# Introduction to ebXML

Presented by developerWorks, your source for great tutorials

**ibm.com/developerWorks**

## Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

# Section 1. About this tutorial

## Should I take this tutorial?

This tutorial is for developers who want to get a feel for the overall architecture and use of electronic business XML (ebMXL), but who may not know where to start. It is also for developers who have a general understanding of ebXML, but who want to start building ebXML-related applications and need to understand how all of the pieces fit together.

Developers must have at least a basic understanding of XML and of XML validation using schemas and/or DTDs. For basic XML information, see the *Introduction to XML* tutorial. Other resources can be found in the Resources on page 32 section at the end of the tutorial.

No programming experience is required.

---

## What is this tutorial about?

Whereas Electronic Data Interchange (EDI) for years has provided a usable but expensive way for companies to exchange information in an automated manner, ebXML now provides a means for companies to integrate their processes much more easily. Based on XML, it provides a methodology for businesses to determine what information they should exchange and how, as well as a set of specifications to allow automation of the process.

Before you can even consider building an ebXML-related application, you must understand what the pieces are and how they fit together.

This tutorial begins with a look at how businesses can use ebXML and a general overview of how all of the pieces fit together to form a complete architecture. The tutorial next discusses analysis and the process of creating Business Process Specifications and Business Documents. It then moves on to Collaboration Protocol Profiles (CPPs) and Collaboration Protocol Agreements (CPAs), and a discussion of ebXML Registries (where all of this information is stored). Finally, it finishes up with a look at the actual messages that are sent between Trading Partners, and how they're constructed.

It's important to note that at the time of this writing, ebXML implementations have never been completed because parts of the specifications are not yet complete. The existing implementations use the pieces that are available and fill in the holes as necessary.

---

## Tools

You will perform no actual programming in the course of this tutorial, so no particular software packages are required.

However, during the tutorial, you will have the option of viewing the OASIS ebXML v2 Registry Reference Implementation. This is not required, but should you choose to do so, you must have Java 1.4 installed in your machine. You can download Java 1.4 from *http://java.sun.com/j2se/1.4/download.html*.

---

# About the author

Nicholas Chase has been involved in Web site development for companies such as Lucent Technologies, Sun Microsystems, Oracle, and the Tampa Bay Buccaneers. Nick has been a high school physics teacher, a low-level radioactive waste facility manager, an online science fiction magazine editor, a multimedia engineer, and an Oracle instructor. More recently, he was the Chief Technology Officer of Site Dynamics Interactive Communications in Clearwater, Fla., and is the author of three books on Web development, including *Java and XML from Scratch* (Que). He loves to hear from readers and can be reached at *nicholas@nicholaschase.com*.

# Section 2. What is ebXML?

## Why businesses talk to each other

Businesses inevitably talk to each other in a variety of ways. Often their only contact with other businesses (either as suppliers or as customers) is through forms sent in the mail or phone calls to a helpful clerk. Performing these communications electronically eliminates the need for paper, saves man-hours, and streamlines the process, shaving significant time off of otherwise manual processes.

The functionality to communicate this way already exists, of course. Many large companies communicate automatically through EDI, which allows two companies to communicate using predetermined signals.

The trouble with EDI is that it's expensive. Originally created for the mainframe world, EDI requires skills that are rare, and it's generally impossible for all but the largest companies to consider using it.

XML provides a solution to that problem.

---

## Using XML to communicate

Part of the answer to the cost and maintenance issues of EDI is to use XML. XML has the following advantages:

- It is simpler than EDI.
- It has many more uses than just data exchange between companies.
- It is fairly easy to find developers who are familiar with it.
- It is a platform-neutral language.
- You can build applications to read and send XML virtually anywhere.

However, this doesn't solve the main problem for which EDI was created: Web Services can be built to communicate across multiple applications and vendors, but what do they say and how do they say it?

---

## ebXML to the rescue

This is where ebXML comes in.

Unlike many other XML derivatives, such as MathML or Scalable Vector Graphics (SVG), ebXML doesn't simply define an XML grammar and vocabulary. Instead, it defines not only an entire architecture, but also a new way of thinking about business,

and more importantly, documenting it.

ebXML consists of a group of related specifications that are maintained by the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT, the overseers of EDI) and OASIS.

You can define how companies conduct business using a specific vocabulary in Business Process Specifications. Predefined documents are built from core components. Messages are sent using standard formats and protocols. And just about everything is stored in ebXML registries, where companies can find the information and structures they need rather than reinventing the wheel.

---

# Finding Trading Partners

Registries also hold another important item: information on potential *Trading Partners*.

In ebXML, companies conduct business through the exchange of documents, which can take the form of purchase orders, administrative information, or even the goods themselves. For example, a company looking for a news clipping service might send a purchase order to a supplier and receive a list of news items in response.

Registries hold information on potential Trading Partners in the form of Collaboration Protocol Profiles (CPPs). CPPs are XML documents that use a specific vocabulary to identify business processes that a company is willing and able to take part in, the roles that it can play, and technical information about its capabilities. For example, searching the CPPs in an ebXML registry can uncover a business that can provide news clippings through an HTTP interface and that is willing to accept purchase orders online.

---

# Reaching agreement

Finding a Trading Partner is just the beginning. Next, you must configure the two systems to work together to complete the appropriate transactions. Fortunately, you can easily accomplish this through the use of a Collaboration Protocol Agreement (CPA). A CPA is composed from the CPPs of each trading partner, specifying what collaborations take place and the specifics about them.

These specifics may include information on technical issues such as protocols to be used, or they might include requirements such as acknowledgment and verification.

Once the companies have generated and agreed to the CPA, this single document can be used to configure the application, or *Business Service Interface*, on both sides. In this way, both Trading Partners are working from the same information, and there is no confusion about who should be doing what.

---

# Advanced ebXML features

To cover all of the aspects of ebXML would take a large book. Some features that are beyond the scope of this tutorial include:

- **Legal bindings**: When the specifications are complete, ebXML will include the ability to specify transactions that are legally binding on one or both parties. Naturally, messages must be tamper-proof and signed.
- **Non-repudiation**: If non-repudiation is required, copies of various pieces of information must be kept to avoid situations in which one partner argues that they never agreed to a particular thing.
- **Security**: As you might imagine, performing critical business functions over the Internet using Web Services brings up a host of issues. ebXML includes a technical report, *Technical Architecture Risk Assessment v 1.0*, which covers various relevant issues. (See Resources on page 32 ).

# Section 3. Architecture

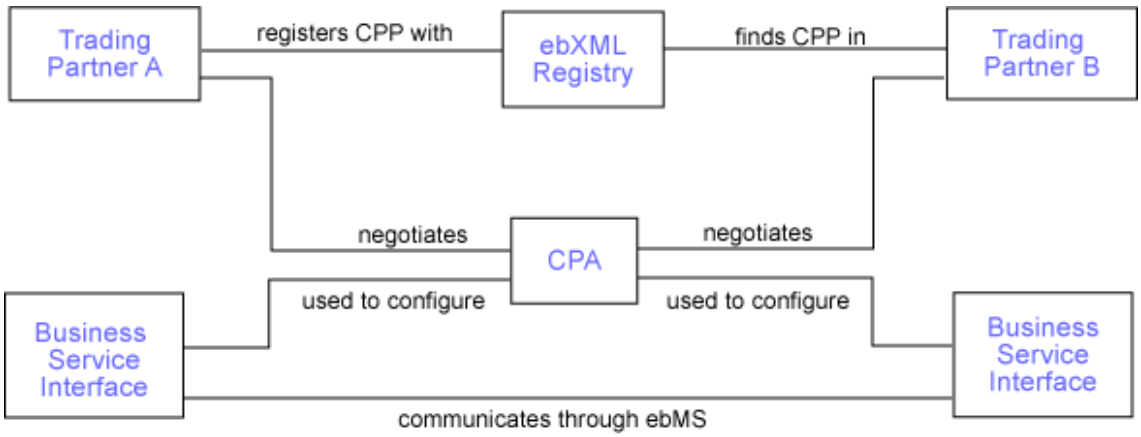
## System overview

Now that you've got a basic idea of how ebXML works, you can learn about the actual architecture, which involves a number of overlapping technologies working together.

