cXML User's Guide

VERSION 1.1

JUNE, 2000

Table of Contents

Preface

This document describes how to use cXML (commerce eXtensible Markup Language) for communication of data related to electronic commerce.

Audience and Prerequisites

This document is intended for programmers designing cXML-enabled applications. It is oriented toward suppliers that are modifying their e-commerce Websites for punchout.

cXML is an open, versatile language for the transaction requirements of:

- Electronic product catalogs
- cXML punchout catalogs
- Procurement applications
- Buying communities

Readers should have a working knowledge of e-commerce concepts and the HTTP Web communication standard.

This document does not describe how to use specific procurement applications or network e-commerce hubs.

Which Chapters to Read

- **E-commerce Business Managers**—For an overview of cXML capabilities, read Chapter 1, “Introduction to cXML.”
- **Web Programmers—**Web programmers who are implementing e-commerce sites should read all chapters.

- **Punchout Site Administrators**—Web engineers experienced with punchout Websites should read Appendix B, “New Features in cXML 1.1.”

Chapter 1
Introduction to cXML

This chapter introduces cXML (commerce eXtensible Markup Language) for electronic-commerce transactions.

This chapter provides an overview of cXML. It discusses the following topics:

- cXML Capabilities
- Types of Applications that Use cXML
- Content Delivery Strategy
- Validation Against DTDs
- Profile Transaction
- XML Utilities

cXML Capabilities

cXML allows buyers, suppliers, aggregators, and intermediaries to communicate using a single, standard, open language.

Successful business-to-business electronic commerce (B2B e-commerce) portals depend upon a flexible, widely-adopted protocol. cXML is the key to providing the widest access to your products and services, because it is a well-defined, robust language designed specifically for B2B e-commerce, and it is the choice of high volume buyers and suppliers.

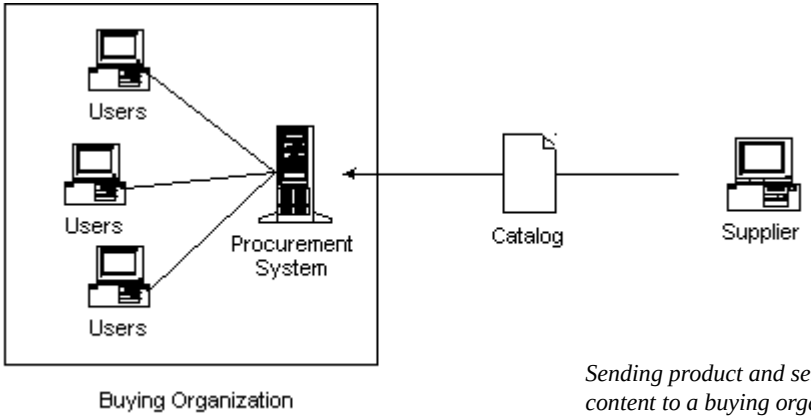
cXML transactions consist of *documents*, which are simple text files with well defined format and contents. Most types of cXML documents are analogous to hardcopy documents traditionally used in business.

The following subsections describe the main types of cXML documents.

Catalogs

Catalogs are files that convey product and service content to buying organizations. They describe the products and services you offer and the prices you charge, and they are the main communication channel from you to your customers.

You create catalogs so that organizations that use procurement applications can see your product and service offerings and buy from you. Procurement applications read your catalogs and store them internally in their databases. After a buying organization approves your catalogs, your content is visible to users, who can choose items and add them to purchase requisitions.



Sending product and service content to a buying organization

You can create catalogs for any product or service, regardless of how it is measured, priced, or delivered.

For each item in a catalog, there is basic information that is required, and optional information that enables advanced catalog features, such as multi-language descriptions.

Punchout

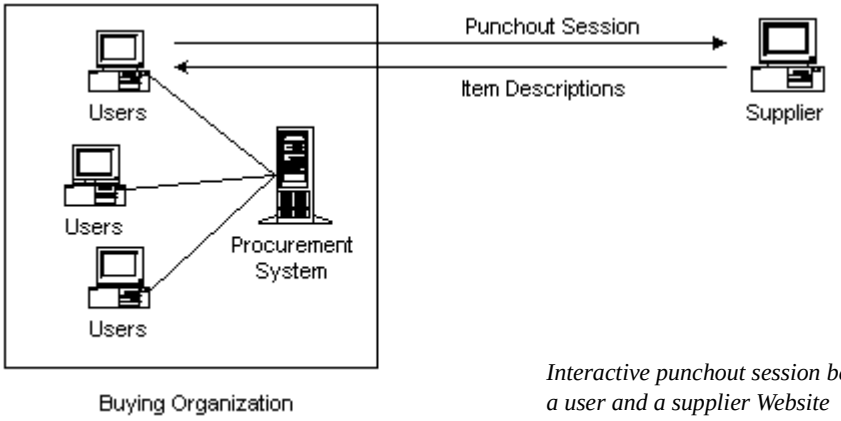
Punchout gives you an alternative to static catalog files. Punchout sites are live, interactive catalogs running on your Website.

If you have an e-commerce Website, you can modify it to support punchout. Punchout sites communicate with procurement systems over the Internet by using cXML.

For more information:

Chapter 2, "Implementing a Punchout Site."

For punchout sites, procurement applications display a button instead of product or pricing details. When users click this button, their Web browsers display pages from your local Website. Depending on how you implement your pages, users can browse product options, specify configurations, and select delivery methods. When users are done selecting items, they click a button that returns the order information to the procurement application. The fully configured products and their prices appear within users' purchase requisitions.



Interactive punchout session between a user and a supplier Website

Your Website can offer previously agreed-upon contract products and prices.

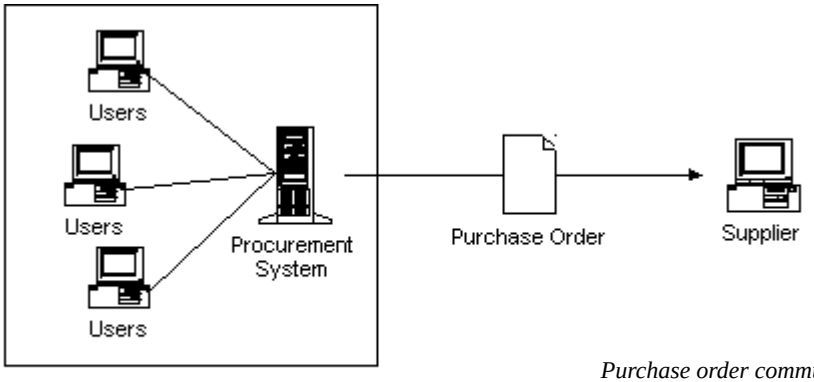
Purchase Orders

Buying organizations send purchase orders to suppliers to request fulfillment of a contract.

For more information:

Chapter 3, “Receiving cXML Purchase Orders.”

They can be routed to suppliers through an e-commerce network platform, such as Ariba Network.



Purchase order communicated to a supplier

cXML is just one of several possible formats for purchase orders. Other formats include e-mail, fax, and EDI (X.12 Electronic Data Interchange). However, cXML is the ideal format for purchase orders because it is flexible, inexpensive to implement, supports the widest array of data and attachments, and can be used by automated systems.

Types of Applications that Use cXML

cXML can be used by any e-commerce application. It is currently in use by buying organizations, vertical and horizontal buying communities, suppliers, and application vendors.

The following subsections describe the main types of applications that currently use cXML.

Procurement Applications

Procurement applications, such as Ariba ORMS (Operating Resource Management System) and Ariba IBX (Internet Business eXchange), use cXML for external transactions.

Ariba ORMS is an enterprise application that is hosted by large organizations for use by their employees over an intranet.

Ariba IBX is an Internet-based service that allows the creation of buying communities that are composed of many small- to medium-sized businesses.

These applications allow communities of users to buy contract products and services from vendors approved by their purchasing managers. Requested purchases are first approved by managers in the communities, and approved purchase orders are transmitted to suppliers through several possible channels, including cXML over the Internet.

Commerce Network Platforms

Commerce network platforms, such as Ariba Network, are Web-based services for connecting buyers and suppliers.

These Web services provide features such as catalog validation and file management, catalog publishing and subscription, automated purchase order routing, and purchase order history.

Communication between these Web services, buyer applications, and supplier applications can occur entirely through cXML over the Internet.

Punchout Catalogs

As described above, punchout catalogs are interactive catalogs, available at supplier Websites. Punchout catalogs are Web server applications, written in a programming language such as ASP (Active Server Pages), JavaScript, or CGI, that manage buyers' punchout sessions.

For more information:

Chapter 2, "Implementing a Punchout Site."

Punchout catalogs accept punchout requests from procurement applications, identify the buying organization, and display the appropriate products and prices in HTML format. Users then select items, configure them, and select options if appropriate.

At the end of the punchout session, the punchout site sends descriptions of the users' selections, in cXML format, to the procurement applications.

Order-Receiving Systems

Order-receiving systems are applications at supplier sites that accept and process purchase orders sent by buying organizations. Order-receiving systems can be any automated system, such as inventory management systems, order-fulfillment systems, or order-processing systems.

For more information:

Chapter 3, "Receiving cXML Purchase Orders."

Because it is simple to extract information from cXML purchase orders, it is relatively easy to create the adapters that enable your existing order-receiving systems to accept them.

Content Delivery Strategy

Procurement applications present product and service content to users. Suppliers want to control the way their customers view their products or services, because presentation is critical to their sales process. Buying organizations want to make content easily accessible and searchable to ensure high contract compliance.

Buying organizations and suppliers can choose from multiple methods for delivering product and service content. The particular method to use is determined by agreement between a buying organization and a supplier, and the nature of the products or services traded.

The following table lists example categories of commonly procured products and services, and their preferred content delivery methods.

Commodity	Properties	Content Delivery Method
Office Supplies, Internal Supplies	Static content, stable pricing	Static catalogs
Lab Supplies, MRO (Maintenance, Repair, and Operations), Electronic Parts	Requires normalization to be useful	Punchout to a vertical commodity portal
Books, Chemicals	Large number of line Items	Punchout to a supplier hosted site
Computers, Network Equipment, Peripherals	Many possible configurations	Punchout to a supplier configuration tool
Services, Printed Materials	Content has highly variable attributes	Punchout to an electronic form at a supplier site

Buying organizations can either store content locally within the organization, or they can access it remotely on the Internet, through punchout. cXML catalogs support both storage strategies.

As the table above indicates, punchout offers a flexible framework upon which suppliers, depending on their commodity or customer, can provide customized content. The objective of this content strategy is to allow buyers and suppliers to exchange catalog data in the method that make the most sense.

Validation Against DTDs

Because cXML is an XML language, a set of Document Type Definitions (DTDs) thoroughly define it. These DTDs are text files that describe the precise syntax and order of cXML elements. DTDs enable applications to validate the cXML they read or write.

cXML applications are not required to validate cXML documents, although it is recommended.

Getting cXML DTDs

DTDs for all versions of cXML are available at consistent locations on cxml.org:

http://xml.cXML.org/schemas/cXML/<*version*>/cXML.dtd

where <*version*> is the full cXML version number, such as 1.1.007.

Performing Validation

Your applications can use these DTDs to validate all incoming and outgoing cXML documents. XML validation applications are available on the Web. Microsoft Internet Explorer 5 has built-in XML validation capability.

For the most robust transaction handling, validate all cXML documents received. If you detect errors, issue the appropriate error code so the sender can retransmit.

For best performance, cXML clients should not fetch DTDs each time they parse cXML documents. Instead, they should look at the cXML version in the document headers and retrieve DTDs that have not already been stored locally.

Profile Transaction

The Profile transaction communicates basic information about cXML servers. It is the only transaction that all cXML servers must support.

This transaction consists of two documents, ProfileRequest and ProfileResponse. Together, they retrieve server capabilities, including supported cXML version, supported transactions, and options to those transactions.

Note: All cXML 1.1 servers **must** support the Profile transaction.

Clients can use the Profile transaction to “ping” servers to verify that they are available.

ProfileRequest

The ProfileRequest document has no content. It simply routes to the specified cXML server.

ProfileResponse

The server responds with a ProfileResponse document, which lists transactions supported by the cXML server, their locations, and any named options with a string value.

XML Utilities

Utilities for editing and validating XML files are available for free and for purchase on the Web. The following listing describes a few of these utilities:

- **Internet Explorer 5** from Microsoft. An XML-aware Web browser that can validate XML files against DTDs.

  www.microsoft.com/windows/ie/default.htm

- **XML Notepad** from Microsoft. A simple XML editor.

  msdn.microsoft.com/xml/notepad/intro.asp

- **XML Authority** from Extensibility. A Java-based XML DTD editor, with hierarchical and graphical views.

  www.extensibility.com

- **XML Spy** from Icon Information Systems. A tool for maintaining DTDs and XML files, with a grid, source and browser view.

  www.icon-is.com

- **XMetaL** from Softquad Software. A customizable XML authoring tool.

  www.softquad.com

- **CLIP** from Techno2000 USA. An easy-to-use XML authoring tool, with guided editing.

  www.t2000-usa.com

- **XMLwriter** from Wattle Software. A graphical XML authoring tool designed to manage XML projects.

  www.xmlwriter.net

In addition, the following Websites list more XML tools:

www.xmlsoftware.com
www.xml.com/pub/pt/Editors

Chapter 2
Implementing a Punchout Site

Punchout enables users of procurement applications to access supplier contracts for products or services that reside at your Website. It eliminates the need for you to send whole catalogs to buying organizations. Instead, you send them just short files that contain a description of your storefront, product categories, or products.

