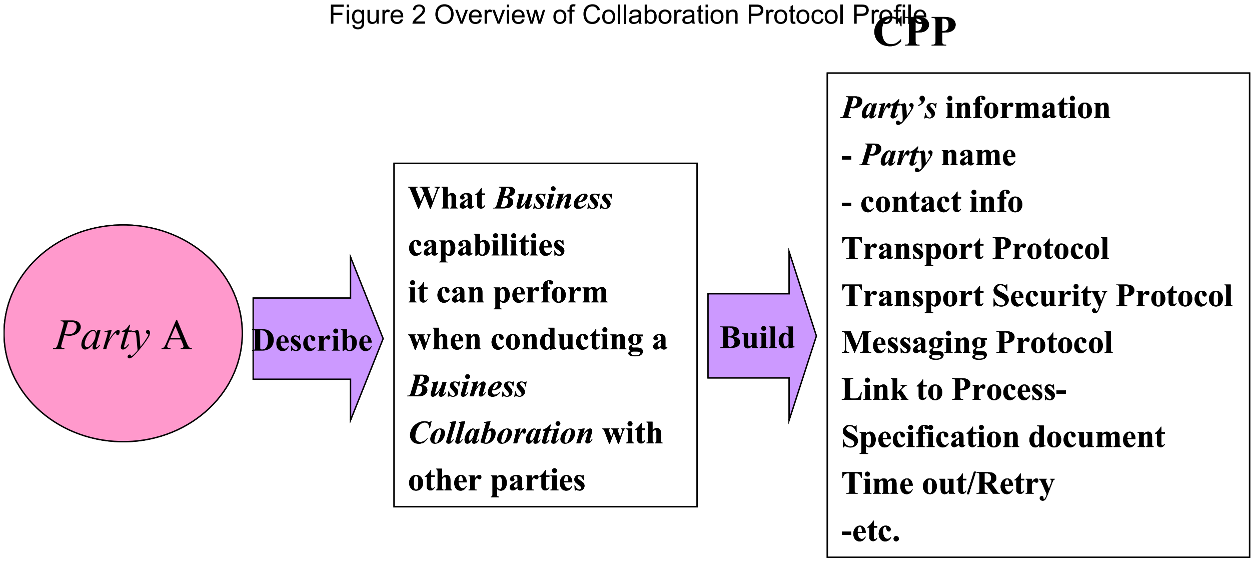
It is a further objective of this specification to facilitate migration of both traditional EDI-based and other legacy applications to platforms based on this specification or for use in tandem with other ebXML or emerging technologies. In particular, the CPP and CPA are components of the migration of applications based on the X12 838 Trading-Partner Profile [X12] to more automated means of setting up and realizing Business relationships.

3.3 Forming a CPA from Two CPPs

This section summarizes the process of discovering a Party to do Business with and forming a CPA from the two Parties' CPPs. In general, this section is an overview of a possible procedure and is not to be considered a normative specification.

Figure 2 illustrates forming a CPP. Party A tabulates the information to be placed in a repository for the discovery process, constructs a CPP that contains this information, and enters it into an ebXML Registry or similar repository along with additional information about the Party. The additional information might include a description of the Businesses of which the Party engages. Once Party A's information is in the repository, other Parties can discover Party A by using the repository's discovery services.

Figure 2 Overview of Collaboration Protocol Profile


CPP
Party A
Describe
What Business capabilities it can perform when conducting a Business Collaboration with other parties
Build
Party's information
- Party name
- contact info
Transport Protocol
Transport Security Protocol
Messaging Protocol
Link to Process-Specification document
Time out/Retry
-etc.

In Figure 3, Party A and Party B use their CPPs to jointly construct a single copy of a CPA by calculating the intersection of some or all of the information in their CPPs. The resulting CPA defines how the two Parties will behave in performing their Business Collaboration.

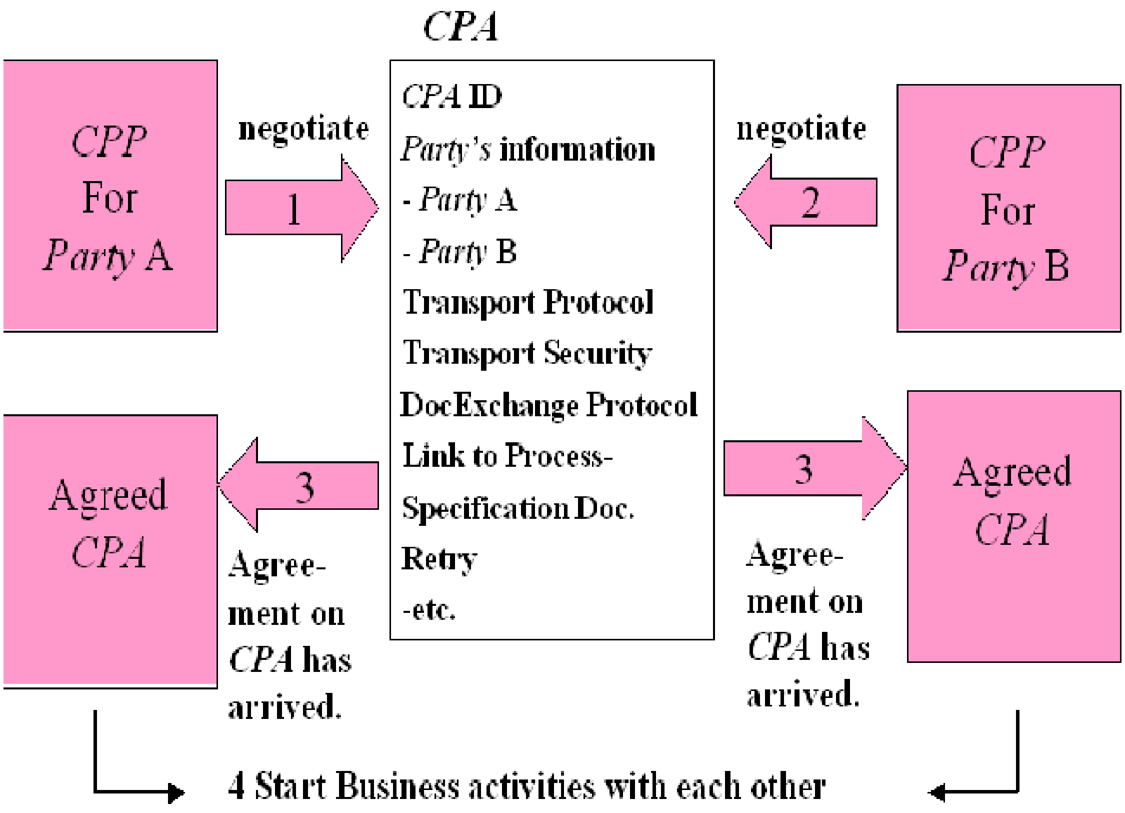
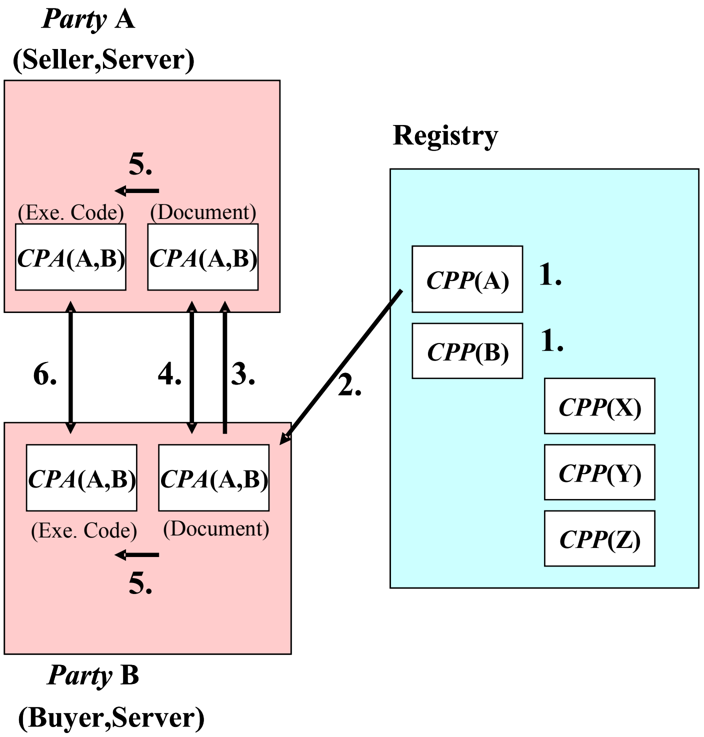


Figure 3 Overview of Collaboration Protocol Agreement

Figure 4 illustrates the entire process. The steps are listed at the left. The end of the process is that the two Parties configure their systems from identical copies of the agreed CPA and they are then ready to do Business.



1. Any *Party* may register its CPPs to an ebXML Registry.

2. *Party* B discovers trading partner A (Seller) by searching in the Registry and downloads *CPP*(A) to *Party* B's server.

3. *Party* B creates *CPA*(A,B) and sends *CPA*(A,B) to *Party* A.

4. *Parties* A and B negotiate and store identical copies of the completed *CPA* as a document in both servers. This process is done manually or automatically.

5. *Parties* A and B configure their run-time systems with the information in the *CPA*.

6. *Parties* A and B do business under the new *CPA*.

Figure 4 Overview of Working CPP/CPA with a Registry such as ebXML Registry

3.4 Forming a CPA from a CPA Template

Alternatively, a CPA template might be used to create a CPA. A CPA template represents one party's "fill in the blanks" proposal to a prospective trading partner for implementing one or more business processes. For example, such a template might contain placeholder values for identifying aspects of the other Party.

To form a CPA from a CPA template, the placeholder values would be replaced by the actual values for the other trading partner. Actual values might be obtained from the other party's CPP, if available, or by data entry in an HTML form, among other possibilities. The current version of this specification does not address how placeholder values might be represented in a CPA. However, the process of filling out a CPA template MUST result in a valid CPA. Further discussion of CPA templates is provided in the appendices. This could be added if we want to provide a reference to Pim's work posted on the web site; otherwise, delete this reference as it is not currently in the appendices.

3.5 How the CPA Works

A CPA describes all the valid visible, and hence possibly technically enforceable, interactions between the Parties and the way these interactions occur. It is independent of the internal processes executed at each Party. Each Party executes its own internal processes and interfaces them with the Business Collaboration described by the CPA and composed in a Process-Specification document(s). The CPA does not expose details of a Party's internal processes to the other Party. The intent of the CPA is to provide a high-level specification that can be easily comprehended by humans and yet is precise enough for computable enforcement by computers.

The information in the CPA is used to configure the Parties' systems to enable exchange of Messages in the course of performing a Business Transactions in a selected Business Collaboration. Typically, the software that performs the Message exchanges and otherwise supports the interactions between the Parties is middleware, software or services that can support any selected Business Collaboration. One component of this technology MAY be the ebXML Message Service Handler ([ebMS] or [ebMS3]). In this specification, the term "run-time system" or "run-time software" is used to denote such technology.

The CPA and the Process-Specification document that it references define a conversation between the two Parties. The conversation represents a single unit of Business as defined by the Business Collaboration component of the Process-Specification document through that document's business activities. The conversation consists of one or more Business Transactions, each of which is a request Message from one Party and a possible response Message from the other Party. The Process-Specification document defines, among other things, the request and response Messages for each Business Transaction and may specify the order in which the Business Transactions occur in a Business Collaboration. See for example, the [ebBPSS] or [ebBP] for a detailed explanation.

The CPA MAY actually reference more than one Process-Specification document. When a CPA references more than one Process-Specification document, each Process-Specification document defines a distinct type of conversation. Any one conversation involves only a single Process-Specification document.

A new conversation is started each time a new unit of Business is started. The Business Collaboration also determines when the conversation ends. From the viewpoint of a CPA between Party A and Party B, the conversation starts at Party A when Party A sends the first request Message to Party B. At Party B, the conversation starts when it receives the first request of the unit of Business from Party A. A conversation ends when the Parties have completed the unit of Business. These also could be applied using one-way type mechanisms such as ebMS v3 pull mode, whereby the conversation is tracked by an identifier.

For ease of construction, the CPA views those units of work as binary interactions although they may be composed in many compound structures in a business process definition.

> *NOTE: The run-time system could provide a mechanism by which the Business application can request initiation and ending of conversations. Typically, run-time software will generate a conversation identifier above the messaging interface.*

3.6 Where the CPA May Be Implemented

PMW: From former Section 3; edit as needed.

Conceptually, a Business-to-Business (B2B) server at each Party's site implements the CPA and Process-Specification documents. The B2B server may include the run-time technology (i.e., the software, services or middleware) that supports communication with the other Party, execution of the functions specified in the CPA, interfacing to each Party's back-end processes, and logging the interactions between the Parties for purposes such as audit and recovery. It could support the concept of a long-running conversation as the embodiment of a single unit of Business between the Parties. To configure the two Parties' systems for Business-to-Business operations, the information in the copy of the CPA and Process-Specification documents at each Party's site is installed in run-time system(s). The static information may be recorded in a local database and other information in the CPA and Process-Specification document may be used in generating or customizing the necessary code to support the CPA.

> *NOTE: It is possible to provide a graphical CPP/CPA-authoring tool that understands both the semantics of the CPP/CPA and the XML syntax. Equally important, the definitions in this specification make it feasible to automatically generate, at each Party's site, the code needed to execute the CPA, enforce computable rules, and interface with the Party's back-end processes.*

3.7 Definition and Scope

This specification defines and explains the contents of the CPP and CPA XML documents. Its scope is limited to these definitions. How to compose a CPA from two CPPs or define CPP and CPA run-time support is outside of the scope of this document. Non-normative suggestions and recommendations are provided where they serve to clarify the CPP and CPA definitions. See Section 2.5 for a discussion of conformance to this specification.

> *NOTE: Conformance in general can be achieved by producing a* CPP *or* CPA *document that conforms to the XML Schema document defined. It is, however, important to understand that the value of this specification lies in its enabling a run-time system that supports electronic commerce between two* Parties *under the guidance of the information in the* CPA. Conformance requirements are addressed in Section 2.5.

4 CPP Definition

A CPP defines the capabilities of a Party to engage in electronic Business with other Parties. These capabilities include both technology and business capabilities, such as supported communication and messaging protocols, what Business Collaborations it uses.

This section defines and discusses the details in the CPP in terms of the individual XML elements with some illustrations. The XML schema for the CPP and/or CPA is available in the specification package and samples provided throughout this document and the appendices (for functional mappings).

The ProcessSpecification, DeliveryChannel, DocExchange, and Transport elements of the CPP describe the processing of a unit of Business (conversation). These elements form a layered structure somewhat analogous to a layered communication model.

Process-Specification layer - The Process-Specification layer defines the heart of the Business agreement between the Parties: the services (Business Transactions) which Parties to the CPA can request of each other and transition rules that determine the order of requests. This layer is defined by the separate Process-Specification document that is referenced by the CPP and CPA, where those process descriptions choreography Business Transactions into Business Collaborations.

Delivery Channels - A delivery channel describes a Party's Message-receiving and Message-sending characteristics. It consists of one document-exchange definition and one transport definition. Several delivery channels MAY be defined in one CPPs.

Document-Exchange Layer - The Document-exchange layer specifies processing of the business documents by the Message-exchange function. Properties specified include encryption, digital signature, and reliable-messaging characteristics. The options selected for the Document-exchange layer are complementary to those selected for the transport layer. For example, if Message security is desired and the selected transport protocol does not provide Message encryption, then Message encryption MUST be specified in the Document-exchange layer. The protocol for exchanging Messages between two Parties is defined by, for example, the messaging services such as ebXML Message Service v3 Specification [ebMS3].

Transport layer - The transport layer identifies the transport protocol to be used in sending messages through the network and defines the endpoint addresses, along with various other properties of the transport protocol. Choices of properties in the transport layer are complementary to those in the document-exchange layer (See "Document-Exchange Layer" directly above.)

> *NOTE: The functional layers encompassed by the* CPP *are independent of the contents of the payload of the* Business *documents*

4.1 CPP Structure

Following is the overall structure of the CPP. Unless otherwise noted, CPP elements MUST be in the order shown here. Subsequent sections describe each of the elements in greater detail.

4.2 CollaborationProtocolProfile element

The ***CollaborationProtocolProfile*** element is the root element of the CPP XML document.

The REQUIRED XML [XML] Namespace [XMLNS] declarations for the basic document are as follows:

- XML
- XML Schema
- CPP/CPA namespace for 2.0
- CPP/CPA namespace for 3.0
- XML Encryption (See 1.2)
- XML Signature (See 1.2)
- Xlink (See 1.2)
- WSDL (for v1.1 and v2.0) (See 1.2 for both)

```
xmlns:wsdl11="http://schemas.xmlsoap.org/wsdl/"
```

```
xmlns:wsdl20="http://www.w3.org/2006/01/wsdl"
```

```
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
```

```
xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:tns="http://docs.oasis-open.org/ebxmlcppa/cppa-3.0"
```

```
xmlns:tp=http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
```

```
targetNamespace="http://docs.oasis-open.org/ebxmlcppa/cppa-3.0"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="3.0">
```

In addition, the CollaborationProtocolProfile element contains a REQUIRED cppid attribute that supplies a unique identifier for the document, plus a REQUIRED version attribute that indicates the version of the schema. Its purpose is to identify the version of the schema of which the CPP conforms. The value of the version attribute SHOULD be a string such as "2_0", "3.0", etc.

> NOTE: The method of assigning unique cppid values is left to the implementation.

The CollaborationProtocolProfile element SHALL follow this syntax:

```
<cppa:CollaborationProtocolProfile
       cppa:cppid=xs:non-empty-string
       cppa:version=xs:non-empty-string
>
       <cppa:PartyInfo/>+
       <cppa:SimplePart/>+
       <cppa:Packaging/>+
       <cppa:BusinessTransactionCharacteristics/>*
       <cppa:MessagingCharacteristics/>*
       <cppa:Signature/>?
       <cppa:Comment/>*
</cppa:CollaborationProtocolProfile>
```

The elements included are:

- PartyInfo: Identifies the organization (or parts of the organization) whose capabilities are described by the CPP.
- SimplePart elements: Describe the constituents used to make up composite Messages.
- Packaging: Describe how the Message Header and payload constituents are packaged for transmittal.
- Signature: Where present, contains the digital signature that signs the CPP document.
- Comment: Where present, describes comments related to PartyInfo.
- MessagingCharacteristics: Where present, describes delivery channel characteristics associated with Composite messages.
- BusinessTransactionCharacteristics: Where present, describes the quality of service characteristics associated with atomic units of work defined in the technical contract for Composite message transmission.

A CPP document MAY be digitally signed so as to provide for a means of ensuring the document has not been altered (integrity) and to provide authentication of the Party. A digitally signed CPP SHALL be signed using technology that conforms to the joint W3C/IETF XML Digital Signature specification[XMLDSIG]. In order to protect business data in a CPP, CPA or CPA template, [XMLENC] SHOULD be used.

4.3 PartyInfo Element

The PartyInfo element identifies the organization whose capabilities are described in this CPP and includes all the details about this Party. More than one PartyInfo element MAY be provided in a CPP if the organization chooses to represent itself as subdivisions with different characteristics.

Each of the sub-elements of PartyInfo is discussed later. The overall syntax of the PartyInfo element is as follows:

```
<cppa:PartyInfo
       partyName=tns:non-empty-string
       defaultMshChannelId=xsd:IDREF
       defaultMshPackageId=xsd:IDREF
>
       <tns:PartyId/>+
       <tns:PartyRef/>+
       <tns:CollaborationRole/>+
       <tns:Certificate/>*
       <tns:SecurityDetails/>*
       <tns:DeliveryChannel/>+
       <tns:Transport/>=
       <tns:DocExchange/>+
       <tns:OverrideMshActionBinding/>*
</cppa:PartyInfo>
```

The PartyInfo element contains a partyName attribute that indicates the common, human readable name of the organization. The value of each partyName SHALL identify the organization, entity, division or group of an organization described in the PartyInfo element in the CPP or CPA document.

The following example illustrates two possible party names.

```
       <tp:PartyInfo tp:partyName="Example, Inc."...</tp:PartyInfo>

       <tp:PartyInfo tp:partyName="Example, Inc. US Western Division">
       ...
       </tp:PartyInfo>
```

The PartyInfo element also contains a **defaultMshChannelId** attribute and a defautMshPackageId attribute.

The defaultMshChannelId attribute identifies the default **DeliveryChannel** to be used for sending standalone message handler such as the Message Service Handler [ebMS] level ones (i.e., Acknowledgment, Error, StatusRequest, StatusResponse, Ping, Pong) to be delivered asynchronously. When synchronous reply mode is in use, these messages are by default returned synchronously. The default can be overridden through the use of OverrideMshActionBinding elements.

The defaultMshPackageId attribute identifies the default Packaging to be used for sending standalone message handler messages.

The PartyInfo element consists of the following child elements, as shown in the schema snippet that follows:

- PartyId: Provides logical identifiers for the organization.
- PartyRef: Points to more information about the Party.
- CollaborationRole: Identifies the role(s) that this Party can play in the context of a Process-Specification document(s).
- Certificate: Identifies the certificates used by this Party in security functions.
- SecurityDetails: Identifies trust anchors and specifies security policy used by this Party in security functions.
- DeliveryChannel: Defines the characteristics that the Party can use to send and/or receive Messages. It includes both the transport and messaging protocols.
- Transport: Defines the characteristics of the transport protocol(s) that the Party can support to send and/or receive Messages.
- DocExchange: Defines the Message-exchange characteristics, such as the signature and encryption protocols that the Party can support.
- OverrideMshActionBinding: Specifies the DeliveryChannel to use for asynchronously delivered message handling messages.

4.3.1 PartyId Element

The PartyID element provides an identifier that SHALL be used to logically identify the Party. The value of the PartyID element is an non-empty string that provides an identifier. This is the convention used, for example, within a CPP or CPA when leveraged with ebMS.

The PartyId element has a single attribute: type that has an anyURI [XMLSCHEMA-2] value. In a CPP or CPA, the type value of the type attribute SHOULD be a URN [RFC2141] that defines a namespace for the value or content of the PartyId element. Typically, the URN would be registered in a well-known directory of organization identifiers.

For a CPP or CPA, if the type attribute is not present, the value or content of the PartyId element MUST be a URI that conforms to [RFC2396].

While an identifier may be understood by both Parties in a CPA, messaging protocols such as ebMS recommend the use of URIs. Within ebMS, the value or content of the PartyId MUST be a URI that conforms to [RFC2396].

Additional PartyId elements MAY be present under the same PartyInfo element so as to provide for alternative logical identifiers for the Party. In a CPP, if the Party has preferences as to which logical identifier is used, the PartyId elements SHOULD be listed in order of preference starting with the most-preferred identifier.

In a CPP that contains multiple PartyInfo elements, different PartyInfo elements MAY contain PartyId elements that define different logical identifiers. This permits a large organization, for example, to have different identifiers for different business or other technical purposes.

If multiple PartyId elements occur under the same PartyInfo element in the *CPA*, all of those PartyInfo elements MUST be included in the Message Header. This applies, for example, when protocol bindings such as ebMS v2 are used.

The value of the PartyId element is any string that provides a unique identifier. The identifier MAY be understood by both Parties to a CPA. Several schemes exist for naming identifier domains. [ISO6523] enumerates values for many domains: the domain is identified by a four-digit numeric sequence. For example, the ISO 6523 ICD 0060 identifies the Dun & Bradstreet domain. Typically, the identifier would be listed in well-known directories such as DUNS (Dun and Bradstreet) or in any naming system specified by [ISO6523].

Alternatively, the code list for ISO 9735 (UN/EDIFACT syntax) D.3 Service simple data element 0007 (routing Identification Code qualifier) can be used to name identifier domains [ISO9735]. ANSI ASC [X12] Data Element I05 (Interchange ID Qualifier) element, used in the ISA Interchange Control Header segment, also provides a list of code values for naming domains.

These three serve as "catalogues" of schemes for naming identifier domains; processes exist by which additional domains can be identified through the Registration Authorities (RA):

- the British Standards Institute (BSI): ISO 6523
- ISO/TC154-UN/CEFACT Joint Syntax Working Group (JSWG): [ISO9735] D.E Service simple data element 0007
- ANSI ASC X12: X12 Data Element I05

The following example shows how URNs are used for the type attribute and for the PartyId element value.

```
<PartyInfo partyName="CompanyA" defaultMshChannelId="asyncChannelA1"
              defaultMshPackageId="CompanyA_MshSignalPackage">
              <PartyId type="urn:oasis:names:tc:ebxml-cppa:partyid-
type:duns">123456789</PartyId>...

 <tp:PartyId>urn:icann:example.com</tp:PartyId>
```

The first example indicates the Party's DUNS number for the organization using a type attribute with a URN value. The value of the PartyId element itself is the DUNS number of the organization, which is a string of digits assigned by the agency. The second example shows an arbitrary URN as a PartyId value. No type is indicated, but the value might be a URN that the Party has registered with the Internet Assigned Numbers Authority [IANA] (http://www.iana.org) to identify itself directly.

When these naming conventions are used, values are generated for the type attribute from information items using a well-known directory of organization identifiers such as ISO6523. A method to standardize URNs used as PartyId@type values using ISO6523 is described. This method adheres to the rules, requirements and recommendations of a CPP or CPA, or ebMS.

An implementation SHOULD provide support, and be able to accommodate, the usage of the type values standardized herein (See four bullets that follow).

- If an abbreviated name is described in the item titled “Name of Coding System” within the ICD list, a type attribute can be constructed by prepending: “urn:oasis:names:tc:ebxml-cppa:partyid-type:” to the abbreviated name and appending a colon “:” followed by the ICD value. For example, using the abbreviated name D-U-N-S Number:

  Abbreviated Name: “D-U-N-S Number”

  Upper-camel-case resultant string: “D-U-N-SNumber”

```
tp:type=" urn:oasis:names:tc:ebxml-cppa:partyid-type:D-U-N-SNumber:0060"
```

Note: “0060” is the ICD value of D-U-N-S Number.

To be consistent with v2 CPP/A, the value that follows remains a valid type attribute value or content for the PartyId element: “urn:oasis:names:tc:ebxml-cppa:partyid-type:duns”.

- Because an abbreviated name may be omitted from the ICD list, the type attribute can always contain the string derived from “Name of Coding System” expressed in upper-camel-case. A value can always be constructed by pre-pending “urn:oasis:names:tc:ebxml-cppa:partyid-type:” to the upper-camel-case name and appending a colon “:” followed by the ICD value. For example, using the formal name of the Name of Coding System: “Data Universal Numbering System”:

  Transformed Camel-case: “DataUniversalNumberingSystem”

```
tp:type=" urn:oasis:names:tc:ebxml-cppa:partyid-type:DataUniversalNumberingSystem:0060"
```

- Punctuation marks in these formal names (such as, “/”, “-“ or “’”) should be included unless they are not allowed in URNs [RFC2141]. If the punctuation characters are not allowed in URNs, then the hexadecimal escaping convention explained in [RFC2141] should be followed for characters not allowed in URNs. However, spaces are not allowed in URNs and should be consumed during the production of an upper-camel-case string, rather than preserved in an escaped form. Words in names that are all upper-case should remain so when converted to an upper-camel-case string.

- The ICD value should be appended as the last field of the URN so that any collision between formal or abbreviated names is avoided.

4.3.2 PartyRef element

The PartyRef element provides a link, in the form of a URI, to additional information about the Party. Typically, this would be the URL from which the information can be obtained. The content and/or format of that information at that URI is outside of the scope of this specification.

The information might be at the Party's web site or in a publicly accessible repository such as an ebXML or UDDI Registry, or a Lightweight Directory Access Protocol[RFC2251] (LDAP) directory. Information available at that URI MAY include contact information like names, addresses, and phone numbers, or context information like geographical locales and industry segments, or perhaps more information about the Business Collaborations that the Party supports. This information MAY be in the form of an ebXML Core Component [ebXML Core Component Overview].

The PartyRef element is an [XLINK]simple link with attributes. The syntax is as follows:

```
<cppa:PartyRef
        type=xsd:anyURI?
        schemaLocation=xsd:anyURI?
        <!-- from tns:XlinkGroup -->
        xlink:type="simple"?
        xlink:href=xlink:href
/>
```

The contents of the document referenced by the PartyRef element are subject to change at any time. The value of the xlink:href SHOULD be dereferenced only when the contents of this document are used rather than cached for significant time periods.