Let's start by looking at the overall process. Then you can look at Business Processes, and at some of the individual technologies behind an ebXML implementation.

On the surface, using an ebXML system is straightforward. It involves the following steps:

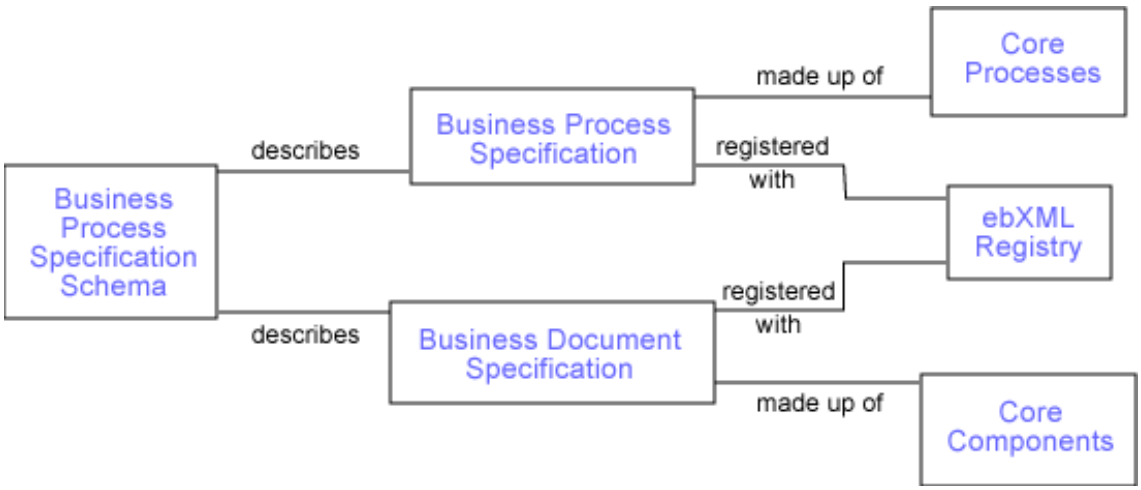
1. Search for a Trading Partner.
2. Create a CPA.
3. Negotiate any issues regarding the CPA.
4. Configure both *Business System Interfaces* using the CPA.
5. Begin performing Business Processes.




## Business Processes and Business Documents

Both *Business Processes* and *Business Documents* are designed and documented prior to their use, and are usually composed from existing components and processes.

For example, Business Processes may be composed from existing Core Processes documented in a business library or other registry. Business Documents are normally composed from existing Core Components in a registry. Both are documented using the *Business Process Specification Schema* (BPSS) and stored in an ebXML registry so that they can be referenced from CPPs, CPAs, and other structures.




## The Business Process Specification Schema

Business models define how business processes are discovered, defined, and documented. In ebXML, you can accomplish this by using the *Unified Modeling Methodology* (UMM). UMM is not required. Just as an XML Schema or DTD provides a vocabulary for data within an XML document, the UMM provides a common language that can be used by those individuals who define business processes.

The BPSS is a subset of the UMM. The BPSS is typically expressed in the *Unified Modeling Language* (UML) and translated to a XML Schema or DTD using production rules. In this way, the common language that the UMM uses to think about and discuss business processes becomes a common language through which you can describe processes using XML.

As seen in the Business Processes and Business Documents on page 7 , the BPSS is used to define both the *Business Processes* and the *Business Documents* they involve. This tutorial discusses the BPSS in the next section, Designing the system: Business Processes on page 10 .

## The Registry Information Model

Once Business Processes and Business Documents are defined, you can store them in an ebXML registry along with CPPs and CPAs, classifications, and other objects. One goal of your ebXML project is to define a structure that organizes all this disparate information that is in one place. That structure is the *Registry Information Model*.

It's important to note that an ebXML registry does not store actual documents or specifications, but rather metadata about documents. It consists of a collection of `RegistryEntry` objects of various types, including `Organizations`, `Packages`, `Slots` (of information about an object), and `Associations` between objects.

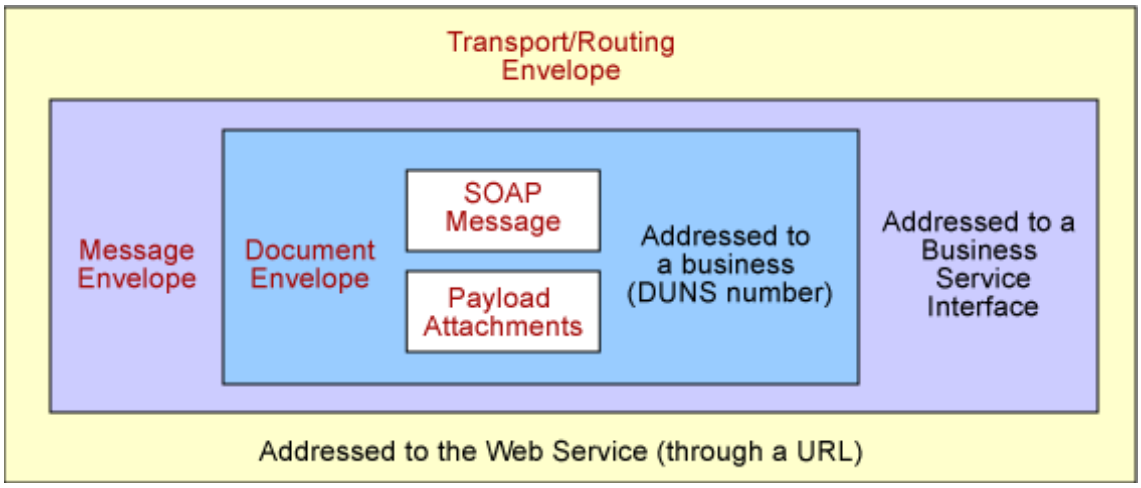
You can query the registry programmatically, and you can update it so that companies can add information.

---

## Message structure

One important goal of the ebXML project is to use an open message format that can accommodate extensions later. The ebXML Message Service (ebMS) is based on the SOAP with Attachments specification. Messages contain a single Message Header. The Message Header consists of a complete SOAP message, and one or more payload attachments. These payload attachments are the actual documents being transferred. The SOAP message simply contains information about the message, such as the `ConversationId`.

This message is packaged in a Document Envelope, which is then packaged in a Transport Envelope. Each of these envelopes contains an appropriate addressing scheme, as seen in the following figure.



# Section 4. Designing the system: Business Processes

## Analyzing your business

To effectively implement an ebXML system, you need to understand just what you're trying to accomplish. It might be something as simple as finding a supplier for cabbages, but in most cases companies execute business functions in a way that is similar to the way they currently work.

The difference is that now they need to express those business processes in a standard way: a *Business Process Specification*.

You can go about this in several ways. You might choose to simply jump in and start creating the Business Process Specification, but unless the process is ludicrously simple, that's usually a recipe for disaster. If you do use a methodology of some sort, the ebXML architecture says it should be the UMM. Then, all parties can communicate effectively about what's discovered and defined. (The UMM is a major topic in itself, well beyond the scope of this tutorial. See Resources on page 32 for more information on the UMM.)

The goal of this analysis is to define Business Processes and Business Information. *Business Processes* are something that a business does. *Business Information* is information that a business uses to do the processes, normally expressed in the form of *Business Documents*.

---

## Business Processes

A Business Process is something that a business does, such as buying hot dog buns or selling a service. It involves the exchange of information between two or more *Trading Partners* in some predictable way.

For example, Bamboo Ltd. wants to buy clipping services from Custom News Inc. Doing that involves several steps. Assume that all of the technical details have already been worked out. First, Bamboo has to choose the categories in which it wants clippings; then it needs to place the order. Once it receives the clippings, Bamboo needs to offer payment information (or perhaps it must offer payment information before receiving the clippings). These decisions are part of the analysis process.