This chapter shows how to modify a Website to support punchout. It discusses the following topics:

- Punchout Requirements
- Punchout Event Sequence
- Punchout Documents
- Modifications to Your Web Pages
- Punchout Website Suggestions

Punchout Requirements

Before buying organizations configure their procurement applications for punchout, or suppliers implement punchout Websites, both parties must evaluate the benefits and requirements of punchout.

Buying Organizations

Setup and testing of cXML-compatible procurement applications with a punchout-enabled supplier can be completed in less than one day.

Because barriers to technical integration are low for buying organizations, the decision to use punchout should be based on the business practices and types of commodities purchased. (See “Content Delivery Strategy” on page 6 for a list of commodities that are well suited for punchout.)

Business Issues

Buying organizations should consider the following questions:

- Do requisitioners and approvers have Internet access? If not, would controlled access to the Internet be allowed?
- Does the buying organization want their suppliers to create and maintain catalog content (including pricing)?
- Do requisitioners currently procure goods on the Internet? If so, do these goods require a supplier-side configuration tool or contain unique attributes that cannot conform to a static content model?
- Does the buying organization use content aggregators for catalogs (for example, Aspect, TPN Register, or Harbinger)?
- Does the buying organization currently procure services (for example, consultants, temp services, or maintenance) through the Internet?
- Does the buying organization currently conduct online sourcing?

If the answer to any of the above questions is yes, then punchout might be appropriate for the buying organization.

Technical Issues

Buying organizations must meet the following technical requirements:

- Direct Internet Access—Users within buying organizations must have direct Internet access. Punchout relies on regular Web browser sessions where the user interacts with live supplier Websites. This communication occurs through regular internet/Internet infrastructure, not through the procurement application.
- Reliable Internet Connection—Internet access must be constantly operational and reliable. If users cannot procure products because of Internet outages, they are likely to make rogue purchases.
- Contracts with Punchout Suppliers—Purchasing agents must have established contracts with punchout-enabled suppliers. Punchout Websites allow access only to known, authenticated buying organizations.

Suppliers

The term *supplier* in the context of punchout encompasses more than the traditional definition of the term. The punchout protocol was designed as a flexible framework capable of transmitting data about virtually any kind of product or service from any kind of supplier, distributor, aggregator, or manufacturer.

Example products and services include:

- Computers direct from a manufacturer or reseller
- Chemicals and reagents from an aggregator
- Office supplies from a distributor
- Contract services from a temp agency

You might already have a transactive Website capable of hosting content and receiving purchase orders. Given this capability, you need to consider both your business practices and technical resources in deciding whether to implement punchout.

Business Issues

Suppliers should consider the following questions:

- Do you currently sell your products or services through the Internet? If so, do you offer customer-specific content (contract pricing) through your Website?
- Do your products and services fall into one of the punchout categories as described in the chart in “Content Delivery Strategy” on page 6? To review, these categories include:

  Highly configurable products (such as computers)
  Large number of line items (such as books)
  Unique product attributes (such as chemicals)
  Normalized data (such as MRO Supplies)

- Do you prefer to receive purchase orders and/or payment through your Website?

If the answer to any of the above questions is yes, then punchout might be appropriate for your organization.

Technical Issues

Suppliers must meet the following technical requirements:

- Reliable Internet Connection—The Web server infrastructure and Internet connection must be extremely reliable. If users cannot access remote content, they are likely to go to another supplier.
- Competent Website Administrators—The punchout Website and supporting applications will require periodic maintenance and modification. Users' needs and your product offerings will change, so you need personnel to modify your punchout infrastructure.
- Support for Basic Transactions—Punchout Websites do not need to support all cXML functionality, but they must support the following required transactions:

  Profile Transaction
  PunchOutSetupRequest
  PunchOutSetupResponse
  PunchOutOrderMessage

Work Estimate

The following table lists estimates of work required for cXML punchout integration based on estimates from suppliers:

Level of Pre-existing Infrastructure	Estimated Time for Completion
Transactive site with XML infrastructure	3 weeks with in-house IT staff 3-4 weeks with contractors
Transactive site without XML infrastructure	4 weeks with in-house IT staff 4-5 weeks with contractors

Understanding XML

The first step to becoming punchout enabled is to understand XML. XML is a language for describing languages. cXML documents are constructed based on XML Document Type Definitions (DTDs). Acting as templates, DTDs can be used to define content models within a cXML document (for example, the valid order and nesting of elements) and the datatypes of attributes.

About XML

XML (eXtensible Markup Language) is a standard for passing data between Internet applications. XML documents contain data in the form of tag/value pairs. XML has a structure similar to HTML (hypertext markup language), but Internet applications can extract and use data from XML documents more easily than in plain HTML ones.

As Internet applications become more widespread, XML will become more prevalent.

To implement a punchout Website, you must have a fundamental understanding of how to create, parse, query, receive, and transmit XML data to and from a remote source.

The basic tools to process XML documents are XML parsers. These parsers are freely available from Microsoft and other companies (for example, an XML parser is standard in Microsoft Internet Explorer 5). For a list of XML tools, see "XML Utilities" on page 9.

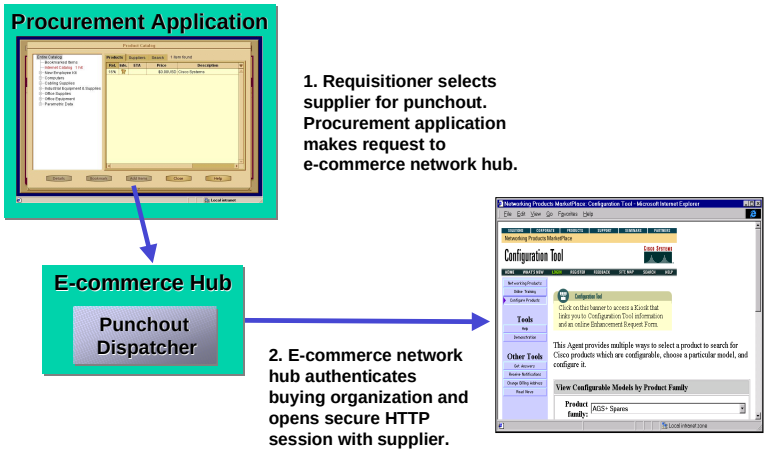
Punchout Event Sequence

A punchout session is comprised of several distinct steps.

Steps 1 & 2: Punchout Request

Users log in to a procurement application and open new purchase requisitions. They find desired items by searching their local catalogs by commodity, supplier, or product description. When they select a punchout item, the procurement application opens a new browser window and logs them into their accounts at your Website.

The following figure illustrates the punchout request steps:



How does it work? When a user clicks a punchout item, the procurement application sends a cXML PunchOutSetupRequest document to a network e-commerce hub. Acting as the trusted third party, the hub accepts the request, verifies the buying organization, and passes the request to your punchout Website.

> **Note:** All cXML documents sent through the Internet can travel through secure, SSL (Secure Socket Layer) 3.0-encrypted HTTPS connections.

The purpose of this request is to notify your Website of the buyer's identity, and to communicate the operation to be performed. Supported operations include the following:

- **create** – Initiates a new punchout session

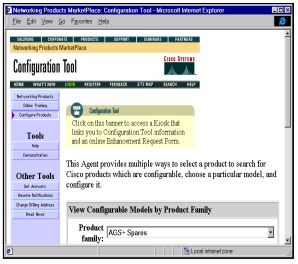
- **edit** – Re-opens a punchout session for editing
- **inspect** – Re-opens a punchout session for inspection (no changes can be made to the data)

After your Website receives a request, it sends back a PunchOutSetupResponse containing a URL that tells the procurement application where to go to initiate a browsing session on your Website.

The procurement application opens a new browser window, which displays a session logged into an account on your Website. This account can be specific to a region, a company, a department, or a user.

Step 3: Product Selection

Users select items from your inventory using all the features and services provided by your Website:



3. Requisitioner uses supplier site to find and configure products.

Depending on the product or customer, these features might include the following:

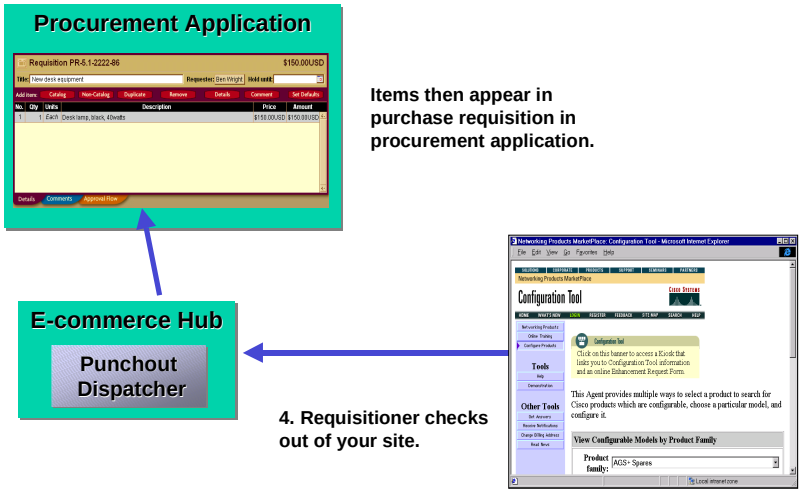
- Configurator tools for building customized products (for example, computers, organic compounds, or personalized products)
- Search engines for finding desired products from large catalogs.
- Views of normalized data for comparing products based on price, features, or availability (for example, MRO products)
- Views of attributes unique to a particular commodity (for example, printed materials, chemical and reagents, or services)
- Real-time pricing, inventory, and availability checking
- Automatic tax and freight calculations based on ship-to destination, size, or quantity of items (not necessary to calculate during the punchout session)

How does it work? After the procurement application directs users to your Website, the shopping experience is the same as if they had logged on to your Website directly. Thus, none of the previously listed features and services require modification.

Step 4: Check Out

Your Website calculates the total cost of the user's selections, including tax, freight, and customer-specific discounts. Users then click your Website's "Check Out" button to send the contents of the shopping cart to the their purchase requisitions within the procurement application.

The following figure illustrates the check-out steps:



How does it work? When users click your "Check Out" button, your Website sends a cXML PunchOutOrderMessage containing product details and prices to the procurement application. You can also send hidden supplier cookies, which can later associate items with a specific shopping session.

Effectively, you have provided a quote for the requested items—you have not yet received a purchase order, so you cannot yet book the order.

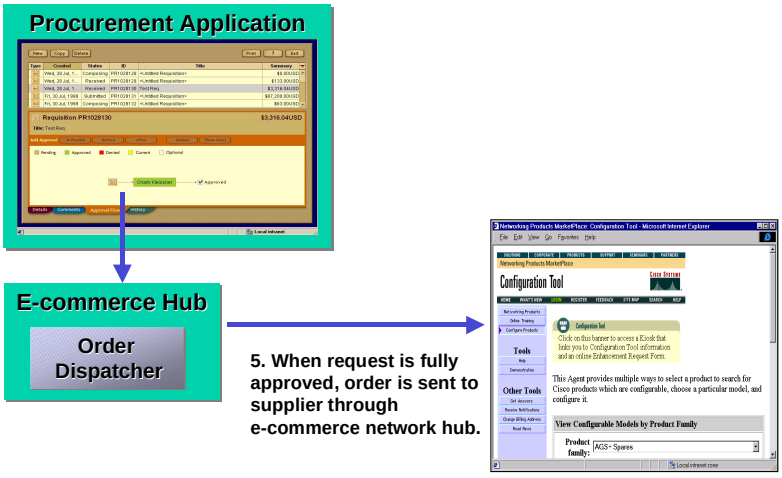
If users later need to edit any of the items in a purchase requisition, you can allow them to "re-punchout" to your Website. The procurement application sends back the contents of the original shopping cart to your Website and users make any changes there. Upon check out, your Website returns the items to the purchase requisition.

Your Website is the information source for all punchout items. Changes to the quantity or the addition of new items to the requisition might alter tax or shipping charges, which would require recalculation at your Website. Thus, any changes to the original items need to be made at your Website, not in the procurement application, hence the need to re-punchout. A re-punchout is simply a PunchOutSetupRequest with "edit" as its operation.

Step 5: Transmittal of Purchase Order

After the contents of the shopping cart have been passed from your Website to the user's purchase requisition, the procurement application approval processes take over. When the purchase requisition is approved, the procurement application converts it into a purchase order and sends it back to your Website for fulfillment. Purchasing card data can be transmitted along with the order, or you can invoice the order upon shipment.