Note, in migration from CPPA v2 to v3, a XlinkGroup was created that accommodates the xlink:type and xlink:ref attributes for ease of construction. tns:XlinkGroup attribute group

The attribute group houses a type and reference to the linked references.

- The xlink:type attribute SHALL have a value of "simple". This identifies the element as being an [XLINK] simple link.

- The xlink:href attribute SHALL have a value that is a URI that conforms to [RFC2396]and identifies the location of the external information about the Party.

The value of the type attribute identifies the document type of the external information about the Party. It MUST be a URI that defines the namespace associated with the information about the Party. If the type attribute is omitted, the external information about the Party MUST be an HTML web page.

The value of the schemaLocation attribute provides a URI for the schema that describes the structure of the external information.

Two examples of the PartyRef element are:

```
<PartyRef xlink:href="http://CompanyA.com/about.html"/>

<PartyRef xlink:type="simple" xlink:href="http://CompanyB.com/about.html"/>
```

determine if this new snippet is sufficient

4.3.3 CollaborationRole element

The CollaborationRole element associates a Party with a specific role in the business collaboration protocol of the Business Collaboration. Generally, the Process-Specification is defined in terms of roles such as "buyer" and "seller". The association between a specific Party and the role(s) it is capable of fulfilling within the context of a Process-Specification is defined in both the CPP and CPA documents. In a CPP, the CollaborationRole element identifies which role the Party is capable of playing in each Process-Specification documents referenced by the CPP.

The CollaborationRole element SHALL follow this syntax:

```
<cppa:CollaborationRole>
        <cppa:ProcessSpecification/>
        <cppa:Role/>
        <cppa:ApplicationCertificateRef/>*
        <cppa:ApplicationSecurityDetailsRef/>?
        <cppa:ServiceBinding/>+
</cppa:CollaborationRole>
```

The ProcessSpecification element identifies the Process-Specification document that applies to a role. The CollaborationRole element SHALL contain the appropriate combination of ProcessSpecification element and Role element.

The Role element identifies which role the Party is capable to support. To indicate that the Party can play roles in more than one collaboration protocol for a Business Collaboration or more than one role in a given Business Collaboration, the PartyInfo element SHALL contain more than one CollaborationRole element. Where more than one role applies, *e*ach CollaborationRole element SHALL contain the appropriate combination of ProcessSpecification element and Role element.

In a business process definition, a Party could play multiple roles which can relate to an external role exposed to other Parties for a Business Collaboration(s). The associations described above map to the external role of the collaboration that maps to the role of the business activities.

The ApplicationCertificateRef element identifies the certificate to be used for application level signature and encryption. The ApplicationSecurityDetailsRef element identifies the references to the trust anchors and security policy that will be applied to any application-level certificate offered by the other Party.

The ServiceBinding identifies the send and receipt elements that include the delivery channels used for sending and receiving business action messages by the Role (or roles) involved. The CanSend and CanReceive elements MAY also be used for specifying ***DeliveryChannels*** for business signal messages.

Each Party SHALL have a default delivery channel for the delivery of standalone message handling signals such as those used by ebMS like (Reliable Messaging Acknowledgments, Errors, StatusRequest, StatusResponse, etc)..

Leave as ebCORE issue. Dale 3/13/2008

4.3.3.1 ProcessSpecification element

In the CPP or CPA, the ProcessSpecification element provides the link to the Process-Specification document that defines the interactions between the two Parties. For rich functionality for Business Collaboration, the ebXML Business Process Specification Schema([ebBPSS] or [ebBP]) SHOULD be used. The Process-Specification document MAY be kept in a Registry such as an ebXML Registry and Repository.

Define is this should be retained. Leave as ebCORE issue.

A mapping between the CPA and either ebBP or ebBPSS is found in the appendices (See Appendices E and F respectively).

> *NOTE: A Party can describe the Business Collaboration using any desired alternative to the ebXML Business Process Specification Schema. When an alternative is used, the Parties to a CPA agree on how to interpret the business process description and the corresponding CPA constructs. These affected elements in the CPA could include the Role element, the CanSend and CanReceive elements if used (for v2 compatibility), the ActionContext or ActionContext2 element, and attributes of the BusinessTransactionCharacteristics element.*

When an alternative business process description is used, the Parties to a CPA MUST agree on how to interpret that description and elements in the CPA that reference information in that description.

An v2 example of the use of the ProcessSpecification element is:

```
<ProcessSpecification version="2.0" name="Notifications" xlink:type="simple"
        xlink:href="http://docs.oasis-open.org/ebcppa/samples/Notification-ebBP20.xml"
        uuid="urn:oasis-open.org:ebcppa:Notification"/>
```

In v3, the ***ProcessSpecification*** element can be extended using a substitution group. The syntax is as follows:

```
<cppa:ProcessSpecification
       name=tns:non-empty-string
       version=tns:non-empty-string
       uuid=xsd:anyURI?
       <!-- from ProcessSpecificationBaseType -->
       ...[namespace="##any"]*
       <!-- from attributeGroup XlinkGroup -->
       xlink:type="simple"?
       xlink:href=xlink:href
>
       <ds:Reference/>*
</cppa:ProcessSpecification>
```

The ProcessSpecification element contains ds:Reference elements in accordance with the XML Digital Signature specification[XMLDSIG] where the syntax and semantics are defined. The first ds:Reference element, if present, relates to the xlink:type and xlink:href attributes of the tns:XLinkGroup attribute group.

In case the CPP (CPA) document is signed, the first ds:Reference element that is present MUST include a ds:URI attribute whose value is identical to that of the xlink:href attribute of the XlinkGroup in the enclosing ProcessSpecification element.

The ds:Reference element specifies a digest method and digest value to enable verification that the referenced Process-Specification document has not changed. Additional ds:Reference elements are needed if the referenced ProcessSpecification in turn includes (i.e., references) other ProcessSpecifications. Essentially, ds:Reference elements MUST be provided to correspond to the transitive closure of all ProcessSpecifications that are referenced directly or indirectly to ensure that none of them has been changed. Other constraints for the ds:Reference element are defined later in this Section.

The ProcessSpecification element MUST include a ***name*** attribute which is a string that identifies the business process definition being performed.

If the Process-Specification document is defined by the ebXML Business Process specification ([ebBPSS] or [ebBP]), then this attribute MUST be set to the ***name*** for the corresponding ***ProcessSpecification*** element within the Business Process Specification instance.

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x ebBP v2.x	ProcessSpecification/@name	ProcessSpecification/@name	ProcessSpeciification/@name in the business process specification instance maps to the ProcessSpecification/@name herein.

On the table above and throughout this document, all CPP/CPA references are to both v2 and v3 unless otherwise specified explicitly.

The ProcessSpecification element includes a version attribute to indicate the version of the Process-Specification document identified by the xlink:href attribute of the tns:XlinkGroup attribute group (and also identified by the ds:Reference element, if any).

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x	ProcessSpecification/@version	ProcessSpecification/@version	Name of the specification version.
ebBP v2.x	ProcessSpecification@specificationVersion	ProcessSpecification/@version	Name of specification

	ProcessSpecification@instanceVersion	N/A	version.

The XlinkGroup attributeGroup follows the same convention as previously specified. The xlink:type attribute has a FIXED value of "simple". This identifies the element as being an [XLINK] simple link.

The REQUIRED xlink:href attribute SHALL have a value that identifies the Process-Specification document and is a URI that conforms to [RFC2396].

The uuid attribute uniquely identifies the ProcessSpecification. If the Process-Specification document is defined by the ebXML Business Process specification ([ebBPSS] or [ebBP]), then this attribute MUST be set to the uuid for the corresponding **ProcessSpecification** element within the business process specification instance.

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x ebBP v2.x	ProcessSpecification/@uuid	ProcessSpecification/@uuid	

4.3.3.1.1 ds:Reference element

The ds:Reference element identifies the same Process-Specification document as the enclosing ProcessSpecification element's xlink:href attribute of the associated tns:XlinkGroup attributeGroup and additionally provides for verification that the Process-Specification document has not changed since the CPP was created, through the use of a digest method and digest value as described below.

NOTE: CPP or CPA validity may be tested such as:

- *For a CPP and the referenced Process-Specification documents at the time composition of a CPA begins in case they have changed since they were created*
- *For a CPA and the referenced Process-Specification documents at the time a CPA is installed into a Party's system*
- *For a CPA at intervals after the CPA has been installed into a Party's system. The CPA and the referenced Process-Specification documents MAY be processed by an installation tool into a form suited to the particular software, services or middleware. Therefore, alterations to the CPA and the referenced Process-Specification documents do not necessarily affect ongoing run-time operations. Such alterations might not be detected until it becomes necessary to reinstall the CPA and the referenced Process-Specification documents.*

The [XMLDSIG] defines a ds:Reference element that can have a ds:Transforms child element, which in turn has an ordered list of one or more ds:Transform child elements to specify a sequence of transforms.

This specification SHALL adhere to the Canonical XML [XMLC14N] transform and SHALL forbid other transforms as follows:

- The ds:Reference element MUST have a ds:Transforms child element.
- That ds:Transforms element MUST have exactly one ds:Transform child element.
- That ds:Transform element MUST specify the Canonical XML [XMLC14N] transform via the following value for its ds:Algorithm attribute: http://www.w3.org/TR/2001/Rec-xml-c14n-20010315.

NOTE: Implementation of Canonical XML is REQUIRED by the XML Digital Signature specification[XMLDSIG].

To enable verification that the identified and transformed Process-Specification document has not changed, the ds:DigestMethod element specifies the digest algorithm applied to the Process-Specification document, and the ds:DigestValue element specifies the expected value. The Process-Specification document is presumed to be unchanged if and only if the result of applying the digest algorithm to the Process-Specification document results in the expected value.

A ds:Reference element in a ProcessSpecification element has implications for CPP validity:

- A CPP MUST be considered invalid if any ds:Reference element within a ProcessSpecification element fails reference validation as defined by the XML Digital Signature specification[XMLDSIG]

- A CPP MUST be considered invalid if any ds:Reference element within it cannot be dereferenced.

Other validity implications of such ds:Reference elements are specified in the description of the Signature element in Section 5.8.

4.3.3.2 Role element

The Role element identifies which role in the ProcessSpecification the Party is capable of supporting via the ServiceBinding element(s) siblings within this CollaborationRole element.

The Role element has the following syntax:

```
<cppa:Role
       name=tns:non-empty-string
       <!-- from attributeGroup tns:XlinkGroup -->
       xlink:type="simple"?
       xlink:href=xlink:href
/>
```

The name attribute is a string that gives a name to the Role [See 4.3.3.1].

Process Definition	Process Reference	CPP/CPA Reference	Comments

ebBPSS v1.x	BinaryCollaboration/Role/@name	CollaborationRole/Role/@name	
ebBP v2.x	BusinessCollaboration/Role/@name BinaryCollaboration/Role/@name MultiPartyCollaboration/Role/@name	CollaborationRole/Role/@name	A top-level collaboration is one whose isInner Collaboration = false.

The xlink:type attribute has a FIXED value of "simple". This identifies the element as being an [XLINK] simple link.

The REQUIRED xlink:href attribute SHALL have a value that is a URI that conforms to [RFC2396]. It identifies the location of the element or attribute within the Process-Specification document that defines the role in the context of the Business Collaboration. An example is:

```
xlink:href="http://www.rosettanet.org/processes/3A4.xml#Buyer"
```

Where "Buyer" is the value of the ID attribute of the element in the Process-Specification document that defines the role name.

4.3.3.3 ApplicationCertificateRef element

The ApplicationCertificateRef element, if present, identifies a certificate for use by the business process/application layer for exchanging keys. When ebMS v2 is used, this certificate is not used, but is included in the CPP so that it can be considered in any CPA negotiation process.

> *NOTE: Application software on both sides of a collaboration determine the intended/allowed usage of an application certificate by inspecting the key usage extension within the certificate itself.*
>
> *NOTE: This element ,* ApplicationCertificateRef*, is included in the* CPP/CPA *to support interoperability with legacy systems that already perform cryptographic functions such as digital signature or encryption. Implementers should understand that use of* ApplicationCertificateRef *is necessary only in cases where interoperability with such legacy systems is required.*

For the CollaborationRole element, the ApplicationCertificateRef element has the following syntax derived from its complexType:

```
<cppa:ApplicationCertificateRef
       certId=xsd:IDREF
/>
```

The certId attribute is an [XML] IDREF that associates the CollaborationRole element with a certificate. It MUST have a value equal to the value of the certId attribute of one of the Certificate elements under PartyInfo.

4.3.3.4 ApplicationSecurityDetailsRef element

The element ApplicationSecurityDetailsRef, if present, identifies the trust anchors and security policy that this Party will apply to any application-level certificate offered by the other Party. When ebMS v2 is used, these trust anchors and policy are not used by ebMS v2, but included in the CPP to be considered if any CPA negotiation process applies.

The ApplicationSecurityDetailsRef element follows this syntax:

```
<cppa:ApplicationSecurityDetailsRef
        securityId=xsd:IDREF
/>
```

The securityId attribute is an [XML]IDREF that associates the CollaborationRole with a SecurityDetails element that specifies a set of trust anchors and a security policy. It MUST have a value equal to the value of the securityId attribute of one of those SecurityDetails elements under PartyInfo.

4.3.3.5 ServiceBinding element

The ServiceBinding element identifies the delivery details around the business Message traffic that is to be sent or received by the Party within the context of the identified Process-Specification document. A service comprises action bindings around the delivery of business Message traffic.

There are two approaches to the content of a Service binding – one for compatibility for v2 includes the ActionContext CanSend and CanReceive child elements. In v2, To flatten the hierarchy in v3, the recommended use is an element ActionBinding in the ActionBindingBaseHead substitution group.

```
<ServiceBinding>
        <tns:Service>
                (
                        (
                                <tns:CanSend/>*
                        |
                                <tns:CanReceive/>*
                        )+
                |
                        ( PMW: ??
                                <ActionBindingBaseType
                                        ...[namespace="##any" processContents="strict"]
                                />
                                <tns:ActionBinding/>
                                <tns:ThisPartyActionBinding/>
                        )*
                )
        </tns:Service>
</ServiceBinding>
```

4.3.3.5.1 Service element

The Service element identifies what collaboration is targeted. This establishes the requesting and responding business activities in a business process definition that describe the actions to be implemented.

> *NOTE: A “top-level” BusinessCollaboration, MultiPartyCollaboration or BinaryCollaboration element is one whose isInnerCollaboration attribute value is false. The v3.0 schema also adds two Xlink attributes to the* ***Service*** *element to identify which top-level BusinessCollaboration, MultiPartyCollaboration or BinaryCollaboration element is referenced from a business process definition (See table herein in 4.3.3.5.1 that follows).*

The value of the Service element is a string that SHALL be used as the value of the Service element in the ebMS ([ebMS] or [ebMS3]), or a similar element in the Message Header of another supported message service as shown in the appendices..

In this specification, the action message functions have been enhanced and extended to simplify the overall structure, allow for greater extensibility, and support previous v2 CPP*s/*CPAs. The action binding constructs provide ease of implementation and flexibility for existing and anticipated implementations. The Service element follows this syntax:

```
<cppa:Service
       type=tns:non-empty-string?
/>
```

> *NOTE: The purpose of the Service element is to provide application level routing information for the ebMS or other supported Message Header. The CollaborationRole element and its child elements identify the information in the ProcessSpecification document relevant to the CPP or CPA. The Service element could be used along with the CanSend and CanReceive elements (and their descendants) or the flattened structure to provide routing of received messages to the correct application entry point.*

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x	ProcessSpecification/@uuid	Service/@name	The value of the Service element MUST be as specified. This is the RI in business process definition instance document.
ebBP v2.x	BusinessCollaboration/@name BinaryCollaboration/@name MultiPartCollaboration/@name	Service/@name	The value of the Service element MUST be as specified.

This allows you to create part of CPP using the business process definition information (Party Info).

If the type attribute is present, it indicates that the Parties sending and receiving the Message know, by some other means, how to interpret the value of the Service element.

If the type attribute is not present, the value of the Service element MUST be a URI[RFC2396]

If using ebBPSS for the business process definition:

Process Definition	CPP/CPA Reference	Comments
ebBPSS v1.x	Service/@type	Must be an URI [RFC2396].

4.3.3.5.2 Action Bindings

This specification provides mechanisms to support a simplified action binding structure as well as one compatible with v2 CPP/CPA. This has resulted in enhancements for service and action binding structures. For v3, these bindings have been simplified using substitution groups to allow backward compatibility using CanSend and CanReceive and a flattened structure for service and bindings to provide ease of implementation.

This supports the definition of the capabilities of sending and receiving messages through the DeliveryChannels and Packaging.

For v3, the flattened action binding syntax is as follows:

```
<!-- Flattens out CPP and CPA structure for ActionBinding -->
<cppa:ActionBinding
        id=xsd:ID?
        action=xsd:NMTOKEN?
        sendOrReceive=xsd:NMTOKEN?
        correlativeBinding=xsd:IDREF?
        packageId=xsd:IDREF?
        <!-- from ActionBindingBaseType -->
        ...[namespace="##any" processContents="strict"]*
>
        <tns:BusinessTransactionCharacteristics/>
        <tns:ActionContextHead/>?
        <tns:ChannelId/>+
</cppa:ActionBinding>
```

This structure integrates into the CanSend and CanReceive capabilities that a Party can provide. Implementations for v2 are also supported, using the same syntax as shown above for v3.

4.3.3.5.2.1 CanSend and CanReceive elements

When present, the CanSend element identifies an action message that a Party is capable of sending. This element is used particularly for CPP or CPA v2.

```
<cppa:CanSend
       id=xsd:ID?
>
       <tns:ThisPartyActionBinding/>
       <tns:OtherPartyActionBinding/>?
       <tns:CanReceive/>*
</cppa:CanSend>
```

The ThisPartyActionBinding identifies the DeliveryChannel and Packaging that the Party uses to send the invocation action message for both CPPs and CPAs. The OtherPartyActionBinding element is only used in CPAs and CPA templates where the DeliveryChannels and Packaging used/expected by the two Parties MUST be compatible.

The CanReceive element identifies an action invocation message that a Party is capable of receiving. This element is used particularly for CPP or CPA v2.

As with the CanSend element, the bindings for the CanReceive element have been simplified using substitution groups to allow backward compatibility and a flattened structure for ease of implementation.

```
<cppa:CanReceive
       id=xsd:ID?
>
       <tns:ThisPartyActionBinding/>
       <tns:OtherPartyActionBinding/>?
       <tns:CanSend/>*
</cppa:CanReceive>
```

When the CanReceive element is present, it indicates that one or more synchronous response actions are expected.

> *NOTE: While the schema permits arbitrary nesting levels under the CanReceive or CanSend element, use cases have identified only two in actual use. For example, a Request with a synchronously returned Response could also specify an additional synchronous Acknowledgment.*

4.3.3.5.2.2 ThisPartyActionBinding element

The ThisPartyActionBinding specifies one or more DeliveryChannels and Packaging for Messages for a selected *action* that are to be sent or received by the Party in the context of the ProcessSpecification for the CollaborationRole that is associated with the ServiceBinding elements.

The ThisPartyActionBinding syntax is as follows:

```
<cppa:ThisPartyActionBinding
       id=xsd:ID
       action=xsd:NMTOKEN
       sendOrReceive=xsd:NMTOKEN?
       correlativeBinding=xsd:IDREF?
       packageId=xsd:IDREF
       <!-- from ActionBindingBaseType -->
       ...[namespace="##any" processContents="strict"]*
```

```
>
        <tns:BusinessTransactionCharacteristics/>
        <tns:ActionContextHead/>?
        <tns:ChannelId/>+
</cppa:ThisPartyActionBinding>
```

Under a given ServiceBinding element, there MAY be multiple action binding child elements, and when used, **CanSend** or **CanReceive**. In this specification, an extensible substitution model has been used for action bindings whereby backward compatibility is retained where CanSend and CanReceive elements exist or for other correlated actions.

When CanSend and/or CanReceive elements are used and both the ThisPartyActionBinding and OtherPartyActionBinding elements are present (such as in some CPAs), they MUST have identical action values or equivalent ActionContext or ActionContext2 elements (see 4.3.3.5.2.6).

These action bindings could encompass the same action to allow different software entry points and Transport options. In such a scenario, the DeliveryChannels referred by the ChannelID child elements of ThisPartyActionBinding SHALL point to distinct EndPoints for the receiving message handler to uniquely identify the DeliveryChannel being used for this particular message exchange.

Verify this is correct given extensibility point.

> *NOTE: An implementation could provide the capability of dynamically assigning delivery channels on a per Message basis during performance of a Business Collaboration.*

The delivery channel selected would be chosen, based on present conditions, from those identified by CanSend elements or action bindings that refer to the action that is sending the Message. On the receiving side, the message handler could use the distinct EndPoints to identify the DeliveryChannel used for this message exchange.

The action attribute links the hierarchical naming associated with a business process or application to the corresponding action in the messaging service header.

The action attribute is a string or type of string that identifies the business document exchange to be associated with the DeliveryChannel identified (see syntax in 4.3.3.5.2). The value of the action attribute SHALL be used as the value of the Action element in the message header from ebMS or another messaging service.

The details of this mapping MAY be implemented by using the ActionContext or the ActionContext2 element [see 4.3.3.1).

Should the ebBPSS or ebBP business process definitions be used:

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x ebBP v2.x	RequestingBusinessActivity/@name RespondingBusinessActivity/@name	ThisPartyActionBinding/@ction	Business transactions are used in the same context and are not

			reused. Use identical names.
ebBPSS v1.x ebBP v2.x	RequestingBusinessActivity/@name RespondingBusinessActivity/@name	Provide ActionContext or ActionContext2 using RequestingBusinessActivity or RespondingBusinessActivity names, or create distinct names.	When business transactions are reused, ActionContext or ActionContext2 should exist or distinct names used that reflect collaboration hierarchy.

Update table from v3 CPA changes on action bindings.

Business signals can be sent individually rather than being bundled with response documents in synchronous reply mode. The value used for the action attribute SHALL be: ReceiptAcknowledgment, AcceptanceAcknowledgment, or Exception. In addition, they SHOULD specify a Service that is the same as the Service used for the original message. Signals can have their own delivery channel configured with their own action binding mechanism.

ebCORE: Decide if more exception definition is required. We have WSDL faults, error handling in ebMS, business level acknowledgments in ebBP or others.

> *NOTE: In general, the action name chosen by the two Parties to represent a particular requesting business activity or responding business activity in the context of a Business Collaboration that makes use of nested Business Collaborations MAY not be identical. Therefore, when composing two CPPs to form a CPA, it is necessary to make use of information from the associated* ActionContext or ActionContext2 (See 4.3.3.5.2.6) in order to determine if two different action names from the two CPPs actually represent the same ActionContext or ActionContext2. *Also see preceding table regarding business transactions used.*

For id attribute Need text. Need to address use="required" for v2 while v3 is optional.

The packageId attribute is an [XML]IDREF that identifies the Packaging element to be associated with the Message identified by the action attribute. In v3, this attribute is optional rather than its use explicitly required as in v2.

Need to address use="required" for v2 only if more needs to be said.

> *NOTE: Software for producing draft CPAs is encouraged to guarantee that ID values in one CPP are distinct from ID values in the other CPP so that no IDREF references collide when the CPPs are merged.*

In v3, two optional attributes have been added for the flattened action binding structure: sendOrReceive and correlativeBinding.

The optional sendOrReceive attribute has been added were the flattened action binding structure is used for a particular Message. This attribute explicitly specifies the abstract role the Party assumes. This attribute MUST be used when engaging the flattened action binding structure.

In v3, an optional correlativeBinding attribute has also been added where the flattened action binding structure is used for a particular Message. This identifies a reference to the channel for the other Party to be engaged and their configuration. Correlation support is described later in this specification.

In v2 only, the xlink:href attribute, if present, SHALL provide an absolute [XPOINTER]URI expression as specified.

Process Definition	Process Reference	CPP/CPA Reference (v2 only)	Comments
ebBPSS v1.x	RequestingBusinessActivity	ThisPartyActionBinding@action	Attribute specifically identifies each activity.
ebBP v2.x	RespondingBusinessActivity	ThisPartyActionBinding@action	

The xlink:type attribute has a FIXED value of "simple" (i.e. a [XLINK] simple link).

4.3.3.5.2.3 OtherPartyActionBinding element

The OtherPartyActionBinding element is used CPAs and CPA templates and SHALL NOT be used in a CPP. For v3, the element syntax is as follows:

```
<cppa:OtherPartyActionBinding>xsd:IDREF</cppa:OtherPartyActionBinding>
```

This element of type IDREF identifies a corresponding (matching, equal or compatible) ThisPartyActionBinding element found with the collaboration partner. It indirectly identifies the DeliveryChannel the other Party will use for sending or receiving the action message and the expected Packaging.

4.3.3.5.2.4 BusinessTransactionCharacteristics element

Characteristics of Business Transactions and their realization as Business Transaction Activities can be found in the business process definition and could be referenced by a CPP (see 4.2) or CPA. The reusable BusinessTransactionCharacteristics for a CPP or CPA references those. In addition, the BusinessTransactionCharacteristics element exists on the ActionBinding where the two-Party interactions are realized.

For the action bindings, the BusinessTransactionCharacteristics element describes the security characteristics and other attributes of the delivery channel, as derived from the ProcessSpecification(s) whose messages are transported. The attributes of the BusinessTransactionCharacteristics element bound to the action bindings, MAY be used to override the values of the corresponding attributes in the Process-Specification document referenced in a CPP or CPA.

In v3, these characteristics have been enhanced to allow for greater extensibility in the use of other business process definitions, accommodate ebXML Business Process Specification Schema(ebBP) v2.x, and to modularize the business transaction definitions (businessTransactionAttributes).

This construction also allows reference and reuse of such attributes in a CPP or CPA.

Information on alternative Business Collaboration descriptions is found later in this specification (See 4.3.3.1).

The BusinessTransactionCharacteristics element syntax is as follows:

```
<cppa:BusinessTransactionCharacteristics>
       <!-- from attributeGroup businessTransactionAttributes -->
       isNonRepudiationRequired=xsd:boolean
       isNonRepudiationReceiptRequired=xsd:boolean
       isConfidential=tns:persistenceLevelType
       isAuthenticated=tns:persistenceLevelType
       isTamperProof=tns:persistenceLevelType
       isAuthorizationRequired=xsd:boolean
       isIntelligibleCheckRequired=xsd:boolean
       timeToAcknowledgeReceipt=xsd:duration
       timeToAcknowledgeAcceptance=xsd:duration
       timeToPerform=xsd:duration
       retryCount=xsd:integer
       id=xsd:ID
       idRef=xsd:IDREF
       ...[namespace="##any" processContents="lax"]?
       <!-- from extension tns:BusinessTransactionCharacteristicsBaseType -->
       ...[namespace="##any" processContents="strict"]?
/>
```

These attributes allow parameters specified at the Process-Specification level to be overridden. If one of these attributes is not specified, the corresponding default value can be obtained from the Process-Specification document.

CPP and CPA composition tools and CPA deployment tools SHALL check the delivery channel definitions for the sender and receiver (transport and document-exchange) for internal consistency as well as compatibility between the two partners. Typically, when an attribute has a particular value, sub-elements under the corresponding Transport and DocExchange elements would exist to further describe the implied implementation parameters.

For v3, the sections that follow detail most of the attributes of the businessTransactionAttributes attributeGroup. In addition to those described as follows, ID and IDREFs have been added to the type definition. This allows ease of reference to those transactional expectations.

A BusinessTransactionCharacteristics element that is an immediate descendant of the root CollaborationProtocolProfile or a CollaborationProtocolAgreement MUST have a unique ID value for the id attribute, and will be referenced using the idRef attribute of a BusinessTransactionCharacteristics element.

Other attributes MAY also be defined by extension using the ##any namespace extension.

4.3.3.5.2.4.1 isNonRepudiationRequired and isNonRepudiationReceiptRequired attributes

The ***isNonRepudiationRequired*** attribute is a Boolean with possible values "true" or "false". If the value is "true" then the delivery channel MUST specify that the Message is to be digitally signed using the certificate of the Party sending the Message*,* and archived by both Parties.