Building a Business Process is a bit like building a machine: Each part is a *Business Collaboration* that works in concert with all of the other parts.

---

## Business Collaborations

A Business Collaboration is a choreographed set of *Business Transaction Activities*, in which two Trading Partners exchange documents.

Several variations on Business Collaborations exist. The most common one is a *Binary Collaboration*, in which two partners exchange documents. A *Multiparty Collaboration* takes place when information is exchanged between more than two parties...or does it? Multiparty Collaborations are actually choreographed Binary Collaborations. At all times information is flowing between two and only two parties, though the same information may be sent to different parties in separate Binary Collaborations.

Business Collaborations can also be nested, in that a Business Transaction Activity may also be a Business Collaboration. For example, the process of choosing categories for news clippings might include several steps involving a request to a third party to get a list of available categories. This collaboration (like that between Bamboo Ltd. and the category provider) is a Business Collaboration that acts as a *building block* to build the main collaboration.

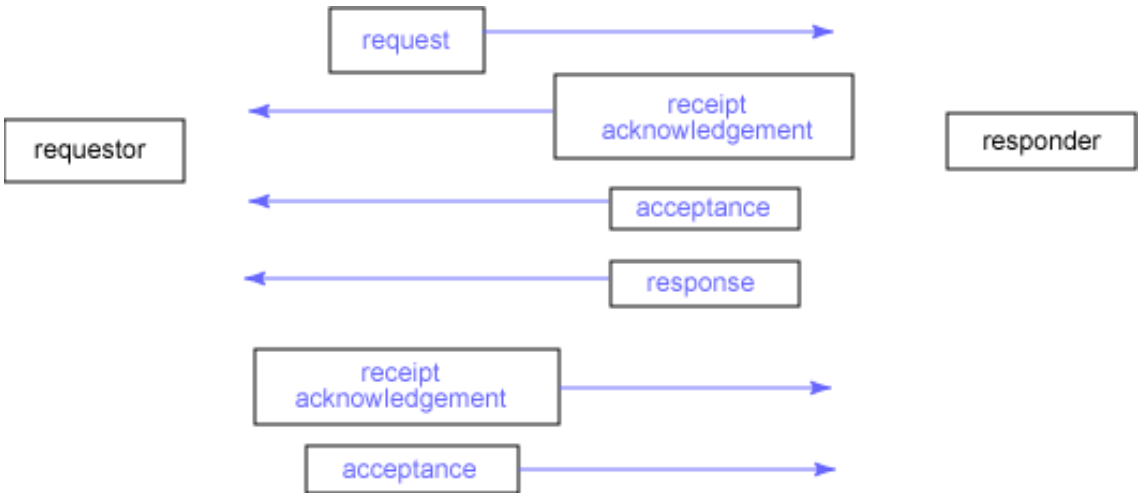
At its lowest level, a Business Collaboration can be broken down into *Business Transactions*.

---

# Business Transactions

A Business Transaction is the atomic level of work in a Business Process. It either succeeds or fails completely.

Business Transactions are transactions in which Trading Partners actually transfer Business Documents. In a Business Transaction, the requesting party, or *requestor*, sends the request to the responding party, or *responder*. This request may be an actual request, in which case the responder then sends back a response -- or it may be a simple notification, in which case the responder simply accepts the information.




---

## Business Transactions and control

The requestor sends the request, and control of the transaction passes to the responder. In this case, the responder sends a receipt acknowledgment, then an acceptance acknowledgment, and then the response itself. At this point, control passes back to the requestor, which sends back a receipt and acceptance acknowledgment.

Note that the requirement of receipts and acceptances is defined at the transactional level; this affects whether and when control passes to each party. For example, in the above case, control stays with the responder until it sends the required response. Then control passes back to the requestor. If neither a response nor an acceptance acknowledgment are required, control is passed back to the requestor as soon as the responder sends the receipt acknowledgment.

---

## Choreography and states

As mentioned in Business Collaborations on page 10 , a Business Collaboration consists of choreographed Business Transactions. That choreography is expressed in terms of states and the transitions between them.

In fact, a *Business Activity* is known as an abstract state, with Business Collaborations and Business Transaction Activities known as concrete states. Auxiliary states include *start*, *fork*, *synchronization*, and *completion* (which takes the form of either *success* or *failure*).

As the collaboration proceeds, it transitions from one state to the next. In some cases in which a particular requirement exists (such as document receipt or validation), a guard *gates* the transition to control whether or not it takes place.

All of the logic for state management is built as part of the Business Service Interface portion of the application.

---

## Business Documents

So where do these Business Documents being transferred come from?

Business Documents are composed of *Business Information Objects*, or smaller chunks of information that have previously been identified. These chunks, or components, don't carry any information, of course. They are merely structures, such as an XML Schema or a DTD, that define information and how it must be presented. The end result is a predictable structure into which information is placed, so that the receiver of the final document can interpret it to extract the information.

One of the goals of ebXML is to create a set of *Core Components* that will be stored in a *Core Library* and be available for the building of Business Documents. As of this writing, the Core Components were not yet complete. It's likely that early adopters of ebXML will find themselves in a situation where the components they need for their Business Documents are not available in any library.

---

## Reusable components

Both Business Processes and Business Documents are always built out of existing components. So what happens if those components are not available?

The first step in building documents is to determine the required attributes and search the registry for documents that already contain them. If the Document contains everything that is needed, simply use it rather than creating a new one. If not, create a new component or add attributes to an existing Document and register the new or changed information with the registry. Use the newly registered components to create the new Document.

Register the completed Document structure with the registry so that it can be used.

Business Processes are built the same way, with Core Processes stored in the registry.

---

## A sample Business Process Specification

The end result of all of this analysis is a Business Process Specification. Here is a simple example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ProcessSpecification SYSTEM "ebXMLProcessSpecification-v1.00.dtd">
<ProcessSpecification name="NewsClips" version="1.2" uuid="[1234-5678]">
    <BusinessDocument name="Clipping Request"/>
    <BusinessDocument name="ClipList"/>
    <Package name="Ordering">
        <BinaryCollaboration name="Request Clippings">
            <InitiatingRole name="requestor"/>
            <RespondingRole name="provider"/>
            <BusinessTransactionActivity name="Clipping Request"
                    businessTransaction="Clipping Request"
                    fromAuthorizedRole="requestor" toAuthorizedRole="provider"/>
        </BinaryCollaboration>
        <BinaryCollaboration name="Fulfillment" timeToPerform="P1D">
            <Documentation>
                 timeToPerform = Period: 1 day from start of transaction
            </Documentation>
            <InitiatingRole name="buyer"/>
            <RespondingRole name="seller"/>
            <BusinessTransactionActivity name="Create Order"
                    businessTransaction="Create Order" fromAuthorizedRole="buyer"
                    toAuthorizedRole="seller"/>
            <BusinessTransactionActivity name="Notify creation"
```