The following figure illustrates purchase order transmittal:



How does it work? The procurement application sends all purchase orders to the e-commerce hub in cXML format. The hub then routes them to you, using your preferred order-routing method. When you acknowledge the receipt of a purchase order, you have effectively booked the order.

For punchout-enabled suppliers, the order routing method should be cXML, because of the following reasons:

- cXML purchase orders allow embedded supplier cookie information to be transmitted back to you. Because the supplier cookie is of data type “any”, it does not easily map to other order routing methods such as fax, e-mail, or EDI.
- Punchout-enabled suppliers are cXML-aware, so accepting cXML purchase orders is a small incremental effort.

Purchase orders are discussed in detail in Chapter 3, “Receiving cXML Purchase Orders.”

Punchout Documents

There are four types of cXML documents:

- Punchout Index Catalog
- PunchOutSetupRequest
- PunchOutSetupResponse
- PunchOutOrderMessage

Punchout Index Catalog

Punchout index catalogs are files that list punchout items and point to your punchout Website.

The following example lists a punchout index catalog:

Type of cXML document and URL of DTD

Your identifier for the punchout item

URL of your punchout Website (launch page)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Index SYSTEM "http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd">
<Index>
    <SupplierID domain="DUNS">83528721</SupplierID>
    <IndexItem>
        <IndexItemPunchout>
            <ItemID>
                <SupplierPartID>5555</SupplierPartID>
            </ItemID>
             <PunchoutDetail>
                <Description xml:lang="en-US">Desk Chairs</Description>
                <Description xml:lang="fr-FR">Chaises de Bureau</Description>
                <URL>http://www.workchairs.com/punchout.asp</URL>
                <Classification domain="UNSPSC">5136030000</Classification>
            </PunchoutDetail>
        </IndexItemPunchout>
```

```
        </IndexItem>
    </Index>
```

SupplierID identifies the supplier organization. You can use any identification domain, but the recommended ones are DUNS (Dun & Bradstreet Universal Naming System) and NetworkID. For more information about DUNS numbers, see www.dnb.com.

Description specifies the text that the procurement application displays in product catalogs. You can provide the description in multiple languages, and the procurement application displays the appropriate one for the user's locale.

Classification specifies the commodity grouping of the line item to the buyer. All your products and services must be mapped and standardized to the UNSPSC schema. For punchout index catalogs, the Classification determines the location of the punchout item within catalogs displayed to users. For a list of UNSPSC codes, see www.unspsc.com.

Creating and Publishing Index Catalogs

Create these catalogs and publish them on an e-commerce hub to your customers. The catalog manager within buying organizations downloads them and stores them for use with procurement applications.

Users see the contents of your punchout index catalogs alongside regular, static catalog items.

Punchout Item Granularity

You can create store-level, aisle-level, or product-level catalogs.

- Store-level catalogs list all of your products and services. Users must search to find the desired item.
- Aisle-level catalogs list related products and services.
- Product-level catalogs list only one product or service. Users need to perform no searching.

To determine how broad to make punchout items, consider your business model, the makeup of your product and service offerings, and the structure of your punchout Website.

The more search and configuration tools you have on your Website, the more broad you can make the punchout items in your index catalogs.

PunchOutSetupRequest

To initiate a punchout session, the user selects your punchout item. The procurement application generates a PunchOutSetupRequest document and sends it to an e-commerce hub, which forwards it to your punchout Website.

The following is a sample PunchOutSetupRequest document:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd">
<cXML version="1.1.007" xml:lang="en-US"
payloadID="933694607118.1869318421@jlee" timestamp="2000-08-15T08:36:47-
07:00">
    <Header>
        <From>                                          Originator (buying organization)
            <Credential domain="DUNS">
                <Identity>65652314</Identity>
            </Credential>
        </From>
        <To>                                            Destination (supplier)
            <Credential domain="DUNS">
                <Identity>83528721</Identity>
            </Credential>
        </To>
        <Sender>                                        Previous relaying entity (Ariba Network in this case)
            <Credential domain="AribaNetworkUserId">
                <Identity>sysadmin@ariba.com</Identity>
                <SharedSecret>abracadabra</SharedSecret>
            </Credential>
            <UserAgent>Ariba ORMS 6.1</UserAgent>
        </Sender>
    </Header>
    <Request>
        <PunchOutSetupRequest operation="create">   Type of request
            <BuyerCookie>1CX3L4843PPZO</BuyerCookie>
            <Extrinsic name="CostCenter">610</Extrinsic>
            <Extrinsic name="User">john_smith</Extrinsic>
            <BrowserFormPost>                       Destination for final PunchOutOrderMessage
                <URL>https://aribaorms:26000/punchout.asp</URL>
            </BrowserFormPost>
            <SupplierSetup>
                <URL>http://www.workchairs.com/punchout.asp</URL>
            </SupplierSetup>
            <SelectedItem>                          Item selected by user
                <ItemOut quantity="1">
                    <ItemID>
                        <SupplierPartID>5555</SupplierPartID>
                    </ItemID>
                </ItemOut>
```

```
            </SelectedItem>
        </PunchOutSetupRequest>
    </Request>
</cXML>
```

The payloadID and timestamp attributes near the beginning are used by cXML clients to track documents and to detect duplicate documents.

The From, To, and Sender elements allow receiving systems to identify and authorize parties. The From and To elements in a document do not change. However, as the document travels to its destination, intermediate nodes (such as Ariba Network) change the Sender element.

Create, Edit, and Inspect Operations

The operation attribute specifies the type of session the buyer initiates. It can create, edit, or inspect.

- create sessions generate new shopping carts, which correspond to new purchase requisitions.
- edit sessions reopen previously created shopping carts for modification. The procurement application sends line-item data as part of the PunchOutSetupRequest. The punchout Website can use this data to re-instantiate the shopping cart created during the original session.
- inspect sessions reopen previously created shopping carts for viewing only. As with the edit operation, the procurement application sends line-item data as part of the PunchOutSetupRequest. However, after re-instantiating the shopping cart, the punchout Website does not allow modification of its contents.

The following example lists an edit request:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd">
<cXML version="1.1.007" xml:lang="en-US"
payloadID="933695135608.677295401@jlee" timestamp="2000-08-15T08:45:35-07:00">
    <Header>
        <From>
            <Credential domain="DUNS">
                <Identity>65652314</Identity>
            </Credential>
        </From>
        <To>
            <Credential domain="DUNS">
                <Identity>83528721</Identity>
            </Credential>
        </To>
```

```
            <Sender>
                <Credential domain="AribaNetworkUserId">
                    <Identity>sysadmin@ariba.com</Identity>
                    <SharedSecret>abracadabra</SharedSecret>
                </Credential>
                <UserAgent>Ariba ORMS 6.1</UserAgent>
            </Sender>
        </Header>
        <Request>
            <PunchOutSetupRequest operation="edit">
                <BuyerCookie>1CX3L4843PPZO</BuyerCookie>
                <Extrinsic name="CostCenter">610</Extrinsic>
                <Extrinsic name="User">john_smith</Extrinsic>
                <BrowserFormPost>
                    <URL>https://aribaorms:26000/punchout.asp</URL>
                </BrowserFormPost>
                <SupplierSetup>
                    <URL>http://www.workchairs.com/punchout.asp</URL>
                </SupplierSetup>
                <ItemOut quantity="2">
                    <ItemID>
                        <SupplierPartID>220-6338</SupplierPartID>
                        <SupplierPartAuxiliaryID>E000028901
                        </SupplierPartAuxiliaryID>
                    </ItemID>
                </ItemOut>
            </PunchOutSetupRequest>
        </Request>
</cXML>
```

If the user initiated the edit session by selecting a catalog item, the PunchOutSetupRequest would contain a SelectedItem element, like a create session.

Authentication by an E-commerce Hub

All PunchOutSetupRequest documents route through an e-commerce hub for authentication and to look up the URL of your punchout Website. The steps are:

1. The hub receives the PunchOutSetupRequest document from the user.
2. The hub verifies the buyer's ID (From and Shared Secret) with that buyer's e-commerce account. It also identifies the requested supplier (To).
3. The hub looks up your shared secret from your account and inserts it (Shared Secret) into the Sender element.
4. The hub finds the URL of your punchout Website in your account and sends the PunchOutSetupRequest document to it.

5. Your Website receives the cXML document and knows that it is authenticated because it contains your own shared secret.

6. Your Website uses information in the From element to identify the requester at the company level (for example, acme.com).

7. You can use the Contact and extrinsic data in the body of the request to uniquely identify the user (for example, John Smith in Finance at acme.com).

The PunchOutSetupRequest and PunchOutSetupResponse documents pass through the e-commerce hub for authentication. The PunchOutOrderMessage document (returning the contents of the shopping basket to the procurement application) travels directly between your Website and the user through standard HTTP or HTTPS.

Supplier Setup URL and SelectedItem

In previous cXML releases, the SupplierSetup element provided the only way to specify the URL of your punchout Website. Beginning with cXML 1.1, the e-commerce hub already knows the URL of your punchout Website.

Also, starting with cXML 1.1, procurement applications can use the SelectedItem element to specify store-, aisle-, or product-level punchout.

The SupplierSetup element has been deprecated. However, your punchout Website must handle both methods until all punchout Websites recognize the SelectedItem element.

Contact Data for Extrinsic Data and User Identification

The PunchoutSetupRequest document can contain detailed user information in the Contact element that your Website can use to authenticate and direct users, such as:

- User name and role
- E-mail address

In addition, the PunchOutSetupRequest might also contain *extrinsic* data that you can use to further identify users, such as:

- User cost center and sub account
- Region
- Supervisor
- Default currency

Buying organizations configure their procurement applications to insert Contact and extrinsic data. Ask your customers what data you can expect to receive.

PunchOutSetupResponse

After receiving a PunchOutSetupRequest, your Website sends a PunchOutSetupResponse. The PunchOutSetupResponse document serves two functions:

- It indicates whether the PunchOutSetupRequest was successful.
- It provides the procurement application with a redirect URL to your Start Page.

It contains a <URL> element that specifies the Start Page URL to pass to the user's Web browser for the interactive browsing session. This URL must contain enough state information to bind to a session context on your Website, such as the identity of the requester and the contents of the BuyerCookie element.

The following example lists a PunchOutSetupResponse document:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd">
<cXML version="1.1.007" xml:lang="en-US" payloadID="933694607739"
timestamp="2000-08-15T08:46:00-07:00">
    <Response>
        <Status code="200" text="success"></Status>
        <PunchOutSetupResponse>
            <StartPage>
                <URL>
                http://xml.workchairs.com/retrieve?reqUrl=20626;Initial=TRUE
                </URL>
            </StartPage>
        </PunchOutSetupResponse>
    </Response>
</cXML>
```

PunchOutOrderMessage

After the user selects items on your Website, configures them, and clicks your "Check Out" button, you send a PunchOutOrderMessage document to communicate the contents of the shopping basket to the buyer's procurement application. This document can contain much more data than the other documents because it needs to be able to fully express the contents of any conceivable shopping basket. This document does not strictly follow the Request/Response paradigm; its use will be explained in detail.

The following example lists a PunchOutOrderMessage:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd">
<cXML version="1.1.007" xml:lang="en-US" payloadID="933695160894"
timestamp="2000-08-15T08:47:00-07:00">
    <Header>
        <From>
            <Credential domain="DUNS">
                <Identity>83528721</Identity>
            </Credential>
        </From>
        <To>
            <Credential domain="DUNS">
                <Identity>65652314</Identity>
            </Credential>
        </To>
        <Sender>
            <Credential domain="workchairs.com">
                <Identity> website 1</Identity>
            </Credential>
            <UserAgent>Workchairs cXML Application</UserAgent>
        </Sender>
    </Header>
    <Message>
        <PunchOutOrderMessage>
            <BuyerCookie>1CX3L4843PPZO</BuyerCookie>
            <PunchOutOrderMessageHeader operationAllowed="edit">
                <Total>
                    <Money currency="USD">763.20</Money>
                </Total>
            </PunchOutOrderMessageHeader>
            <ItemIn quantity="3">
                <ItemID>
                    <SupplierPartID>5555</SupplierPartID>
                    <SupplierPartAuxiliaryID>E000028901
                    </SupplierPartAuxiliaryID>
                </ItemID>
                <ItemDetail>
                    <UnitPrice>
                        <Money currency="USD">763.20</Money>
                    </UnitPrice>
                    <Description xml:lang="en">
                        <ShortName>Excelsior Desk Chair</ShortName>
                        Leather Reclining Desk Chair with Padded Arms
                    </Description>
                    <UnitOfMeasure>EA</UnitOfMeasure>
                    <Classification domain="UNSPSC">5136030000
                    </Classification>
                </ItemDetail>
            </ItemIn>
        </PunchOutOrderMessage>
```