The ***isNonRepudiationReceiptRequired*** attribute is a Boolean with possible values "true" or "false". If the value is "true" then the delivery channel MUST specify that the Message is to be acknowledged by a digitally signed Receipt Acknowledgment signal Message, signed using the certificate of the Party that received the Message, that includes the digest(s) of the payload(s) of the Message being acknowledged.

The ***SenderNonRepudiation*** and ***ReceiverNonRepudiation*** elements under the extensible v3 DocExchange constructs further describe various parameters related to the implementation of non-repudiation of origin and receipt, such as the hashing algorithm, the signature algorithm, the signing certificate, the trust anchor, etc.

Dale review For ebCORE

4.3.3.5.2.4.2 isConfidential attribute

The ***isConfidential*** attribute has the possible values of "none", "transient", "persistent", and "transient-and-persistent". For example, persistent confidentiality is intended to preserve the confidentiality of the message such that only the intended party (application) can see it. If the ebXML Business Process Specification Schema ([ebBPSS] or [ebBP]) are used, these values MUST be interpreted as defined by those business process definitions.

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x	DocumentEnvelope@isConfidential Attachment@isConfidential MAP OTHER ATTRIBUTES?	BusinessTransactionCharacteristics/@isConfidential	This value can not be overridden.
ebBP v2.x	DocumentEnvelope/@isConfidential	BusinessTransactionCharacteristics/@isConfidential	This value can not be overridden.

In general, transient confidentiality can be implemented using a secure transport protocol like SSL; persistent confidentiality can be implemented using a digital envelope mechanism like S/MIME [S/MIME].

More secure transport information is provided in the TransportSender (See 4.3.8.1) and TransportReceiver (See 4.3.8.1.3.6) elements under the Transport element. More persistent encryption information is provided in the SenderDigitalEnvelope and ReceiverDigitalEnvelope element under DocExchange constructs that facilitates a hierarchy of modular containers. (See 4.3.10.4.3.2). In v3, these mechanisms have been modularized and extended to allow for multiple protocols and bindings.

4.3.3.5.2.4.3 isAuthenticated attribute

The isAuthenticated attribute has the possible values of "none", "transient", "persistent", and "persistent-and-transient”. If this attribute is set to any value other than "none", then the receiver MUST be able to verify the identity of the sender. In general, transient authentication can be implemented using a secure transport protocol like SSL (with or without the use of basic or digest authentication); persistent authentication can be implemented using a digital signature mechanism.

Secure transport and persistent authentication information is further provided as detailed in Section 4.3.3.5.2.4.2.

4.3.3.5.2.4.4 isAuthorizationRequired attribute

The isAuthorizationRequired attribute is a Boolean with possible of values of "true" and "false". If the value is "true" then it indicates that the delivery channel MUST specify that the sender of the Message is to be authorized before delivery to the application.

4.3.3.5.2.4.5 isTamperProof attribute

The isTamperProof attribute has the possible values of "none", "transient", "persistent", and "persistent-and-transient". If this attribute is set to a value other than "none", then it must be possible for the receiver to detect if the received message has been corrupted or any tampering has occurred.

In general, transient tamper detection can be implemented using a secure transport like SSL; persistent tamper detection can be implemented using a digital signature mechanism.

Secure transport and persistent authentication information is further provided as detailed in Section 4.3.3.5.2.4.2.

4.3.3.5.2.4.6 isIntelligibleCheckRequired attribute

The isIntelligibleCheckRequired attribute is a Boolean with possible values of "true" and "false". If the value is "true", then the receiver MUST verify that a business document is not garbled (i.e., it passes schema validation) before returning a Receipt Acknowledgment business signal.

4.3.3.5.2.4.7 timeToAcknowledgeReceipt attribute

The timeToAcknowledgeReceipt attribute is of type duration [XMLSCHEMA-2]. It specifies the time period within which the receiving Party has to acknowledge receipt of a business document.

If this attribute is specified, then the Receipt Acknowledgment business signal MUST be used.

We may have an issue here because we don’t ever require a receipt acknowledgment in the ebBP tables.

4.3.3.5.2.4.8 timeToAcknowledgeAcceptance attribute

The timeToAcknowledgeAcceptance attribute is of type duration[XMLSCHEMA-2] It specifies the time period within which the receiving Party has to non-substantively acknowledge acceptance of a business document (i.e., after it has passed business rules validation).

If this attribute is specified, then the Acceptance Acknowledgment business signal MUST be used.

See comment above.

4.3.3.5.2.4.9 timeToPerform attribute

For v2, the timeToPerform attribute is of type duration [XMLSCHEMA-2]. It specifies the time period, starting from the initiation of the RequestingBusinessActivity, within which the initiator of the transaction MUST have received the response, i.e., the business document associated with the RespondingBusinessActivity.

The timeToPerform attribute associated with a BinaryCollaboration in ebBPSS v1.x is not modeled in this specification. Therefore, it cannot be overridden and the value specified at the ebBPSS level MUST be used.

When synchronous reply mode is in use (See 4.3.7.1.1), the timeToPerform value SHOULD be used as the connection timeout for v2 CPP/CPAs.

More advanced business process definitions may include choreographic compositions such as Business Collaborations with timing characteristics. For example, in ebBP v2.x, the TimeToPerform is an extensible element that allows use of condition expressions and external context for timing characteristics focused on business characteristics. Currently, this is handled outside of CPPA.

Core group discussion for timetoPerform.

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x	BinaryCollaboration/@timetoPerform Fork/@timetoPerform BusinessTransactionActivity/@timetoPerform	(v2 and v3) BusinessTransactionCharacteristics /@timeToPerform Not modeled for BinaryCollaboration	Value for this attribute MUST be used.
ebBP v2.x	BusinessCollaboration/TimetoPerform BinaryCollaboration/TimetoPerform MultiPartyCollaboration/TimetoPerform	This currently not modeled in a CPP or CPA for v2 or v3.	This element allows extensibility to link business transaction activities and documents to specific timing requirements. This requirement and the changes in ebBP v2.x are being assessed.

4.3.3.5.2.4.10 retryCount attribute

The **retryCount** attribute is of type integer. It specifies the maximum number of times the Business Transaction MAY be retried should certain error conditions (e.g., timeout waiting for the Receipt Acknowledgment signal) arise during its execution. Such retries MUST not be used when ebXML Reliable Messaging is employed to transport messages in the Business Transaction. In the latter case, retries are governed by the associated ReliableMessaging constructs under DocExchange.

4.3.3.5.2.5 ChannelId element

The ChannelId element identifies one or more DeliveryChannel elements that can be used for sending or receiving the corresponding action messages. Multiple ChannelId elements can be used to associate DeliveryChannel elements with different characteristics with the same service and action bindings. Note, this differs from the channelID attribute that identifies that channel.

For example, a Party that supports both HTTP [HTTP] and SMTP [SMTP] for sending the same action can specify different channelId attribute values for the corresponding channels. If using multiple DeliveryChannel elements, different EndPoint elements MUST be used, so that the receiving message handler can uniquely determine the DeliveryChannel element being used for this message exchange.

Details on DeliveryChannel element characteristics and the relationship to the transport, channel and document exchange are found later (See 4.3.7).

4.3.3.5.2.6 ActionContext and ActionContext2 elements

In v3, the ActionContext or ActionContext2 elements have been further enhanced to accommodate more business process definitions where collaboration activities are used. These elements are extensible using substitution groups whereby nested collaborations are further supported. The syntax is as follows:

```
<cppa:ActionContext2
        <!-- from complexType tns:ActionContextBaseType -->
        ...[namespace="##any" processContents="lax"]?
        <!-- from extension tns:ActionContext2Type -->
        businessTransactionActivity=tns:non-empty-string
        requestOrResponseAction=tns:non-empty-string
        role=tns:non-empty-string?
>
        <tns:CollaborationLevelHead/>?
        ...[namespace="##other" processContents="lax"]*
</cppa:ActionContext2>
```

In v2, the ActionContext concentrated support on collaborations only between two parties and activities (using the binaryCollaboration attribute) as follows:

```
<cppa:ActionContext
        <!-- from tns:ActionContextBaseType -->
        ...[namespace="##any" processContents="lax"]?
        <!-- from tns:ActionContextType -->
        binaryCollaboration=tns:non-empty-string
        businessTransactionActivity=tns:non-empty-string
        requestOrResponseAction=tns:non-empty-string
>
        <tns:CollaborationActivity/>?
        ...[namespace="##other" processContents="lax"]*
</ActionContext>
```

These constructs provide a mapping from the action attribute in the ThisPartyActionBinding element to any corresponding Business Process implementation-specific naming strategy such as those defined for ebBPSS or ebBP. Should these business process definitions be used, the ActionContext or ActionContext 2 element MUST be present.

Information in these constructs and the action attribute can facilitate message routing decisions by an implementation of a business process or application.

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x	TBD@name	ActionContext/@role	
ebBP v2.x	BusinessTransactionActivity/RequestingParty/@name BusinessTransactionActivity/RespondingParty/@name	ActionContext2/@role (for both)	

If business process definitions other than ebBPSS or ebBP are used, the action attribute of the parent ThisPartyActionBinding element and/or the [XMLSCHEMA-1] wildcard element within the ActionContext or ActionContext2 elements MAY be used to make routing decisions above the message handling level.

4.3.3.5.2.6.1.1 binaryCollaboration and businessTransactionActivity attributes

In v3, the collaboration activity structure is further enhanced and extended for nested collaboration and business transaction activities that exist in emerging business process definitions.

For v2 only, the binaryCollaboration attribute is a string that identifies the BinaryCollaboration for which the parent ThisPartyActionBinding is defined.

The businessTransactionActivity attribute is a string that identifies the Business Transaction for which the parent ThisPartyActionBinding is defined.

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x	BinaryCollaboration/@name	ActionContext/@binaryCollaboration (v2)	The value of the binaryCollaboration attribute MUST match the value of the name attribute of the BinaryCollaboration element as defined in the ebBPSS.

ebBPSS v1.x	BusinessTransactionActivity/@name	ActionContext/@businessTransactionActivity (v2)	The value of the businessTransactionActivity attribute MUST match the value of the name attribute of the BusinessTransactionActivity element, whose parent is the BinaryCollaboration referred to by the binaryCollaboration attribute.
ebBP v2.x	CollaborationActivity/@name ComplexBusinessTransactionActivity/@name	ActionContext2/CollaborationLevel/@collaborationLevelConstruct="CollaborationActivity" ActionContext2/CollaborationLevel/@collaborationLevelConstruct="ComplexBusinessTransactionActivity" (v3	
ebBP v2.x	BusinessTransactionActivity/@name	ActionContext2/@businessTransactionActivity (v3)	

How to handle v2.x ebBP for binaryCollaboration? How to handle specifying name attribute group?

4.3.3.5.2.6.1.2 requestOrResponseAction attribute

The requestOrResponseAction attribute is a string that identifies either the Requesting or Responding Business Activity for which the parent ThisPartyActionBinding is defined.

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x	RequestingBusinessActivity/@name	ActionContext/@requestorResponseAction	For request: The value of the requestOrResponseAction attribute MUST match the value of the name

	RespondingBusinessActivity/@name		attribute of the RequestingBusinessActivity element corresponding to the Business Transaction specified in the businessTransactionActivity attribute (See 4.3.3.5.2.6.1.1). For response: The value of the requestOrResponseAction attribute MUST match the value of the name attribute of the RespondngBusinessActivity element corresponding to the Business Transaction specified in the businessTransactionActivity attribute (See 4.3.3.5.2.6.1.1).
ebBP v2.x	RequestingBusinessActivity/@name RespondingBusinessActivity/@name	ActionContext2/@requestorResponseAction	TBD

See our draft correspondence on this table snippet in August 2007 – the mapping of, for example, RespondingBusinessActivity/@name => ThisPartyActionBinding/@action.

4.3.3.5.2.6.2 CollaborationActivity elements

As stated previously, v3 extends and expands the nested collaboration and activity structure to accommodate such functionality in emerging business process definitions. Collaboration levels can be expressed, extended and further allow for use of other business process definitions.

For v2, the CollaborationActivity element supports the ActionContext element by providing the ability to map nested BinaryCollaborations to the action attribute.

An example of the CollaborationActivity element in use is:

```
<tp:CollaborationActivity tp:name="Credit Check"/>
```

Process Definition	Process Reference	CPP/CPA Reference	Comments
ebBPSS v1.x	TBD	CollaborationActivity/@name Need to know if we show the base and head level	Maps nested collaborations to the action attribute as defined in ebBPSS.

		substitutions in this reference.	The CollaborationActivity element MUST be present when the BinaryCollaboration referred to by the binaryCollaboration attribute has a CollaborationActivity defined in the business process definition (See 4.3.3.5.2.6.1.1). The value of the name attribute MUST match the value of the name attribute of the CollaborationActivity within the BinaryCollaboration.
ebBP v2.x	CollaborationActivity/@name	ActionContext2/CollaborationLevel/@name	TBD

Attribute group?

4.3.4 MessagingCharacteristics element

The MessagingCharacteristics element can be used to describe the messaging attributes intended to be reused by the Party (see 4.2) and for action message exchange using the DeliveryChannel element.

In v3, the MessagingCharacteristics have been modularized and extended to allow for multiple protocols and bindings. The Messaging Characteristics element syntax is as follows:

```
<MessagingCharacteristics
       <!-- from attributeGroup tns:messagingAttributes -->
       syncReplyMode=("mshSignalsOnly"|"responseOnly"|"signalsAndResponse"
                     |"signalsOnly"|"none")?
       ackRequested=("always"|"never"|"perMessage")[default="perMessage"]?
       ackSignatureRequested=("always"|"never"|"perMessage")[default="perMessage"]?
       duplicateElimination=("always"|"never"|"perMessage")[default="perMessage"]?
       actor=("urn:oasis:names:tc:ebxml-msg:actor:nextMSH"
              |"urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH")?
       id=xsd:ID?
       idRef=xsd:IDREF?
       ...[namespace="##any" processContents="lax"]?
       <!-- from MessagingCharacteristicsBaseType -->
       ...[namespace="##any" processContents="strict"]?
/>
```

In v3, the extensible MessagingCharacteristics element has the following attributes as specified in messagingAttributes attributeGroup:

- syncReplyMode
- ackRequested
- ackSignatureRequested
- duplicateElimination
- actor

Attributes for ID and IDREFs have also been added to the type definition to allow ease of reference. Other attributes can also be defined by extension using the ##any attribute extensions.

MessagingCharacteristics elements that are immediate descendants of the root CollaborationProtocolProfile or a CollaborationProtocolAgreement MUST have a unique ID value for the id attribute, and will be referenced using the idRef attribute of a MessagingCharacteristics element.

A detailed description of the attributes in the ***messagingAttributes*** attributeGroup are found when the ***MessagingCharacteristics*** element is used for the ***DeliveryChannel*** element (see 4.3.7.1).

A channel can be created to assign a specific ***DeliveryChannel*** to return the acknowledgements asynchronously, in response to user message requests such as those using ebMS. The following rules explain how the messaging characteristics and attributes that interact with other information values such as defaultMSHChannelId and syncReplyMode that pertain to the channel used for returning acknowledgements:

If the rules are not retained, the text above should not be either. If retained move to MessagingCharacteristics on DeliveryChannel.

Need guidance on if and how to recapture this using the overrides on action binding that exist and the relationship with messaging characteristics.

4.3.5 Certificate element

Under the PartyInfo element**,** the Certificate element defines certificate information for use in this CPP. One or more Certificate elements can be provided for use in the various security functions in the CPP. An example of the Certificate element is:

```
<!-- Certificates used by the Contractor company -->
   <Certificate certId="CompanyA_SigningCert">
     <ds:KeyInfo>
       <ds:KeyName>CompanyA_SigningCert_Key</ds:KeyName>
     </ds:KeyInfo>
   </Certificate>
```

The Certificate element syntax is as follows:

```
<Certificate
        certId=xsd:ID
>
        <ds:KeyInfo/>
```

```
</Certificate>
```

The ds:KeyInfo element may contain a complete chain of certificates, but the leaf certificate is the Certificate element containing the key used in various asymmetric cryptographic operations. (The leaf certificate will be one that has been issued but has not been used to issue certificates.) If the leaf certificate has been issued by an intermediate certificate authority, the complete chain to the root certificate authority SHOULD be included because it aids in testing certificate validity with respect to a set of trust anchors.

The certId attribute is an [XML] ID that is referred to by a CertificateRef element associated with the CollaborationRole in the CPP. Here is an example of how a CertificateRef would refer to the Certificate element:

```
        <SigningCertificateRef certId="CompanyA_SigningCert"/>
```

4.3.5.1 ds:KeyInfo element

The ds:KeyInfo element defines the certificate information. The content of this element and any sub-elements are defined by the XML Digital Signature specification [XMLDSIG].

> *NOTE: Software for creation of* CPPs *and* CPAs *should recognize the* ***ds:KeyInfo*** *element and insert the sub-element structure necessary to define the certificate.*

4.3.6 SecurityDetails element

The SecurityDetails element defines a set of TrustAnchors and an associated SecurityPolicy for use in this CPP. One or more SecurityDetails elements can be provided for use in the various security functions in the CPP.

The SecurityDetails element syntax is as follows:

```
<SecurityDetails
        securityId=xsd:ID
>
        <tns:TrustAnchors/>?
        <tns:SecurityPolicy/>?
</SecurityDetails>
```

The SecurityDetails element contains a TrustAnchors element that identifies a set of certificates that are trusted by the Party. It also can contain a SecurityPolicy element.

The SecurityDetails element allows agreement to be reached on what root certificates will be used in checking the validity of the other Party's certificates. It can also specify policy regarding operation of the public key infrastructure. An example of the ***SecurityDetails*** element is:

```
<SecurityDetails securityId="CompanyA_TransportSecurity">
        <TrustAnchors>
               <AnchorCertificateRef certId="TrustedRootCertA1"/>
               <AnchorCertificateRef certId="TrustedRootCertA2"/>
        </TrustAnchors>
</SecurityDetails>
```

The securityId attribute is an [XML] ID that is referred to by a SecurityDetailsRef element within the CollaborationRole in the CPP.

An example of how a SigningSecurityDetailsRef would refer to the SecurityDetails element as shown in the previous section follows:

```
<SigningSecurityDetailsRef securityId="CompanyA_MessageSecurity"/>
```

4.3.6.1 TrustAnchors element

The TrustAnchors element contains one or more AnchorCertificateRef elements, each of which refers to a Certificate element (under PartyInfo) that represents a certificate trusted by this Party. These trusted certificates are used in the process of certificate path validation. If a certificate in question does not “chain” to one of this Party’s trust anchors, it is considered invalid.

The TrustAnchors element syntax is as follows:

```
<TrustAnchors>
        <AnchorCertificateRef>
                <tns:AnchorCertificateRef/>+
        </AnchorCertificateRef>
</TrustAnchors>
```

The TrustAnchors element eventually resolves into XMLDSIG KeyInfo elements. These elements may contain several certificates (a chain), and may refer to those certificates using the RetrievalMethod element to convey a reference to KeyInfo information that is stored at another location.

When there is a chain, the trust anchor is the “leaf” certificate with respect to the “root” issuing certificate authority (CA) certificate. The root CA will be a self-issued and self-signed certificate, and using the Issuer information and perhaps key usage attributes, the leaf certificate (“issued but not issuing” within the chain) can be determined. The chain is included for convenience in that validity checks typically will chain to a “root” CA.

Please note that the inclusion of a root CA in a chain does not mean that the root CA is being announced as a trust anchor. It is possible a PKI policy exists in which some, but not all, intermediate CAs are trusted. If a root CA were accepted as a trust anchor, all of its intermediate CAs, and all the certificates they issue, would be validated. Care should be taken if this is the intent.

4.3.6.2 SecurityPolicy element

The SecurityPolicy element enables the Party to specify its policy and compliance regarding specific components of its public key infrastructure. For example, it might stipulate revocation checking procedures or constraints related to name, usage, or path length.

In v3, the SecurityPolicy element has expanded from a placeholder (in v2) to an extensible container using substitution groups. The element syntax is as follows:

Do we want an example using ebMS 3?

```
<SecurityPolicy
        ...[namespace="##any" processContents="lax"]
```

```
/>
```

4.3.7 DeliveryChannel element

A delivery channel is a combination of a Transport element and a DocExchange element that describes the Party's Message communication characteristics. The CPP SHALL contain one or more DeliveryChannel elements, one or more Transport elements, and one or more DocExchange elements. Each delivery channel SHALL refer to any combination of a DocExchange element and a Transport element.

The same DocExchange element or the same Transport element can be referred to by more than one delivery channel. Two delivery channels can use the same transport protocol and the same document-exchange protocol and differ only in details such as communication addresses or security definitions.

Decide if here or earlier we expand on messaging characteristics extensibility (see comments defined below in 4.3.7.1)?

Figure 5 illustrates three delivery channels. The delivery channels have ID attributes with values of "DC1", "DC2", and "DC3". Each delivery channel contains one of both a transport and a document-exchange definition. Each definition also has an ID attribute whose value is shown in the figure. Note that delivery channel DC3 illustrates that a delivery channel can refer to the same definition used by other delivery channels but a different combination. In this case delivery channel DC3 is a combination of transport definition T2 (also referred to by delivery channel DC2) and document-exchange definition X1 (also referred to by delivery channel DC1).

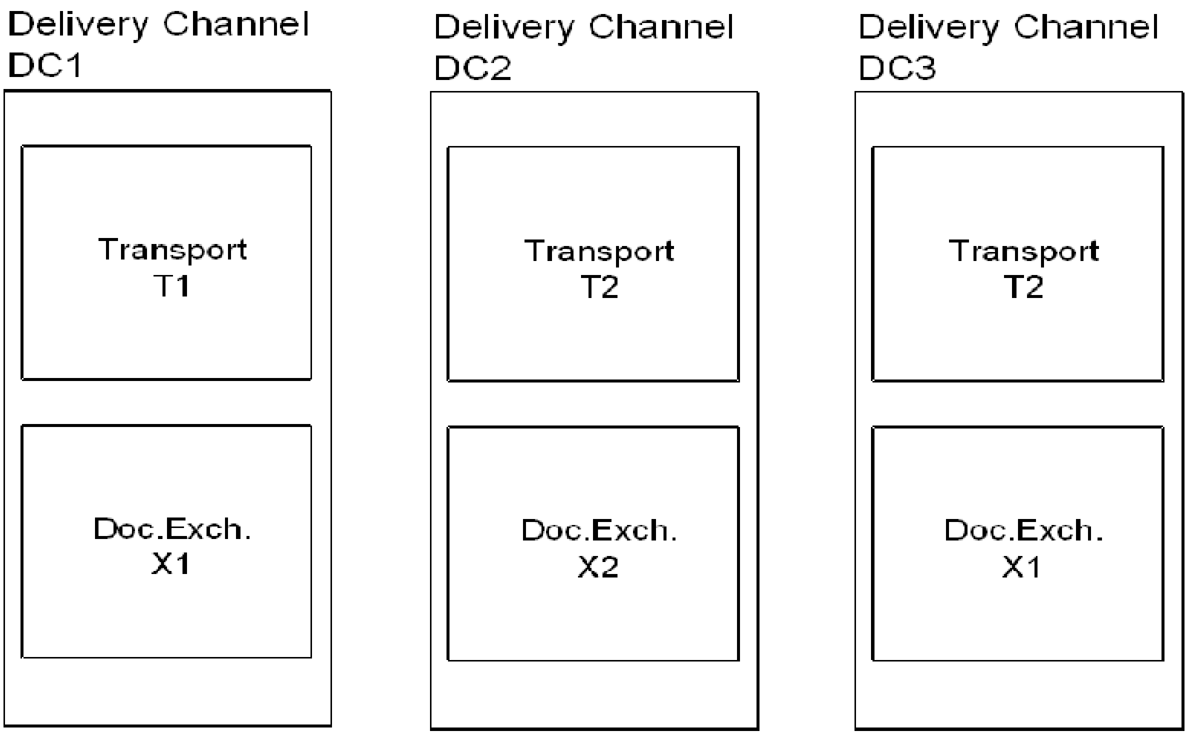


Figure 5 Three Delivery Channels

These elements have been enhanced in v3 to allow greater extensibility via substitution groups. The DeliveryChannel element syntax is as follows:

```
<cppa:DeliveryChannel
        channelId=xsd:ID
        transportId=xsd:IDREF
        docExchangeId=xsd:IDREF
>
        <tns:MessagingCharacteristics/>
</cppa:DeliveryChannel>
```

Each DeliveryChannel element identifies one Transport element and one DocExchange element that together make up a single delivery channel definition.   An example of a DeliveryChannel follows:

Verify text above.

```
<!-- An asynchronous delivery channel with ebMS pull mode-->
    <DeliveryChannel channelId="asyncChannelA1" transportId="transportA1"
      docExchangeId="docExchangeA1">
      <MessagingCharacteristics syncReplyMode="none" ackRequested="always"
        ackSignatureRequested="never" duplicateElimination="always"/>
    </DeliveryChannel>
```

The DeliveryChannel element has the following attributes:

- channelId: An [XML] ID attribute that uniquely identifies the DeliveryChannel element for reference, using IDREF attributes, from other parts of the CPP or CPA.

- transportId: An [XML] IDREF that identifies the Transport element that defines the transport characteristics of the delivery channel. It MUST have a value that is equal to the value of a transportId attribute of one of the extensible Transport elements elsewhere within the CPP document.

- docExchangeId: An [XML] IDREF that identifies the DocExchange element that defines the document-exchange characteristics of the delivery channel. It MUST have a value that is equal to the value of a docExchangeId attribute of one of the extensible DocExchange elements elsewhere within the CPP document.

4.3.7.1 MessagingCharacteristics element

On the ***DeliveryChannel*** element, the ***MessagingCharacteristics*** element describes the attributes associated with messages delivered over a given delivery channel. The collaborating Parties can stipulate that these attributes be fixed for all messages sent through the delivery channel, or they can agree that these attributes be variable on a “per message” basis.

This element has been enhanced in v3 to allow greater extensibility to enable message protocols using substitution groups. More details on this element are found where is it referenced for reuse in the root CollaborationProtocolProfile and/or CollaborationProtocolAgrement, in Section 4.3.4.

CPP and CPA composition tools and CPA deployment tools SHALL check the delivery channel definition (transport and document-exchange) for consistency with the defined attributes.

4.3.7.1.1 syncReplyMode attribute

syncReplyMode: An enumeration attribute comprised of these possible values: “mshSignalsOnly”, “signalsOnly”, “responseOnly”, “signalsAndResponse”, and “none”.

When syncReplyMode is present, it indicates what the sending application expects in a synchronous response. The delivery channel MUST be bound to a synchronous communication protocol such as HTTP when syncReplyMode is other than "none".

These values are defined as follows:

- mshSignalsOnly: Indicates the response returned (on the HTTP 200 response using HTTP) will only contain standalone message handling level messages such as Acknowledgment (for Reliable Messaging) and Error messages. All other application level responses are to be returned asynchronously (using a DeliveryChannel element determined by the service and action in question).

- signalsOnly: Indicates the response returned (on the HTTP 200 response using HTTP) will only include one or more Business signals found in or user defined in a business process specification such as [ebBPSS] or [ebBP], in addition to any piggybacked message handler level signals.

  A Business-response Message is not included. If a business process definition requires such, the Business-response Message MUST be returned asynchronously.

  If an Acceptance Acknowledgment signal is not required, then the Action element in the synchronously returned Message MUST be set to "ReceiptAcknowledgment" if ebMS is used. Otherwise, the Action element in the synchronously returned Message (which includes both Receipt Acknowledgment and Acceptance Acknowledgment signals) MUST be set to "AcceptanceAcknowledgment".

- responseOnly: Indicates any Business signals, even if they are indicated in the ProcessSpecification, are to be omitted and only the Business-response Message will be returned synchronously, plus any piggybacked message handler level signals. To be consistent where these attributes are used, the timeToAcknowledgeReceipt and timeToAcknowledgeAcceptance attributes under the corresponding BusinessTransactionCharacteristics element SHOULD be set to zero to indicate they are not to be used.