```
                businessTransaction="Notify of information creation"
                fromAuthorizedRole="buyer" toAuthorizedRole="seller"/>
            <Start toBusinessState="Create Order"/>
            <Transition fromBusinessState="Create Order"
                        toBusinessState="Notify creation"/>
            <Success fromBusinessState="Notify creation"
                        conditionGuard="Success"/>
            <Failure fromBusinessState="Notify creation"
                        conditionGuard="BusinessFailure"/>
        </BinaryCollaboration>
        <BusinessTransaction name="Clipping Request">
            <RequestingBusinessActivity name="">
                <DocumentEnvelope isPositiveResponse="true"
                        businessDocument="Clipping Request"/>
            </RequestingBusinessActivity>
            <RespondingBusinessActivity name="">
                <DocumentEnvelope isPositiveResponse="true"
                        businessDocument="Clipping List"/>
            </RespondingBusinessActivity>
        </BusinessTransaction>
        <BusinessTransaction name="Create Order">
            <RequestingBusinessActivity name="" isNonRepudiationRequired="true"
                    timeToAcknowledgeReceipt="P1D"
                    timeToAcknowledgeAcceptance="P1D">
                <DocumentEnvelope isPositiveResponse="true"
                                businessDocument="Purchase Order"/>
            </RequestingBusinessActivity>
            <RespondingBusinessActivity name="" isNonRepudiationRequired="true"
                    timeToAcknowledgeReceipt="P1D">
                <DocumentEnvelope isPositiveResponse="true"
                                businessDocument="PO Acknowledgement"/>
            </RespondingBusinessActivity>
        </BusinessTransaction>
        <BusinessTransaction name="Notify Creation">
            <RequestingBusinessActivity name="">
                <DocumentEnvelope isPositiveResponse="true"
                                businessDocument="ClipList"/>
            </RequestingBusinessActivity>
            <RespondingBusinessActivity name="" timeToAcknowledgeReceipt="P1D"/>
        </BusinessTransaction>
    </Package>
</ProcessSpecification>
```

# Section 5. CPPs and CPAs

## Collaboration Protocol Profiles (CPPs)

Communicating with a Trading Partner requires understanding what they can and cannot do. The CPP identifies the Business Processes in which an organization takes part, and the role (for example, `buyer` or `insurer`) they play within that collaboration. It also defines the delivery channels and transport protocols (such as HTTP) that the organization supports.

Depending on the level of security desired, this document may also be digitally signed.

This document is stored in an ebXML Registry with a Globally Unique Identifier (GUID) that becomes part of the metadata for the entry. Note that the registry does not and cannot insert the GUID into the document, because that would invalidate any signatures on the document.

The information within the CPP is available to be searched on, so a potential Trading Partner can determine whether the organization has the capabilities to do business.

---

## Overall structure of a CPP

The structure of a CPP consists of a root `CollaborationProtocolProfile` element with `PartyInfo`, `Packaging`, `Signature`, and `Comment` elements:

```
<CollaborationProtocolProfile
         xmlns="http://www.ebxml.org/namespaces/tradePartner"
         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
         xmlns:xlink="http://www.w3.org/1999/xlink"
         version="1.1">
    <PartyInfo>
        ...
        <!--REQUIRED, Repeatable-->
        ...
    </PartyInfo>
    <Packaging id="ID">
        ...
        <!--REQUIRED-->
        ...
    <Packaging>
    <ds:Signature>
        ...
        <!--OPTIONAL-->
        ...
    </ds:Signature>
    <Comment>
        ...
        <!-- OPTIONAL -->
        ...
    </Comment>
</CollaborationProtocolProfile>
```

The root element, `CollaborationProtocolProfile`, requires three namespace declarations:

- `http://www.ebxml.org/namespaces/tradePartner` is the default namespace.
- `http://www.w3.org/2000/09/xmldsig#`, is the namespace for XML Digital Signature, and is included to allow signing of CPPs.
- `http://www.w3.org/1999/xlink` is the XLink namespace, which allows the CPP to reference external information.

As for the content of the document itself, only the `PartyInfo` and `Packaging` elements are required.

---

# PartyInfo

The `PartyInfo` element provides information about the organization. Multiple `PartyInfo` elements can be added to include information about different parts of an organization.

The `PartyInfo` element includes:

- One or more `PartyId` elements. These elements provide a logical identifier for the organization, such as a DUNS number.
- One `PartyRef` element. This element points to an external resource with more information about the organization.
- One or more `CollaborationRole` elements. These elements are the heart of the CPP, providing information on the Business Processes in which the party engages, and the roles it plays within those processes. The `CollaborationRole` element directly references a Business Process Specification stored in the registry.
- One or more `Certificate` elements. These elements identify the party's security certificates.
- One or more `DeliveryChannel` elements. These elements define the ways in which the party can receive messages, including references to both a document exchange, or message protocol, and a transport protocol layer described below.
- One or more `Transport` elements. These elements provide specifics for the transport layers referenced in the `DeliveryChannel` elements. Transport layers may include HTTP, SMTP, or other transport protocols.
- One or more `DocExchange` elements. These elements provide specifics for the document exchanges referenced in the `DeliveryChannel` elements. The document exchange represents the messaging protocol, such as ebMS.

Each of these elements has its own child elements, as seen in the sample CPP document.

---

# Packaging

The Packaging element provides information about the way in which messages are actually constructed. Messages are processed as SOAP Messages with Attachments, and the `Packaging` element provides information on how these messages are organized.

The Packaging element has three potential child elements:

- The `ProcessingCapabilities` element is an empty element with two required attributes, `generate` and `parse`, which indicate whether the system is capable of creating or reading messages.
- The `SimplePart` element defines message pieces that consist of a certain Multipurpose Internet Mail Extenstions (MIME) type. Pieces are identified so that they can be referenced within the `CompositeList` element.
- The `CompositeList` element provides information about composites or encapsulations of `SimpleParts`. This element is optional, and will not appear if parts are sent individually.

These elements are seen in the sample CPP that follows.

---

# A sample CPP

This sample from the ebXML site shows the various parts of a complete CPP:

```
<?xml version="1.0" encoding="UTF-8" ?>
<tp:CollaborationProtocolProfile
         xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"
         xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
         xmlns:xlink="http://www.w3.org/1999/xlink"
         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
         xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
       http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd"
        tp:version="1.1">
   <tp:PartyInfo>
      <tp:PartyId tp:type="DUNS">123456789</tp:PartyId>
      <tp:PartyRef tp:href="http://example.com/about.html" />
      <tp:CollaborationRole tp:id="N00">
         <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
                xlink:type="simple"
                xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
         <tp:Role tp:name="buyer" xlink:type="simple"
                xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>
         <tp:CertificateRef tp:certId="N03" />
         <tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402">
         <tp:Service tp:type="uriReference"
                  >uri:example.com/services/buyerService</tp:Service>
             <tp:Override tp:action="orderConfirm" tp:channelId="N07"
                tp:packageId="N0402" xlink:type="simple"
           xlink:href="http://ebxml.org/processes/buySell.xml#orderConfirm"/>
         </tp:ServiceBinding>
```

```
      </tp:CollaborationRole>
      <tp:Certificate tp:certId="N03">
        <ds:KeyInfo />
      </tp:Certificate>
       <tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
                           tp:docExchangeId="N06">
           <tp:Characteristics tp:syncReplyMode="none"
                   tp:nonrepudiationOfOrigin="true"
                   tp:nonrepudiationOfReceipt="false"
                   tp:secureTransport="true" tp:confidentiality="true"
                   tp:authenticated="true" tp:authorized="false" />
       </tp:DeliveryChannel>
       <tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
                            tp:docExchangeId="N06">
           <tp:Characteristics tp:syncReplyMode="none"
                   tp:nonrepudiationOfOrigin="true" tp:confidentiality="true"
                   tp:nonrepudiationOfReceipt="false"
                   tp:secureTransport="false" tp:authenticated="true"
                   tp:authorized="false" />
       </tp:DeliveryChannel>
       <tp:Transport tp:transportId="N05">
          <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
          <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
          <tp:Endpoint tp:uri="https://www.example.com/servlets/ebxmlhandler"
                       tp:type="allPurpose" />
          <tp:TransportSecurity>
             <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
             <tp:CertificateRef tp:certId="N03" />
          </tp:TransportSecurity>
       </tp:Transport>
       <tp:Transport tp:transportId="N08">
          <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
          <tp:ReceivingProtocol tp:version="1.1">SMTP</tp:ReceivingProtocol>
          <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
                       tp:type="allPurpose" />
       </tp:Transport>
       <tp:DocExchange tp:docExchangeId="N06">
          <tp:ebXMLBinding tp:version="0.98b">
               <tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
                        tp:idempotency="true"
                        tp:messageOrderSemantics="Guaranteed">
                   <tp:Retries>5</tp:Retries>
                   <tp:RetryInterval>30</tp:RetryInterval>
                   <tp:PersistDuration>P1D</tp:PersistDuration>
               </tp:ReliableMessaging>
               <tp:NonRepudiation>
                   <tp:Protocol
                     >http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
                   <tp:HashFunction
                    >http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
                   <tp:SignatureAlgorithm
               >http://www.w3.org/2000/09/xmldsig#dsa-sha1</tp:SignatureAlgorithm>
                   <tp:CertificateRef tp:certId="N03" />
               </tp:NonRepudiation>
               <tp:DigitalEnvelope>
                   <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
                   <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
                      <tp:CertificateRef tp:certId="N03" />
               </tp:DigitalEnvelope>
           </tp:ebXMLBinding>
       </tp:DocExchange>
    </tp:PartyInfo>
    <tp:Packaging tp:id="N0402">
        <tp:ProcessingCapabilities tp:parse="true" tp:generate="true" />
```