```
        </Message>
</cXML>
```

BuyerCookie enables the procurement application to associate a given PunchOutOrderMessage with its originating PunchOutSetupRequest. Therefore, your Website should return this element whenever it appears. Do not use the BuyerCookie to track punchout sessions, because it changes for every session, from create, to inspect, to edit.

SupplierPartAuxiliaryID acts as a supplier cookie. This field allows you to transmit additional data, such as quote number or another cXML document. The procurement application passes it back to you in any subsequent PunchOutSetupRequest edit or inspect sessions, and in the resulting cXML purchase order. You can use the supplier cookie to associate items in a purchase requisition with the corresponding items in a shopping cart at your Website.

UnitOfMeasure describes how the product is packaged or shipped. It must conform to UN/CEFACT Unit of Measure Common Codes. For a list of UN/CEFACT codes, see www.unece.org/cefact.

Classification lists the UNSPSC (United Nations Standard Product and Service Code) commodity code for each selected item. These codes are used by back-end systems within buyer and supplier organizations for accounting and report generation. For the list of UNSPSC codes, see www.unspsc.org.

Modifications to Your Web Pages

To receive or send the three cXML punchout documents, you might need to modify or create four pages on your Website:

- Launch Page
- Start Page
- Sender Page
- Order Receiver Page

To illustrate how you might implement these pages, simple Active Server Page (ASP) code samples and the Microsoft Internet Explorer 5 XML Parser will be used. Actual implementation of these pages will vary depending on the supplier development environment (for example, CGI, JavaScript, or WebObjects).

Launch Page

The Launch Page receives all authenticated PunchOutSetupRequest documents from the e-commerce hub. It reads the HTTP stream sent from the hub and validates the cXML request imbedded within that stream against the cXML DTD (in the case of ASP, using method calls to the Internet Explorer 5 XML parser).

After validation, your Launch Page extracts elements from the document in order to:

1. Identify the user and determine where to redirect that user.
2. Compose a PunchOutSetupResponse document and return it to the sender.

Your Launch Page should store the following data for use by your Start Page:

- Identity of the requester (Sender)
- Identity of the language of the user (xml:lang) so you can provide localized content
- Type of the request (create, edit, or inspect)
- Any extrinsic data that further identifies the user and the user location

The following is a sample Launch Page. This code does not use an XML parser to dynamically generate the PunchOutSetupResponse, but instead uses a static XML template into which line item data is filled. **This code is intended for illustrative purposes only.**

```
script language=JScript RUNAT=Server>
function elementValue(xml, elem)
{
        var begidx;
        var endidx;
        var retStr;

        begidx = xml.indexOf(elem);
        if (begidx > 0) {
                endidx = xml.indexOf('</',begidx);
                if (endidx > 0)
                        retStr =  xml.slice(begidx+elem.length,
                        endidx);
                        return retStr;
                }
                return null;
}

function twoChar( str )
{
        var retStr;
```

```
        str = str.toString();
        if ( 1 == str.length ) {
                retStr = "0" + str;
        } else {
        retStr = str;
        }
        return retStr;
        }

function timestamp( dt )
        {
        var str;
        var milli;
        str = dt.getFullYear() + "-" + twoChar( 1 + dt.getMonth() ) + "-";
        str += twoChar( dt.getDate() ) + "T" + twoChar( dt.getHours() ) + ":";
        str += twoChar( dt.getMinutes() ) + ":" + twoChar( dt.getSeconds() ) + ".";
        milli = dt.getMilliseconds();
        milli = milli.toString();
        if ( 3 == milli.length ) {
                str += milli;
        } else {
                str += "0" + twoChar( milli );
        }
        str += "-08:00";
        return str;
        }

function genProlog( cXMLvers, randStr )
        {
        var dt;
        var str;
        var vers, sysID;
        var nowNum, timeStr;
        if ( 1.1 > parseFloat( cXMLvers )) {
                vers = "1.0";
                sysID = "cXML.dtd";
        } else {
                vers = "1.1.007";
        sysID = "http://xml.cXML.org/schemas/cXML/" + vers + "/cXML.dtd";
        }
        dt = new Date();
        nowNum = dt.getTime();
        timeStr = timestamp( dt );
        str = '<?xml version="1.1.007" encoding="UTF-8"?>\n';
        str += '<!DOCTYPE cXML SYSTEM "' + sysID + '">\n';
        str += '<cXML version="' + vers + '" payloadID="' + nowNum + ".";
        str += randStr + '@' + Request.ServerVariables("LOCAL_ADDR");
        str += '" timestamp="' + timeStr + '">';
        return str;
}
```

```
</script>
REM Create data needed in prolog.
%<
Randomize
randStr = Int( 100000001 * Rnd )
prologStr = genProlog( "1.0", randStr )
Response.ContentType = "text/xml"
Response.Charset = "UTF-8"
%>
<%
REM This receives the PunchOutSetup request coming from the e-commerce hub.
REM It takes the ORMSURL and buyercookie, attaches them to the Start Page URL,
REM and sends the response back to the requester.
REM punchoutredirect.asp?bc=2133hfefe&url="http://workchairs/com/..&redirect="
      Dim ret
      Dim punch
      Dim statusText
      Dim statusCode
      Dim cookie
      Dim url
      Dim xmlstr
      Dim fromUser
      Dim toUser
      cookie = ""
      url = ""
      xmlstr = ""
      dir = ""
      path = Request.ServerVariables("PATH_INFO")
      dir = Left(path, InstrRev(path, "/"))
      if IsEmpty(dir) then
            dir = "/"
      end if

REM This command reads the incoming HTTP cXML request
      xml = Request.BinaryRead(Request.TotalBytes)
      for i = 1 to Request.TotalBytes
            xmlstr = xmlstr + String(1,AscB(MidB(xml, i, 1)))
      Next
      cookie = elementValue(xmlstr, "<BuyerCookie>")
      url = elementValue(xmlstr, "<URL>")
      fromUser = elementValue(xmlstr, "<Identity>")
      newXMLStr = Right(xmlstr, Len(xmlstr) - (InStr(xmlstr,"<Identity>") +
Len("<Identity>")))
      toUser = elementValue(newXMLStr, "<Identity>")
%>
REM This formats the cXML PunchOutSetupReponse
<% if IsEmpty(cookie) then %>
<%= prologStr %>
      <Response>
```

```
            <Status code="400" Text="Bad Request">Invalid Document. Unable to extract
            BuyerCookie.</Status>
        </Response>
</cXML>
<% else %>
<%= prologStr %>
        <Response>
            <Status code="200" text="OK"/>
            <PunchOutSetupResponse>
                <StartPage>
                    <URL>http://<%=
Request.ServerVariables("LOCAL_ADDR")%>/<%= dir%>/punchoutredirect.asp?bc=<%=
cookie%>&amp;amp;url="<%= url%>"&amp;amp;from=<%= fromUser%>&amp;amp;to=<%=
toUser%>&amp;amp;redirect=<%= StartPage%></URL>
                </StartPage>
            </PunchOutSetupResponse>
        </Response>
</cXML>
<%end if%>
```

Your Launch Page should return a StartPage URL that is unique for that punchout session. In addition, this URL should be valid for only a limited amount of time. By deactivating this URL, you make it more difficult for unauthorized users to access your Start Page.

Remember to implement functionality for subsequent edit and inspect sessions. Users cannot change order details for punchout items (such as quantity) within their procurement application. They must re-punchout with an edit session. For the greatest benefit to users, inspect sessions that occur after you receive the order should display order status.

Start Page

Your Start Page logs the requester into an account on your Website. From your Start Page, users begin their shopping experience. This page might already exist at your Website, so modify it to query user name and password information from the PunchOutSetupRequest document.

Allow only authorized users into your Start Page. If you wait until the check-out step to authenticate them, you do not protect your confidential pricing or terms.

If you use HTTP browser cookies to track user preferences, destroy them after sending the PunchOutOrderMessage to buyers. Destroying these cookies prevents the possibility of offering privileged features to unauthorized users.

Sender Page

The Sender Page sends the contents of the user's shopping cart to the user. As described earlier, after users fill their shopping carts, they click your "Check Out" button.

Below is a simple ASP implementation of this feature. This code does not use an XML parser to dynamically generate the PunchOutOrderMessage, but instead uses a static XML template into which line item data is filled. **This code is intended for illustrative purposes only.**

This is a portion of a supplier's Website product page:

```
<!--#include file="punchoutitem.inc"-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- saved from
url=(0093)https://secure1.shore.net/wbird/cgi/vsc.cgi/wbird/houses/urban.htm?L+wbird+w
adt4101+928011405 -->

<TABLE border=0>
      <TBODY>
            <TR>
                  <TD><IMG src="UrbanHouses_files/uhjm.gif"> </TD>
                  <TD><STRONG>Jefferson Memorial</STRONG>- A birdfeeder with a
rotunda! This famous American monument will be a unique addition to any garden or yard.
It attracts small to medium sized birds and its dimensions are 11" x 9 1/2" x 8" H.
                  </TD>
            </TR>
      </TBODY>
</TABLE><BR>
-Jefferson Memorial<STRONG>
$139.95</STRONG><BR>
<% AddBuyButton 139.95,101,"Bird Feeder, Jefferson Memorial",5 %>
<BR>
<HR>
```

The AddBuyButton function sends the PunchOutOrderMessage back to the user.

The following listing is the include file (punchoutitem.inc) referenced above:

```
<%
REM This asp is included in items.asp, which specifies the item parameters, formats
REM a cXML document, and allows the user to proceed with a checkout of the item.
function CreateCXML(toUser, fromUser, buyerCookie, unitPrice, supPartId, desc)
%>
<?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?>
<!DOCTYPE cXML SYSTEM
&quot;http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd&quot;>
```

```
<cXML version=&quot;1.1.007&quot; payloadID=&quot;<%= Now &"@"&
Request.ServerVariables("LOCAL_ADDR")%>&quot; timestamp=&quot;<%= Now
%>&quot;>
    <Header>
        <From>
            <Credential domain=&quot;ariba.com&quot;>
                <Identity><%= toUser%></Identity>
            </Credential>
        </From>
        <To>
            <Credential domain=&quot;ariba.com&quot;>
                <Identity><%= fromUser%></Identity>
            </Credential>
        </To>
        <Sender>
            <Credential domain=&quot;ariba.com&quot;>
                <Identity><%= toUser%></Identity>
            </Credential>
            <UserAgent>PunchoutSite</UserAgent>
        </Sender>
    </Header>
    <Message>
        <PunchOutOrderMessage>
            <BuyerCookie><%= buyerCookie%></BuyerCookie>
            <PunchOutOrderMessageHeader
            operationAllowed=&quot;edit&quot;>
                <Total>
                    <Money currency=&quot;USD&quot;><%=
                    unitPrice%></Money>
                </Total>
            </PunchOutOrderMessageHeader>
            <ItemIn quantity=&quot;1&quot;>
                <ItemID>
                    <SupplierPartID><%= supPartId%></SupplierPartID>
                    <SupplierPartAuxiliaryID><%= supPartAuxId%>
                    </SupplierPartAuxiliaryID>
                </ItemID>
                <ItemDetail>
                    <UnitPrice>
                        <Money currency=&quot;USD&quot;><%= unitPrice%>
                        </Money>
                    </UnitPrice>
                    <Description xml:lang=&quot;en&quot;><%= desc%>
                    </Description>
                    <UnitOfMeasure>EA</UnitOfMeasure>
                    <Classification
                        domain=&quot;SupplierPartID&quot;><%= supPartId%>
                    </Classification>
                </ItemDetail>
            </ItemIn>
```