  The Action element in the synchronously returned Message is determined by the name of the action in the CPA that corresponds to the appropriate RespondingBusinessActivity in the business process definition.

- signalsAndResponse: Indicates the application will synchronously return the Business-response Message in addition to one or more Business signals, plus any piggybacked message handler level signals. Here, each signal and response bundled into the same message must appear as a separate MIME part (i.e., be placed in a separate payload container).

  To be consistent, the timeToAcknowledgeReceipt and timeToPerform (only for v2 CPP/CPA) attributes under the corresponding BusinessTransactionCharacteristics element SHOULD have identical values. The timeToAcknowledgeAcceptance attribute, if specified, SHOULD also have the same value as these two timing attributes.

The Action element in the synchronously returned Message is determined by the name of the action in the CPA that corresponds to the appropriate RespondingBusinessActivity in the business process definition.

NOTE: For HTTP 1.1 clients and servers, two HTTP requests and replies will have to be sent and received on the same connection. Implementations that implicitly assume a HTTP connection will be closed after a single synchronous request reply interchange will not be able to support the "signalsAndResponse" synchronous reply mode.

May need to revise on timeToPerform.

The Receipt Acknowledgment signal for the Business-response Message, sent from the request initiator back to the responder, if required by the business process definition, MUST also be delivered over the same synchronous connection.

- none: An implied default value in the absence of the syncReplyMode attribute. “none” indicates responses and business signals will not be returned synchronously (neither the Business-response Message nor any Business signal(s)). Here, all message handling and Business level messages will be returned as separate asynchronous messages.The ebMS **SyncReply** element is included in the SOAP Header whenever the **syncReplyMode** attribute has a value other than "none". If the delivery channel identifies a transport protocol that has no synchronous capabilities (such as SMTP), the **BusinessTransactionCharacteristics** element SHALL NOT have a **syncReplyMode** attribute with a value other than "none".

  When the value of the syncReplyMode attribute is other than "none", a synchronous delivery channel SHALL be used to exchange all messages necessary for conducting a business transaction activity. If the business process definition uses non-repudiation of receipt for the response message, then the initiator is expected to return a signed ReceiptAcknowledgment signal for the responder’s response message.

4.3.7.1.2 ackRequested attribute

The ackRequested attribute is an enumeration comprised of these possible values: “always”, “never” and “perMessage”.

- perMessage: Whether the AckRequested element in the SOAP Header is present or absent can be varied on a "per message" basis.

- always: Every message sent over the delivery channel MUST have an AckRequested element in the SOAP Header.

- never: Every message sent over the delivery channel MUST NOT have an AckRequested element in the SOAP Header.

If the ackRequested attribute is a value other than "never", then the ReliableMessaging element MUST be present under the corresponding DocExchange element to provide the necessary Reliable Messaging parameters.

4.3.7.1.3 ackSignatureRequested attribute

The ackSignatureRequested attribute determines how the signed attribute within the AckRequested element in the SOAP Header is to be set. This attribute is an enumeration comprised of these possible values: “always”, “never” and “perMessage”.

- perMessage: The signed attribute in the AckRequested element in the SOAP Header can be set to either “true” or “false” on a "per message" basis.

- always: Every message sent over the delivery channel that has an AckRequested element in the SOAP Header MUST have its signed attribute set to "true".

- never: Every message sent over the delivery channel that has an AckRequested element in the SOAP Header MUST have its signed attribute set to "false". If the ackRequested attribute is set to "never", the setting of the ackSignatureRequested attribute has no effect.

  NOTE: By enabling the use of signed Acknowledgment for reliably delivered messages, a different (weaker) form of non-repudiation of receipt can be supported. With this form, the Receipt Acknowledgment signal is not subject to schema validation on the payload prior to the return of the Acknowledgment.

The ackSignatureRequested attribute can be set independent of the value for the isNonRepudiationReceiptRequired attribute under the BusinessTransactionCharacteristics element. Thus, even if the original Process-Specification specifies that non-repudiation of receipt is to be performed, the CPP and/or CPA can override this requirement, set isNonRepudiationReceiptRequired to "false" and ackSignatureRequested to "always". There, the weaker form of non-repudiation of receipt has been achieved.

Let’s discuss this paragraph and whether or not we include it. If so, how to position it differently. In addition, this conflicts with RFC language in Section 6.4.3.6.1.2.1 (MUST be digitally signed).

4.3.7.1.4 duplicateElimination attribute

The duplicateElimination attribute determines whether the DuplicateElimination element within the MessageHeader element in the SOAP Header is to be present. In ebMS v3, duplicate elimination maps to reliable messaging constructs that provide delivery assurance. This attribute is an enumeration comprised of these possible values: "always", “never” and “perMessage”.

- always: Every message sent over the delivery channel MUST have a DuplicateElimination element in the SOAP Header or equivalent functionality.

- never: Every message sent over the delivery channel MUST NOT contain a DuplicateElimination element in the SOAP Header.

  If the duplicateElimination attribute is other than "never", the PersistDuration element MUST be present under the corresponding DocExchange element to provide the necessary persistent storage parameter. The default value "perMessage" indicates the DuplicateElimination element within the SOAP Header can be present or absent on a "per message" basis.

In v3, the modularization of the DocExchange element also allows extensibility when duration applies (See 4.3.10).

4.3.7.1.5 actor attribute

The actor attribute contains a URI used as the actor attribute value in the AckRequested element if present in the SOAP Header, as governed by the ackRequested attribute within the MessagingCharacteristics element in the CPA. This attribute is an enumeration of these possible values:

- "urn:oasis:names:tc:ebxml-msg:actor:nextMSH"
- "urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH"

Discuss if this changes for ebMS v3.

If the ackRequested attribute is set to "never", the setting of the actor attribute has no effect (See 4.3.7.1.2).

4.3.8 Transport element

The Transport element defines the Party's network communication capabilities. One or more Transport elements MUST be present in a CPP, each of which describes mechanisms the Party uses to send or receive messages, or both.

In v3, the Transport element is extended using substitution groups. This allows more extensibility for user-defined options for message transport and transport protocols. The Transport element syntax is as follows:

```
<cppa:Transport
       transportId=xsd:ID
>
       <!-- from TransportUserType -->
       ...[namespace="##other" processContents="lax"]*
       (
              <tns:TransportSender/>
       |
              <tns:TransportReceiver/>
       ){1,2}        PMW: Cardinality is specified by text, not schema.
</cppa:Transport>
```

An example of the use of this element is as follows:

```
<!-- Pull mode features for Sender -->
<Transport transportId="transportA1">
       <TransportSender>
              <TransportProtocol version="1.1"
                     method="HTTP response">HTTP</TransportProtocol>
              <AccessAuthentication>basic</AccessAuthentication>
              <Endpoint uri="https://www.CompanyA.com/ebxml3handler"
                     type="allPurpose"/>
              <!-- need transport security details here -->
              <TransportServerSecurity>
                     <TransportSecurityProtocol
                            version="1.1">TLS</TransportSecurityProtocol>
                     <ServerCertificateRef certId="CompanyA_ServerCert"/>
```

```
                      <ClientSecurityDetailsRef
                              securityId="CompanyA_TransportSecurity"/>
               </TransportServerSecurity>
        </TransportSender>
</Transport>
```

A Transport that contains both TransportSender and TransportReceiver elements is bi-directional, as it can be used for sending and receiving messages. If the Party communicates in synchronous mode (where replies are returned over the same TCP connections on which messages are sent), its CPP MUST provide a ServiceBinding that contains ActionBindings bound to a DeliveryChannel that uses a bi-directional Transport (See 4.3.7.1.1).

A Transport that contains either a TransportSender or a TransportReceiver element, but not both, is unidirectional. A unidirectional Transport can only be used for sending or receiving messages (not both) depending on which element is included.

A CPP contains as many Transport elements as are needed to fully express the Party's inbound and outbound communication capabilities. For example, if the Party can send and receive messages via HTTP and SMTP, its CPP would contain multiple Transport elements with its HTTP and SMTP.

A transportId attribute is an [XML] ID and refers to a Transport element elsewhere in the CPP. The transportId in the Transport element is referred to the corresponding transportId held for the DeliveryChannel.

An example of a DeliveryChannel that refers to the Transport element follows.

```
<tp:DeliveryChannel tp:channelId="channelA1"
        tp:transportId="transportA1"
        tp:docExchangeId="docExchangeA1">
        <tp:MessagingCharacteristics . . . />
</tp:DeliveryChannel>
```

4.3.8.1 TransportSender and TransportReceiver elements

The TransportSender element contains properties related to the sending side of a DeliveryChannel. The sender is determined by the Party producing the business data being delivered. Another party could actually make the TCP connection or write the stream. For TransportSender, the TransportProtocol element specifies the transport protocol used for sending messages. If present, Endpoint elements specify logical addresses where messages can be sent. If present, the AccessAuthentication element(s) specifies the type(s) of access authentication supported by the client. The TransportSender element syntax is:

```
<tns:TransportSender>
        <tns:TransportProtocol/>
        <tns:AccessAuthentication/>*
        <tns:Endpoint/>*
        (
               <tns:TransportClientSecurity/>
        |
               <tns:TransportServerSecurity/>
        )?
</tns:TransportSender>
```

The TransportReceiver element contains properties related to the receiving side of a DeliveryChannel. The receiver is determined by the Party consuming the business data being communicated. A receiver may or may not make a TCP connection when such a protocol is used. Its TransportProtocol element specifies the transport protocol used for receiving messages. If present, Endpoint elements specify logical addresses where messages can be received. The actual endpoint would be identified by the messaging transport protocol. If present, the AccessAuthentication element(s) indicates the type(s) of access authentication supported by the server. The TransportReceiver element syntax is:

```
<TransportReceiver>
        <tns:TransportProtocol/>
        <tns:AccessAuthentication/>*
        <tns:Endpoint/>*
        (
                <tns:TransportClientSecurity/>
        |
                <tns:TransportServerSecurity/>
        )?
</TransportReceiver>
```

Given the transport selected, either TransportClientSecurity or TransportServerSecurity can be used for TransportSender or TransportReceiver. If present, the TransportClientSecurity element defines the Party's provisions for client-side transport layer security. If present, the TransportServerSecurity element defines the Party's provisions for server-side transport layer security.

> *NOTE: See the note in Section 4.3.8.1.2 regarding the relevance of the **TransportServerSecurity** element when synchronous replies are in use.*

4.3.8.1.1 TransportProtocol element

The TransportProtocol element identifies a transport protocol the Party can use to send or receive Business data. This and other elements are derived from the ProtocolType complexType. The syntax is as follows:

```
<cppa:TransportProtocol
        version=tns:non-empty-string?
        method=tns:non-empty-string?
        mep=tns:non-empty-string?
        ...[namespace="##any" base="tns:non-empty-string" processContents="lax"]?
/>
```

In v3, additional and extensible attributes have been provided for the TransportProtocol as shown above, that supports many message transports and transport protocols.

> *NOTE: One goal for this specification is enable support for any transport capable of carrying MIME content using the vocabulary defined in this specification.*

4.3.8.1.2 AccessAuthentication element

If present, the **AccessAuthentication** element indicates the authentication mechanism that MAY be used by a transport server to challenge a client request and by a client to provide authentication information to a server. Its syntax is:

```
<AccessAuthentication>(basic|digest)</AccessAuthentication>
```

For example, [RFC2617] specifies two access authentication schemes for HTTP: "basic" and "digest". A client that supports both would have two **AccessAuthentication** elements, as shown in the example that follows. When multiple schemes are supported, the order in which they are specified in the CPP indicates the order of preference.

Shown below is an example of a TransportSender which provides an ordered choice of two authentication mechanisms:

```
<tp:TransportSender>
       <tp:TransportProtocol tp:version="1.1">HTTP</tp:TransportProtocol>
       <tp:AccessAuthentication>digest</tp:AccessAuthentication>
       <tp:AccessAuthentication>basic</tp:AccessAuthentication>
       <tp:TransportClientSecurity>
              ...
       </tp:TransportClientSecurity>
</tp:TransportSender>
```

A CPA will contain, for each TransportSender or TransportReceiver, only the agreed-upon AccessAuthentication elements. NOTE: For basic authentication, the userid and password values are configured through means outside of this specification.

4.3.8.1.3 TransportClientSecurity and TransportServerSecurity elements

Where present, the TransportClientSecurity element provides information about this Party's transport client needed by the other Party's transport server to enable a secure connection to be established between the two. It contains a TransportSecurityProtocol element, and MAY also contain client certificate, server security and encryption algorithm elements using the syntax as follows:

```
<cppa:TransportClientSecurity>
       <TransportSecurityProtocol/>
       <ClientCertificateRef/>?
       <ServerSecurityDetailsRef/>?
       <EncryptionAlgorithm/>*
</cppa:TransportClientSecurity>
```

Where present, the TransportServerSecurity element provides information about this Party's transport server needed by the other Party's transport client to enable a secure connection to be established between the two. It contains a TransportSecurityProtocol element, a ServerCertificateRef element, may contain a ClientSecurityDetailsRef element, and also may contain one or more EncryptionAlgorithm elements. A description of the EncryptionAlgorithm element is found later in this section (See 4.3.8.1.3.6).

```
<TransportServerSecurity>
       <tns:TransportSecurityProtocol>
       <tns:ServerCertificateRef>
       <tns:ClientSecurityDetailsRef/>?
       <tns:EncryptionAlgorithm/>*
</TransportServerSecurity>
```

In asynchronous messaging mode, the sender will always be a client to the receiver's server. In synchronous messaging mode, the message handling level reply (and possibly a bundled business signal and/or business response) is returned over the same connection of which the initial business message arrived.

In such cases, where the sender is the server, the receiver is the client and the connection already exists, the sender's TransportClientSecurity and the receiver's TransportServerSecurity elements SHALL be ignored.

4.3.8.1.3.1 TransportSecurityProtocol element

The TransportSecurityProtocol element identifies the transport layer security protocol supported by the parent Transport. Its syntax is the same as that for TransportProtocol, in Section 4.3.8.1.1.

For encryption, the protocol is TLS Version 1.0 [RFC2246], which uses public-key encryption. Appendix E of the TLS Version 1.0 specification [RFC2246] covers backward compatibility with SSL [SSL].

4.3.8.1.3.2 ClientCertificateRef element

For TransportClientSecurity element, the ClientCertificateRef element identifies the certificate used by the client's transport security module. The IDREF attribute certId identifies the certificate used by referring to the Certificate element (under PartyInfo) that has the matching ID attribute value. For example, a TLS-capable HTTP client uses this certificate to authenticate itself with receiver's secure HTTP server.

If present, the ClientCertificateRef element indicates mutual authentication between client and server (i.e., initiator and responder of the HTTP connection) MUST be performed. Its syntax is the same as found in Section 4.3.3.3 for the ApplicationCertificateRef element.

4.3.8.1.3.3 ServerSecurityDetailsRef element

For TransportClientSecurity element, the ServerSecurityDetailsRef element references the security details that this Party will apply to the other Party's server authentication certificate. Its syntax is the same as found in Section 4.3.3.4 for the ApplicationSecurityDetailsRef element.

4.3.8.1.3.4 ServerCertificateRef element

For TransportServerSecurity element, the ServerCertificateRef element, if present, identifies the certificate to be used by the server's transport security module.

As other elements, this element reuses the CertificateRefType complexType (See syntax in 4.3.3.3).

The IDREF attribute certId identifies the certificate to be used by referring to the Certificate element (under PartyInfo) that has the matching ID attribute value. A TLS-enabled HTTP server, for example, uses this certificate to authenticate itself with the sender's TLS client.

The ServerCertificateRef element MUST be present if the transport security protocol uses certificates. It MAY be omitted otherwise (e.g. if authentication is by password).

4.3.8.1.3.5 ClientSecurityDetailsRef element

For TransportServerSecurity element, the ClientSecurityDetailsRef element, if present, identifies references to the the trust anchors and security policy that this Party will apply to the other Party’s client authentication certificate.

For v3, the ClientSecurityDetailsRef element syntax is the same as found in Section 4.3.3.4 for the ApplicationSecurityDetailsRef element.

4.3.8.1.3.6 Encryption Algorithm

EncryptionAlgorithm elements MAY be included under the TransportClientSecurity or TransportServerSecurity element. Multiple elements are more useful in a CPP context, to announce capabilities or preferences; normally, a CPA will contain the agreed upon context.

When zero or more than one element is present in a CPA, the Parties agree to allow an automatic negotiation capability of the TransportSecurityProtocol element to determine the actual algorithm used.

The EncryptionAlgorithm element syntax is as follows:

```
<EncryptionAlgorithm
        minimumStrength=xsd:integer?
        oid=tns:non-empty-string?
        w3c=tns:non-empty-string?
        enumerationType=tns:non-empty-string?
/>
```

The minimumStrength attribute describes the effective strength the encryption algorithm MUST provide for “effective” or random bits. This value is less than the key length in bits when check bits are used in the key. For example, the 8 check bits of a 64-bit DES key would not be included in the count, and to require a minimum strength the same as that supplied by DES would be reported by setting minimumStrength to 56.

The elements' ordering will reflect the preference for algorithms. A primary reason for including this element is to permit use of the minimumStrength attribute; a large value for this attribute can indicate high encryption strength is desired or has been agreed upon for the TransportSecurityProtocol.

The oid attribute serves as a way to supply an object identifier for the encryption algorithm. The formal definition of OIDs comes from ITU-T recommendation X.208 (ASN.1), chapter 28; the assignment of the "top of the tree" is given in Appendices B, C, and D of X.208 (http://www.itu.int/POD/). Commonly used values (in the IETF dotted integer format) for encryption algorithms include:

- 1.2.840.113549.3.2 (RC2-CBC)
- 1.2.840.113549.3.4 (RC4 Encryption Algorithm)
- 1.2.840.113549.3.7 (DES-EDE3-CBC)
- 1.2.840.113549.3.9 (RC5 CBC Pad)
- 1.2.840.113549.3.10 (DES CDMF)

- 1.2.840, 1.3.14.3.2.7 (DES-CBC)

The w3c attribute serves as a way to supply an object identifier for the encryption algorithm. The definitions of these values are in the [XMLENC] specification. Expected values include:

- http://www.w3.org/2001/04/xmlenc#3des-cbc
- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes256-cbc

The enumerationType attribute specifies a way of interpreting the text value of the EncryptionAlgorithm element. This attribute is for identifying future algorithm identification schemes and formats.

For SSL and TLS, it is customary to specify cipher suite values as text values for the EncryptionAlgorithm element. These values include, but are not limited to:

- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA

Consult the original specifications for enumerations and discussions of these values (See 1.2 or 1.3).

4.3.8.2 Endpoint element

Endpoint elements MAY be provided for each TransportReceiver or TransportSender element. When present, each Endpoint specifies a logical address and an indication of the types of messages that can be received at that location.

In v3, the ***Endpoint*** element derives from a complexType using the following syntax:

```
<Endpoint
        uri=xsd:anyURI
        type=("login"|"request"|"response"|"error"|"allPurpose")[default="allPurpose"]?
/>
```

The uri attribute specifies a URI identifying the address of a resource. The value of the ***uri*** attribute SHALL conform to the syntax for expressing URIs as defined in [RFC2396].

The type attribute identifies the purpose of this endpoint. The enumerated values for type are:

- login: Used for the address for the initial Message between the two Parties.

- request: Used for a request Message.

- response: Used for a response Message.

- error: Enables error message to be received.

- intermediary: In v3, this value specifies the endpoint as an intermediary.

- allPurpose: If the type attribute is omitted, this is the default. When this value is used, no other Endpoint of any type should be included.

To enable error Messages to be received, each Transport element SHALL contain at least one endpoint of type "error", "response", or "allPurpose". The types of Endpoint element within each TransportReceiver or TransportSender elements MUST be mutually exclusive.

4.3.9 Transport protocols

In the following sections, we discuss the specific details of each supported transport protocol.

4.3.9.1 Hypertext Transfer Protocol

For HTTP [HTTP], the endpoint is a URI that SHALL conform to [RFC2396]. There MAY be one or more endpoints, whose use is determined by the application.

An example of an HTTP endpoint is:

```
<Endpoint uri="https://www.CompanyA.com/ebxml3handler" type="allPurpose"/>
```

The "request" and "response" endpoints can be dynamically overridden for a particular request or asynchronous response by application-specified URIs in Business documents exchanged under the CPA.

For a synchronous response, the "response" endpoint is ignored if present. A synchronous response is always returned on the existing connection, i.e. to the URI that is identified as the source of the connection.

4.3.9.2 Simple Mail Transfer Protocol

For SMTP[SMTP], use, Multipurpose Internet Mail Extensions[MIME] MUST be supported. For SMTP, the communication address is the fully qualified mail address of the destination Party as defined by [RFC2822].

An example of an SMTP endpoint is:

```
<tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
    tp:type="request"/>
```

NOTE: The SMTP Mail Transfer Agent (MTA) can encode binary data when the receiving MTA does not support binary transfer. In general, SMTP transfer may involve coding and recoding of Content-Transfer-Encodings as a message moves along a sequence of MTAs. Such changes can in some circumstances invalidate some signature types even though no malicious actions or transmission errors have occurred.

NOTE: SMTP by itself (without any authentication or encryption) is subject to denial of service and masquerading by unknown Parties. It is strongly suggested that those Parties who choose

SMTP as their transport layer also use adequate encryption and authentication either in the document-exchange or the transport layer such as [S/MIME].

NOTE: SMTP is an asynchronous protocol that does not guarantee a particular quality of service. A transport-layer acknowledgment (i.e. an SMTP acknowledgment) to the receipt of a mail Message constitutes an assertion on the part of the SMTP server that it knows how to deliver the mail Message and will attempt to do so at some point in the future. However, the Message is not hardened and might never be delivered to the recipient. Furthermore, the sender will see a transport-layer acknowledgment only from the nearest node. If the Message passes through intermediate nodes, SMTP does not provide an end-to-end acknowledgment. Therefore receipt of an SMTP acknowledgment does not guarantee that the Message will be delivered to the application and failure to receive an SMTP acknowledgment is not evidence that the Message was not delivered.

A reliable messaging protocol such as those supported in the ebMS SHOULD be used with SMTP.

4.3.9.3 File Transfer Protocol

The use of FTP for electronic business collaborations, though common, has not been standardized in too many specifc protocols. Ediint AS3 does provide a description of three sample patterns of configurations for exchanging a business message and its acknowledgement message, which is a Message Disposition Notification, or MDN, in Section 15 of [*RFC* 4823].

For each configuration, party A is sending a message containing a business document to party B and receives a MDN (signed or unsigned) in acknowledgement.

In one configuration, called peer to peer, each party has a server, but party A has a FTP server that is self-contained but local to the Ediint application, while party B has an embedded FTP server in its Ediint application. The TransportSender for Party A might then be:

```
<TransportSender>       <!--  A sends Message to B -->
        <TransportProtocol method="PUT">FTP</TransportProtocol>
        <Endpoint uri="ftps://embedded.B.com/AS3/Message"/>
        <TransportServerSecurity>
                <TransportSecurityProtocol version="1.0">TLS</TransportSecurityProtocol>
                <ServerCertificateRef certId="ID2005"/>
                <ClientSecurityDetailsRef securityId="ID2005"/>
                <EncryptionAlgorithm minimumStrength="168"
                        w3c="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
                >des_EDE3_CBC</EncryptionAlgorithm>
        </TransportServerSecurity>
</TransportSender>
```

And the TransportReceiver element might then be:

```
<TransportReceiver>       <!--  A gets MDNs from A off its local server -->
       <TransportProtocol  method="GET">FTP</TransportProtocol>
       <Endpoint uri="ftps://local.A.com/AS3/MDN"/>
       <TransportServerSecurity>
              <TransportSecurityProtocol version="1.0">TLS</TransportSecurityProtocol>
              <ServerCertificateRef certId="ID2005"/>
              <ClientSecurityDetailsRef securityId="ID2005"/>
              <EncryptionAlgorithm minimumStrength="168"
                     w3c="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
              >des_EDE3_CBC</EncryptionAlgorithm>
       </TransportServerSecurity>
</TransportReceiver>
```

For party B in a peer to peer configuration, the TransportSender for B sending a MDN might be:

```
<TransportSender>      <!--  B sends MDNs to A -->
       <TransportProtocol version="1.0" method="PUT">FTP</TransportProtocol>
       <Endpoint uri="ftps://local.A.com/AS3/MDN"/>
       <TransportServerSecurity>
              <TransportSecurityProtocol version="1.0">TLS</TransportSecurityProtocol>
              <ServerCertificateRef certId="ID2005"/>
              <ClientSecurityDetailsRef securityId="ID2005"/>
              <EncryptionAlgorithm minimumStrength="168"
                     w3c="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
              >des_EDE3_CBC</EncryptionAlgorithm>
       </TransportServerSecurity>
</TransportSender>
```

And for Party B receiving a Message, the TransportReceiver element might indicated by:

```
<TransportReceiver>  <!-- B picks up messages from its internal ftp receiver-->
       <TransportProtocol version="1.0"  method="GET">FTP</TransportProtocol>
       <Endpoint uri="ftps://embedded.B.com/AS3/Message"/>
       <TransportServerSecurity>
              <TransportSecurityProtocol version="1.0">TLS</TransportSecurityProtocol>
              <ServerCertificateRef certId="ID2005"/>
              <ClientSecurityDetailsRef securityId="ID2005"/>
              <EncryptionAlgorithm minimumStrength="168"
                     w3c="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
              >des_EDE3_CBC</EncryptionAlgorithm>
       </TransportServerSecurity>
</TransportReceiver>
```

[An Ediint collaboration gateway might have an implementation-dependent way of acquiring its data from its embedded server, so access to the data might not involve use of the GET method over the ftps communications transfer protocol.]

The RFC illustrates a quite different configuration for the use of FTP in collaboration, called “third-party hosting.” Here both A and B make use of GETs and PUTs to communicate. The case of A sending a message to B, with B returning a MDN might have the following Transport models.

A might send a message to B in accordance with:

```
<TransportSender>
        <TransportProtocol method="PUT">FTP</TransportProtocol>
        <Endpoint uri="ftps://hosted.com/B/AS3/Message"/>
        <TransportServerSecurity>
                <TransportSecurityProtocol version="1.0">TLS</TransportSecurityProtocol>
                <ServerCertificateRef certId="ID2005"/>
                <ClientSecurityDetailsRef securityId="ID2005"/>
                <EncryptionAlgorithm minimumStrength="168"
                        w3c="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
                >des_EDE3_CBC</EncryptionAlgorithm>
        </TransportServerSecurity>
</TransportSender>
```

And A might receive MDNs from B in accordance with:

```
<TransportReceiver>
        <TransportProtocol  method="GET">FTP</TransportProtocol>
        <Endpoint uri="ftps://hosted.com/A/AS3/MDN"/>
        <TransportServerSecurity>
                <TransportSecurityProtocol version="1.0">TLS</TransportSecurityProtocol>
                <ServerCertificateRef certId="ID2005"/>
                <ClientSecurityDetailsRef securityId="ID2005"/>
                <EncryptionAlgorithm minimumStrength="168"
                        w3c="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
                >des_EDE3_CBC</EncryptionAlgorithm>
        </TransportServerSecurity>
</TransportReceiver>
```

B might send MDNs to A by using the following parameters:

```
<TransportSender>
        <TransportProtocol version="1.0" method="PUT">FTP</TransportProtocol>
        <Endpoint uri="ftps://hosted.com/A/AS3/MDN"/>
        <TransportServerSecurity>
                <TransportSecurityProtocol version="1.0">TLS</TransportSecurityProtocol>
                <ServerCertificateRef certId="ID2005"/>
                <ClientSecurityDetailsRef securityId="ID2005"/>
                <EncryptionAlgorithm minimumStrength="168"
                        w3c="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
                >des_EDE3_CBC</EncryptionAlgorithm>
        </TransportServerSecurity>
</TransportSender>
```

And B could receive Messages from A using the Transport configuration:

```
<TransportReceiver>
        <TransportProtocol version="1.0" method="GET">FTP</TransportProtocol>
        <Endpoint uri="ftps://hosted.com/B/AS3/Message"/>
        <TransportServerSecurity>
                <TransportSecurityProtocol version="1.0">TLS</TransportSecurityProtocol>
                <ServerCertificateRef certId="ID2005"/>
                <ClientSecurityDetailsRef securityId="ID2005"/>
                <EncryptionAlgorithm minimumStrength="168"
                        w3c="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
                >des_EDE3_CBC</EncryptionAlgorithm>
        </TransportServerSecurity>
</TransportReceiver>
```

4.3.10 DocExchange Element

Messaging protocols for business collaboration generally tend to take distinctive approaches to the configurable information items needed or useful for collaboration protocol agreements. In addition, the configurable parameters for receivers and senders may differ. Therefore, the DocExchange element contains specific containers to indicate the protocol type and the protocol role (sender or receiver). The DocExchange element provides information that the Parties MUST agree on for their document exchanges. This information includes the messaging service properties (e.g. ediint). [May need to delete]

In v3, the DocExchange structures for message protocols are modularized where characteristics can be reused and/or combined, given the binding(s) which logically apply.