```
        <tp:SimplePart tp:id="N40" tp:mimetype="text/xml">
            <tp:NamespaceSupported
              tp:location=
              "http://ebxml.org/project_teams/transport/messageService.xsd"
              tp:version="0.98b"
           >http://www.ebxml.org/namespaces/messageService</tp:NamespaceSupported>
            <tp:NamespaceSupported tp:location=
             "http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"
              tp:version="1.0"
                 >http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>
        </tp:SimplePart>
        <tp:SimplePart tp:id="N41" tp:mimetype="text/xml">
            <tp:NamespaceSupported tp:version="1.0"
                 tp:location="http://ebxml.org/processes/buysell.xsd"
               >http://ebxml.org/processes/buysell.xsd</tp:NamespaceSupported>
        </tp:SimplePart>
        <tp:CompositeList>
            <tp:Composite tp:id="N42" tp:mimetype="multipart/related"
                  tp:mimeparameters="type=text/xml;">
                <tp:Constituent tp:idref="N40" />
                <tp:Constituent tp:idref="N41" />
            </tp:Composite>
        </tp:CompositeList>
    </tp:Packaging>
    <tp:Comment tp:xml_lang="en-us">buy/sell agreement between example.com and
          contrived-example.com</tp:Comment>
</tp:CollaborationProtocolProfile>
```

---

# Collaboration Protocol Agreements (CPAs)

The CPA is, in many ways, simply the intersection of two CPPs. For example, if one partner agrees to act as buyer and the other as seller, their roles will mesh like gears. The only remaining concerns involve ironing out issues such as packaging and other details.

Once both sides have reached agreement, they each take an electronic copy of the same CPA and use it to configure their systems. The CPA may also be added to the registry for reference, but this is not a requirement.

---

# Overall structure of a CPA

The structure of the CPA is similar to that of the CPP:

```
<CollaborationProtocolAgreement
       xmlns="http://www.ebxml.org/namespaces/tradePartner"
       xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
       xmlns:xlink = "http://www.w3.org/1999/xlink"
       cpaid="http://www.example.com/cpas/clipCPA"
       version="1.7">
    <Status value = "proposed"/>
    <Start>1988-04-07T18:39:09</Start>
    <End>1990-04-07T18:40:00</End>
```

```
    <ConversationConstraints invocationLimit = "250"
                                concurrentConversations = "5"/>
    <PartyInfo>
        ...
        <!--REQUIRED, repeatable-->
        ...
    </PartyInfo>
    <PartyInfo>
        ...
        <!--REQUIRED, repeatable-->
        ...
    </PartyInfo>
    <Packaging id="N20">
        ...
        <!--REQUIRED, repeatable-->
        ...
    </Packaging>
    <ds:Signature>
        <!--OPTIONAL-->
    </ds:Signature>
    <Comment xml:lang="en-gb">
        <!--OPTIONAL-->
    </Comment>
</CollaborationProtocolAgreement>
```

Like the CPP, the CPA defines namespaces on its root element (in this case the `CollaborationProtocolAgreement`) and a version to distinguish any subsequent changes. The CPA also includes a `cpaid` attribute that both parties use. The CPP/CPA specification recommends that this attribute consist of a unique URI value.

The `PartyInfo`, `Packaging`, `Signature`, and `Comment` elements have the same meaning they have for a CPP, except that `PartyInfo` elements are included for both parties involved in the agreement.

Typically, one party generates a CPA and offers it to the other party for approval, so the `Status` element shows where the document is in this process. The possible values are `proposed`, `agreed`, and `signed`.

The `Start` and `End` elements represent, in Coordinated Universal Time, the beginning and end of the period during which this CPA is active. (Note that if an `invocationLimit` value is given on the `ConversationConstraints` element, the CPA may expire before the end date.)

Finally, the optional `CoversationConstraints` element defines the finite number of *conversations* that may be held under this CPA, and the number that may be held concurrently.

---

# A sample CPA

This example from the ebXML site builds on the previous CPP:

```
<?xml version="1.0" ?>
<tp:CollaborationProtocolAgreement
        xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"
```

```
        xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
        xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
         http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        tp:cpaid="http://www.example.com/cpas/clipCPA"
        tp:version="1.2">
    <tp:Status tp:value="proposed" />
    <tp:Start>2001-05-20T07:21:00Z</tp:Start>
    <tp:End>2002-05-20T07:21:00Z</tp:End>
    <tp:ConversationConstraints tp:invocationLimit="100"
                                tp:concurrentConversations="100"/>
    <tp:PartyInfo>
        <tp:PartyId tp:type="DUNS">123456789</tp:PartyId>
        <tp:PartyRef xlink:href="http://example.com/about.html"/>
        <tp:CollaborationRole tp:id="N00">
            <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
                 xlink:type="simple"
                 xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
            <tp:Role tp:name="buyer" xlink:type="simple"
                 xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>
            <tp:CertificateRef tp:certId="N03" />
            <tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402">
                 <tp:Service tp:type="uriReference"
                     >uri:example.com/services/buyerService</tp:Service>
                 <tp:Override tp:action="orderConfirm" tp:channelId="N08"
                     tp:packageId="N0402" xlink:type="simple" xlink:href=
                     "http://ebxml.org/processes/buySell.xml#orderConfirm"/>
            </tp:ServiceBinding>
        </tp:CollaborationRole>
        <tp:Certificate tp:certId="N03">
            <ds:KeyInfo />
        </tp:Certificate>
        <tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
            tp:docExchangeId="N06">
            <tp:Characteristics tp:syncReplyMode="none"
                tp:nonrepudiationOfOrigin="true"
                tp:nonrepudiationOfReceipt="false" tp:secureTransport="true"
                tp:confidentiality="true" tp:authenticated="true"
                tp:authorized="false" />
        </tp:DeliveryChannel>
        <tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
            tp:docExchangeId="N06">
            <tp:Characteristics tp:syncReplyMode="none"
                tp:nonrepudiationOfOrigin="true" tp:secureTransport="false"
                tp:nonrepudiationOfReceipt="false" tp:confidentiality="true"
                tp:authenticated="true" tp:authorized="false" />
        </tp:DeliveryChannel>
        <tp:Transport tp:transportId="N05">
            <tp:SendingProtocol
                     tp:version="1.1">HTTP</tp:SendingProtocol>
            <tp:ReceivingProtocol
                     tp:version="1.1">HTTP</tp:ReceivingProtocol>
            <tp:Endpoint tp:type="allPurpose"
                     tp:uri="https://www.example.com/servlets/ebxmlhandler"/>
            <tp:TransportSecurity>
                <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
                <tp:CertificateRef tp:certId="N03" />
            </tp:TransportSecurity>
        </tp:Transport>
        <tp:Transport tp:transportId="N18">
            <tp:SendingProtocol
                         tp:version="1.1">HTTP</tp:SendingProtocol>
            <tp:ReceivingProtocol
```