The v3 DocExchange element syntax is as follows:

```
<DocExchange
         docExchangeId=xsd:ID
>
        <DocExchangeBindingType
                version=tns:non-empty-string
        >
                (
                        <tns:DocExchangeModuleHead/>*
                |
                        <tns:DocExchangeModuleDurationHead/>*
                |
                        <tns:DocExchangeModuleStringHead/>*
                )+
        </DocExchangeBindingType>*
</DocExchange>
```

An example of the basic structure of the DocExchange element of the CPP follows using an ebXMLSenderBinding (See 4.3.10.1.1).

Need to discuss what to update below?

```
<tp:DocExchange tp:docExchangeId="docExchangeB1">
    <tp:ebXMLSenderBinding tp:version="2.0"><!-- 0 or 1 -->
        <tp:ReliableMessaging>                      <!-- 0 or 1 -->
        . . .
        </tp:ReliableMessaging>
        <tp:PersistDuration>                    <!-- 0 or 1 -->
        . . .
        </tp:PersistDuration>
        <tp:SenderNonRepudiation>                   <!-- 0 or 1 -->
        . . .
        </tp:SenderNonRepudiation>
        <tp:SenderDigitalEnvelope>                  <!-- 0 or 1 -->
        . . .
        </tp:SenderDigitalEnvelope>
        <tp:NamespaceSupported>                     <!-- 0 or more -->
        . . .
        </tp:NamespaceSupported>
    </tp:ebXMLSenderBinding>
    <tp:ebXMLReceiverBinding tp:version="2.0">      <!-- 0 or 1 -->
        <tp:ReliableMessaging>                      <!-- 0 or 1 -->
        . . .
        </tp:ReliableMessaging>
        <tp:PersistDuration>                    <!-- 0 or 1 -->
        . . .
        </tp:PersistDuration>
        <tp:ReceiverNonRepudiation>                 <!-- 0 or 1 -->
        . . .
        </tp:ReceiverNonRepudiation>
        <tp:ReceiverDigitalEnvelope>        <!-- 0 or 1 -->
        . . .
        </tp:ReceiverDigitalEnvelope>
        <tp:NamespaceSupported>                     <!-- 0 or more -->
        . . .
        </tp:NamespaceSupported>
    </tp:ebXMLReceiverBinding>
</tp:DocExchange>
```

The DocExchange element has a docExchangeId attribute that is an [XML] ID that provides a unique identifier that can be referenced from elsewhere within the CPP document.

The DocExchange element can be comprised of sender and receiver binding types such as web services, ebMS or ediint. It MUST have at least one child element. CPP and CPA composition tools and CPA deployment tools SHALL verify the presence of a child element.

The schema doesn't constrain that one child element be present. Is this a semantic constraint outside of schema?

> *NOTE: The document-exchange section can be extended to messaging services other than the ebXML Message service by adding additional xxxSenderBinding and xxxReceiverBinding elements and their child elements that describe the other services, where xxx is replaced by the name of the additional binding. An example is XMLPSenderBinding/XMLPReceiverBinding, which might define support for the future XML Protocol specification.*

Mapping to particular messaging protocols and to a given business process definition are found in the Appendices.

4.3.10.1 DocExchange Containers for Protocol Bindings and Characteristics

The DocExchange structures typically apply for, as indicated, senders and receivers. These containers derive from the abstract global element whereby the underlying elements in the substitution group are subtypes of that abstract head.

Currently three major types of messaging protocol families are found within the substitution group of DocumentExchangeBindingHead, and each family of module types has a sender and receiver role. These roles are viewed from the perspective of which party is the sender or receiver of business data. When business data is absent, then business or other protocol signal origin determines who is the sender. Note, this information may differ from the underlying communication transfer or transport protocol that first writes or connects.

Each protocol-family, protocol-role combination provides a versioned container for DocExchangeModule types. These modules are all members of the same substitution group for simplicity. Nevertheless, not all of the modules are appropriate for each DocExchangeBinding container.

The syntax for v3 DocExchange module types is as follows:

```
PMW:?? <!-- Distinct DocExchange Module Types -->
       DocExchangeModuleBaseType">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
       </xsd:complexType>
       DocExchangeModuleDurationType">
              base="xsd:duration">
              <xsd:anyAttribute namespace="##any" processContents="lax"/>
       DocExchangeModuleStringType">
              base="xsd:string">
              <xsd:anyAttribute namespace="##any" processContents="lax"/>
```

The extensibility model allows integration and use of bindings with their associated individual or combined characteristics such as persistence, policy constructs and duration. For example, ediint sender and receiver binding types are typically defined for use together. That protocol binding uses combined module characteristics such as compression, digital enveloping for a senders and receiver. In addition, ediint could also leverage non-repudiation as supporting protocol. Another example of a supporting protocol is reliable messaging.

The sections that follow detail the following:

1. Protocol binding types (i.e., ebXML, ediint and web services) for applicable role bindings (i.e., sender or receiver). Those role bindings are usually used together (sender and receiver). Other supporting message protocols can be used with these primary protocol binding types such as reliable messaging for ebMS v3 or non-repudiation.

2. Module characteristic(s) used for those protocol bindings

Each module has a type constructed to adhere to technical constraints for simple and complexTypes. The most typical modules for each container of protocol bindings follow. More detailed mapping between CPP/CPA and a messaging protocol are found in the Appendices.

4.3.10.1.1 ebXMLSenderBinding and ebXMLReceiverBinding elements

Derived from the DocExchangeBindingType, the ebXMLSenderBinding element describes properties for sending messages using ebMS ([ebMS] or [ebMS3]). The syntax that supports sender and receiver bindings is as follows:

```
<ebXMLSenderBinding>
        <DocExchangeBindingType/>+
</ebXMLSenderBinding>

<ebXMLReceiverBinding>
        <DocExchangeBindingType/>+
</ebXMLReceiverBinding>
```

The ebXMLReceiverBinding element describes properties related to receiving messages with the ebMS. The ebXMLReceiverBinding element can be also comprised with extensible capabilities through the DocExchangeBindingType. These capabilities typically could include reliable messaging (See 4.3.10.1.4), non-repudiation, a digital envelope and supported namespaces (See Error: Reference source not found).

A brief overview of the typical modular characteristics used by this protocol binding for ebMS v2 [ebMS] and [ebMS3] are:

DocExchange Module Lists for Message Protocol Bindings ebXMLSenderBinding and ebXMLReceiverBinding			
Binding	Binding Version	Usage Type	Applicable Usage Details
ebXMLSenderBinding	ebMS 2.0	Typical Modules	ReliableMessaging PersistDuration NamespaceSupported SenderNonRepudiation SenderDigitalEnvelope
ebXMLSenderBinding	ebMS 3.0	Typical Modules	ReliableMessaging PersistDuration NamespaceSupported SenderNonRepudiation SenderDigitalEnvelope SenderProcessingMode WSDLOperation SenderPolicy
ebXMLReceiverBinding	ebMS 2.0	Typical Modules	ReliableMessaging PersistDuration NamespaceSupported ReceiverNonRepudiation ReceiverDigitalEnvelope
ebXMLReceiverBinding	ebMS 3.0	Typical Modules	ReliableMessaging PersistDuration NamespaceSupported ReceiverNonRepudiation ReceiverDigitalEnvelope ReceivierProcessingMode

			WSDLOperation ReceiverPolicy

A CPA could be valid regardless whether these capabilities are used in this binding type.

For this specification, mappings between a CPA and either ebMS v2 or v3 are found in the Appendices (See Appendices XXX).

4.3.10.1.2 ediintSenderBinding and ediintReceiverBinding elements

This protocol binding supports Applicability Statement 2 (AS2) [RFC4130] and AS3 [RFC4823] functionality (See Appendix XXX). The syntax for sender and receiver bindings is as follows:

```
<EdiintSenderBinding>
       <DocExchangeBindingType/>+
</EdiintSenderBinding>

<EdiintReceiverBinding>
       <DocExchangeBindingType/>+
</EdiintReceiverBinding>
```

This protocol binding is typically supported by the use of compression and Message Disposition Notification (MDN) [RFC3798]elements detailed in this specification. It also leverages PKCS7 security [RFC3852] for MDN as shown in the mapping tables in the appendices. (See Appendix G).

A brief overview of the typical modular characteristics used by this protocol binding for AS1 [RFC3335], AS2 [RFC4130] and AS3 [RFC4823] are:

DocExchange Module Lists for Message Protocol Bindings ediintSenderBinding and ediintReceiverBinding			
Binding	Binding Version	Usage Type	Applicable Usage Details
ediintSenderBinding	1.0 1.1 1.2 etc.	Typical Modules	SenderCompression SenderRequestedMDNStyle SenderNonRepudiation SenderDigitalEnvelope
ediintSenderBinding	1.0, 1.1 1.2 etc.	Recommended	PersistDuration ReliableMessaging
ediintSenderBinding	1.0, 1.1 1.2 etc.	Optional	NamespaceSupported
ediintReceiverBinding	1.0, 1.1 1.2 etc.	Typical Modules	ReceiverCompression ReceiverAcceptedMDNStyle ReceiverNonRepudiation ReceiverDigitalEnvelope

ediintReceiverBinding	1.0, 1.1 1.2 etc.	Recommended	PersistDuration ReliableMessaging
ediintReceiverBinding	1.0, 1.1 1.2 etc.	Optional	NamespaceSupported

For this specification, a mapping between the CPA and these protocol bindings is found in the appendices (See Appendix XXX).

4.3.10.1.3 WSSenderBinding and WSReceiverBinding elements

The WSSenderBinding and WSReceiverBinding elements are typically used with WSDLOperation element, and may be used with policy constructs detailed in this specification. The syntax for these web services bindings is as follows:

```
<WSSenderBinding>
        <DocExchangeBindingType/>+
</WSSenderBinding>

<WSReceiverBinding>
        <DocExchangeBindingType/>+
</WSReceiverBinding>
```

A brief overview of the typical modular characteristics used by this protocol binding to leverage SOAP 1.1 [SOAP11] and SOAP 1.2 [SOAP12] are:

DocExchange Module Lists for Message Protocol Bindings WSSenderBinding and WSReceiverBinding			
Binding	Binding Version	Usage Type	Applicable Usage Details
WSSenderBinding	SOAP 1.1 SOAP 1.2	Typical Modules	WSDLOperation SenderPolicy
WSSenderBinding	SOAP 1.1 SOAP 1.2	Recommended	PersistDuration ReliableMessaging SenderNonRepudiation SenderDigitalEnvelope
WSSenderBinding	SOAP 1.1 SOAP 1.2	Optional	NamespaceSupported
WSReceiverBinding	SOAP 1.1 SOAP 1.2	Typical Modules	WSDLOperation ReceiverPolicy
WSReceiverBinding	SOAP 1.1 SOAP 1.2	Recommended	PersistDuration ReliableMessaging ReceiverNonRepudiation ReceiverDigitalEnvelope

WSReceiverBinding	SOAP 1.1 SOAP 1.2	Optional	NamespaceSupported

For this specification, a mapping between the CPA and these protocol bindings is found in the appendices (See Appendix H).

4.3.10.1.4 ReliableMessaging element

Where used for protocol binding types, the ReliableMessaging element specifies the properties of reliable Message exchange. The ReliableMessaging element syntax is as follows:

```
<cppa:ReliableMessaging>
        (
                <Retries>xsd:integer</Retries>
                <RetryInterval>xsd:duration</RetryInterval>
        )?  PMW: Cardinality specified by text; schema has each individually optional.
        <MessageOrderSemantics/>
        <ReliabilityGroup/>?
</cppa:ReliableMessaging>
```

An example of the ReliableMessaging element is:

```
<ReliableMessaging>
        <Retries>3</Retries>
        <RetryInterval>PT2H</RetryInterval>
        <MessageOrderSemantics>Guaranteed</MessageOrderSemantics>
        <ReliabilityGroup startGroup="true" terminateGroup="true"/>
</ReliableMessaging>
```

Semantics of reliable messaging are defined in other specifications such as ebMS v3 which can make use of other reliable messaging functionality. The protocol bindings that leverage reliable messaging within the scope of CPA are specified herein.

4.3.10.1.4.1 Retries and RetryInterval elements

The Retries and RetryInterval elements specify the permitted number of retries and the interval, expressed as an XML Schema [XMLSCHEMA-2]duration, between retries of sending a reliably delivered Message following a timeout waiting for the Acknowledgment.

The RetryInterval is the time between sending of the original message and the first retry, and, if applicable, between all subsequent retries. To increase success rates, the RetryInterval element allows retry deferral whereby any temporary conditions causing the error could be corrected.

The Retries and RetryInterval elements MUST either be included or omitted together. If they are omitted, the values of the corresponding quantities (number of retries and retry interval) are a local matter at each Party.

4.3.10.1.4.2 MessageOrderSemantics element

The MessageOrderSemantics element defines if ordering of messages is to be preserved. This element is an enumeration comprised of these possible values: Guaranteed and NotGuaranteed.

- Guaranteed: The Message MUST contain a MessageOrder element in the SOAP Header. This infers use of duplicate elimination. The PersistDuration element MUST also appear if MessageOrderSemantics is Guaranteed.

- NotGuaranteed: Converse to Guaranteed, the Message MUST NOT contain a MessageOrder element in the SOAP Header.

The presence of a MessageOrderSemantics element in the SOAP Header for Messages determines if the ordering of messages sent from the From Party needs to be preserved so that the To Party receives those messages in the order in which they were sent.

```
<MessageOrderSemantics>(Guaranteed|NotGuaranteed)</MessageOrderSemantics>
```

4.3.10.1.4.3 ReliabilityGroup element

In order to correlate conversations when reliability is enabled, the ReliabilityGroup element is used in this specification. The ReliabilityGroup element derived from the ReliabilityGroupType syntax is as follows:

```
<cppa:ReliabilityGroupType
        startGroup=xsd:boolean?
        terminateGroup=xsd:boolean?
>
        <tns:CorrelationPath/>?
</cppa:ReliabilityGroupType>
```

4.3.10.1.5 WSDLOperation element

The WSDLOperation element is typically used with the constructs defined in the following section. The element syntax is as follows:

```
<WSDLOperation
        version=tns:non-empty-string?
        operationRef=xsd:anyURI?
        messageRef=tns:wsdlRefListType?
        faultRef=tns:wsdlRefListType?
>
        ...[namespace="http://www.w3.org/2006/01/wsdl http://schemas.xmlsoap.org/wsdl/"
               processContents="lax"]+
</WSDLOperation>
```

This element also provides the portType or Interface when used in a CPP or CPA. The protocol bindings that leverage web services operations within the scope of CPA are specified herein.

4.3.10.1.6 ReceiverPolicy and SenderPolicy elements

To enable emerging models for expressing policies for domain-specific capabilities and characteristics for web services based-systems, new elements have been added in the extensible DocExchange structure. These elements are typically used with the WSDLOperation element. The syntax supports policy constructs for senders and receivers, with ***SenderPolicy*** and ***ReceiverPolicy*** elements. The syntax is as follows:

```
<ReceiverPolicy
         version=xsd:string?
>
        (
               ...[namespace="http://www.w3.org/ns/ws-policy" processContents="lax"]+
        |
               ...[namespace="http://schemas.xmlsoap.org/ws/2004/09/policy"
                       processContents="lax"]+
        )?
</ReceiverPolicy>
```

```
<SenderPolicy
         version=xsd:string?
>
        (
               ...[namespace="http://www.w3.org/ns/ws-policy" processContents="lax"]+
        |
               ...[namespace="http://schemas.xmlsoap.org/ws/2004/09/policy"
                       processContents="lax"]+
        )?
</SenderPolicy>
```

The protocol bindings that leverage web services policies within the scope of CPA are specified herein.

> *NOTE: Policy attachment functionality is implementation-specific.*

4.3.10.1.7 ReceiverCompression and SenderCompression elements

To enable compression of business data when necessary, a CompressionType complexType is used. These constructs are typically used with other protocol bindings particularly ediintSenderBinding and ediintReceiverBinding elements. The protocol bindings that leverage compression formats within the scope of CPA are specified herein. The syntax supports compression constructs for senders and receivers, with SenderCompression and ReceiverCompression elements. The syntax is as follows:

```
<SenderCompression
        version=tns:non-empty-string[default="1.1"]?
        mechanismType=tns:non-empty-string[default="zlib"]?
/>
<ReceiverCompression
        version=tns:non-empty-string[default="1.1"]?
        mechanismType=tns:non-empty-string[default="zlib"]?
```

```
/>
```

4.3.10.1.8 ReceiverProcessingMode and SenderProcessingMode elements

Further extensibility is provided for processing modes and characteristics including extension properties using ProcessingModeType complexType. The protocol bindings that leverage processing modes within the scope of CPA are specified herein. The syntax is as follows:

```
<ReceiverProcessingMode
        conformanceLevel=tns:non-empty-string
        deliveryErrorNotifyConsumer=xsd:boolean?
        processErrorNotifyConsumer=xsd:boolean?
        processErrorNotifyProducer=xsd:boolean?
        <!-- from DocExchangeModuleBaseType -->
        ...[namespace="##other" processContents="lax"]?
>
        <tns:AccessToken/>*
        <tns:MEP>xsd:anyURI</tns:MEP>
        <tns:MEPBinding>xsd:anyURI</tns:MEPBinding>
        <tns:SOAPVersion>tns:non-empty-string</tns:SOAPVersion>
        <tns:MPC>xsd:anyURI</tns:MPC>*
        <tns:ExtensionProperty/>*
</ReceiverProcessingMode>
```

4.3.10.1.8.1 AccessToken element

An AccessToken is a username and password. The type is declared so that validity can be checked when the information is present either as an encrypted element or as a decrypted content.

The AccessToken element syntax is as follows:

```
<AccessToken>
        (
                (
                        <tns:Username>tns:non-empty-string</tns:Username>
                        <tns:Password>tns:non-empty-string<tns:Password/>
                        <tns:Nonce>xsd:boolean[default="false"]</tns:Nonce>?
                        <tns:Digest>xsd:boolean</tns:Digest>?
                        <tns:CreatedTimestamp>xsd:boolean
                                [default="false"]</tns:CreatedTimestamp>?
                )
        |
                <xenc:EncryptedData/>
        )
</AccessToken>
```

4.3.10.1.8.2 MEP element

4.3.10.1.8.3 MEPBinding element

4.3.10.1.8.4 SOAPVersion element

4.3.10.1.8.5 MPC element

4.3.10.1.8.6 ExtensionProperty element

The ExtensionProperty element syntax is as follows:

```
<tns:ExtensionProperty
       name=tns:non-empty-string?
       description=tns:non-empty-string?
       expected=xsd:boolean?
       dataType=tns:non-empty-string?
/>
```

4.3.10.1.9 ReceiverAcceptedMDNStyle and SenderRequestedMDNStyle elements

In v3, receipt functionality is available for Message Disposition Notification (MDN) using the SenderRequestedMDNStyle and ReceiverAcceptedMDNStyle elements. This functionality is typically used with the ediintSenderBinding and ediintReceiverBinding protocol binding elements. The protocol bindings that leverage MDN styles for disposition within the scope of CPA are specified herein. The syntax is as follows:

```
<ReceiverAcceptedMDNStyle
       receiptType=tns:non-empty-string?
       mdnRequested=tns:perMessageCharacteristicsType?
       mdnDestination=tns:non-empty-string?
>
       <HashFunction>tns:non-empty-string</HashFunction>
</ReceiverAcceptedMDNStyle>
```

4.3.10.2 SenderNonRepudiation element

The SenderNonRepudiation element conveys the message sender’s requirements and certificate for non-repudiation. Non-repudiation both proves who sent a Message and prevents later repudiation of the contents of the Message. Nonrepudiation is based on signing the Message using XML Digital Signature [XMLDSIG]. The SenderNonRepudiation element syntax is as follows:

```
<SenderNonRepudiation>
       <!-- from tns:NonRepudiationBaseType -->
       <tns:NonRepudiationProtocol/>
       <tns:HashFunction/>
       <tns:SignatureAlgorithm/>+

       <!-- from SenderNonRepudiationType -->
       <tns:SigningCertificateRef/>
</SenderNonRepudiation>
```

If this element is omitted, the Messages are not digitally signed. An example of the SenderNonRepudiation is as follows:

```
<SenderNonRepudiation>
        <NonRepudiationProtocol version="1.1">WSS</NonRepudiationProtocol>
        <HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</HashFunction>
        <SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</SignatureAlgorithm>
        <SigningCertificateRef certId="CompanyA_SigningCert"/>
</SenderNonRepudiation>
```

4.3.10.2.1 NonRepudiationProtocol element

The **NonRepudiationProtocol** element identifies the technology used to apply a digital signature to a Message.

```
<NonRepudiationProtocol
        <!-- from ProtocolType -->
        version=tns:non-empty-string?
        method=tns:non-empty-string?
        mep=tns:non-empty-string?
        ...[namespace="##any" processContents="lax"]?
/>
```

4.3.10.2.2 HashFunction element

The HashFunction element identifies the algorithm used to compute the digest, or hash, of the Message being signed.

```
<tns:HashFunction>tns:non-empty-string</tns:HashFunction>
```

4.3.10.2.3 SignatureAlgorithm element

The SignatureAlgorithm element identifies the algorithm used to compute the value of the digital signature. Expected values include: RSA-MD5, RSA-SHA1, DSA-MD5, DSA-SHA1, SHA1withRSA and MD5withRSA.

```
<SignatureAlgorithm
        oid=tns:non-empty-string?
        w3c=tns:non-empty-string?
        enumerationType=tns:non-empty-string?
/>
```

When used within a CPP, the SignatureAlgorithm element can be repeated to indicate supported capabilities. The order within a repeated list of elements indicates comparative preference, from most to least preferred. When used within a CPA, the SignatureAlgorithm value is the agreed upon signature algorithm (See also 5.5.2).

> *NOTE: Implementations should be prepared for values in upper and/or lower case and with varying usage of hyphens and conjunctions.*

The oid attribute supplies an object identifier for the signature algorithm. The formal definition of OIDs exists in the ITU-T recommendation X.208 (ASN.1), chapter 28; the assignment of the "top of the tree" is given in Appendices B, C, and D of X.208 (http://www.itu.int/POD/). Commonly used values (in the IETF dotted integer format) for signature algorithms include:

```
1.2.840.113549.1.1.4 - MD5 with RSA encryption
1.2.840.113549.1.1.5 - SHA-1 with RSA Encryption
```

The w3c attribute also supplies an object identifier for the signature algorithm. The definitions of these values are found in the [XMLDSIG] or [XMLENC] specifications. Expected values for signature algorithms include:

```
http://www.w3.org/2000/09/xmldsig#dsa-sha1
http://www.w3.org/2000/09/xmldsig#rsa-sha1
```

The enumerationType attribute enables future means to interpret the text value of the SignatureAlgorithm element. This attribute identifies future signature algorithm identification schemes and formats.

4.3.10.2.4 SigningCertificateRef element

The SigningCertificateRef element identifies the certificate the sender uses for signing messages. Its required IDREF attribute, certId, refers to the Certificate element (under PartyInfo) that has the matching ID attribute value (See 4.3.5). The SigningCertificateRef element syntax is:

```
<SigningCertificateRef
        <!-- from CertificateRefType -->
        certId=xsd:IDREF
/>
```

4.3.10.3 ReceiverNonRepudiation element

The ReceiverNonRepudiation element conveys the message receiver’s requirements for non-repudiation. Non-repudiation both proves who sent a Message and prevents later repudiation of the contents of the Message. Non-repudiation is based on signing the Message using XML Digital Signature [XMLDSIG]. The ReceiverNonRepudiation element syntax is as follows:

```
<ReceiverNonRepudiation>
        <!-- from NonRepudiationBaseType -->
        <tns:NonRepudiationProtocol/>
        <tns:HashFunction/>
        <tns:SignatureAlgorithm/>+

        <!-- from ReceiverNonRepudiationType -->
        <tns:SigningSecurityDetailsRef/>?
</ReceiverNonRepudiation>
```

An example of ReceiverNonRepudiation is as follows:

```
<ReceiverNonRepudiation>
        <NonRepudiationProtocol>http://www.w3.org/2000/09/xmldsig#</NonRepudiationProtocol>
        <HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</HashFunction>
        <SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-sha1</SignatureAlgorithm>
        <SigningSecurityDetailsRef securityId="CompanyA_MessageSecurity"/>
</ReceiverNonRepudiation>
```

The receiver is responsible for non-repudiation of receipt (see 4.3.3.5.2.4.1).

If the ReceiverNonRepudiation element is omitted, the Messages are not digitally signed (See NonRepudationProtocol Error: Reference source not found).

The syntax for the elements common to both SenderNonRepudiation and ReceiverNonRepudiation (NonRepudiationProtocol, HashFunction, SignatureAlgorithm) is the same as described in the subsections of Error: Reference source not found.

4.3.10.3.1 SigningSecurityDetailsRef element

The SigningSecurityDetailsRef element identifies the trust anchors and security policy that this (receiving) Party will apply to the other (sending) Party's signing certificate. Its IDREF attribute, securityId, refers to the SecurityDetails element (under PartyInfo) that has the matching ID attribute value.

```
<tns:SigningSecurityDetailsRef
       <!-- from SecurityDetailsRefType -->
       securityId=xsd:IDREF
/>
```

4.3.10.4 Module Characteristics for DocExchange

In v3, modular characteristics associated with messaging protocols (protocol binding types) are used and enabled using abstract global element types for durations and strings, respectively, DocExchangeModuleDurationHead and DocExchangeModuleStringHead.

4.3.10.4.1 PersistDuration element

For duration, module characteristics are handled from an abstract global element. The PersistDuration element is the minimum length of time, expressed as an XML Schema [XMLSCHEMA-2] duration, that data from a reliably sent Message is persisted by a Messaging Service implementation that receives that Message to facilitate duplicate elimination. This duration also applies to response messages that are persisted to allow automatic replies to duplicate messages without their repeated application processing.

The PersistDuration element syntax is as follows:

```
<cppa:PersistDuration
       <!-- from xsd:extension DocExchangeModuleDurationType -->
       ...[namespace="##any" processContents="lax" base=""]?
>xsd:duration</cppa:PersistDuration>
```

See rules that govern the PersistDuration element are found in 4.3.10.1.4.2.

4.3.10.4.2 NamespaceSupported element

For string module characteristics, the NamespaceSupported element may be included when substituted for the abstract string module element and extension type (See 4.3.10.1) for a DocExchange element. Each occurrence of the NamespaceSupported element identifies one namespace supported by the messaging service implementation.

```
<cppa:NamespaceSupported
        location=xsd:anyURI
        version=tns:non-empty-string
        <!-- from extension tns:DocExchangeModuleStringType -->
        ...[namespace="##any" processContents="lax"]?
>xsd:string</cppa:NamespaceSupported>
```

The location attribute supplies a URI for retrieval of the schema associated with the namespace. The version attribute provides a version value for the namespace, when one exists. While the NamespaceSupported element can be used to list the namespaces that could be expected to be used during document exchange, the motivation is primarily for extensions, version variants, and other enhancements that might not be expected, or have only recently emerged into use.

For example, support for Security Assertion Markup Language [SAML2] would be defined as follows:

```
<tp:NamespaceSupported
tp:location="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-
assertion-27.xsd" tp:version="1.0">
http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-
27.xsd</tp:NamespaceSupported>
```

Need to determine how to handle URN in NamespaceSupported location:
urn:oasis:names:tc:SAML:2.0:assertion on OASIS site at: http://www.oasis-open.org/specs/index.php#samlv2.0.