```
                        tp:version="1.1">SMTP</tp:ReceivingProtocol>
            <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
                         tp:type="allPurpose" />
        </tp:Transport>
        <tp:DocExchange tp:docExchangeId="N06">
            <tp:ebXMLBinding tp:version="0.98b">
                <tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
                                      tp:idempotency="true"
                   tp:messageOrderSemantics="Guaranteed">
                    <tp:Retries>5</tp:Retries>
                    <tp:RetryInterval>30</tp:RetryInterval>
                    <tp:PersistDuration>P1D</tp:PersistDuration>
                </tp:ReliableMessaging>
                <tp:NonRepudiation>
                    <tp:Protocol
                          >http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
                    <tp:HashFunction
                    >http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
                    <tp:SignatureAlgorithm
           >http://www.w3.org/2000/09/xmldsig#dsa-sha1</tp:SignatureAlgorithm>
                    <tp:CertificateRef tp:certId="N03" />
                </tp:NonRepudiation>
                <tp:DigitalEnvelope>
                    <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
                    <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
                    <tp:CertificateRef tp:certId="N03" />
                </tp:DigitalEnvelope>
            </tp:ebXMLBinding>
        </tp:DocExchange>
    </tp:PartyInfo>
    <tp:PartyInfo>
        <tp:PartyId tp:type="DUNS">987654321</tp:PartyId>
        <tp:PartyRef xlink:type="simple"
                     xlink:href="http://contrived-example.com/about.html" />
        <tp:CollaborationRole tp:id="N30">
            <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
                xlink:type="simple"
                xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
            <tp:Role tp:name="seller" xlink:type="simple"
                xlink:href="http://ebxml.org/processes/buySell.xml#seller" />
           <tp:CertificateRef tp:certId="N33" />
           <tp:ServiceBinding tp:channelId="N34" tp:packageId="N0402">
               <tp:Service tp:type="uriReference"
                    >uri:example.com/services/sellerService</tp:Service>
           </tp:ServiceBinding>
        </tp:CollaborationRole>
        <tp:Certificate tp:certId="N33">
            <ds:KeyInfo />
        </tp:Certificate>
        <tp:DeliveryChannel tp:channelId="N34" tp:transportId="N35"
                            tp:docExchangeId="N36">
            <tp:Characteristics tp:nonrepudiationOfOrigin="true"
                            tp:nonrepudiationOfReceipt="false"
              tp:secureTransport="true" tp:confidentiality="true"
                                    tp:authenticated="true"
              tp:authorized="false"/>
        </tp:DeliveryChannel>
        <tp:Transport tp:transportId="N35">
            <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
            <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
            <tp:Endpoint
                   tp:uri="https://www.contrived-example.com/servlets/ebxmlhandler"
                   tp:type="allPurpose" />
            <tp:TransportSecurity>
```