In addition, the NamespaceSupported element can be used to identify the namespaces associated with the message body parts (See Section 4.4), and especially when these namespaces are not implicitly indicated through parts of the ProcessSpecification or when they indicate extensions of namespaces for payload body parts.

Verify with Dale this is still valid – seems reasonable.

4.3.10.4.3 Containers for Digital Envelope structures

For v3, modular constructs for digital envelopes are defined for sender and receiver structures through an extensibility model, in a manner compatible with v2.

4.3.10.4.3.1 ReceiverDigitalEnvelope element

The ReceiverDigitalEnvelope element provides the receiver's requirements for message encryption using the [DIGENV] digital-envelope method. Digital-envelope is a procedure in which the Message is encrypted by symmetric encryption (shared secret key) and the secret key is sent to the Message recipient encrypted with the recipient's public key. The ReceiverDigitalEnvelope element syntax is as follows:

```
<ReceiverDigitalEnvelope
        <!-- from DocExchangeModuleBaseType -->
        ...[namespace="##other" processContents="lax"]?
>
        <!-- from DigitalEnvelopeBase -->
        <tns:DigitalEnvelopeProtocol/>
        <tns:EncryptionAlgorithm/>+
        <!-- from ReceiverDigitalEnvelopeType -->
        <tns:EncryptionCertificateRef
                <!-- from CertificateRefType -->
                certId=xsd:IDREF
        />
</ReceiverDigitalEnvelope>
```

An example of the ReceiverDigitalEnvelope element is as follows:

```
<ReceiverDigitalEnvelope>
        <DigitalEnvelopeProtocol version="2.0">S/MIME</DigitalEnvelopeProtocol>
        <EncryptionAlgorithm>DES-CBC</EncryptionAlgorithm>
        <EncryptionCertificateRef certId="CompanyA_EncryptionCert"/>
</ReceiverDigitalEnvelope>
```

4.3.10.4.3.1.1 DigitalEnvelopeProtocol element

The DigitalEnvelopeProtocol element identifies the message encryption protocol to be used.

```
<tns:DigitalEnvelopeProtocol
        <!-- from ProtocolType -->
        version=tns:non-empty-string?
        method=tns:non-empty-string?
        mep=tns:non-empty-string?
        ...[namespace="##any" processContents="lax"]?
/>
```

4.3.10.4.3.1.2 EncryptionAlgorithm element

The EncryptionAlgorithm element identifies the encryption algorithm to be used (See syntax and attributes in 4.3.8.1.3.6 as these are reused).

```
<tns:EncryptionAlgorithm
        <!- from EncryptionAlgorithmType -->
        minimumStrength=xsd:integer?
        oid=tns:non-empty-string?
        w3c=tns:non-empty-string?
        enumerationType=tns:non-empty-string?
/>
```

4.3.10.4.3.1.3 EncryptionCertificateRef element

The EncryptionCertificateRef element identifies the certificate the sender uses for encrypting messages. IDREF attribute, certId refers to the Certificate element (under PartyInfo) that has the matching ID attribute value.

4.3.10.4.3.2 SenderDigitalEnvelope element

The SenderDigitalEnvelope element provides the sender’s requirements for message encryption using the [DIGENV] digital-envelope method. Digital-envelope is a procedure in which the Message is encrypted by symmetric encryption (shared secret key) and the secret key is sent to the Message recipient encrypted with the recipient's public key.

In v3, the SenderDigitalEnvelope element syntax is as follows:

```
<SenderDigitalEnvelope
        <!-- from DocExchangeModuleBaseType -->
        ...[namespace="##other" processContents="lax"]?
>
        <-- from tns:SenderDigitalEnvelopeType -->
        <EncryptionSecurityDetailsRef
                <!-- from tns:SecurityDetailsRefType -->
                securityId=xsd:IDREF
        />?
        <!-- from tns:DigitalEnvelopeBase -->
        <tns:DigitalEnvelopeProtocol/>
        <tns:EncryptionAlgorithm/>+
</SenderDigitalEnvelope>
```

An example of the SenderDigitalEnvelope element contains is as follows:

```
<SenderDigitalEnvelope>
        <DigitalEnvelopeProtocol version="2.0">S/MIME</DigitalEnvelopeProtocol>
        <EncryptionAlgorithm>DES-CBC</EncryptionAlgorithm>
        <EncryptionSecurityDetailsRef securityId="CompanyA_MessageSecurity"/>
</SenderDigitalEnvelope>
```

4.3.10.4.3.2.1 EncryptionSecurityDetailsRef element

The EncryptionSecurityDetailsRef element identifies the trust anchors and security policy that this (sending) Party will apply to the other (receiving) Party's encryption certificate. Its IDREF attribute, securityId, refers to the SecurityDetails element (under PartyInfo) with the matching ID attribute value.

4.3.11 OverrideMshActionBinding element

Under PartyInfo, the OverrideMshActionBinding element identifies when another DeliveryChannel is used, rather than the default, and the action which applies. The element syntax is as follows:

```
<OverrideMshActionBinding
        action=tns:non-empty-string
        channelId=xsd:IDREF
/>
```

The action attribute identifies the messaging handling action whose delivery is not to use the default DeliveryChannel, while the channelId attribute specifies the DeliveryChannel to be used.

4.4 SimplePart element

The SimplePart element provides a repeatable list of the constituent parts, primarily identified by the MIME content-type value.

```
<SimplePart
        <!-- from attributeGroup PackageGroup -->
        id=xsd:ID
        mimetype=tns:non-empty-string
        mimeparameters=tns:non-empty-string?

        <!-- from SimplePartType -->
        xlink:role
>
        <tns:NamespaceSupported/>*
</SimplePart>
```

From within the PackageGroup attribute Group, the id attribute value will be used later to reference this Message part when specifying how the parts are packaged into composites, should they be present. The mimetype attribute can provide actual values of content-type for the simple Message part being specified.

The attribute values may also make use of an asterisk wildcard, “*”, to indicate either an arbitrary top-level type, an arbitrary sub-type, or a completely arbitrary type, “*/*”. SimpleParts with wildcards in types can be used in indicating more open packaging processing capabilities. Is this still true?

The mimeparameters attribute provides the values of any significant MIME parameter (such as "type=application/xml") that is needed to understand the processing demands of the content-type (See 4.5.2).

These are examples of SimplePart elements:

```
<!-- SimplePart corresponding to the SOAP 1.2 Envelope -->
<SimplePart id="Notification" mimetype="application/soap+xml">
        <NamespaceSupported
                location="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200703/TBD"
                version="3.0">http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200703/
        </NamespaceSupported>
</SimplePart>

<!-- SimplePart corresponding to the SOAP 1.1 Envelope -->
<SimplePart id="MsgHdr" mimetype="text/xml">
        <NamespaceSupported
                location="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200703/TBD"
                version="3.0">http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200703/
        </NamespaceSupported>
</SimplePart>

<!-- SimplePart corresponding to a business signal -->
<SimplePart id="Exception" mimetype="application/xml">
        <NamespaceSupported
                location="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-
header-2_0.xsd"
                version="2.0"> http://www.oasis-open.org/committees/ebxml-msg/schema/msg-
header-2_0.xsd
        </NamespaceSupported>
</SimplePart>
```

4.4.1 NamespaceSupported element

The SimplePart element can have zero or more NamespaceSupported elements. Each of these identifies any namespace supported for the XML that is packaged in the parent simple body part.

SimplePart also has an xlink:role attribute which identifies some resource that describes the mime part or its purpose on use of this value within a messaging protocol. If present, it SHALL have a value that is a valid URI in accordance with the [XLINK] specification (See 1.2).

With the use of Packaging, it could alleviate the need to list all the namespaces used in a SimplePart. For example, when defining the SimplePart for a SOAP envelope, as part of an ebXML Message, it is not necessary to list all the namespaces.

If any unique extensions such as for security, new versions or atypical security extensions are present, for example, consider announcing these explicitly in the packaging. It is not, however, incorrect to list all namespaces used in a SimplePart, even where these they have been mandated by a given messaging protocol.

```
<tns:NamespaceSupported
        <!-- from NamespaceSupportedType -->
        location=xsd:anyURI
        version=tns:non-empty-string
        <!-- from DocExchangeModuleStringType -->
        ...[namespace="##any" processContents="lax"]?
/>
```

By convention, when a full listing of namespaces is supplied within a SimplePart element, the first NamespaceSupported element identifies the schema for the SimplePart while subsequent NamespaceSupported elements represent namespaces that are imported by that schema. Any additional NamespaceSupported elements indicate extensions.

> *NOTE: The explicit identification of imported namespaces is discretionary. Thus, the* CPP *and* CPA *could or may choose not to explicitly identify the ebMS namespace but omit the SOAP envelope and XML Digital Signature namespaces that are imported into the schema for the ebMS namespace.*

The same ***SimplePart*** element can be referenced from or reused in multiple ***Packaging*** elements.

4.5 Packaging element

The subtree of the Packaging element provides specific information about how the Message Header and payload constituent(s) are packaged for transmittal over the transport, including important information about what document-level security packaging is used and how security features have been applied. Typically the subtree under the Packaging element indicates the specific way in which constituent parts of the Message are organized. MIME processing capabilities are typically the capabilities or agreements described in this subtree. The Packaging element provides information about MIME content types, XML namespaces, security parameters, and MIME structure of the data exchanged between Parties.

```
<Packaging
        id=xsd:ID
>
        <ProcessingCapabilities
                parse=xsd:boolean
                generate=xsd:boolean
        />
        (
                <tns:CompositeList>
                        (
                                <tns:Encapsulation
```

```
                                     <!-- from attributeGroup PackageGroup -->
                                     id=xsd:ID
                                     mimetype=tns:non-empty-string
                                     mimeparameters=tns:non-empty-string?
                              >
                                     <tns:Constituent
                                            <!-- See definition below -->
                                     />
                              </tns:Encapsulation>
                       |
                              <tns:Composite/>
                      )+
               </tns:CompositeList>
        |
               <tns:Constituent/>
        )+
</Packaging>
```

This example of a Packaging element references the SimplePart elements discussed in Section 4.4:

Review for update

```
<!-- Simple ebXML S/MIME Packaging for application-based payload encryption -->
<tp:Packaging>
        <tp:ProcessingCapabilities tp:generate="true" tp:parse="true"/>
        <tp:CompositeList>
                <tp:Encapsulation
                        <!--I002 is the payload being encrypted -->
                        tp:id="I003"
                        tp:mimetype="application/pkcs7-mime"
                        tp:mimeparameters="smime-type=&quot;enveloped-data&quot"
                >
                        <Constituent tp:idref="I002"/>
                </tp:Encapsulation>
                <tp:Composite tp:id="I004"
                        <!-- I001 is the SOAP envelope. The ebXML message is made
                        up of the SOAP envelope and the encrypted payload. -->
                        tp:mimetype="multipart/related"
                        tp:mimeparameters="type=&quot;text/xml&quot;
version=&quot;1.0&quot"
                >
                        <tp:Constituent tp:idref="I001"/>
                        <tp:Constituent tp:idref="I003"/>
                </tp:Composite>
        </tp:CompositeList>
</tp:Packaging>
```

The Packaging element has an id attribute. It is referred to in the ThisPartyActionBinding (v2) or ActionBinding (v3) element, by using the IDREF attribute, packageId.

The child elements of the Packaging element are ProcessingCapabilities, CompositeList, and Constituent. The ProcessingCapabilities is found in each Packaging element, and a repeatable choice over CompositeList elements or Constituent element follows.

4.5.1 ProcessingCapabilities element

The ProcessingCapabilities element has two attributes, parse and generate, with Boolean values of either "true" or "false". Typically, these attributes will both have values of "true" to indicate the packaging constructs specified in the other child elements can be both produced and processed at the software Message service layer. At least one of the generate or parse attributes MUST be true.

4.5.2 Choice of CompositeList or Constituent element

A choice element exists in Packaging: a CompositeList or Constituent element. Need Dale to review entire section as there are many semantics I am unfamiliar with.

The CompositeList is a container for the specific way that the simple parts are combined into groups (MIME multiparts) or encapsulated within security-related MIME content-types.

When the CompositeList element is present, the content model for the CompositeList element is a repeatable sequence of choices of Composite or Encapsulation elements. The Composite and Encapsulation elements can appear intermixed as desired. The sequence in which the choices are presented is important because, given the recursive character of MIME packaging, composites or encapsulations can include previously mentioned composites (or rarely, encapsulations) in addition to the Message parts characterized within the SimplePart subtree. Therefore, the "top-level" packaging will be described last in the sequence. The Composite element has one child element, Constituent.

> *NOTE: In previous versions, a convention for indicating Packaging that consisted of one SimplePart was as follows: The mimetype value for the Composite and the mimetype value of the SimplePart referred to by the Constitutent were identical. In that case, there will be only one Constituent element in the Composite, and only one Composite in the CompositeList. The mimeparameters attribute on the Composite element would typically be omitted.*

The 3 attributes (id, mimetype and mimeparameters, exist in the PackageGroup attributeGroup in this specification while previous versions used the pkg.grp attributeGroup.

The mimetype attribute provides the MIME content-type for this Message part. This will be a MIME composite type, such as "multipart/related" or "multipart/signed". The id attribute refers to this composite if it needs to be mentioned as a constituent of some later element in the sequence.

The mimeparameters attribute could provides any significant MIME parameter needed to understand the processing demands of the content-type.

> *NOTE: In v2, this convention indicated Packaging of a single SimplePart* whereby *the Composite mimetype attribute could have simple mime types, such as "text/xml," "application/xml," and others as appropriate. For backward compatibility, this convention is being retained. This allows accessible conversion mechanisms for previous version instances. For version 3.0, this convention is deprecated, and the choice of a Constituent element is used instead.*

When only one Constituent element occurs, the Packaging consists of just the SimplePart to which the Constituent refers.

The CompositeList element SHALL be omitted from Packaging when no security encapsulations or composite multi-parts are used, and the Constituent is used instead.

```
<Constituent
        idref=xsd:IDREF?
        excludedFromSignature=xsd:boolean[default="false"]
        excludedFromDataConfidentiality=xsd:boolean[default="true"]
        minOccurs=xsd:nonNegativeInteger
        maxOccurs=xsd:nonNegativeInteger
        maxSizeInKBytes=xsd:nonNegativeInteger
>
        (
                <tns:SignatureTransforms>
```

```
                       <ds:Transform>+
               </tns:SignatureTransforms>?
               <tns:EncryptionTransforms>
                       <ds:Transform>+
               </tns:EncryptionTransforms>?
               <tns:ElementRef
                      signed=xsd:boolean
                      encrypted=xsd:boolean
                      signBeforeEncrypt=xsd:boolean[default="true"]
               >xsd:anyURI</tns:ElementRef>*
        )+
</Constituent>
```

The idref attribute value is that of the id attribute of a previous Composite, Encapsulation, or SimplePart element. This sequence of Constituents indicates both the contents and order of what is packaged within the current Composite or Encapsulation.

The excludeFromSignature attribute indicates this Constituent is not to be included as part of the message [XMLDSIG] signature. In other words, the signature generated by the message handling should not include a ds:Reference element to provide a digest for this Constituent of the message. This attribute is applicable only if the Constituent is part of the top-level Composite that corresponds to the entire message.

In this specification, two new attributes excludedFromDataConfidentiality and maxSizeInKBytes have been added. The excludedFromDataConfidentiality indicates this Constiuent is not to be included as part of the message for data confidentiality [XMLENC]. It has a boolean value and a default of “true”. The maxSizeInKBytes is optional to limit the size of the packaged contents.

The minOccurs and maxOccurs attributes serve to specify the value or range of values that the referred to item may occur within Composite. When unused, it is understood that the item is used exactly once.

4.5.2.1 Encapsulation element

For the CompositeList, the Encapsulation element is typically used to indicate MIME security mechanisms, such as [S/MIME] or Open-PGP [RFC2015]. A security body part can encapsulate a MIME part that has been previously characterized. For convenience, all such security structures are under the Encapsulation element, even when the data is not "inside" the body part. (In other words, the so-called clear-signed or detached signature structures possible with MIME multipart/signed are for simplicity found under the Encapsulation element.)

Another possible use of the Encapsulation element is to represent the application of a compression algorithm such as gzip [RFC1950] to some part of the payload, prior to its being encrypted and or signed.

The Encapsulation element reuses the PackageGroup attributesGroup.

In the attributeGroup, the mimetype attribute provides the value of the MIME content-type for this messagepart, such as "application/pkcs7-mime". The id attribute refers to this encapsulation if it needs to be mentioned as a constituent of some later element in the sequence. The mimeparameters attribute provides the values of any significant MIME parameter(s) needed to understand the processing demands of the content-type.

4.5.2.2 SignatureTransform and EncryptionTransforms elements

The Constituent element may contain at most one of each SignatureTransform and one EncryptionTransform child elements.

The SignatureTransform element is intended for use with XML Digital Signature [XMLDSIG]. When present, it identifies the transforms that must be applied to the source data before a digest is computed.

The EncryptionTransform element is intended for use with XML Encryption [XMLENC]. When present, it identifies the transforms that must be applied to a [XMLENC] CipherReference before decryption can be performed.

The SignatureTransforms element and the EncryptionTransforms element each contains one or more ds:Transform [XMLDSIG] elements.

4.5.2.3 ElementRef element

In this specification, an ElementRef element has been added where XPointer or XPath can be used to identify elements to be signed and/or encrypted.

4.6 Signature element

The Signature element enables the CPA to be digitally signed using technology that conforms with the XML Digital Signature specification [XMLDSIG]. The Signature element is the root of a subtree of elements used for signing the CPP. The Signature element syntax is as follows:

```
<Signature>
       <ds:Signature/>{1,3}
</Signature>
```

The Signature element contains up to three ds:Signature elements. This allows signature by both Parties involved, and an overall signature. The content of the ds:Signature element and any sub-elements are defined by the XML Digital Signature specification (See 5.8 for a detailed discussion).

> *NOTE: It is necessary to wrap the ds:Signature elements with a Signature element in the target namespace to allow for the possibility of having wildcard elements (with namespace="##other") within the CollaborationProtocolProfile and CollaborationProtocolAgreement elements. The content model would be ambiguous without the wrapping.*

The following additional constraints on ds:Signature are imposed:

- A CPP MUST be considered invalid if any ds:Signature element fails core validation as defined by the XML Digital Signature specification [XMLDSIG].
- Whenever a CPP is signed, each ds:Reference element within a ProcessSpecification element MUST pass reference validation and each ds:Signature element MUST pass core validation.

> *NOTE: In case a CPP is unsigned, software could validate the ds:Reference elements within ProcessSpecification elements and report any exceptions.*
>
> *Software for creation of* CPPs and CPAs *may recognize ds:Signature and automatically insert the element structure necessary to define signing of the CPP and CPA. Signature generation is*

outlined in Section 5.8.1.1. Details of the cryptographic process are outside the scope of this specification.

See non-normative note in Section 4.3.3.1 for a discussion of times at which validity tests could apply.

4.7 Comment element

The CollaborationProtocolProfile and CollaborationProtocolAgreement elements can contain Comment element(s). This element is a textual note serves any purpose the author desires and is extensible. In v2, the language of the Comment was identified by an xml:lang attribute, which can be used if desired in this specification. The Comment element syntax is as follows:

Dale verify that anyAttribute can be extended to use xml:lang.

```
<Comment
        ...[processContents="lax"]?
>xsd:anyType</Comment>
```

An example of the ***Comment*** element is as follows:

```
<tp:Comment xml:lang="en-US">This is a CPA between A and B</tp:Comment>
```

When a CPA is composed from two CPPs, all ***Comment*** elements from both CPPs SHALL be included in the CPA unless the two Parties agree otherwise.

5 CPA Definition

A Collaboration-Protocol Agreement (CPA) defines the capabilities that two Parties need to agree upon to enable them to engage in electronic Business for the purposes of the particular CPA. This section defines and discusses the details of the CPA.

Most of the elements in this section are described in detail in Section 4, "CPP Definition". In general, this section does not repeat that information. The discussions in this section are limited to those elements that are not in the CPP or for which additional discussion is needed in the CPA context.

5.1 CPA Structure

An example of a ***CollaborationProtocolAgreement*** is as follows:*PMW: example removed, in favor of top-level pseudocode in next section.*

5.2 CollaborationProtocolAgreement element

The ***CollaborationProtocolAgreement*** element is the root element of a CPA.   The ***cpaid*** attribute supplies a unique identifier for the document. The value of the ***cpaid*** attribute SHALL be assigned by one Party and used by both. The value of the ***cpaid*** attribute SHOULD be a URI. The value of the ***cpaid*** attribute SHALL be used as the value of an agreement reference or similar element in the message header, such as in ebMS (CPAid or AgreementRet for v2 and v3 respectively)>.

NOTE: Each Party could associate a local identifier with the ***cpaid*** attribute.

In addition, the ***CollaborationProtocolAgreement*** element has a ***version*** attribute. This attribute indicates the version of the schema to which the CPA conforms. The value of the ***version*** attribute SHOULD be a string such as "2_0a", "2_0b", “2.0”, “2.0a”, etc.

*NOTE: The method of assigning unique **cpaid** values is left to the implementation.*

The ***CollaborationProtocolAgreement*** element has [XML] Namespace [XMLNS] declarations that are defined in Section 4, "CPP Definition".

The ***CollaborationProtocolAgreement*** element syntax is as follows:

```
<CollaborationProtocolAgreement
        cpaid=tns:non-empty-string
        version=tns:non-empty-string
>
        <tns:Status/>
        <tns:Start/>
        <tns:End/>
        <tns:ConversationConstraints/>?
        <tns:PartyInfo/>{2,2}
        <tns:SimplePart/>+
        <tns:Packaging/>+
        <tns:BusinessTransactionCharacteristics/>*
        <tns:MessagingCharacteristics/>*
        <tns:Signature/>?
        <tns:Comment/>*
</CollaborationProtocolAgreement>
```

Child elements are:

- Status: Identifies the state of the process that creates the CPA,
- Start: Records the date and time that the CPA goes into effect,
- End: Records the date and time after which the CPA MUST be renegotiated by the Parties,
- ConversationConstraints element: Documents certain agreements about conversation processing,
- PartyInfo: One for each Party to the CPA,
- SimplePart elements,
- Packaging elements,
- BusinessTransactionCharacteristics elements: Relates business transactions and quality of service expectations to CPA,
- MessagingCharacteristics elements: Identifies characteristics of actions and bindings,
- Signature element: Provides for signing of the CPA using the XML Digital Signature[XMLDSIG] standard,
- Comment elements.

5.2.1 Status Element

The Status element records the state of the composition/negotiation process that creates the CPA. The The status value indicates the status of this CollaborationProtocolAgreement. Values are found in the simpleType,"statusValueType" Four values are specified: template, proposed, agreed, and signed. The values represent the state of the CPA in the agreement process.

A CPA in a template state may be associated with a NegotiationDescriptionDocument indicating what is eligible for change and possibly what choices exist for those changes.

Status element syntax is as follows:

```
<Status
        value=("agreed"|"signed"|"proposed"|"template")
/>
```

The Status element has a value attribute that records the current state of composition of the CPA. The allowed values are:

- proposed: The CPA is still being negotiated by the Parties,

- agreed: The contents of the CPA have been agreed to by both Parties,
- signed: The CPA has been signed by one or more of the Parties. This takes the form of a digital signature (See 5.6). The value of the Status element's value attribute is typically set to "signed" before the first Party signs. It is preferable to change the attribute value to "signed" prior to the first signature, and maintain it as "signed" for any subsequent signatures.

NOTE: The *Status* element could be used by a CPA composition and negotiation tool to assist it in the process of building a CPA.

5.3 CPA Lifetime: Start and End elements

The lifetime of the CPA is given by the Start and End elements. The element syntax is as follows:

```
<Start>xsd:dateTime</Start>

<End>xsd:dateTime</End>
```

An example of the Start and End elements is as follows:

```
<Start>2010-05-20T07:21:00Z</Start>
<End>2015-05-20T07:21:00Z</End>
```

5.3.1 Start element

The Start element specifies the starting date and time of the CPA. The Start element SHALL be a string value that conforms to the content model of a canonical dateTime type as defined in the XML Schema Datatypes Specification. For example, to indicate 1:20 pm UTC (Coordinated Universal Time) on May 31, 1999, a Start element would have the following value: 1999-05-31T13:20:00Z.

The Start element SHALL be represented as Coordinated Universal Time (UTC).

5.3.2 End element

The End element specifies the ending date and time of the CPA. The End element SHALL be a string value that conforms to the content model of a canonical dateTime type as defined in the XML Schema Datatypes Specification [XMLSCHEMA-2]. For example, to indicate 1:20 pm UTC (Coordinated Universal Time) on May 31, 1999, an End element would have the following value: 1999-05-31T13:20:00Z.

The End element SHALL be represented as Coordinated Universal Time (UTC).

What follows is CPA lifetime guidance:

Conversations: When the end of the CPA lifetime is reached, any Business Transactions still in progress SHALL be allowed to complete and no new Business Transactions SHALL be started. When all in-progress Business Transactions on each conversation are completed, the Conversation SHALL be terminated whether or not it was completed.

It might be unfeasible to wait for outstanding conversations to terminate before ending the CPA since there is no upper limit on conversations (See 5.4.1).

If a Business application defines a conversation as consisting of multiple Business Transactions, such a conversation MAY be terminated with no error indication when the end of the lifetime is reached.

> *NOTE: The run-time system could provide an error indication to the application if a conversation is terminated.*

An error indication SHOULD be returned to both Parties when a new Business Transaction is started under this CPA after the date and time specified in the End element.

How do we compare a conversation and a Business Collaboration?

Lifetime validation: When a CPA is signed, software for signing the agreements SHALL warn if any signing certificate’s validity expires prior to the proposed time for ending the CPA. The opportunity to renegotiate a CPA End value or to in some other way align certificate validity periods with CPA validity periods SHALL be made available. (Other ways to align these validity periods would include reissuing the signing certificates for a longer period or obtaining new certificates for this purpose.)

Signing software SHOULD also attempt to align the validity periods of certificates referred to within the CPA that perform security functions so as to not expire before the CPA expires. This alignment can occur in several ways including making use of ds:KeyInfo’s content model ds:RetrievalMethod so that a new certificate can be installed and still be retrieved in accordance with the information in ds:RetrievalMethod. If no alignment can be attained, signing software MUST warn the user of the situation that the CPA validity exceeds the validity of some of the certificates referred to within the CPA.

5.4 ConversationConstraints Element

The ConversationConstraints element places limits on the number of conversations under the CPA. The ConversationConstraints element syntax is as follows:

```
<ConversationConstraints
       invocationLimit=xsd:integer?
       concurrentConversation=xsd:integer?
/>
```

An example of this element follows:

```
<tp:ConversationConstraints tp:invocationLimit="100"
    tp:concurrentConversations="100"/>
```

5.4.1 invocationLimit and concurrentConversations attributes

The invocationLimit attribute defines the maximum number of conversations that can be processed under the CPA. When this number has been reached, the CPA is terminated and MUST be renegotiated. If no value is specified, there is no upper limit on the number of conversations and the lifetime of the CPA is controlled solely by the End element.

The invocationLimit attribute sets a limit on the number of units of Business that can be performed under the CPA. It is a Business parameter, not a performance one. A CPA expires whichever terminating condition (End or invocationLimit) is first reached.

The concurrentConversations attribute defines the maximum number of conversations that can be in process under this CPA at the same time. If no value is specified, processing of concurrent conversations is handled locally.

> *NOTE: The concurrentConversations attribute provides a parameter for the* Parties *to use when it is necessary to limit the number of conversations that can be concurrently processed under a particular* CPA*. For example, the back-end process might only support a limited number of concurrent conversations. If a request for a new conversation is received when the maximum number of conversations allowed under this* CPA *is already in process, an implementation could reject the new conversation or enqueue the request until an existing conversation ends. If no value is given for concurrentConversations, how to handle a request for a new conversation for which there is no capacity is handled locally.*

5.5 PartyInfo Element

The general characteristics of the PartyInfo element are discussed in Section 4.3.

The CPA SHALL have one PartyInfo element for each Party to the CPA. The PartyInfo element specifies the Parties' agreed terms for engaging in the Business Collaborations defined by the Process-Specification documents referenced by the CPA. If a CPP has more than one PartyInfo element, the appropriate PartyInfo element SHALL be selected from each CPP when composing a CPA.

Should we consider allowing more than 2 PartyInfo?

In the CPA, there SHALL be one or more PartyId elements under each PartyInfo element. The values of these elements are the same as the values of the PartyId or comparable elements in messaging service specification, such as ebMS. These PartyId elements SHALL be used within a To or From Head*er* element of an ebXML Message.

The ebMS v3 dictates this is the originating and destination party without RFC language. Can we dictate this last statement or change to SHOULD?

5.5.1 ProcessSpecification element

The ProcessSpecification element identifies the Business Collaboration that the two Parties have agreed to perform. The syntax of the ProcessSpecification element is described in Section 4.3.3.1. There can be one or more ProcessSpecification elements in a CPA. Each SHALL be a child element of a separate CollaborationRole element (See 4.3.3).

5.5.2 DocExchange Constraints for the CPA

Within a CPA, the SenderNonRepudiation and ReceiverNonRepudiation elements shall have exactly one SignatureAlgorithm element whose value is the agreed upon signature algorithm (See 4.3.10.2.3).

5.6 SimplePart element

The CollaborationProtocolAgreement element SHALL contain one or more SimplePart elements (See 4.4).

5.7 Packaging element

The CollaborationProtocolAgreement element SHALL contain one or more Packaging elements (See 4.5).

5.8 Signature element

A CPA document can be digitally signed by one or more of the Parties as a means of ensuring its integrity and to express the agreement. This is similar to a corporate officer's signature on a paper document. If signatures are being used to digitally sign a CPA or CPP document, then [XMLDSIG] SHALL be used to digitally sign the document.

The Signature element, if present, consists of up to three ds:Signature elements. The CPA can be signed by one or both Parties. Both Parties SHOULD sign the CPA. For signing by both Parties, one Party initially signs. The other Party then signs over the first Party's signature. The resulting CPA MAY then be signed by a notary. The ds:Signature element is the root of a subtree of elements used for signing the CPP.

The content of this element and any sub-elements are defined by the XML Digital Signature specification [XMLDSIG]. The following additional constraints on ds:Signature are imposed:

- A CPA MUST be considered invalid if any ds:Signature fails core validation as defined by the XML Digital Signature specification [XMLDSIG].

- Whenever a CPA is signed, each ds:Reference within a ProcessSpecification MUST pass reference validation and each ds:Signature MUST pass core validation (See suggestions in 4.3.3.1).

> NOTE: In case a CPA is unsigned, software could validate the ds:Reference elements within ProcessSpecification elements and report any exceptions.
>
> *Note: Software for creation of CPPs and CPAs SHALL recognize ds:Signature and automatically insert the element structure necessary to define signing of the CPP and CPA. Signature creation itself is a cryptographic process that is outside the scope of this specification.*

This is at least one place we have software guidance that is normative but is not a NOTE. Should we change any implementation guidance as the original sections say they are non-normative in nature for software implementation.

5.8.1 Persistent Digital Signature

If [XMLDSIG] is used to sign an ebXML CPP or CPA, the process defined in this section of the specification SHALL be used.

5.8.1.1 Signature Generation

Following are the steps that SHOULD be used to create a digital signature:These steps appear valid per the XMLDSIG, any changes?

1. Create a SignedInfo element, a child element of ds:Signature. SignedInfo SHALL have child elements SignatureMethod, CanonicalizationMethod, and Reference as prescribed by

[XMLDSIG].

2. Canonicalize and then calculate the SignatureValue over SignedInfo based on algorithms specified in SignedInfo as specified in [XMLDSIG].
3. Construct the Signature element that includes the SignedInfo, KeyInfo (RECOMMENDED), and SignatureValue elements as specified in [XMLDSIG].
4. Include the namespace qualified Signature element in the document just signed, following the last PartyInfo element.

5.8.1.2 ds:SignedInfo element

The ***ds:***SignedInfo element SHALL be comprised of zero or one ds:CanonicalizationMethod element, the ds:SignatureMethod element, and one or more ds:Reference elements.

5.8.1.3 ds:CanonicalizationMethod element

The ds:CanonicalizationMethod element as defined in [XMLDSIG], can occur one time, meaning that the element need not appear in an instance of a ds:SignedInfo element. The default canonicalization method that is applied to the data to be signed is [XMLC14N] in the absence of a ds:CanonicalizationMethods:CanonicalizationMethod element that specifies otherwise. This default SHALL also serve as the default canonicalization method for the CPP and CPA documents.

XMLDSIG indicates that canonicalizationmethod is occurrence of 1.

5.8.1.4 ds:SignatureMethod element

The ds:SignatureMethod element SHALL be present and SHALL have an Algorithm attribute. The value for the Algorithm attribute SHOULD be:

```
        "http://www.w3.org/2000/09/xmldsig#rsa-sha1"
```

This value SHALL be supported by conformant CPP or CPA software implementations.

5.8.1.5 ds:Reference element

The ds:Reference element for the CPP or CPA document SHALL have a URI attribute value of "" to provide for the signature to be applied to the document that contains the ds:Signature element (the CPA or CPP document). In accordance with XMLDSIG, the ds:Reference element for the CPP or CPA document can include a type attribute that has a value of:

```
"http://www.w3.org/2000/09/xmldsig#Object"
```

This attribute is purely informative and MAY be omitted. Implementations of software designed to author or process an CPA or CPP document SHALL be prepared to handle either case. The ds:Reference element can include the id attribute, type ID, by which this ds:Reference element is referenced from a ds:Signature element.

5.8.1.6 ds:Transform element

The ds:Reference element for the CPA or CPP document SHALL include a descendant ds:Transform element that excludes the containing ds:Signature element and all its descendants. This exclusion is achieved by means of specifying the ds:Algorithm attribute of the Transform element (of [XMLDSIG] as:

```
"http://www.w3.org/2000/09/xmldsig#enveloped-signature" determine if example changes are
needed.
```

For example:

```
<ds:Reference ds:URI="">
       <ds:Transforms>
              <ds:Transform
ds:Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
       </ds:Transforms>
       <ds:DigestMethod
 ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <ds:DigestValue>...</ds:DigestValue>
</ds:Reference>
```

The ds:Transform element SHALL include a ds:Algorithm attribute that has a value of:

```
            http://www.w3.org/2000/09/xmldsig#enveloped-signature
```

When digitally signing a CPA, each Party SHOULD sign the document as outlined.

When the two Parties sign the CPA, the first Party that signs the CPA SHALL sign only the CPA contents, excluding their own signature. The second Party SHALL sign over the contents of the CPA as well as the ds:Signature element that contains the first Party's signature. If necessary, a notary can then sign over both signatures.

5.9 Comment element

The CollaborationProtocolAgreement element may contain Comment element(s) (See 4.7).

5.10 Composing a CPA from Two CPPs

This section discusses normative issues in composing a CPA from two CPPs.

5.10.1 ID Attribute Duplication

In composing a CPA from two CPPs, there is a hazard that ID attributes from the two CPPs might have duplicate values. When a CPA is composed from two CPPs, a test for duplicate ID attribute values SHALL occur. If a duplicate ID attribute value is present, one of the duplicates SHALL be given a new value and the corresponding IDREF attribute values from the corresponding CPP SHALL be corrected.

> *NOTE: A party can seek to prevent ID/IDREF reassignment in the CPA by choosing ID and IDREF values which are likely to be unique among its trading partners. For example, the following Certificate element found in a CPP has a certId attribute is generic and could conflict with a certId attribute found in a collaborating party's CPP:*

```

<tp:Certificate tp:certId="EncryptionCert"><ds:KeyInfo/></tp:Certificate>
```

To prevent reassignment of this ID (and its associated IDREFs) in a CPA, a better choice of certId in Company A's CPP would be:

```
<!-- Certificates used by the Contractor company -->
<Certificate certId="CompanyA_SigningCert">
       <ds:KeyInfo>
```

```
               <ds:KeyName>CompanyA_SigningCert_Key</ds:KeyName>
       </ds:KeyInfo>
</Certificate>
```

5.11 Modifying Parameters of the Process-Specification Document Based on Information in the CPA

A Process-Specification document contains a number of parameters, expressed as XML attributes. An example is the security attributes that are counterparts of the attributes of the CPA BusinessTransactionCharacteristics element. The values of these attributes can be considered to be default values or recommendations. When a CPA is created, the Parties could accept the recommendations in the Process-Specification or agree on values of these parameters that better reflect their needs. Any overrides SHOULD be related to quality of service aspects for specific purposes.

Need to discuss this as there may be caveats to Override. Is this language sufficient?

When a CPA is used to configure a run-time system, choices specified in the CPA MUST always assume precedence over choices specified in the referenced Process-Specification document. In particular, all choices expressed in a CPA’s BusinessTransactionCharacteristics and Packaging elements MUST be implemented as agreed to by the Parties.   These choices SHALL override the default values expressed in the Process-Specification document. The process of installing the information from the CPA and Process-Specification document MUST verify all resulting choices are mutually consistent and, if not, MUST signal an error.

> *NOTE: There are several ways of overriding the information in the Process-Specification document by information from the CPA. For example:*
>
> - *The CPA composition tool can create a separate copy of the Process-Specification document. The tool can then directly modify the Process-Specification document with information from the CPA. An advantage of this method is that the override process is performed entirely by the CPA composition tool.*
>
> - *A CPA installation tool can dynamically override parameters in the Process-Specification document using information from the corresponding parameters in the CPA at the time the CPA and Process-Specification document are installed in the Parties' systems. This eliminates the need to create a separate copy of the Process-Specification document.*
>
> - *Other possible methods might be based on XSLT transformations of the parameter information in the CPA and/or the Process-Specification document.*
>
> - *Extensibility mechanisms may be used for the Process-Specification document that reflect the agreement of the parties.*
>
> *Of the options above, the creation of a distinct Process-Specification document is preferred.*

6 Provisions for CPP and CPA Extensibility

In v2, several areas of extensibility were assumed. The extensibility provisions of CPP/CPA v3 are explicitly described and illustrated. These illustrations can serve as patterns for future extensibility and use; separate specifications can be created to document those new extensions and their specific semantic contents.

Suggest this be in the introductory sections instead.

6.1 Goals for Extensibility Framework

The ebXML component design intended each functional specification be aligned with others (highly aligned), but work standalone or be capable of working with emerging technologies or alternative specifications (loosely coupled). For CPPA, two main types of alternative specifications need to be considered.

In section 4.3.3.1, it was noted that when using alternative Business Collaboration descriptions, it is necessary to reconsider affected elements such as the roles, actions and business transaction functionality.

Likewise, in section 4.3.10, other messaging alternatives (other than ebXML Message Service [ebMS] for example) are considered. Additional messaging services have been defined and others can be defined using the convention *SenderBinding or *ReceiverBinding (such as WSSenderBinding). Customized content models can describe these services The “*” can be replaced by a suitable identifier for the additional binding.

The basic approach in v3 of this specification is to use substitution groups to support these extensions, and the v3 schema differs primarily from the v2 schema in that it implements a substitution group approach for making CPPA extensibility explicit.

New abstract elements (substitution group heads) are introduced to serve as the global elements that are referenced when declaring extension elements. The extensibility approach taken in CPPA v3 is new substitution group members are not required to have types that are subtypes of v2 elements. Because the members of a substitution group have to satisfy a subtype constraint [XMLSCHEMA-2] that requires that the types of substitution group members must be subtypes of the substitution group head element’s type, the newly created substitution group head elements make minimal commitments with content models of an empty sequence. As a result, types of substitution group elements can be subtypes through derivation by extension.

The result of this approach is to preserve the component layering in the CPP and CPA used to organize the areas of user selectable functionality in a collaboration protocol.

To maintain compatibility with v2, the original elements are redeclared as members of the newly introduced substitution groups in v3. This approach allows previous instances of CPPs and CPAs to validate against the current schema, once the instances are modified to be placed in the new target namespace and attribute prefixes modified to conform to the new unqualified default. The substation group head element is also referenced within an including type to indicate where in the overall CPP or CPA the extensions can occur.

6.2 Principal Substitution Groups

In principle, any global element declaration can be a head for a substitution group. However, as previously explained, both previous v2 elements and new v3 elements are placed into new substitution groups with head elements that are abstract (and cannot be instantiated directly).

These new substitution group heads are as follows:

- ProcessSpecificationHead, ActionContextHead, CollaborationLevelHead
- ActionBindingBaseHead, BusinessTransactionCharacteristicsHead
- DocExchangeBindingHead
- DocExchangeModuleHead, DocExchangeModuleDurationHead, DocExchangeModuleStringHead
- MessagingCharacteristicsHead, SecurityPolicyHead
- TransportUserHead

The substitution groups formed around each of the previous abstract elements are as follows:

6.2.1 ProcessSpecificationHead

The group referencing ProcessSpecificationHead contains ProcessSpecification.

Alternative business process specification languages can add an element to the ProcessSpecificationHead substitution group in order to provide information that identifies the type of process description and references to the specific description that pertains to a given business message to be sent.

6.2.2 ActionContextHead

The group referencing ActionContextHead contains ActionContext, ActionContext2.

Alternative business process specification languages can add an element to the ActionContextHead substitution group in order to provide information that identifies the public or private process context in which a given business message is sent.

6.2.3 CollaborationLevelHead

The group referencing CollaborationLevelHead contains CollaborationActivity, CollaborationLevel.

Alternative business process specification languages or defintions can add an element to the CollaborationLevelHead substitution group in order to provide information that identifies the nesting level of the components capturing public or private process order within which a given business message is sent. This substitution group was introduced to handle ebBP 2.x use of either bp2:CollaborationActivity or bp2:ComplexBusinessTransactionActivity. The version 2 CPPA element, CollaborationActivity, has been generalized as a result of the refactoring of its type allowing it to be included in the CollaborationLevelHead substitution group.

6.2.4 ActionBindingBaseHead

The group referencing ActionBindingBaseHead contains ThisPartyActionBinding, ActionBinding.

Alternative business process protocols can add an element to the ActionBindingBaseHead substitution group in order to create a new way to organize information about the capabilities and requirements for a kind of Action binding. In v3, this extension capability is used to add a simplified, less deeply nested, organization for the components in an Action binding. The extension point context is found within the ServiceBindingType within a choice option.

6.2.5 BusinessTransactionCharacteristicsHead

The group referencing BusinessTransactionCharacteristicsHead contains BusinessTransactionCharacteristics.

Alternative models of business quality of service features can add an element to the BusinessTransactionCharacteristicsHead substitution group in order to declare a different set of business quality of service features than used within UNCEFACT Modeling Methodology (UMM) framework.

6.2.6 DocExchangeBindingHead

The group referencing DocExchangeBindingHead contains ebXMLReceiverBinding, ebXMLSenderBinding, WSReceiverBinding, WSSenderBinding, EdiintReceiverBinding, and EdiintSenderBinding.

Alternative collaboration protocol families can add an element to the DocExchangeBindingHead substitution group in order to provide demarcate a distinctive approach to business collaboration messaging configuration. Introducing a new collaboration protocol family will typically occur when several distinctive modules for that protocol need to be added to the substitution group for DocExchangeModuleHead.

6.2.7 DocExchangeModuleHead

The group referencing DocExchangeModuleHead contains WSDLOperation, ReceiverPolicy, SenderPolicy, ReceiverCompression, SenderCompression, ReceiverProcessingMode, SenderProcessingMode, ReceiverAcceptedMDNStyle, SenderRequestedMDNStyle, ReliableMessaging, SenderNonRepudiation, ReceiverNonRepudiation, ReceiverDigitalEnvelope, and SenderDigitalEnvelope.

Alternative collaboration protocol families or new versions of existing collaboration families can add elements to the DocExchangeModuleHead substitution group in order to add distinctive new capabilities or configuration parameters needed for the business collaboration protocol.

6.2.8 DocExchangeModuleDurationHead

The group referencing DocExchangeModuleDurationHead contains PersistDuration.

6.2.9 DocExchangeModuleStringHead

The group referencing DocExchangeModuleStringHead contains NamespaceSupported.

Alternative collaboration protocol families or new versions of existing collaboration families can add elements to the DocExchangeDurationHead or DocExchangeModuleStringHead substitution groups in order to add distinctive new capabilities or configuration parameters needed for the business collaboration protocol. These substitution groups are needed because the XSD type system does not allow derivations to exist between complexTypes and simpleTypes.

6.2.10 MessagingCharacteristicsHead

The group referencing MessagingCharacteristicsHead contains MessagingCharacteristics.

Alternative collaboration protocol families or new versions of existing collaboration families can add elements to the MessagingCharacteristicsHead substitution group in order to add distinctive new capabilities or configuration parameters needed for the business collaboration protocol.

6.2.11 SecurityPolicyHead

The group referencing SecurityPolicyHead contains SecurityPolicy.

Alternative collaboration protocol families or new versions of existing collaboration families can add elements to the SecurityPolicyHead substitution group in order to add distinctive new capabilities or configuration parameters needed for policy pertaining to business risk or trust.

6.2.12 TransportUserHead

The group referencing TransportUserHead contains TransportUser.

Alternative transfer protocols or ways to use known transfer protocols can add elements to the TransportUserHead substitution group in order to add distinctive new capabilities or configuration parameters needed to describe how the communication layer for a collaboration protocol is configured.

6.3 Other Version 3.0 Changes and Generalizations

Several key changes other than those detailed earlier in this section have occurred for this specification. They include:

1. Constituent element: In v3, Packaging including Constituent elements were enhanced for simple parts of payload constituents and further support web services. These Constituent elements payload constituents that are packaged for transmittal (See 4.5.2).

2. SimplePart element errata: After the publication of version 2 of this specification, an errata was approved to redefine SimplePart construction of a single MIME entity (See 1.4).

3. OverrideMSHActionBinding element: A new model for this element was defined in this specification. This element identfies when another DeliveryChannel is used rather than the default and the action which applies. The syntax is the same; what has changed?

4. Transport element: This element has been generalized to further leverage other transport protocols and bindings (See 4.3.8).

5. ebBP (business process definition): In ebBP v2.x, substantive changes were made to allow for greater definition of and extensibility for business processes, particularly business transactions, role bindings, quality of service guidance, conditions and constraints associated with context of Business Transactions, and mapping to service operations. Similar progress has also been made with ebMS v3. To support the extensibility of and modular approach towards support of emerging technologies and services, this specification has expanded support of the CollaborationRole, Service, ActionContext2, for example. Those changes have been captured in explicit tables throughout this specification and in the appendices, where explicit mapping tables exist. These changes also support use of other business process definitions and constructs.

Determine if we wish to leave this section and NOTES for implementers or consolidate in one section here. Recommend we leave NOTES as is and place this at the start of the document.

A. Introduction to Appendices

These appendices address notices and historical information, examples and other details about the relationship to other specifications. They include:

- Acknowledgements
- Revision history
- Glossary terms
- Mapping to other specifications

> *NOTE: In these appendices, functional mappings may be defined. A conditional obligation exists where a particular functional mapping is defined and used by a run-time system. For example: "If multiple* ***PartyID*** *elements occur under the same* ***PartyInfo*** *element in the CPA, all of those* ***PartyID*** *elements MUST be included in the Message Header". This is a conditional obligation when, for example, a message protocol such as ebMS3 is used whereby the mapping defined herein applies.*

- Extension descriptions
- Other non-normative guidance

B. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

[Participant Name, Affiliation | Individual Member]

[Participant Name, Affiliation | Individual Member]

C. Revision History

Revision Number	Date	By Whom	Overview
2.5	15-March-2008	Dale Moberg Monica J. Martin	Changes from v2 errata, ***PartyID*** request from user community, integration of new protocol bindings, extensibility framework for new message protocols and business process definitions, and updates related to ebBP, ebMS and other protocol binding compatibility. Preparation for integration into new ebXML Core TC in OASIS. Use new OASIS template format.
2.6	15-July-2008	Dale Moberg	Continued to refine definition and use of the business collaboration protocol consistent with CPPA and with potential associated business process options.

D. Correspondence Between CPA and ebXML Messaging Version 2 Parameters

The following table shows the correspondence between elements used in the ebXML Messaging Service v2 Message Header and their counterparts in the CPA.

Message Header Element / Attribute	Corresponding CPA Element / Attribute
PartyId element	PartyId element; if multiple PartyID elements occur under the same PartyInfo element in the CPA, all of them MUST be included in the Message Header
Role element	Role/@name value.
CPAId element	cpaid attribute in CollaborationProtocolAgreement element
ConversationId element	No equivalent; SHOULD be generated by software above the Message Service Interface (MSI)
Service element	Service element
Action element	action attribute in ThisPartyActionBinding element
TimeToLive element	Computed as the sum of Timestamp (in message header) + PersistDuration (under DocExchange/ebXMLReceiverBinding)
MessageId element	No equivalent; generated by the MSH per message
Timestamp element	No equivalent; generated by the MSH per message
RefToMessageId element	No equivalent; usually passed in by the application where applicable; SHOULD be used for correlating response messages with request messages

SyncReply element	syncReplyMode attribute in MessagingCharacteristics element; the SyncReply element is included if and only if the syncReplyMode attribute is not “none”
DuplicateElimination element	duplicateElimination attribute in MessagingCharacteristics element; the DuplicateElimination element is included if the duplicateElimination attribute under MessagingCharacteristics is set to “always”, or if it is set to “perMessage” and the application indicates to the MSH that duplicate elimination is desired
Manifest element	Packaging element; each Reference element under Manifest SHOULD correspond to a SimplePart that is referenced from one of the CompositeList elements under Packaging
xlink:role attribute in Reference element	xlink:role attribute in SimplePart element
AckRequested element	ackRequested attribute in MessagingCharacteristics element; an AckRequested element is included in the SOAP Header if the ackRequested attribute is set to “always”; if it is set to “perMessage”, input passed to the MSI is to be used to determine if an AckRequested element needs to be included; likewise, the signed attribute under AckRequested will be appropriately set based on the ackSignatureRequested attribute and possibly determined by input passed to the MSI
MessageOrder element	messageOrderSemantics attribute in ReliableMessaging element; the MessageOrder element will be present if the AckRequested element is present, and if the messageOrderSemantics attibute in the ReliableMessaging element is set to "Guaranteed"
ds:Signature element	ds:Signature will be present in the SOAP Header if the isNonRepudiationRequired attribute in the BusinessTransactionCharacterisitcs element is set to “true”; the relevant parameters for constructing the signature can be obtained from the SenderNonRepudiation and ReceiverNonRepudiation elements

The following table shows the implicit parameters employed by the ebXML Messaging Service v2 that are not included in the Message Header and how those parameters can be obtained from the CPA.

Implicit Messaging Parameters	Corresponding CPA Element / Attribute
Retries (not in Message Header) but used to govern Reliable Messaging behavior in sender	Retries element (under ReliableMessaging element)
RetryInterval (not in Message Header) but used to govern Reliable Messaging behavior in sender	RetryInterval element (under ReliableMessaging element)
PersistDuration (not in Message Header) but used to govern Reliable Messaging behavior in receiver	PersistDuration element (under ebXMLReceiverBinding element)
Endpoint (not in Message Header) but used for sending SOAP message	Endpoint element (under TransportReceiver); the type of message being sent MUST be passed in to the MSI; an appropriate endpoint can then be selected from among the Endpoints included under the TransportReceiver element
Use Service and Action to determine the corresponding DeliveryChannel	DeliveryChannel
Use ReceiverDigitalEnvelope to determine the encryption algorithm and key	ReceiverDigitalEnvelope
Use SenderNonRepudiation to determine signing certificate(s) and ReceiverNonRepudiation to determine the trust anchors and security policy to apply to the signing certificate	SenderNonRepudiation and ReceiverNonRepudiation
Use Packaging to determine how payload containers ought to be encapsulated. Also use Packaging to determine how an individual SimplePart ought to be extracted and validated against its schema	Packaging
Use TransportClientSecurity and TransportServerSecurity to determine certificates to	TransportClientSecurity and TransportServerSecurity

be used by server and client for authentication purposes	
Use the DeliveryChannel identified by defaultMshChannelId for standalone MSH level messages like Acknowledgment, Error, StatusRequest, StatusResponse, Ping, Pong, unless overridden by OverrideMshActionBinding	defaultMshChannelId attribute in PartyInfo element, and OverrideMshActionBinding

E. Correspondence Between CPPA and ebMS Version 3 Parameters

ebMS Header or PMode Information Item	CPPA v 3.0 Information Source
AgreementRef Agreement	/CollaborationProtocolAgreement/@cpaId
PartyId,Initiator.Party, Responder.Party	//PartyInfo/PartyId
PartyId/@type	//PartyInfo/PartyId/@type

MEP	//SenderProcessingMode/MEP |
	//ReceiverProcessingMode/MEP
	URI value such as http://www.oasis-open.org/committees/ebxml-msg/one-way.

MEPbinding	//SenderProcessingMode/MEPBinding |
	//ReceiverProcessingMode/MEPBinding
	URI such as http://www.oasis-open.org/committees/ebxml-msg/push

[Initiator|Responder].Authorization.
username
	//SenderProcessingMode/AccessToken-/Username |
	//ReceiverProcessingMode/AccessToken-/Username
	[Can be found within //ProcessingMode/AccessToken-/EncryptedData also.]

[Initiator|Responder].Authorization.	//SenderProcessingMode/AccessToken-/Password |

password	//ReceiverProcessingMode/AccessToken-/Password [Can be found within //ProcessingMode/AccessToken/-EncryptedData also.]
	//SenderProcessingMode-/@conformanceLevel | //ReceiverProcessingMode-/@conformanceLevel
ProtocolAddress	//TransportSender/Endpoint/@uri //TransportReceiver/Endpoint/@uri
Protocol.SOAPversion	//SenderProcessingMode/SOAPVersion | //ReceiverProcessingMode/SOAPVersion
BusinessInfo.Service	//ServiceBinding/Service
BusinessInfo.Action	//ActionBinding@action (version 3) //ThisPartyActionBinding/@action (version 2)

BusinessInfo.Role	//CollaborationRole/Role/@name
	or when present,
	//ActionContext2/@role

BusinessInfo.PayloadProfile.MimeMediaType	//SimplePart/@mimetype
BusinessInfo.PayloadProfile.namespaces	//SimplePart/NamespaceSupported

BusinessInfo.PayloadProfile.optional	//Constitutent/@minOccurs equals 0
BusinessInfo.PayloadProfile.maxSize	//Constituent/@maxSizeInKBytes
BusinessInfo.MPC	//SenderProcessingMode/MPC | //ReceiverProcessingMode/MPC
BusinessInfo.Properties	//SenderProcessingMode/ExtensionProperty | //ReceiverProcessingMode-/ExtensionProperty
ErrorHandling.SenderErrorsTo	//TransportReceiver/Endpoint/@uri when //TransportReceiver/Endpoint/@type is “error” or “allPurpose” | Endpoint value associated with defaultMshChannelId| Endpoint/@uri value associated with OverrideMshActionBinding/@channelId when OverrideMshActionBinding/@action value is the action for this configuration.
ErrorHandling.ReceiverErrorsTo	//TransportSender/Endpoint/@uri when //TransportReceiver/Endpoint/@type is “error” or “allPurpose” | Endpoint/@uri value associated with defaultMshChannelId | Endpoint/@uri value associated with OverrideMshActionBinding/@channelId when OverrideMshActionBinding/@action value is the action for this configuration.
ErrorHandling.AsResponse	//MessagingCharacteristics/-@syncReplyMode equals mshSignalsOnly, signalsOnly, or

	signalsAndResponse.
ErrorHandling.ProcessErrorNotifyConsumer	//SenderProcessingMode/-@processErrorNotifyConsumer //SenderProcessingMode/-@processErrorNotifyConsumer
ErrorHandling.ProcessErrorNotifyProducer	//SenderProcessingMode/-@processErrorNotifyProducer //ReceiverProcessingMode/-@processErrorNotifyProducer
ErrorHandling.DeliveryFailuresNotifyProducer	//SenderProcessingMode/-@deliveryErrorNotifyProducer //ReceiverProcessingMode/-@deliveryErrorNotifyProducer
atLeastOnce.Contract	//MessagingCharacteristics.AckRequested
atLeastOnce.AckOnDelivery	//MessagingCharacteristics.AckRequested
atLeastOnce.AcksTo	//MessagingCharacteristics/-@syncReplyMode equals mshSignalsOnly, signalsOnly, or signalsAndResponse imply that HTTPResponse is used for Acks. Otherwise, Endpoint/@uri value associated with defaultMshChannelId or Endpoint/@uri value associated with OverrideMshActionBinding/@channelId when OverrideMshActionBinding/@action value is the action for this configuration.
atLeastOnce.AckResponse	//MessagingCharacteristics.AckRequested (but for Response Action)
atLeastOnce.ReplyPattern	Endpoint/@uri value associated with

	defaultMshChannelId or Endpoint/@uri value associated with OverrideMshActionBinding/@channelId when OverrideMshActionBinding/@action value is the action in this configuration record.
atMostOnce.Contract	MessagingCharacteristics/@duplicateElimination contains the value, true.
inOrder.Contract	ReliableMessaging/MessageOrderSemantics contains the value “Guaranteed”
StartGroup	//ReliableMessaging/ReliabilityGroup/-@startGroup.
Correlation	//ReliableMessaging/ReliabilityGroup/-CorrelationPath
TerminateGroup	//ReliableMessaging/ReliabilityGroup/-@terminateGroup
WSSVersion	//SenderNonRepudiation/NonRepudiation-Protocol/@version where //SenderNonRepudiation/NonRepudiation-Protocol identifies protocol value as “WSS” or “http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1”
Signature.Element	//Constituent/ElementRef/@signed contains value, “true”
Signature.Attachment	//Constituent/@excludedFromSignature contains the value, false.
Signature.Certificate	//SenderNonRepudiation/SigningCertificateRef points to certificate.
Signature.HashFunction	//SenderNonRepudiation/NonRepudiation-Protocol/

	HashFunction
Signature.Algorithm	//SenderNonRepudiation/SignatureAlgorithm/-@w3c
Encryption.Element	//Constituent/ElementRef/@encrypted contains value, “true” [When same element is both signed and encrypted, use the value of //Constituent/ElementRef/@signBeforeEncrypt to decide order.]
Encryption.Certificate	//ReceiverDigitalEnvelope/-EncryptionCertificateRef
Encryption.Algorithm	//ReceiverDigitalEnvelope/EncryptionAlgorithm/@w3c //SenderDigitalEnvelope/EncryptionAlgorithm/@w3c
Encryption.MinimumStrength	//ReceiverDigitalEnvelope/EncryptionAlgorithm/@minimumStrength //SenderDigitalEnvelope/EncryptionAlgorithm/-@minimumStrength

WSS.username	//SenderProcessingMode/AccessToken-/Username |
	//ReceiverProcessingMode/AccessToken-/Username
	[Can be found within //ProcessingMode/AccessToken/EncryptedData also.]