```
                <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
                <tp:CertificateRef tp:certId="N33" />
            </tp:TransportSecurity>
        </tp:Transport>
        <tp:DocExchange tp:docExchangeId="N36">
            <tp:ebXMLBinding tp:version="0.98b">
                <tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
                       tp:idempotency="true"
                       tp:messageOrderSemantics="Guaranteed">
                    <tp:Retries>5</tp:Retries>
                    <tp:RetryInterval>30</tp:RetryInterval>
                    <tp:PersistDuration>P1D</tp:PersistDuration>
                </tp:ReliableMessaging>
                <tp:NonRepudiation>
                    <tp:Protocol
                         >http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
                    <tp:HashFunction
                       >http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
                    <tp:SignatureAlgorithm
         >http://www.w3.org/2000/09/xmldsig#dsa-sha1</tp:SignatureAlgorithm>
                    <tp:CertificateRef tp:certId="N33" />
                </tp:NonRepudiation>
                <tp:DigitalEnvelope>
                    <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
                    <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
                    <tp:CertificateRef tp:certId="N33" />
                </tp:DigitalEnvelope>
            </tp:ebXMLBinding>
        </tp:DocExchange>
    </tp:PartyInfo>
    <tp:Packaging tp:id="N0402">
        <tp:ProcessingCapabilities tp:parse="true" tp:generate="true" />
        <tp:SimplePart tp:id="N40" tp:mimetype="text/xml">
            <tp:NamespaceSupported
               tp:location=
               "http://ebxml.org/project_teams/transport/messageService.xsd"
               tp:version="0.98b"
        >http://www.ebxml.org/namespaces/messageService</tp:NamespaceSupported>
            <tp:NamespaceSupported tp:version="1.0"
         tp:location=
               "http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"
               >http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>
        </tp:SimplePart>
        <tp:Simplepart tp:id="N41" tp:mimetype="text/xml">
               <tp:NamespaceSupported tp:version="1.0"
                    tp:location="http://ebxml.org/processes/buysell.xsd"
           >http://ebxml.org/processes/buysell.xsd</tp:NamespaceSupported>
        </tp:SimplePart>
        <tp:CompositeList>
            <tp:Composite tp:id="N42" tp:mimetype="multipart/related"
                tp:mimeparameters="type=text/xml;">
                <tp:Constituent tp:idref="N40" />
                <tp:Constituent tp:idref="N41" />
             </tp:Composite>
        </tp:CompositeList>
    </tp:Packaging>
    <tp:Comment xml:lang="en-us">buy/sell agreement between example.com and
contrived-example.com</tp:Comment>
</tp:CollaborationProtocolAgreement>
```

# Section 6. Using the registry

## ebXML Registries

A constant theme throughout this tutorial has been the use of an ebXML Registry to store information for current and future use. When used with a repository, which actually stores the referenced objects, an ebXML Registry provides a means for finding organizations, CPPs, components, Business Process Specifications, classification schemes, and even software or other objects.

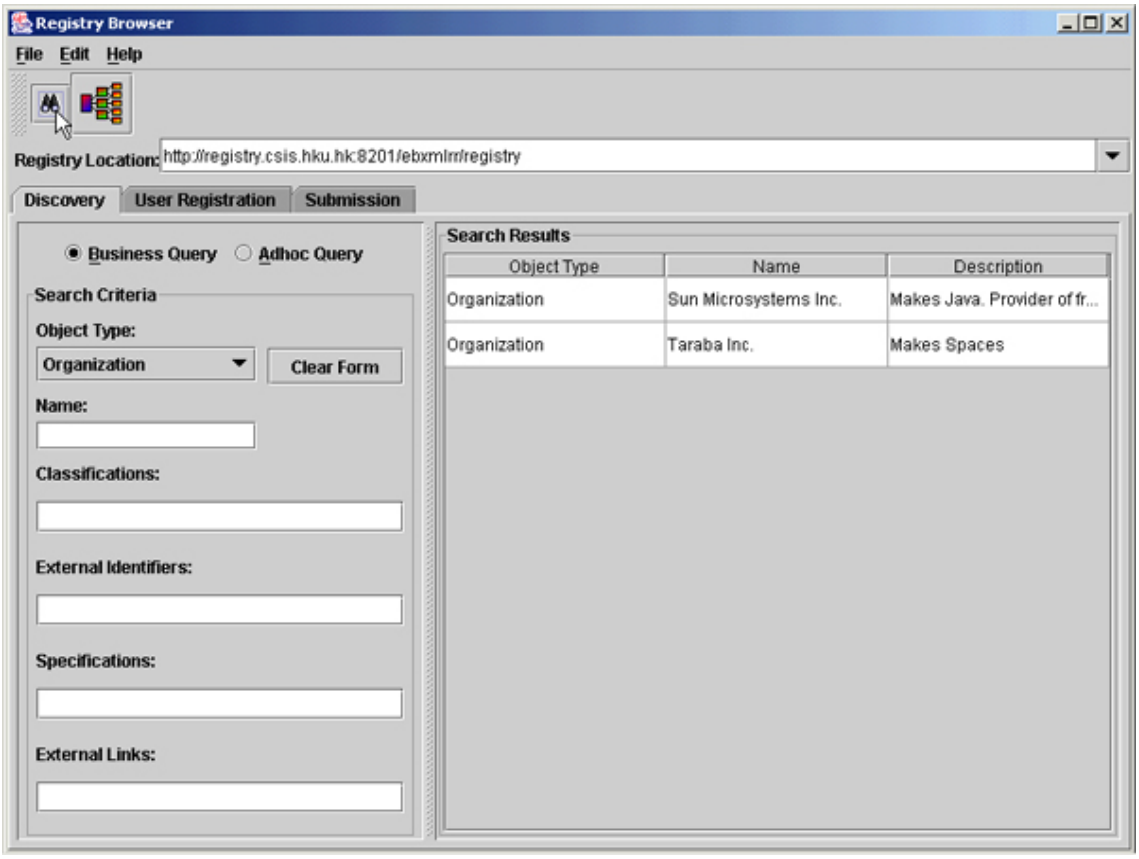
At least one commercial ebXML Registry implementation is already on the market, but most implementations are still fairly rudimentary. (See Resources on page 32 for a list of registry implementations.) The Center for E-Commerce Infrastructure Development is hosting the OASIS ebXML v2 Registry Reference Implementation. You can take a look at it to get an idea of how registries work.

Open your browser and point it at *http://ebxmlrr.sourceforge.net/registryBrowser/registryBrowser.jnlp*. Java Web Start downloads the required components for the registry browser to your computer. (Java 1.4 is required to run the registry browser.)

---

## Browsing the registry

Once Java Web Start downloads the software, a window opens to reveal the main registry browser. This software is just an example; you can build your own using the techniques briefly described in later panels.



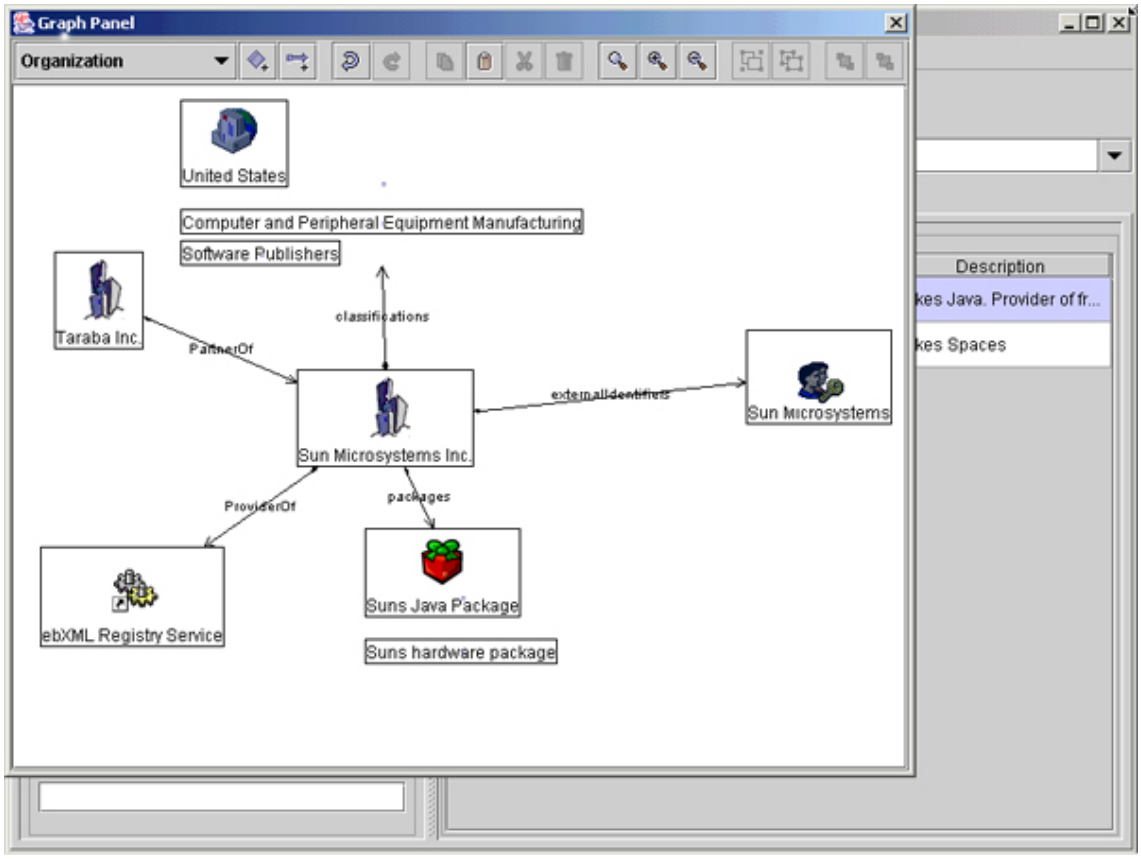

Choose the CECID registry and click the binoculars in the upper left-hand corner to execute a search. Because the object type is set to *Organization*, only organizations are shown. (Unfortunately, the registry doesn't provide write access, and there isn't a lot of data in it.)

Unless you enter specific parameters, the browser shows all information in the registry. Choose different object types from the pulldown menu and look at the data returned.

---

# Registry associations

Information in an ebXML Registry typically doesn't exist in a vacuum. Instead, objects are associated with other objects in various ways. To view some of these relations, right-click the entry for **Sun Microsystems** and select **Browse RegistryObject**. After a few (or in some cases, quite a few) seconds, a second window appears. Right-click the **Sun Microsystems icon** and select **Show Related Objects**.




Notice that in addition to information about the organization itself, which you can get by simply clicking the original entry, you can also see related objects, and how they're related. For example, `Classifications` are entries unto themselves within the registry, but they're also associated with other objects. Sun provides an ebXML Registry Service, which a CPP would show.

# Section 7. The ebXML Message Service

## The ebXML Message Service

Now that you've read about creating messages, it's time to find out about sending them.

Considering that everything in ebXML rests on the concept of sending messages, the ebMS is surprisingly simple. Each message consists of a number of MIME parts. The first MIME part consists of a SOAP message that provides identification for the message and other information crucial to its processing, and the rest consist of payload parts that carry the documents being transferred.

The actual implementation is known as the *Message Service Handler* (MSH). It starts with a layer on top of the actual application, called the *Message Service Interface* (MSI). Once requests come through the MSI, the message header is created, including information such as timestamps, digital signatures, and relevant information from the CPA. The delivery module then packs up and prepares the message itself for delivery.

ebMS does not specify a particular protocol for delivering messages. Although HTTP is the most common protocol, messages can be bound to virtually any protocol, such as SMTP or IIOP.

---

## The SOAP message

The message header consists of a SOAP message that contains the relevant information about the message.

A SOAP message consists of an `Envelope` that carries a `Header` and a `Body`. Extensions, like those for ebXML-specific information (such as a CPA identifier), can be added to the header through the use of namespaces. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
 <SOAP:Envelope  xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
      xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
   http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd" >
       <SOAP:Header xmlns:eb=
     "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
        xsi:schemaLocation=
     "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd" >

...

     </SOAP:Header>
     <SOAP:Body  xmlns:eb=
     "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
```

```
        xsi:schemaLocation=
     "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd" >

...

     </SOAP:Body>
 </SOAP:Envelope>
```

Notice that all of the necessary namespaces are defined, and that the schema location is included so that the message can be validated at the receiving end, if necessary.

---

# The SOAP header

Now add the ebXML-specific information:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
       xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
    <SOAP:Header xmlns:eb=
    "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
          xsi:schemaLocation=
    "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:MessageHeader SOAP:mustUnderstand="1" eb:version="2.0">
        <eb:From>
            <eb:PartyId>urn:duns:987654321</eb:PartyId>
        </eb:From>
        <eb:To>
            <eb:PartyId>urn:duns:123456789</eb:PartyId>
        </eb:To>
        <eb:CPAId>http://www.example.com/cpas/clipCPA</eb:CPAId>
        <eb:ConversationId
              >http://www.example.com/cpas/clipCPA@25430</eb:ConversationId>
        <eb:Service>urn:services:ClipProcessing</eb:Service>
        <eb:Action>OrderClips</eb:Action>
        <eb:MessageData>
            <eb:MessageId
            >http://www.example.com/cpas/clipCPA@25430@648124</eb:MessageId>
            <eb:Timestamp>2001-05-16T12:01:00</eb:Timestamp>
        </eb:MessageData>
        <eb:DuplicateElimination/>
    </eb:MessageHeader>
    </SOAP:Header>
    <SOAP:Body xmlns:eb=
         "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
         xsi:schemaLocation=
         "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">

...