WSS.password	//SenderProcessingMode/AccessToken-/Password |
	//ReceiverProcessingMode/AccessToken-/Password
	[Can be found within //ProcessingMode/AccessToken/EncryptedData

	also.]

WSS.Token.digest	//SenderProcessingMode/AccessToken/Digest |
	//ReceiverProcessingMode/AccessToken/Digest
	[Can be found within //ProcessingMode/AccessToken/-EncryptedData also.]

WSS.Token.nonce	//SenderProcessingMode/AccessToken/Nonce |
	//ReceiverProcessingMode/AccessToken/Nonce
	[Can be found within //ProcessingMode/AccessToken/-EncryptedData also.]

WSS.Token.created	//SenderProcessingMode/AccessToken/-CreatedTimestamp |
	//ReceiverProcessingMode/AccessToken/-CreatedTimestamp
	[Can be found within //ProcessingMode/AccessToken/-EncryptedData also.]

WSS.PModeAuthorize	//BusinessCharacteristics/@isAuthorizationRequired .
SendReceipt	MessagingCharacteristics/@signAck and BusinessTransactions/@isNonReudiationofReceipt Required

SendReceipt.ReplyPattern	MessagingChacteristics/@syncReplyMode is signalsOnly or signalsAndResponse Or, signals can have a delivery channel configured within their own ActionBinding.

F. Correspondence Between CPPA Versions 2.0 and 3.0 with BPSS Versions 1.* Parameters

ebBPSS Version 1.*	ebCPPA Version 2.0	Status
ProcessSpecification/@uuid	ProcessSpecification/@uuid	Required
ProcessSpecification/@uuid	Service/@name	Required
BinaryCollaboration/Role/@name	CollaborationRole/Role/@name	Recommended
RequestingBusinessActivity/@name	ThisPartyActionBinding@action	Recommended
RespondingBusinessActivity/@name	ThisPartyActionBinding@action	Recommended

ebBPSS Version 1.*	ebCPPA Version 3.0	Status
ProcessSpecification/@uuid	ProcessSpecification/@uuid	Required
ProcessSpecification/@uuid	Service/@name	Required
BinaryCollaboration/Role/@name	CollaborationRole/Role/@name	Recommended
RequestingBusinessActivity/@name	ThisPartyActionBinding@action	Recommended
RespondingBusinessActivity/@name	ThisPartyActionBinding@action	Recommended

G. Correspondence Between CPPA Version 3.0 and ebBP Version 2.0 Parameters

ebBP (BPSS) Version 2.0	ebCPPA Version 3.0	Status
ProcessSpecification/@uuid	ProcessSpecification/@uuid	Required
BusinessCollaboration/@name BinaryCollaboration/@name MultiPartyCollaboration/@name	Service	Recommended (a potentially different value for each top level Collaboration)
BusinessCollaboration/Role/@name BinaryCollaboration/Role/@name MultiPartyCollaboration/Role/@name	CollaborationRole/Role/@name	Required
RequestingBusinessActivity/@name	ThisPartyActionBinding/@action ActionBinding/@action	Recommended
RespondingBusinessActivity/@name	ThisPartyActionBinding/@action ActionBinding/@action	Recommended
BusinessTransactionActivity/ RequestingParty/@name	ActionContext2/@role	Recommended
BusinessTransactionActivity/ RespondingParty/@name	ActionContext2/@role	Recommended
BusinessTransactionActivity/@name	ActionContext2/ @businessTransactionActivity	Recommended

CollaborationActivity/@name	ActionContext2/Collaboration Level/@name	Recommended
ComplexBusinessTransactionActivity/@name	ActionContext2/Collaboration Level/@name	Recommended
ComplexBusinessTransactionActivity	ActionContext2/Collaboration Level/@collaborationLevelConstruct= ”ComplexBusinessTransactionActivity”	Recommended
CollaborationActivity	ActionContext2/Collaboration Level/@collaborationLevelConstruct= ”CollaborationActivity”	Recommended

H. Using CPPA for EDIINT Configuration

The CPPA extensibility framework can be used in creating CPPs and CPAs for the EDIINT (AS1, AS2, and AS3) collaboration protocols [RFC3335],[RFC4130] [AS3] [Compression]

EDIINT protocols are peer-to-peer collaboration protocols using IETF transfer protocols (SMTP, HTTP, FTP), S/MIME security [S/MIME][CMS], and acknowledgments using Message Disposition Notifications [RFC3798].

EDIINT messaging defines its own (signed) acknowledgment message (MDN) and defines a compression option. Because these features are not found in ebXML 2.0 Messaging, it is necessary to add "modules" that document configuration details for these features.

Two new DocExchange binding elements are introduced for the senders and receivers of EDIINT messages, called EdiintReceiverBinding and EdiintSenderBinding. Each of these elements contain the new modules mentioned above as well as modules also used for other messaging protocols, such as in ebXML Messaging.

EDIINT Mapping Attribute	**Description**
EdiintSenderBinding/@version	The value of this attribute identifies the version of the EDIINT specification being used. These values include: 1.0, 1.1, and 1.2 and are used in the ASx-Version header. It has a default value of 1.1, which is the ASx-Version value for compression support. The 1.2 value is used when the EDIINT-Features header is supported. Use of EDIINT-Features is intended to announce supported features instead of using values greater than 1.2
PartyInfo/PartyId	The AS2 and AS3 protocols have special headers ("AS2-To", "AS2-From", "AS3-To", "AS3-From") that identify the participants. The PartyInfo/PartyId values supply these values. In AS1, the Endpoint/@uri values will also contain the identifying email addresses of the parties within the mailto: URLs.

EDIINT Mapping Attribute	Description
SenderCompression/@mechanismType	The mechanismType attribute has a default value of "zlib," which is the value defined by the compression I-D. Any other value is implementation dependent. In a CPA, this value matches ReceiverCompression/@mechanismType.
SenderCompression/@version	Used to declare the version of compression support, relevant should future versions emerge. It has a default value of 1.1, which is the ASx-Version value indicating support for compression. In a CPA, this value matches ReceiverCompression/@version.
SenderRequestedMDNStyle/HashFunction	The value used in the transfer protocol header Disposition-notification-options: signed-receipt-protocol=optional,pkcs7-signature; signed-receipt-micalg=optional,sha1 In a CPA, the value matches the value in ReceiverRequestedMDNStyle/HashFunction. There can be comma separated list of acceptable hashes as in the parameter: signed-receipt-micalg=optional,sha1,md5.
SenderRequestedMDNStyle/@receiptStyle	The receiptType attribute with values "signed" or "unsigned" that indicate whether the MDN is to be signed. If the attribute is omitted, no receipt should be requested, and so no Disposition-notification-to header should be included in the transfer protocol headers. In CPAs, matches value in ReceiverRequestedMDNStyle/@receiptStyle

EDIINT Mapping Attribute	Description
SenderRequestedMDNStyle/@mdnRequested	The attribute mdnRequested is used to indicate whether an MDN is always, never, or sent on a perMessage basis. The default is "perMessage" For the AS2 protocol, if the MDN is to be sent on a separate TCP connection, the "Disposition-Notification-To" header should be included, with values from the mdnDestination attribute below. In CPAs, matches value in ReceiverRequestedMDNStyle/@mdnRequested
SenderRequestedMDNStyle/@mdnDestination	The mdnDestination attribute's value is a URL or email address that indicates where the MDN is to be sent, and the value for a header: Receipt-delivery-option: http://www.example.com When asynchronous MDNs are used, the value of the attribute "mdnDestination" is the proposed or agreed upon destination. In CPAs, matches value in ReceiverRequestedMDNStyle/@mdnDestination
ServiceBinding/ActionBinding/-BusinessTransaction-Characteristics/@isConfidential	Support for or agreement upon an encrypting digital envelope is declared by setting to a value of either "persistent" or "transient-and-persistent". Either value indicates use of [S/MIME]. The use of SSL/TLS data protection would be indicated by "transient-and-persistent" instead of simply "persistent".
EdiintReceiverBinding/-ReceiverDigitalEnvelope/-EncryptionCertificateRef	The encryption certificate that can be or will be used is found by matching this IDREF value to the element whose ID value it equals.
EdiintSenderBinding/-SenderDigitalEnvelope/EncryptionSecurity-DetailsRef	Trust anchors for checking the encryption certificate can be located using the EncryptionSecurityDetailsRef IDREF value as part of the SecurityDetails element having an ID value equal to the IDREF value.

EDIINT Mapping Attribute	Description
EdiintReceiverBinding/-ReceiverDigitalEnvelope/Encryption-Algorithm/@oid or EdiintReceiverBinding/-ReceiverDigitalEnvelope/Encryption-Algorithm/@w3c.	The encryption symmetric encryption algorithm is declared using one or more of these attributes; the oid syntax is preferable. In a CPA, these Receiver values need to match values found at: DocExchange/EdiintSenderBinding/- SenderDigitalEnvelope/EncryptionAlgorithm/@oid or DocExchange/EdiintSenderBinding/- SenderDigitalEnvelope/EncryptionAlgorithm /@w3c Example: <EncryptionAlgorithm oid="1.2.840.113549.3.7">DES-EDE3-CBC</EncryptionAlgorithm>
EdiintReceiverBinding/-ReceiverDigitalEnvelope/EncryptionAlgorithm/@minimumStrength	The digital envelope's encryption strength supported or agreed upon is indicated using @minimumStrength value.
ServiceBinding/ActionBinding/Business-TransactionCharacteristics/-@isNonRepudiationRequired	Digital signature capabilities or agreements for data origin authentication are declared using a "true" value for @isNonRepudiationRequired.
ServiceBinding/ActionBinding/Business-TransactionCharacteristics/-@isNonRepudiationReceiptRequired	A true value indicates support for or agreement to use a signed MDN and conducting checks on the returned hash value for the sent message.
EdiintSenderBinding/-SenderNonRepudiation/SigningCertificateRef	The SigningCertificate used in data origin authentication is found by finding the Certificate element whose ID equals the IDREF value in this element.

EDIINT Mapping Attribute	Description
EdiintReceiverBinding/-ReceiverNonRepudiation/Signing-SecurityDetailsRef	The receiver indicates trust anchors used in validating trust in the sender's signing certificate.
EdiintSenderBinding/SenderNonRepudiation/-HashFunction	Example would be: <HashFunction>SHA1</HashFunction>
EdiintSenderBinding/SenderNonRepudiation/-SignatureAlgorithm	Example would be: <SignatureAlgorithm oid="1.2.840.113549.1.1.5">RSA-SHA1</SignatureAlgorithm>
EdiintSenderBinding/SenderNonRepudiation-/NonRepudiationProtocol	Example might be: <NonRepudiationProtocol>EDIINT</NonRepudiationProtocol> or <NonRepudiationProtocol>AS2</NonRepudiationProtocol>
MessagingCharacteristics/@syncReplyMode	The MDN is viewed as a business acknowledgement or signal, and not as a MSH signal response. For this reason, when MDNs are returned synchronously in AS2, the value for @syncReplyMode should be either "responseOnly" or better "signalsAndResponse" When the MDN is returned in a separate connection, @syncReplyMode should be "none".
Packaging	Packaging formats will conform with the specifications for MIME formats of the offered cryptographic services [RFC3335] and other relevant specifications [RFC3798, S/MIME]. Examples are presented in discussion at end of this table.
SimplePart/NamespaceSupported	Available for use with XML payloads..

EDIINT Mapping Attribute	Description
PersistDuration	This element can be used when there is an agreement on checking for duplicates when using the Message-Id value. This feature is outside of the core EDIINT specifications but is often supported.
Transport/TransportReceiver/Endpoint/@uri	Example: <Endpoint uri="https://www.CompanyB.com/AS2" type="allPurpose"/> AS1: The URL uses the “mailto:“ scheme to exchange RFC 2822 adresses. AS2: The URL uses either “http:” or “https:” schemes. AS3: The URL uses “ftp:” or “ftps:” to indicate the scheme.
Transport/TransportSender/Transport-Protocol/@method	Example: <TransportProtocol version="1.1" method="POST">HTTP</TransportProtocol> POST method is what AS2 uses. For AS3, the method might be STOR or RETR.

The EDIINT collaboration messaging protocol is specified in three application statements, AS1, AS2 and AS3, that differ primarily in which IETF transfer protocol is used. AS1, AS2, and AS3 use SMTP, HTTP/HTTPS, and FTP/FTP-over-SSL respectively. Similar Packaging documentation is used for all transfer protocols. The details of packaging depend upon the specific security and MDN response modes of operation that are selected.

Some simple types referenced in Packaging could be:

```
   <SimplePart id="X12SimplePart" mimetype="application/EDI-X12"/>
    <SimplePart id="MdnText" mimetype="text/plain"/>
    <SimplePart id="MdnMessage" mimetype="message/disposition-notification"/>
```

A signed MDN Packaging element could be:

```
<Packaging id="SignedMdn">
        <ProcessingCapabilities generate="true" parse="true"/>
        <CompositeList>
            <Composite id="MdnComposite" mimeparameters="report-
type=disposition-notification"
                mimetype="multipart/report">
                <Constituent excludedFromSignature="false" idref="MdnText"
maxOccurs="1"
                    minOccurs="1"/>
                <Constituent excludedFromSignature="false" idref="MdnMessage"
maxOccurs="1"
                    minOccurs="1"/>
            </Composite>
            <Encapsulation id="DefaultEncapsulation3"
mimetype="application/pkcs7-signature">
                <Constituent excludedFromSignature="false"
idref="MdnComposite" maxOccurs="1"
                    minOccurs="1"/>
            </Encapsulation>
            <Composite id="DefaultComposite4"
                mimeparameters="protocol=&quot;application/pkcs7-
signature&quot;"
                mimetype="multipart/signed">
                <Constituent excludedFromSignature="false"
idref="DefaultEncapsulation3"
                    maxOccurs="1" minOccurs="1"/>
            </Composite>
        </CompositeList>
    </Packaging>
```

Examples of packaging are included in the package of examples and sample referenced business process descriptions accompanying this specification.

SSL and PKI aspects for Transport Security are documented by means of components of the TransportServerSecurity or TransportClientSecurity, such as ServerCertificateRef, ClientCertificateRef, ClientSecurityDetailsRef, ServerSecurityDetailsRef.

Examples of this standard PKI documentation are included in the package of examples and sample referenced business process descriptions accompanying this appendix.

I. Using CPPA for Web Service Agreements

In order to announce capabilities or document agreements about aspects of web service collaboration protocols, web services DocExchange extension modules are added to a CPP or CPA using either the WSSenderBinding or WSReceiverBinding elements. Within these elements, the WSDLOperation and WSDLPolicy elements are used as containers for WSDL 1.1 [WSDL11] and 2.0 [WSDL20] or WS-Policy [WSPolicy] information. Version – used 1.5?

Each WSDLOperation only needs to include or reference the "interface" aspects of WSDL. These aspects are found within portType or interface and any QName referenced other elements. (The precise elements needed differ in WSDL versions 1.1 and 2.0).

For each ActionBinding, the interface's operation first needs to be identified. The WSDLOperation/@operationRef uses a URI Reference to identify the operation.

This reference is constructed by appending a fragment identifier to the wsdl file location as discussed in WSDL 2.0 or a the W3C Note for WSDL 1.1 constructed using an XPointer fragment syntax:

```
http://example.org/hello1.wsdl#wsdl.interfaceOperation(Hello/sayHello)
```

where the interface local name "Hello" starts the path, and is followed by the local name of the operation, "sayHello".

Because an operation can involve both requests and responses, the identification of the specific document that is being sent is specified as the value of the WSDLOperation@messageRef attribute, which contains an Xpointer value indicating the relevant document information. The list of faults is identified by WSDLOperation@faultRef Xpointer fragment identifiers.

WSDL content may also be placed verbatim within the WSDLOperation content model. In this case, a fragment identifier (beginning with a "#" and followed by the ID value) indicates that the WSDL is found within the element content associated with the fragment identifier.

The question of how to make use of wsdl:service, wsdl:binding or wsdl extensions left to the implementations that consume CPAs. It is expected that installed CPAs be checked so configuration information is consistent.

On the wsdl consumer side, the capability to act as a client is marked by an entirely empty WSDLOperation element. An empty element indicates that the client is capable of conforming or agrees to conform with whatever the wsdl specifies pertaining to input, output, and faults.

Support for CPPs and CPAs that make use of WSDL for describing Message Exchange Patterns (MEPs) between a sender and a receiver is provided by extensions to the DocExchange components. Because each DocExchange module pertains to some aspect of an ebXML Action. WSDL provides message descriptions from the standpoint of the service and so for the most commonly used MEPs.

WSSenderBinding	The WSSenderBinding element describes properties related to sending messages by means of web services. The WSSenderBinding element MUST include the

	WSDLOperation element
WSSenderBinding/@version	Version attribute is initially 3.0 to indicate that semantics are given as part of CPPA version 3.0.
WSReceiverBinding	The WSReceiverBinding element describes properties related to receiving messages using the Web Services Message Service. The WSReceiverBinding element is an alternative to the ebXMLReceiverBinding element and MUST include the WSDLOperation element
WSReceiverBinding/@version	Version attribute is initially 3.0 to indicate that semantics are given as part of CPPA version 3.0.
WSDLOperation/@version	The REQUIRED version attribute identifies the version of the WSDL specification being used.
WSDLOperation/@operationRef	The REQUIRED operationRef attribute provides an Xpointer fragment identifier to the wsdl operation [WSDL20] or to the wsdl portType [WSDL11],[WSDL11Ref].
WSDLOperation@messageRef	The REQUIRED messageRef attribute provides a list of Xpointer [XPOINTER] fragment identifiers are used for this reference. For [WSDL11], the relevant wsdl:parts of wsdl:message should be referenced. For [WSDL20], the relevant global element declarations will be referenced each of the form: wsdl.interfaceMessageReference(interface/operation/message)
WSDLOperation@faultRefList	The IMPLIED faultRef attribute provides a list of Xpointer [XPOINTER] fragment identifiers are used for this list of references each of the form: wsdl.interfaceFaultReference(interface/operation/message/fault)
SenderPolicy	Container for WS-Policy that applies to the Action of the ActionBinding that references this element. In a CPPA, this will be an agreed upon PolicyAlternative that holds between

	Sender and Receiver for the Action that is bound to this policy. In a CPP, this will be a policy that in canonical form can include many PolicyAlternatives.
ReceiverPolicy	Container for WS-Policy that applies to the Action of the ActionBinding that references this element. In a CPPA, this will be an agreed upon PolicyAlternative that holds between Sender and Receiver for the Action that is bound to this policy. In a CPP, this will be a policy that in canonical form can include many PolicyAlternatives.
SenderPolicy/@version	Version of WS-Policy.
ReceiverPolicy/@version	Version of WS-Policy.

J. Glossary of Terms

Term	Definition
AGREEMENT	An arrangement between two partners that specifies in advance the conditions under which they will trade (terms of shipment and/or payment, collaboration protocols, constraints etc.) Any specific economic commitments are outside of the scope of this specification.
APPLICATION	Software above the level of the message handling that implements a Service(s) by processing one or more of the Messages in the Document Exchanges associated with the(those) Service(s).
AUTHORIZATION	A right or a permission that is granted to a system entity to access a system resource.
BUSINESS ACTIVITY	A business activity is used to represent the state of the business process of one of the partners. For instance the requester is either in the state of sending the request, waiting for the response, or receiving. For CPPA, a Business Activity typically relates to and is focused on a Business Transaction Activity.
BUSINESS COLLABORATION	A set of roles that business partners assume in a Business Activity through the exchange of business messages in a peer-to-peer environment rather than a controlled environment to achieve a business goal. An business collaboration activity is conducted between two or more parties for the purpose of achieving a specified outcome. In the context of CPPA, the business collaboration is decomposed into business activities (or business transaction activities) between two partners.
BUSINESS COLLABORATION PROTOCOL	Defines the business messages and signals (if used) that insure that technical state alignment for a Business Collaboration instance is strictly identical for the participating parties. In the context of CPPA, the business collaboration protocol is decomposed into business activities between two partners.
BUSINESS DOCUMENT	The set of information components that are interchanged as part of a business activity.

BUSINESS PARTNER

An entity that engages in business transactions with another business partner(s).

BUSINESS PROCESS

The means by which one or more activities are accomplished in operating business practices.

BUSINESS PROCESS SPECIFICATION SCHEMA

Defines the necessary set of elements for run-time aspects and configuration parameters to drive the partners' systems used in the collaboration. The goal of the BP Specification Schema is to provide the bridge between the eBusiness process modeling and specification of eBusiness software components.

BUSINESS TRANSACTION

A business transaction is a logical unit of business conducted by two or more parties that generates a computable success or failure state. The community, the partners, and the process, are all in a definable, and self-reliant state prior to the business transaction, and in a new definable, and self-reliant state after the business transaction. In other words if you are still 'waiting' for your business partner's response or reaction, the business transaction has not completed.

CLIENT

Software that initiates a connection with a Server.

COLLABORATION

Two or more parties working together under a defined set of rules.

COLLABORATION PROTOCOL

The protocol that defines for a Collaborative Process: 1. The sequence, dependencies and semantics of the Documents that are exchanged between Parties in order to carry out that Collaborative Process, and 2. The Messaging Capabilities used when sending documents between those Parties. Note that a Collaborative Process can have more than one Collaboration Protocol by which it can be implemented.

COLLABORATION PROTOCOL AGREEMENT (CPA)

Information agreed between two (or more) Parties that identifies or describes the specific Collaboration Protocol that they have agreed to use. A CPA indicates what the involved Parties “will” do when carrying out a Collaborative Process. A CPA is representable by a Document.

A CPPA realizes concretely the technical contract around business transaction activities using business transactions defined in a business process definition through one or more business collaboration activity protocols. The BTA are typically choreographed in a business process definition and can be composed as Business Collaboration activities. For simplicity of construction and flexibility for reuse, those Business Collaboration Protocol activities are concretely defined as logical units of work with binary interactions in the CPPA, leaving the actual choreographic definitions to the business process that applies (and therefore where the business requirements are held).

COLLABORATION PROTOCOL PROFILE (CPP)

Information about a Party that can be used to describe one or more Collaborative Processes and associated Collaborative Protocols that the Party supports. A CPP indicates what a Party “can” do in order to carry out a Collaborative Process. A CPP is representable by a Document. While logically, a CPP is a single document, in practice, the CPP might be a set of linked documents that express various aspects of the capabilities. A CPP is not an agreement but represents the capabilities of a Party.

COLLABORATIVE PROCESS

A shared process by which two Parties work together in order to carry out a process. The Collaborative Process can be defined by an ebXML Collaboration Model.

CONFORMANCE

Fulfillment of a product, process or service of all requirements specified; adherence of an implementation to the requirements of one or more specific standards or technical specifications.

DIGITAL SIGNATURE

A digital code that can be attached to an electronically transmitted message that uniquely identifies the sender

DOCUMENT

A Document is any data that can be represented in a digital form.

DOCUMENT EXCHANGE

An exchange of documents between two parties.

ENCRYPTION

Cryptographic transformation of data (called "plaintext") into a form (called "ciphertext") that conceals the data's original meaning to prevent it from being known or used. If the transformation is reversible, the corresponding reversal process is called "decryption", which is a transformation that restores encrypted data to its original state.

EXTENSIBLE MARKUP LANGUAGE

XML is designed to enable the exchange of information (data) between different applications and data sources on the World Wide Web and has been standardized by the W3C.

IMPLEMENTATION	An implementation is the realization of a specification. It can be a software product, system or program.
MESSAGE	The movement of a document from one party to another.
MESSAGE HEADER	A specification of the structure and composition of the information necessary for an ebXML Messaging Service to successfully generate or process an ebXML compliant message.
MESSAGING CAPABILITIES	The set of capabilities that support exchange of Documents between Parties. Examples are the communication protocol and its parameters, security definitions, and general properties of sending and receiving messages.
MESSAGING SERVICE	A framework that enables interoperable, secure and reliable exchange of Messages between Trading Partners.
PACKAGE	A general-purpose mechanism for organizing elements into groups. Packages can be nested within other packages.
PARTY	A Party is an entity such as a company, department, organization or individual that can generate, send, receive or relay Documents.
PARTY DISCOVERY PROCESS	A Collaborative Process by which one Party can discover CPP information about other Parties.
PAYLOAD	A section of data/information that is not part of the ebXML wrapping.
PAYLOAD CONTAINER	A container used to envelope the real payload of an ebXML message. If a payload is present, the payload container consists of a MIME header portion (the ebXML Payload Envelope) and a content portion (the payload itself).
PAYLOAD ENVELOPE	The specific MIME headers that are associated with a MIME part.
RECEIVER	Recipient of a Message.

REGISTR	A mechanism whereby relevant repository items and metadata about them can be registered such that a pointer to their location, and all their metadata, can be retrieved as a result of a query.
REQUESTER	Initiator of a Business Transaction.
RESPONDER	A counterpart to the initiator in a Business Transaction.
ROLE	The named specific behavior of an entity participating in a particular context. A role could be static (e.g., an association end) or dynamic (e.g., a collaboration role).
SECURITY POLICY	A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources.
SENDER	Originator of a Message.
SERVER	Software that accepts a connection initiated by a Client.
UNIQUE IDENTIFIER	The abstract concept of utilizing a standard mechanism and process for assigning a sequence of alphanumeric codes to ebXML Registry items, including: Core Components, Aggregate Information Entities, and Business Processes.
UNIVERSALLY UNIQUE IDENTIFIER (UUID)	An identifier that is unique across both space and time, with respect to the space of all UUIDs. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.