    </SOAP:Body>
</SOAP:Envelope>
```

The SOAP header (as opposed to the ebMS header) holds information such as the sender and receiver of the document, the CPA governing the message, and the `ConversationId`.

The `ConversationId` lets the receiver know specifically what transaction or collaboration this message is part of. Similarly, the `MessageId` identifies this particular message. You can use `MessageId` with `DuplicateElimination` to make sure that duplicates will be discarded if multiple copies of the message are sent (for example, because acknowledgment was never received).

The SOAP header also holds information such as the `Service` and `Action`, which help to identify the appropriate Business Process and the action to take within it.

---

# The SOAP body

The `Body` part of the SOAP message provides the information on the actual document being transferred.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
       xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
    <SOAP:Header xmlns:eb=
           "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
           xsi:schemaLocation=
           "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">

...

    </SOAP:Header>
    <SOAP:Body xmlns:eb=
        "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
           xsi:schemaLocation=
        "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
          <eb:Manifest eb:version="2.0">
              <eb:Reference xlink:href="cid:payload1" xlink:role="XLinkRole"
                  xlink:type="simple">
                  <eb:Description xml:lang="en-US">Clip List</eb:Description>
              </eb:Reference>
          </eb:Manifest>
    </SOAP:Body>
</SOAP:Envelope>
```

The `Manifest` element can refer to documents that are located externally, such as those on the Web, or (more commonly) documents that are part of the payload of the ebMS message. Here it refers to the `Content-Id` MIME header.

---

# Pulling it together

The actual message is a series of MIME parts. MIME is best known for the *MIME type*, which is carried by every single page, image, Flash movie, and any other item transferred over the Web as any given type. This information helps an application receiving data to know what to do with it. For example, a Web page has a MIME-type of `text/html`, so the browser knows to render it as HTML. A JPEG image on that page might have a MIME type of `image/x-jpeg`, so the browser knows to assemble it into an image and display it.

In this case, MIME is used for something closer to its original purpose: pulling together disparate resources into a single message. For example:

```
Content-type: multipart/related; boundary="boundaryValue";
type="text/xml"; start="<ebxmlheader@example.com>"


--boundaryValue
Content-ID: <ebxmlheader@example.com>
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
       xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
    <SOAP:Header xmlns:eb=
           "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
           xsi:schemaLocation=
           "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
        <eb:MessageHeader SOAP:mustUnderstand="1" eb:version="2.0">
            <eb:From>
                <eb:PartyId>urn:duns:987654321</eb:PartyId>
            </eb:From>
            <eb:To>
                <eb:PartyId>urn:duns:123456789</eb:PartyId>
            </eb:To>
            <eb:CPAId>http://www.example.com/cpas/clipCPA</eb:CPAId>
            <eb:ConversationId
             >http://www.example.com/cpas/clipCPA@25430</eb:ConversationId>
            <eb:Service>urn:services:ClipProcessing</eb:Service>
            <eb:Action>OrderClips</eb:Action>
            <eb:MessageData>
                <eb:MessageId>
                     http://www.example.com/cpas/clipCPA@25430@648124</eb:MessageId>
                <eb:Timestamp>2001-05-16T12:01:00</eb:Timestamp>
            </eb:MessageData>
            <eb:DuplicateElimination/>
        </eb:MessageHeader>
    </SOAP:Header>
<SOAP:Body xmlns:eb=
           "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
              xsi:schemaLocation=
           "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
        <eb:Manifest eb:version="2.0">
```

```
            <eb:Reference xlink:href="cid:payload1" link:role="XLinkRole"
                xlink:type="simple">
                <eb:Description xml:lang="en-US">Clip List</eb:Description>
            </eb:Reference>
        </eb:Manifest>
    </SOAP:Body>
</SOAP:Envelope>

--boundaryValue
Content-ID: <payload1>
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<purchase_order>
<po_number>1</po_number>
<part_number>123</part_number>
<price currency="USD">500.00</price>
</purchase_order>
--boundaryValue--
```

# Section 8. Summary and Resources

## Wrap up

ebXML is a group of related specifications that cover:

- Analysis of Business Processes and Business Documents
- Documentation of a company's capabilities in such a way that automated agreement is possible
- Document transfer between Trading Partners to achieve specific business goals.

Businesses document their Business Processes and Business Document structures using the BPSS. They document their capabilities using a CPP, and their agreements with Trading Partners using a CPA. Any or all of this information may be stored in an ebXML Registry, where it is searchable.

Messages are sent from one application, known as the Business Service Interface, to another using ebMS. Specifically, the MSH sends messages using the SOAP with Attachments specification so that information about the message is contained in a SOAP message, and the documents themselves are contained in various payload attachments.

---

## Resources

ebXML covers a great variety of interrelated technologies. Now that you've got an idea of how they all fit together, here are some places you can go for more information.

**developerWorks Tutorials**

*developerWorks* provides a number of tutorials that cover the background necessary for building ebXML applications, including:

- Learn the basics of what XML is and how to use it with *Introduction to XML* (http://www6.software.ibm.com/reg/devworks/dw-xmlintro-i).
- Get the background on XML messaging with *Introduction to XML messaging* (http://www6.software.ibm.com/reg/devworks/dw-coxmsg-i), then learn where SOAP fits in with *XML messaging with SOAP* (http://www6.software.ibm.com/reg/devworks/dw-cosoap-i). Java technology developers should read *Introducing the Java Message Service* (http://www6.software.ibm.com/reg/devworks/dw-jms-i ).
- Learn to add security to your applications with *Digital signatures for SOAP messages* (http://www6.software.ibm.com/reg/devworks/dw-wsdsst-i).
- Gain an understanding of Web services in general with *Creating a complete Web service* (http://www6.software.ibm.com/reg/devworks/dw-wsaggr-i) and *Implementing Web services with the WSTK 3.0.1*

(http://www6.software.ibm.com/reg/devworks/dw-wstk30-i).

### Other IBM resources

- Get a quick overview of ebXML and more resources with David Mertz's article "*Understanding ebXML: Untangling the business Web of the future*" (http://www-106.ibm.com/developerworks/library/x-ebxml/index.html) (*developerWorks*, June 2001).
- Read IBM's original proposal for *Trading Partner Agreements* (http://ebxml.org/project_teams/trade_partner/tpaml106.zip), and read about IBM's *plans* for integrating business-to-business communications (http://www.ibm.com/developer/xml/tpaml/b2b-integration-with-tpa.pdf).
- Take a look at *IBM WebSphere Studio Application Developer* (www-4.ibm.com/software/ad/studioappdev/), an easy-to-use, integrated development environment for building, testing, and deploying J2EE applications, including generating XML documents from DTDs and schemas.
- Find out how you can become an *IBM Certified Developer in XML and related technologies* (http://www-1.ibm.com/certify/certs/adcdxmlrt.shtml).

### ebXML and related resources

- Read the ebXML specifications and technical reports at *http://www.ebxml.org/specs/index.htm*. These extensive documents provide details into the BPSS, Core Components, the Registry Information Model, the ebMS, and every other specification that OASIS and UN/CEFACT produce. Specifications provide normative details, and technical reports provide detailed discussions on how to use them.
- Learn about the UMM at *http://www.gefeg.com/tmwg/n090r10.htm*.
- Learn about the UML at *http://www.omg.org/uml/*.
- Read the W3C's recommendation for Resource Description Framework at *http://www.w3.org/RDF/*.
- Read about the W3C's work on standardizing Web services at *http://www.w3.org/2002/ws/*.
- Read about the W3C XML Signature Working Group's work on security at *http://www.w3.org/Signature/*.

---

# Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you!

---

## Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT

extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.