```
            </PunchOutOrderMessage>
        </Message>
</cXML>
<% end function

function AddBuyButton(unitPrice, supPartId, supPartAuxId, desc)

toUser = Session("toUser")
fromUser = Session("fromUser")

buyerCookie = Session("buyercookie")
url = Session("urlToPost")
if not IsEmpty(buyerCookie) then
        %>
        <FORM METHOD=POST ACTION=<%= url%>>
        <INPUT TYPE=HIDDEN NAME="cxml-urlencoded" VALUE="<% CreateCXML
toUser, fromUser, buyerCookie, unitPrice, supPartId, supPartAuxId, desc%>">
        <INPUT TYPE=SUBMIT  value=BUY>
        </FORM>
<%else%>
        </p>
        <%
end if
end function
%>
```

The AddBuyButton function contains the FORM POST that sends the URL-encoded PunchOutOrderMessage back to the user.

HTTP Form Encoding

To send a PunchOutOrderMessage, you use *HTML form encoding*, which is a different transport model than the traditional HTTP request/response model. This different transport facilitates easier integration between your Website and the procurement application. It also enables buying organizations to receive XML data without requiring them to have a Web server available through a firewall.

Instead of sending a PunchOutOrderMessage directly to the procurement application, your Website encodes it as a hidden HTML Form field and posts it to the URL specified in the BrowserFormPost element of the PunchOutSetupRequest. The hidden HTML Form field must be named either cxml-urlencoded or cxml-base64 (these names are case insensitive).

This encoding permits you to design a checkout Web page that contains the cXML document. When users click your “Check Out” button, your Website presents the data (invisible to users) to the procurement application as an HTML Form Submit.

Cancelling Punchout

You might want to add a “Cancel” button to your pages so that users can cancel their punchout session. The “Cancel” button sends an empty PunchOutOrderMessage that tells the procurement application that no items will be returned, and to delete exiting punchout items from the requisition. You can also use it to perform any housekeeping needed by your Website, such as clearing the shopping cart and closing the user session.

Order Receiver Page

The Order Receiver Page accepts cXML purchase orders sent by buying organizations. It could be similar to the Launch Page discussed above.

For information about receiving purchase orders, see Chapter 3, “Receiving cXML Purchase Orders.”

Punchout Website Suggestions

When planning the implementation of a punchout Website, consider the following suggestions.

Implementation Guidelines

Follow these guidelines when developing your punchout Website:

- Study the cXML specification.
- Use an XML parser and validate documents against the cXML DTD.
- Use the xml:lang= property to identify users’ languages so you can provide localized content.
- Use the From credential to identify buying organizations.
- Send a unique, temporary URL for the session on redirect.
- Do not persist browser cookies.
- Do not overburden your customers with extrinsic data requirements.
- For each line item, use UNUOM (United Nations Units of Measure) and UNSPSC (United Nations Standard Product and Service Codes).

- Provide real value to your customers. Display product availability, order status, and special promotions.
- Checkout should be easy and intuitive. Ideally, users should need to click only three buttons to buy.
- Code for subsequent edit and inspect sessions. Users cannot change order details for punchout items (such as quantity) within their procurement application. They must re-punchout with an edit session.
- For the greatest benefit to users, inspect sessions should display order status.
- Test your punchout Website. Allow time for testing with your customers' procurement applications.
- Punchout transactions produce quotes, not purchase orders. Implement a cXML purchase-order-receiving page to accept orders.

Buyer and Supplier Cookies

The buyer and supplier cookies enable both buyers and suppliers to re-instantiate their own line-item data for their back-end systems.

- Return the buyer cookie (BuyerCookie) you receive. Do not change it.
- Make use of the supplier cookie (SupplierPartAuxiliaryID).

The buyer cookie is analogous to a purchase requisition number; it conveys state that allows the buying organization's system to maintain the relationship between a requisition and a shopping basket.

Likewise, the supplier cookie is analogous to a quote number; it conveys state that allows your system to maintain a relationship between a shopping basket and the buyer's requisition and purchase order. Procurement applications pass the supplier cookie back to you in subsequent punchout edit or inspect sessions, and in the resulting purchase order. Your Website should take advantage of the supplier cookie to eliminate the need to pass visible, supplier-specific data back to the buyer.

Personalization

The header of the PunchOutSetupRequest always identifies the buying organization, but the request might also contain Contact and Extrinsic data (such as user's cost center, user's location, or product category) that you can use to determine the dynamic URL to serve to the user.

Although not all buying organizations send this extrinsic data, it can enable you to customize your Web store beyond the simple organization level. For example, you could provide a separate Web store for each cost center within the buying organization (or each product category or each user).

You could also store and display the user's previous quotes. You could allow users to reuse quotes, check the status of orders, and create reports on past activity. To avoid security problems, store quote history only at the per-user level.

A key consideration during planning is the amount of effort required to implement a highly dynamic and customized punchout Website. You need to balance between customization and complexity—a complex Website takes longer to implement and maintain, but it could offer more value to users. It is recommended that you start with a simple punchout Website and enhance it over time.

Chapter 3
Receiving cXML Purchase Orders

This chapter describes how to set up a Website to receive cXML-format purchase orders. It also describes how to send purchase order status messages to buying organizations or marketplaces.

Purchase Order Process

Procurement applications convert approved purchase requisitions into one or more purchase orders. A purchase order is a formal request from a buying organization to a supplier to fulfill a contract.

cXML is just one format for transmitting purchase orders. Other common formats are e-mail, fax, and EDI (X.12 Electronic Data Interchange). cXML is the best format for purchase orders because it allows you to easily automate order processing. cXML's well-defined structure allows order-processing systems to easily interpret the elements within a purchase order. With little or no human intervention, the appropriate data within purchase orders can be routed to your shipping, billing, and sales departments, as needed.

In addition, the cXML order-routing method allows the transmittal of any supplier cookies (SupplierPartAuxiliaryID) and purchase order attachments.

When you configure your account on an e-commerce network hub, you specify a URL to which all cXML purchase orders will be sent. Upon receiving a purchase order, you send it to your internal order management system and fulfill it as you normally would. Your Website must also return an Order Response acknowledgement to the e-commerce network hub, which tells the buyer that you successfully received and parsed the purchase order.

You do not need a punchout Website in order to receive cXML purchase orders; punchout and cXML-order-receiving are distinct capabilities. However, the infrastructure and applications required for supporting punchout are the same for receiving cXML purchase orders.

Receiving Purchase Orders

There are two types of cXML documents used for communicating purchase orders. The procurement application sends an OrderRequest, and you respond with an OrderResponse. These documents pass through the e-commerce network hub.

OrderRequest

The OrderRequest document is analogous to a purchase order. The following example shows an OrderRequest for an item:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd">
<cXML version="1.1.007" xml:lang="en-US" payloadID="93369535150910.10.57.136"
timestamp="2000-08-03T08:49:11+07:00">
<Header>
    <From>
        <Credential domain="AribaNetworkUserId">
            <Identity>admin@acme.com</Identity>
        </Credential>
    </From>
    <To>
        <Credential domain="DUNS">
            <Identity>114315195</Identity>
        </Credential>
    </To>
    <Sender>
        <Credential domain="AribaNetworkUserId">
            <Identity>sysadmin@ariba.com</Identity>
            <SharedSecret>abracadabra</SharedSecret>
        </Credential>
        <UserAgent>Ariba Network V1.1</UserAgent>
    </Sender>
</Header>
<Request>
    <OrderRequest>
        <OrderRequestHeader orderID="DO102880"
        orderDate="2000-08-03T08:49:09+07:00"  type="new">
            <Total>
                <Money currency="USD">4688.00</Money>
            </Total>
            <ShipTo>
                <Address isoCountryCode="US" addressID="1000467">
                    <Name xml:lang="en">Acme, Inc.</Name>
                    <PostalAddress name="default">
                        <DeliverTo>John Q. Smith</DeliverTo>
                        <DeliverTo>Buyers Headquarters</DeliverTo>
                        <Street>123 Main Street</Street>
```

```
                    <City>Mountain View</City>
                    <State>CA</State>
                    <PostalCode>94089</PostalCode>
                    <Country>United States</Country>
                </PostalAddress>
                <Email name="default">john_smith@acme.com</Email>
                <Phone name="work">
                    <TelephoneNumber>
                        <CountryCode isoCountryCode="US">1
                        </CountryCode>
                        <AreaOrCityCode>800</AreaOrCityCode>
                        <Number>5555555</Number>
                    </TelephoneNumber>
                </Phone>
            </Address>
        </ShipTo>
        <BillTo>
            <Address isoCountryCode="US" addressID="12">
                <Name xml:lang="en">Acme Accounts Payable</Name>
                <PostalAddress name="default">
                    <Street>124 Union Street</Street>
                    <City>San Francisco</City>
                    <State>CA</State>
                    <PostalCode>94128</PostalCode>
                    <Country isoCountryCode="US">US</Country>
                </PostalAddress>
                 <Phone name="work">
                    <TelephoneNumber>
                        <CountryCode isoCountryCode="US">1
                        </CountryCode>
                        <AreaOrCityCode>415</AreaOrCityCode>
                        <Number>6666666</Number>
                    </TelephoneNumber>
                </Phone>
            </Address>
        </BillTo>
        <Shipping>
            <Money currency="USD">12.34</Money>
            <Description xml:lang="en-us">FedEx 2-day</Description>
        </Shipping>
        <Tax>
            <Money currency="USD">10.74</Money>
            <Description xml:lang="en">CA State Tax</Description>
        </Tax>
        <Payment>
            <PCard number="1234567890123456" expiration="2002-03-12"/>
        </Payment>
    </OrderRequestHeader>
    <ItemOut quantity="2" >
        <ItemID>
```

```
                    <SupplierPartID>220-3165</SupplierPartID>
                    <SupplierPartAuxiliaryID>E000028901</SupplierPartAuxiliaryID>
                </ItemID>
                <ItemDetail>
                    <UnitPrice>
                        <Money currency="USD">2344.00</Money>
                    </UnitPrice>
                    <Description xml:lang="en">Laptop Computer Notebook Pentium® II
                    processor w/AGP, 300 MHz, with 12.1&amp;quot; TFT XGA Display
                    </Description>
                    <UnitOfMeasure>EA</UnitOfMeasure>
                    <Classification domain="UNSPSC">43171801</Classification>
                    <URL>http://www.supplier.com/Punchout.asp</URL>
                    <Extrinsic name="ExtDescription">Enhanced keyboard</Extrinsic>
                </ItemDetail>
                <Distribution>
                    <Accounting name="DistributionCharge">
                        <Segment type="Account" id="7720"
                            description="Office Supplies"/>
                        <Segment type="CostCenter" id="610"
                            description="Engineering Management"/>
                    </Accounting>
                    <Charge>
                        <Money currency="USD">4688.00</Money>
                    </Charge>
                </Distribution>
            </ItemOut>
        </OrderRequest>
</Request>
</cXML>
```

OrderResponse

The OrderResponse document acknowledges that you received the purchase order and that it parses correctly. It is not a commitment to execute the purchase order; it confirms that you received it and that it is a valid cXML document.

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd">
<cXML version="1.1.007" payloadID="8/3/2000 8:49:30 PM@205.180.14.45"
timestamp="2000-08-03T08:49:30+07:00">
    <Response>
        <Status code="200" text="OK"/>
    </Response>
</cXML>
```

Accepting Order Attachments

Buyers often need to clarify purchase orders with associated memos, drawings, or faxes. They can attach files of any type to cXML purchase orders by using MIME (Multipurpose Internet Mail Extensions).

cXML contains only references to external MIME parts sent within one multipart MIME envelope (with the cXML document, in an e-mail or faxed together).

The e-commerce network hub receives the attachments, and can forward them to the supplier or store them for online retrieval.

For more information about purchase order attachments, see “Attachment Transmission” on page 51.

For more information about the MIME standard, see the following Websites:

www.hunnysoft.com/mime
www.rad.com/networks/1995/mime/mime.htm

Appendix A
cXML Language Specification

This appendix describes the protocol and data formats of cXML (commerce eXtensible Markup Language). It contains all the information you need to implement any of the supported transactions from either the client or the server system perspective. Both the protocol interactions and business documents contained in the transactions are discussed in depth.

Additionally, examples of actual implementations illustrate and clarify the use of cXML.

This appendix contains the following sections:

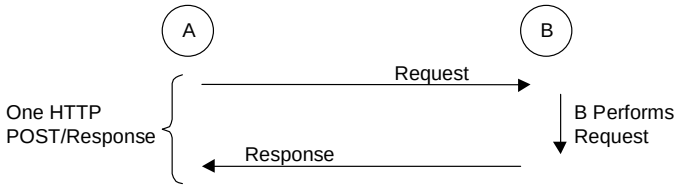
- Protocol Specification
- Basic Elements
- Profile Transaction
- Order Definitions
- Punchout Transaction
- Later Status Changes
- Catalog Definitions
- Subscription Management Definitions
- Message Retrieval Definitions

Protocol Specification

There are two communications models for cXML transactions: Request-Response and One-Way. These two models enable simple implementation because the operations required are strictly described. Both models are required because there are situations when one model would not be appropriate.

Request-Response Model

Request-Response transactions can be performed only over an HTTP connection. The following figure illustrates the steps in a Request-Response interaction between parties A and B:



A Request-Response Transaction

This transaction contains the following steps:

1. A initiates an HTTP/1.x connection with B on a predetermined URL that represents B's address.
2. A uses a POST operation to send the cXML document through the HTTP connection.
3. A waits for a response to return through the HTTP connection.
4. B has an HTTP/1.x-compliant server that dispatches the HTTP Request to the resource specified by the URL used in step 1. This resource can be any valid location known to B's HTTP server, for example, a CGI program or an ASP page.
5. B's resource identified in step 4 reads the cXML document contents and maps the Request to the appropriate handler for that request.
6. B's handler for the cXML Request performs the work that the Request specifies and formats a cXML document as a Response.
7. B sends the cXML Response to A through the HTTP connection established in step 1.

8. A reads the cXML Response and returns it to the process that initiated the Request.

9. A closes the HTTP connection established in step 1.

This process is then repeated for further Request/Response cycles.

To simplify the work in the above steps, cXML documents are divided into two distinct parts:

- Header—Contains authentication information and addressing.
- Request or Response data—Contains a specific request or response and the information to be passed.

Both of these elements are carried in a parent envelope element. The following example shows the structure of a cXML Request document:

```
<cXML>
    <Header>
        Header information here…
    </Header>
    <Request>
        Request information here…
    </Request>
</cXML>
```

The following example shows the structure of a cXML Response document:

```
<cXML>
    <Response>
        Response information here…
    </Response>
</cXML>
```

The Response structure does not use a Header element. It is not needed because the Response always travels in the same HTTP connection as the Request.

XML Conventions

cXML uses elements to describe discrete items, often properties in traditional business documents. Information with obvious subdivisions and relations between those subdivisions such as an address are also described using elements.

cXML makes extensive use of attributes.

In cXML, all elements and attribute names use whole words with capitals (not hyphens) separating the words. Element names begin with an uppercase letter; attribute names begin with a lowercase letter, for example:

Elements: Sender, Credential, Payment, ItemDetail
Attributes: version, payloadID, lineNumber, domain

cXML Envelope

The envelope element is the root of the cXML document structure and it contains all other elements. The cXML element is present in each cXML transaction.

The following example shows a fully specified cXML element:

```
<cXML version="1.1.007" xml:lang="en-US"
    payloadID=1234567.4567.5678@test.ariba.com
    timestamp="1999-03-31T18:39:09-08:00">
```

cXML has the following attributes:

version (optional)	Specifies the version of the cXML protocol. A validating XML parser could also determine the version attribute from the referenced DTD. However, all cXML documents should include the version explicitly to assist applications using non-validating parsers.
xml:lang (optional)	The locale used for all free text sent within this document. The receiver should reply or display information in the same or a similar locale. For example, a client specifying xml:lang="en-UK" in a request might receive "en" data in return.
payloadID	A unique number with respect to space and time, used for logging purposes to identify documents that might have been lost or had problems. This value should not change for retry attempts. The recommended implementation is: datetime.process id.random number@hostname
timestamp	The date and time the message was sent, in ISO 8601 format. This value should not change for retry attempts. The format is YYYY-MM-DDThh:mm:ss-hh:mm (for example, 1997-07-16T19:20:30+01:00).

Locale Specified by xml:lang

The xml:lang attribute also appears with most free text elements (such as Description and Comments). While the XML specification allows the locale for an element to default to that specified for any parent element, such defaults result in inefficient queries of the document tree. cXML attempts to keep the locale identifiers together with the affected strings.

The xml:lang attributes appearing throughout the cXML protocol have no effect upon formatted data such as numbers, dates and times. As described below for the timestamp attribute, such discrete values are formatted according to their data types. Longer strings (and referenced Web pages) not intended for machine processing might contain a locale-specific numeric or date format that matches a nearby xml:lang attribute.

Time and other Data Types

The timestamp attribute (and all other dates and times in cXML) must be formatted in the restricted subset of ISO 8601 described in the Word Wide Web Consortium (W3C) Note entitled “Date and Time Formats” available at www.w3.org/TR/NOTE-datetime-970915.html.

Timestamps require a minimum of a complete date plus hours, minutes and seconds. Fractions of a second are optional. This protocol requires times expressed in local time with a time-zone offset from UTC (Coordinated Universal Time, also known as Greenwich Mean Time). The “Z” time zone designator is not allowed.

For example, 2000-04-14T013:36:00-08:00 corresponds to April 14, 2000, 1:36 p.m., U.S. Pacific Standard Time.

Further references for the date, time, and other data type formats used by cXML are:

- Microsoft’s XML Data Types Reference site, msdn.microsoft.com/xml/reference/schema/datatypes.asp
- The original XML Data proposal to the Word Wide Web Consortium (W3C), www.w3c.org/TR/1998/NOTE-XML-data-0105

Wrapping Layers

The cXML element is the body of a normal XML document. A document might begin:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.1.007/cXML.dtd">
<cXML version="1.1.007" xml:lang="en-US"
      payloadID="0c300508b7863dcclb_13550"
      timestamp="2000-01-09T01:36:05-08:00">
      …
```

This document is normally transmitted via HTTP with a MIME media type of text/xml and a charset parameter matching the encoding in the document. Because HTTP is eight-bit clean, any encoding supported by the recipient parser can be used without a content-transfer-encoding such as base64 or quoted-printable. All XML parsers support the UTF-8 encoding, which includes all Unicode characters. Applications should therefore use it when transmitting cXML documents.

> **Note:** According to RFC 2376 “XML Media Types,” the MIME charset parameter overrides any encoding specified in the XML declaration. Further, the default encoding for the text/xml media type is us-ascii, not UTF-8 as mentioned in Section 4.3.3 of the XML Specification. For clarity, cXML documents should include an explicit encoding in the XML declaration. MIME envelopes should use a matching charset parameter for the text/xml or application/xml media type.

An HTTP transmission of a cXML document might include the following MIME and HTTP headers:

```
POST /cXML HTTP/1.0
Content-type: text/xml; charset="UTF-8"
Content-length: 1862
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
User-Agent: Java1.1
Host: localhost:8080
Connection: Keep-Alive

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

Attachment Transmission

When sending an OrderRequest that references external files, the referenced files can either reside on a server accessible by the supplier, or they can be transmitted along with the cXML document. This second option requires the use of a multipart MIME envelope. One cXML requirement for this envelope (over the basics described in RFC 2046 “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types”) is the inclusion of Content-ID headers with each attached file.

The following example shows the required skeleton of a cXML document with an attached JPEG image (without the HTTP headers shown above):

```
POST /cXML HTTP/1.0
Content-type: multipart/mixed; boundary=something unique

--something unique
Content-type: text/xml; charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
…
--something unique
Content-type: image/jpeg
Content-ID: <uniqueCID@cxml.org>
…
--something unique--
```

This skeleton is also all that a receiving MIME parser must be able to process. Applications that make use of the media type described in RFC 2387 “The MIME Multipart/Related Content-type” will get much more information if the skeleton is enhanced:

```
POST /cXML HTTP/1.0
Content-type: multipart/related; boundary=something unique;
      type="text/xml"; start=<uniqueCIDmain@cxml.org>

--something unique
Content-type: text/xml; charset="UTF-8"
Content-ID: <uniqueCIDmain@cxml.org>

<?xml version="1.0" encoding="UTF-8"?>
…
--something unique
Content-type: image/jpeg
Content-ID: <uniqueCID@cxml.org>
…
--something unique--
```

Receiving MIME parsers that do not understand the multipart/related media type must treat the two examples above identically. Each part of the MIME transmission can additionally have a Content-transfer-encoding and use that encoding. This addition is not necessary for HTTP transmission. Content-description and Content-disposition headers are optional within the cXML protocol, although they provide useful documentation.

For more information about attaching external files to purchase orders, see "Attachment" on page 69.

Header

The Header element contains addressing and authentication information. The Header element is the same regardless of which specific Request or Response is contained in the body of the cXML message. Applications need the requestor's identity, but not validation that the information provided for identity is correct.

The following example shows the Header element:

```
<Header>
    <From>
        <Credential domain="AribaNetworkUserId">
            <Identity>admin@acme.com</Identity>
        </Credential>
    </From>
    <To>
        <Credential domain="DUNS">
            <Identity>012345678</Identity>
        </Credential>
    </To>
    <Sender>
        <Credential domain="AribaNetworkUserId">
            <Identity>sysadmin@ariba.com</Identity>
            <SharedSecret>abracadabra</SharedSecret>
        </Credential>
        <UserAgent>Ariba Network 1.1</UserAgent>
    </Sender>
</Header>
```

The From and To elements are synonymous with From and To in SMTP mail messages; they are the logical source and destination of the messages. Sender is the party that opens the HTTP connection and sends the cXML document.

Sender contains the Credential element, which allows the receiving party to authenticate the sending party. This allows strong authentication without requiring a public-key end-to-end digital certificate infrastructure. Only a user name and password need to be issued by the receiving party to allow the sending party to perform Requests.

Initially, Sender and From are the same, However, if the cXML document travels through e-commerce network hubs, the Sender element changes to indicate current sending party.

From

This element identifies the originator of the cXML request. It can optionally contain more than one Credential element, allowing requestors to identify themselves using multiple identification methods. This use of multiple credentials is analogous to sending both SMTP and X.400 addresses in an e-mail message.

To

This element identifies the destination of the cXML request. Like the From element, it can contain more than one Credential to help identify the target.

Sender

This element allows the receiving party to identify and authenticate the party that opened the HTTP connection. It contains a stronger authentication Credential than the ones in the From or To elements, because the receiving party must authenticate who is asking it to perform work.

Credential

This element contains identification and authentication values used in cXML messages.

Credential has the following attributes:

domain	Specifies the type of credential. This attribute allows documents to contain multiple types of credentials for multiple authentication domains. For messages sent on Ariba Network, for instance, the domain is usually AribaNetworkUserId or DUNS.
type (optional)	Requests to or from a marketplace identify both the marketplace and the member company in From or To Credential elements. In this case, the credential for the marketplace uses the type attribute, which is set to the value “marketplace”.

Credential contains an Identity element and optionally a SharedSecret or DigitalSignature element. The Identity element states who the Credential represents, while the optional authentication elements verify the identity of the party.

The SharedSecret element is used when the Sender has a username/password combination that the requester recognizes.

The DigitalSignature element can be used if the two parties agree on a common certificate format and authority. The type attribute on a DigitalSignature element specifies the type of certificate being used.

> **Note:** Do not use authentication elements in documents sent through one-way communication. This transport routes through users' browsers, so users would be able to see the document source (including Credential elements).

Request

Clients send requests for operations. Only one Request element is allowed for each cXML envelope element, which simplifies the server implementations, because no de-multiplexing needs to occur when reading cXML documents. The Request element can contain virtually any type of XML data.

Request has the following attribute:

deploymentMode (optional)	Indicates whether the request is a test request or a production request. Allowed values are "production" (default) or "test".

Response

Servers send responses to inform clients of the results of operations. Because the result of some requests might not have any data, the Response element can optionally contain nothing but a Status element. A Response element can also contain any application-level data. In the punchout scenarios, that means a PunchOutSetupResponse element.

Status

This element conveys the success or failure of a request operation.

Status has the following attributes:

code	The status code of the request. This follows the HTTP status code model. For example, 200 represents a successful request.
text	The text of the status message. This text aids user readability in logs, and it consists of canonical strings in English.
xml:lang (optional)	The language of the data in the Status element. Optional for compatibility with cXML 1.1. Might be required in future versions of cXML.

The attributes of the Status element indicate what happened to the request.

The data within the Status element can be any data needed by the requestor. For a 200/OK status code, there might be no data. However, for a 500/Internal Server Error status code, it is strongly recommended that the actual XML parse error or application error be presented. This error allows better one-sided debugging and interoperability testing.

Servers should not include additional response elements (for example, a PunchOutSetupResponse element) unless the status code is in the 200 range (for example, 200/OK).

The HTTP 1.1 specification includes many status codes that are inappropriate for cXML. Because cXML is layered above HTTP in most cases, many errors (such as 404/Not Found) will be handled by the transport. The 200/OK and 500/Internal Server Error status codes are most likely. Validation errors in parsing a Request document would normally result in a transport error, such as an HTTP 400/Bad Request error.

The following table includes other HTTP codes that can be used.

cXML includes very few non-HTTP status codes:

- 550 – Unable to reach next cXML server to complete a transaction requiring upstream connections. An intermediate hub can return this code when a supplier site is unreachable. (If upstream connections complete, intermediate hubs should return errors directly to the client.)
- 551 – Unable to forward request due to supplier misconfiguration. For example, an intermediate hub failed to authenticate itself to a supplier. Clients cannot rectify this error, but this error might be resolved before the client retries.

- 560 – Temporary server error. For example, a server might be down for maintenance. Client should retry later.

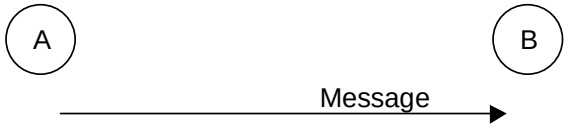
Status	Canonical text	Meaning
200	OK	The server was able to execute this Request, although the returned Response might contain application warnings or errors.
201	Accepted	Some processing might not yet have completed. As mentioned in "StatusUpdateRequest" on page 82, the client should expect later StatusUpdate transactions if this status is returned in response to an OrderRequest.
204	No Content	All Request information was valid and recognized. The server has no Response data of the type requested. In a PunchOutOrderMessage, this status indicates that the punchout session ended without change to the shopping cart (or client requisition).
400	Bad Request	Request unacceptable to the server, although it parsed correctly.
401	Unauthorized	Credentials provided in the Request (the Sender element) were not recognized by the server.
402	Payment Required	This Request must include a complete Payment element.
403	Forbidden	The user has insufficient privileges to execute this Request.
406	Not Acceptable	An alias for code 400: Request unacceptable to the server, although it parsed correctly.
409	Conflict	The current state of the server or its internal data prevented the (update) operation request. An identical Request might succeed in the future, especially after another operation has executed.
412	Precondition Failed	A precondition of the Request (for example, a punchout session appropriate for an PunchOutSetupRequest edit) was not met. This status normally implies the client ignored some portion of a previous transmission from a server (for example, the operationAllowed attribute of a PunchOutOrderMessageHeader).
417	Expectation Failed	Request implied a resource condition that was not met. One example might be a SupplierDataRequest asking for information about a supplier unknown to the server. This status might imply lost information at the client or server.
500	Internal Server Error	Server was unable to complete the Request.
501	Not Implemented	The server does not implement the particular Request. For example, PunchOutSetupRequest or the requested operation might not be supported. Status normally implies the client has ignored the server's profile.

When receiving unrecognized codes, cXML clients must handle them according to their class. Therefore, older clients should treat all new 2*xx* codes as 200, 4*xx* codes as 400 and 5*xx* codes as 500. This behavior allows for both further expansions of the cXML protocol and server-specific codes without loss of interoperability.

One-Way (Asynchronous) Model

Unlike Request-Response transactions, One-Way messages are not restricted to the HTTP transport.

One-way messages are for situations when an HTTP channel (a synchronous request-response type operation) is not appropriate. The following figure shows an example of how A and B might communicate with messages instead of the Request-Response transaction.



One-Way Message (Asynchronous)

In this case, a possible scenario would be:

1. A formats and encodes a cXML document in a transport that B understands.
2. A sends the document using the known transport. A does not (and cannot) actively wait for a response to come back from B.
3. B receives the cXML document and decodes it out of the transport stream.
4. B processes the document.

In the One-Way model, A and B do *not* have an explicit Request-Response cycle. For example, between One-Way messages, messages from other parties might arrive and other conversations could take place.

To fully specify a one-way transaction, the transport used for the message must also be documented. For the cXML transactions that use the one-way approach, the transport and encoding are specified. A common example of a transaction that uses one-way is the PunchOutOrderMessage.

One-way messages have a similar structure to the Request-Response model:

```
<cXML>
    <Header>
```

```
        Header information here…
    </Header>
    <Message>
        Message information here…
    </Message>
</cXML>
```

The Header element is treated exactly as it is in the Request-Response case. The cXML element is also identical to the one described above. The easiest way to tell the difference between a one-way message and a Request-Response message is the presence of a Message element (instead of a Request or Response element). The following section discusses the Message element in more detail.

Message

This element carries all the body level information in a cXML message. It can contain an optional Status element, identical to that found in a Response element—it would be used in messages that are logical responses to request messages.

Message has the following attributes:

deploymentMode (optional)	Indicates whether the request is a test request or a production request. Allowed values are “production” (Default) or “test”.
inReplyTo (optional)	Specifies to which Message this Message responds. The contents of the inReplyTo attribute would be the payloadID of a Message that was received earlier. This would be used to construct a two-way conversation with many messages.

The inReplyTo attribute can also reference the payloadID of an earlier Request or Response document. When a Request-Response transaction initiates a “conversation” through multiple one-way interactions, the first message can include the payloadID of the most recent relevant Request or Response that went in the other direction. For example, a Message containing a PunchOutOrderMessage might include an inReplyTo attribute containing the payloadID of the PunchOutSetupRequest that started the punchout session. (The BuyerCookie included in the punchout documents performs a similar function to such a use of the inReplyTo attribute.)

Transport Options

There are two commonly used transports for one-way messages: HTTP and URL-Form-Encoding. These are just two of the well-defined transports today; more could become supported in the future.

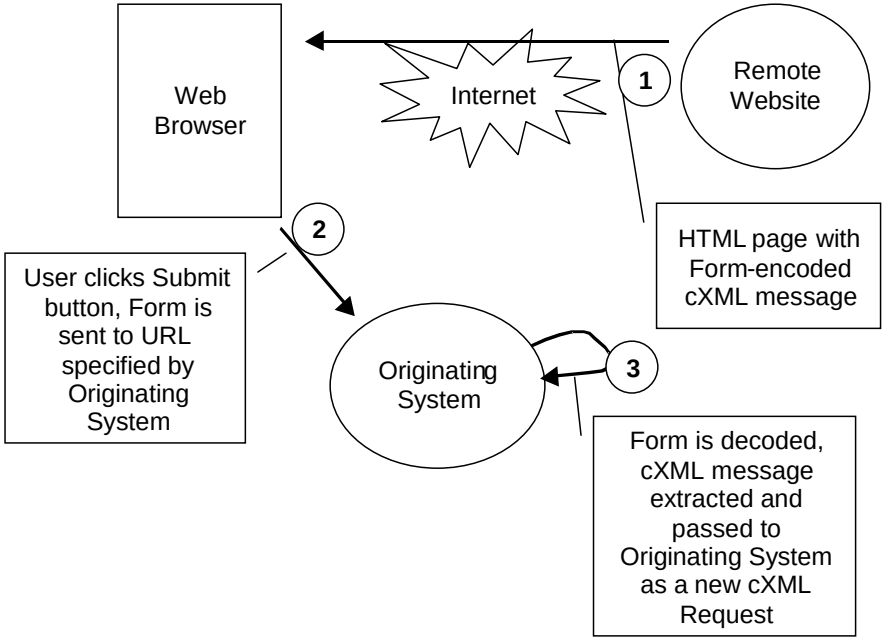
HTTP

HTTP is used for one-way communication to allow procurement applications to pull information. The one type of transaction that uses one-way HTTP communication is GetPendingRequest, which is discussed on page 96.

URL-Form-Encoding

This transport is best understood by examining how the PunchOutOrderMessage transaction is performed. URL-Form-Encoding enables integration between a remote Website and procurement applications. It also serves as a way to avoid requiring a listening server on the buyer's system that is directly accessible through the Internet.

The PunchOutOrderMessage cXML message is not directly sent to the procurement application by the remote Website, but is encoded as a hidden HTML Form field and posted to the URL specified in the BrowserFormPost element of the PunchOutSetupRequest. When the user clicks Check Out, the Website sends the data to the procurement application as an HTML Form Submit. The following diagram illustrates what happens:



The semantics of packing and unpacking are described below.

Form Packing

The PunchOutOrderMessage document is URL-Encoded (per the HTTP specification) and assigned to a hidden field on the Form named cXML-urlencoded. The HTML Form element is assigned a METHOD of POST and an ACTION consisting of the URL passed in the BrowserFormPost element of the PunchOutSetupRequest. For example:

```
<FORM METHOD=POST
        ACTION="http://workchairs.com:1616/punchoutexit">
    <INPUT TYPE=HIDDEN NAME="cXML-urlencoded"
        VALUE="URL-Encoded PunchOutOrderMessage document">
    <INPUT TYPE=SUBMIT VALUE="Proceed">
</FORM>
```

Additional HTML tags on the page might contain the above fragment to describe the contents of the shopping basket in detail.

> **Note:** When Web servers send the cXML-urlencoded field, it is not yet URL encoded. This encoding is required only when the form is submitted by Web browsers (when users click Check Out in the above example). Web browsers themselves meet this requirement. The Web server must HTML-encode only the field value, escaping quotation marks and other special characters, so the form displays properly for the user.

The name cXML-urlencoded is case insensitive.

For cXML-urlencoded data, the receiving parser cannot assume a charset parameter beyond the default for media type text/xml. No character encoding information for the posted data is carried in an HTTP POST. The receiving Web server cannot determine the encoding of the HTML page containing the hidden field. The cXML document forwarded in this fashion must therefore use us-ascii character encoding. Any characters (including those “URI encoded” as “%XX”) found in the XML source document must be in the “us-ascii” set. Other Unicode symbols can be encoded using character entities in that source document.

Base64 Encoding

The cXML-base64 hidden field eases handling of international documents. cXML documents containing symbols outside of “us-ascii” should use this field instead of the cXML-urlencoded hidden field. This alternative has almost identical semantics, but the entire document is base64-encoded throughout transport and not HTML-encoded to the browser or URL-encoded to the receiving Web server. Base64-encoding is described in RFC 2045 “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.”

The name cXML-base64 is case insensitive.

Base64-encoding from the supplier site through the browser and to the receiving Web server at the client maintains the original character encoding of a cXML document. Though no charset parameter arrives with the posted information, the decoded document (after the transfer encoding is removed) can be treated as the media type application/xml. This encoding allows the receiving parser to honor any encoding attribute specified in the XML declaration. For this field (as for any application/xml documents), the default character encoding is UTF-8.

Either of these hidden fields (cXML-urlencoded or cXML-base64) must appear in the data posted to the procurement application. Though recipients should first look for cXML-base64 in the data, it is wasteful to send both fields.

Form Unpacking and Processing

The procurement application, which previously provided the appropriate URL, receives an HTML Form POST containing the Form data as described above. The Form POST processor would first look for the cXML-base64 variable, extract the value and base64-decode its contents. If that field does not exist in the data, the Form POST processor would look for the cXML-urlencoded variable, extract the URL-encoded cXML message and URL-decode it. The decoded content of the field is then processed as if it had been received through a normal HTTP Request/Response cycle.

The implied media type of the document after decoding varies, with different possible character encodings:

- The cXML-urlencoded variable is of media type text/xml with no charset attribute. It is thus restricted to the us-ascii character encoding. The receiving parser must ignore any encoding attribute in the XML declaration of the cXML document because the browser might have changed the encoding.
- The cXML-base64 variable is of media type application/xml and thus might have any character encoding (indicated by the encoding attribute of the contained XML declaration, if any).

The primary difference between this transaction and a normal Request-Response transaction is that there is no response that can be generated, because there is no HTTP connection through which to send it.

Basic Elements

The following entities and elements are used throughout the cXML specification. Most of the definitions here are basic vocabulary with which the higher-order business documents are described. The common type entities and the common elements representing low-level objects are defined here.

Type Entities

Most of these definitions are from the XML-Data note submission to the World Wide Web Consortium (W3C). A few higher-level type entities that are also defined here are not from XML-Data. These types are also discussed in “cXML Envelope” on page 48.

isoLangCode

An ISO Language Code from the ISO 639 standard.

isoCountryCode

An ISO Country Code from the ISO 3166 standard.

xmlLangCode

A language code as defined by the XML 1.0 Specification (at www.w3.org/TR/1998/REC-xml-19980210.html). In the most common case, this includes an ISO 639 Language Code and (optionally) an ISO 3166 Country Code separated by a hyphen. Unlike the full XML recommendation, IANA or private language codes should not be used in cXML. IANA and private subcodes are allowed, though they should come after a valid ISO 3166 Country Code.

The recommended cXML language code format is xx[-YY[-zzz]*]? where xx is an ISO 639 Language code, YY is an ISO 3166 Country Code and zzz is an IANA or private subcode for the language in question. Again, use of the Country Code is always recommended. By convention, the language code is lowercase and the country code is uppercase. This is not required for correct matching of the codes.

unitOfMeasure

UnitOfMeasure describes how the product is packaged or shipped. It must conform to UN/CEFACT Unit of Measure Common Codes. For a list of UN/CEFACT codes, see www.unece.org/cefact.

URL

A URL (Uniform Resource Locator) as defined by the HTTP/1.1 standard.

Base Elements

These elements, used throughout the specification, range from generic ones such as Name and Extrinsic to specific ones such as Money.

Profile Transaction

The ProfileRequest and ProfileResponse documents must be supported by cXML 1.1 server implementations. This transaction can be used to retrieve server capabilities, including supported cXML version, transactions, and options on those transactions.

The response should list all Requests supported at a particular Website, not necessarily all those supported by the company. Suppliers that can receive OrderRequest documents and send various messages or initiate Request/Response transactions describe their OrderRequest support in the profile transaction.

The Profile transaction can be used to "ping" a server within the cXML protocol.

ProfileRequest

This element has no content. It is simply routed to the appropriate cXML server using the Header. The server responds with a single ProfileResponse as described below. The only dynamic portions of this response are the payloadId and timestamp attributes of the cXML element itself. In this particular case, a supplier is not required to provide responses in multiple locales.

An example Request of this type is:

```
<Request>
    <ProfileRequest />
</Request>
```

ProfileResponse

This element contains a list of supported transactions, their locations, and any supported options. While no options are yet defined, the following is a possible ProfileResponse:

```
<ProfileResponse effectiveDate="2001-03-03T12:13:14-05:00">
    <Option name="Locale">1</Option>
    …
    <Transaction requestName="PunchOutSetupRequest">
        <URL>http://www.workchairs.com/cXML/PunchOut.asp</URL>
        <Option name="operationAllowed">create inspect</Option>
        <Option name="dynamic pricing">0</Option>
        …
    </Transaction>
    …
</ProfileResponse>
```

A more likely ProfileResponse from a current supplier might be:

```
<ProfileResponse effectiveDate="2000-01-01T05:24:29-08:00">
    <Transaction requestName="OrderRequest">
        <URL>http://workchairs.com/cgi/orders.cgi</URL>
    </Transaction>
    <Transaction requestName="PunchOutSetupRequest">
        <URL>http://workchairs.com/cgi/PunchOut.cgi</URL>
    </Transaction>
</ProfileResponse>
```

ProfileResponse has the following attribute:

effectiveDate	The date and time when these services became available. Dates should not be in the future.

Option

Value for a defined option (either for the overall service or for a specific transaction). No options are yet defined.

Option has the following attribute:

name	The name of this option. This attribute should not be viewed directly (because the profile is intended for machine consumption).

Transaction

The description of a transaction supported by this service. The Profile definition currently indicates the locations to which to send specific requests. Future versions of cXML will add Option definitions and extend the Profile information to include more information about supported requests.

The Transaction element must contain a URL element.

Transaction has the following attribute:

requestName	A specific request that this server accepts at the given URL. Values can be: ProfileRequest, OrderRequest, PunchOutSetupRequest, StatusUpdateRequest, GetPendingRequest, SubscriptionListRequest, SupplierListRequest, SubscriptionContentRequest, SupplierDataRequest

Order Definitions

The cXML ordering documents are OrderRequest and a generic response. OrderRequest is analogous to a purchase order. The response is the acknowledgment that the supplier received the purchase order. It is not a commitment to execute the purchase order, but a confirmation that it was correctly received.

OrderRequest

The following example illustrates the structure of the OrderRequest element:

```
<OrderRequest>
    <OrderRequestHeader … >
        …
    </OrderRequestHeader>
    <ItemOut … >
        …
    </ItemOut>
    <ItemOut … >
        …
    </ItemOut>
</OrderRequest>
```

OrderRequestHeader

The following example shows an OrderRequestHeader in full detail:

```
<OrderRequestHeader orderID="DO1234"
        orderDate="1999-03-12T13:30:23+8.00"
        type="new"
        requisitionID="R1234">
    <Total>
```

```
        <Money currency="USD">12.34</Money>
    </Total>
    <ShipTo>
        <Address>
            <Name xml:lang="en">Acme Corporation</Name>
            <PostalAddress name="Headquarters">
                <DeliverTo>Joe Smith</DeliverTo>
                <DeliverTo>Mailstop M-543</DeliverTo>
                <Street>123 Anystreet</Street>
                <City>Sunnyvale</City>
                <State>CA</State>
                <PostalCode>90489</PostalCode>
                <Country isoCountryCode="US">USA</Country>
            </PostalAddress>
        </Address>
    </ShipTo>
    <BillTo>
        <Address>
            <Name xml:lang="en">Acme Corporation</Name>
            <PostalAddress name="Finance Building">
                <Street>124 Anystreet</Street>
                <City>Sunnyvale</City>
                <State>CA</State>
                <PostalCode>90489</PostalCode>
                <Country isoCountryCode="US">USA</Country>
            </PostalAddress>
        </Address>
    </BillTo>
    <Shipping>
        <Money currency="USD">12.34</Money>
        <Description xml:lang="en-US">FedEx 2-day</Description>
    </Shipping>
    <Tax>
        <Money currency="USD">12.34</Money>
        <Description xml:lang="en">CA State Tax</Description>
    </Tax>
    <Payment>
        <PCard number="1234567890123456" expiration="1999-03-12"/>
    </Payment>
    <Contact role="purchasingAgent">
        <Name xml:lang="en-US">Mr. Smart E. Pants</Name>
        <Email>sepants@acme.com</Email>
        <Phone name="Office">
            <TelephoneNumber>
                <CountryCode isoCountryCode="US">1</CountryCode>
                <AreaOrCityCode>800</AreaOrCityCode>
                <Number>555-1212</Number>
            </TelephoneNumber>
        </Phone>
    </Contact>
```

```
        <Comments xml:lang="en-US">
            Anything well formed in XML can go here.
        </Comments>
        <Followup>
            <URL>http://acme.com/cgi/orders.cgi</URL>
        </Followup>
    </OrderRequestHeader>
```

OrderRequestHeader has the following attributes:

orderID	The identifier for this order. Analogous to the purchase order number.
orderDate	The date and time this order was placed, in ISO 8601 format.
type (optional)	Type of the request: new (default), update, or delete.
requisitionID (optional)	The buyer's requisition identifier for this entire order. It might be the same as orderID, and it might not be included at all. Must not be included if requisitionID is specified in any ItemOut elements.
shipComplete (optional)	A preference against partial shipments. The only allowed value is "yes". By default, items are shipped when available. Because orders might include items with varying ShipTo elements, only groups of items with common shipping locations should be held until complete when shipComplete="yes".

OrderRequestHeader and ItemOut (when extended with ItemDetail) contain similar information. Where OrderRequestHeader includes overall billing (BillTo) and payment (Payment) information, ItemOut instead describes the individual items (in ItemID, ItemDetail and Distribution).

Do not use the information in OrderRequestHeader as the default for item-specific elements. If present, ShipTo, Shipping, Contact and each named Extrinsic must appear either with every ItemOut or in the OrderRequestHeader. Comments and Tax elements can appear simultaneously at both levels. But, the different Comments elements should not duplicate information, and the header-level Tax element contains a total for the order.

Total

This element contains the total monetary amount of the order. It is a container for the Money element.

ShipTo/BillTo

These elements contain the addresses of the Ship To and Bill To entities on the OrderRequest.

One order must be billed to a single entity. Therefore, the BillTo element appears only in the OrderRequestHeader. Items from an order can be sent to multiple locations. Like the Shipping element (see next section), the ShipTo element can therefore appear either in the OrderRequestHeader or in individual ItemOut elements.

Shipping

This element describes how to ship the items in the request and the cost of doing so. If the Shipping element is present in the OrderRequestHeader, it must not appear on individual ItemOut elements. If it is not present in the OrderRequestHeader, it must appear in the ItemOut elements.

Tax

This element contains the tax associated with the order. This element is present if the buying organization computes tax. When appearing within the OrderRequestHeader, Tax describes the total tax for an order. Tax elements at the item level can describe individual tax amounts.

Payment

This element describes the payment instrument used to pay for the items requested. In the above example, the Payment element contains a PCard element, which encodes a standard purchasing card into the cXML document. In the future, other payment instruments will be defined and supported.

Contact

Contact information the supplier can use to follow up on an order. This element identifies a person and provides a list of ways to reach that person or entity. The only required element is the Name of the contact. Optional and repeating possibilities include PostalAddress (not recommended for immediate correction of order problems), Email, Phone, Fax, and URL.

Buying organizations might choose to use this element to identify the original requestor, the procurement application system administrator, or some other contact who can take responsibility for correcting problems with orders. Contact can differ from both BillTo and ShipTo information for an order.

Contact has the following attribute:

role (optional)	The position of this person within the procurement process. Can take the values endUser, administrator, purchasingAgent, technicalSupport, customerService or sales.

The same Conact role must not appear at both the header and item levels.

There is no default role, due to the disparate contents of the Contact element. So, cXML applications treat a Contact without a role attribute as an additional role.

Comments

Arbitrary human-readable information buyers can send within purchase orders. This string data is not intended for the automated systems at supplier sites.

The Comments element can contain an Attachment element for including external files.

Attachment

Comments can attach external files to augment purchase orders. The Attachment element appears within Comments, and it contains only a reference to the external MIME part of the attachment. All attachments should be sent in a single multipart transmission with the OrderRequest document. Even if this is not possible, the contentID provided by the Attachment element must be usable to retrieve the attachment.

For details about the transfer of attached files, see “Attachment Transmission” on page 51.

Attachment contains a single URL with scheme “cid:”. An attached file in a cXML document might appear as:

```
<Comments>
    <Attachment>
        <URL>cid: uniqueCID@cxml.org</URL>
    </Attachment>
    Please see attached image for my idea of what this
    should look like
    </Comments>
```

The Comments element appears in many places within the cXML protocol, but it can contain the Attachment element only within OrderRequest documents.

Followup

Specifies the URL to which future StatusUpdateRequest documents should be posted. This location is the input location for any later documents that reference the current OrderRequest document.

Extrinsic

This element contains machine-readable information related to the order, but not defined by the cXML protocol. In contrast, the Comments element passes information for human use. Extrinsic elements contain data that is likely to appear in later documents; the Comments element does not. At this level, Extrinsic extends the description of all items contained in the purchase order. Some Extrinsic information might also describe the overall purchase order without affecting the meaning of any contained ItemOut.

Each named Extrinsic can appear only once within the lists associated with the OrderRequestHeader and individual ItemOut elements (within the contained ItemDetail elements). The same name must not appear in both the OrderRequestHeader list and any list associated with the ItemOut elements. If the same Extrinsic name and value is repeated in all ItemOut lists, it should be moved to the OrderRequestHeader.

The Extrinsic element can also appear in the IndexItem, PunchOutSetupRequest and ContractItem elements. These contexts are described later in this document.

ItemOut

The following example shows a minimum valid ItemOut element.

```
<ItemOut quantity="1">
    <ItemID>
        <SupplierPartID>5555</SupplierPartID>
    </ItemID>
</ItemOut>
```

ItemOut has the following attributes:

quantity	The number of items desired. Fractions are allowed for some units of measure. The value might have already been checked by the supplier during a punchout session. Should never be negative.
lineNumber (optional)	Position of this item within an order. This ordinal value increases once per ItemOut in a "new" OrderRequest. Clients should always specify this attribute in an OrderRequest, although it might not be useful in other ItemOut contexts.
requisitionID (optional)	The buyer's requisition identifier for this line item. Must not be included if requisitionID is specified in the OrderRequestHeader.
requestedDeliveryDate (optional)	The date item was requested for delivery, which allows item-level delivery dates in the OrderRequest. It must be in ISO 8601 format.

The lineNumber attribute remains constant for any item through updates to the order. Deletion of items from an order never changes the lineNumber of remaining items. New items have higher numbers than those previously included in the order. A change to an existing item (an increased quantity, for example) does not affect the lineNumber of that item.

The following example shows a more complicated ItemOut.

```
<ItemOut quantity="2" lineNumber="1"
    equestedDeliveryDate="1999-03-12">
    <ItemID>
        <SupplierPartID>1233244</SupplierPartID>
        <SupplierPartAuxiliaryID>ABC</SupplierPartAuxiliaryID>
    </ItemID>
    <ItemDetail>
        <UnitPrice>
            <Money currency="USD">1.34</Money>
        </UnitPrice>
        <Description xml:lang="en">hello</Description>
        <UnitOfMeasure>EA</UnitOfMeasure>
        <Classification domain="SPSC">12345</Classification>
        <ManufacturerPartID>234</ManufacturerPartID>
        <ManufacturerName xml:lang="en">foobar</ManufacturerName>
        <URL>www.bar.com</URL>
    </ItemDetail>
    <ShipTo>
        <Address>
            <Name xml:lang="en">Acme Corporation</Name>
            <PostalAddress name="Headquarters">
                <Street>123 Anystreet</Street>
                <City>Sunnyvale</City>
                <State>CA</State>
                <PostalCode>90489</PostalCode>
                <Country isoCountryCode="US">USA</Country>
            </PostalAddress>
        </Address>
    </ShipTo>
    <Shipping>
        <Money currency="USD">1.34</Money>
        <Description xml:lang="en-US">FedEx 2-day</Description>
    </Shipping>
    <Tax>
        <Money currency="USD">1.34</Money>
        <Description xml:lang="en">foo</Description>
    </Tax>
    <Distribution>
        <Accounting name="DistributionCharge">
            <Segment type="G/L Account" id="23456"
                description="Entertainment"/>
```

```
                    <Segment type="Cost Center" id="2323"
                        description="Western Region Sales"/>
                </Accounting>
                <Charge>
                    <Money currency="USD">.34</Money>
                </Charge>
            </Distribution>
            <Distribution>
                <Accounting name="DistributionCharge">
                    <Segment type="G/L Account" id="456"
                        description="Travel"/>
                    <Segment type="Cost Center" id="23"
                        description="Europe Implementation"/>
                </Accounting>
                <Charge>
                    <Money currency="USD">1</Money>
                </Charge>
            </Distribution>
            <Comments xml:lang="en-US">
                Anything valid in XML can go here.
            </Comments>
        </ItemOut>
```

The ItemDetail element allows additional data to be sent to suppliers instead of just the unique identifier for the item represented by the ItemID.

The ShipTo, Shipping, Tax, Contact, Comments and Extrinsic elements (some nested within ItemDetail) are identical to the ones that can be in the OrderRequestHeader. These elements allow per-item data such as shipping, shipping type, and associated cost to be represented. Use these elements either as the OrderRequestHeader level, or at the ItemOut level, but not at both levels.

Distribution

Distribution divide the cost of an item among multiple parties. Suppliers return the Distribution element on invoices to facilitate the buyer's reconciliation process.

Accounting

The Accounting element groups Segments to identify who is charged.

Accounting has the following attribute:

name	The name for this accounting combination.

Segment has the following attributes:

type	An identifying name for this Segment with respect to the others in the Accounting element.
id	The unique identifier within this Segment type. This value might be the actual account code if the type were “Cost Center”.

Charge

This element specifies the amount to be charged to the entity represented by the Accounting element.

Response to an OrderRequest

This is the response part of the synchronous Request-Response transaction. The following example shows a Response to an OrderRequest document:

```
<cXML version="1.1.007" payloadID="9949494" xml:lang="en"
        timestamp="1999-03-12T18:39:09-08:00">
    <Response>
        <Status code="200" text="OK"/>
    </Response>
</cXML>
```

As shown above, this Response is straightforward. In this case, there is no actual element named “OrderResponse”, because the only data that needs to be sent back to the requestor is the Status part of the Response.

The Response tells the requestor its OrderRequest was successfully parsed and acted on by the remote part of HTTP connection. It does not communicate order-level acknowledgement such as which items can be shipped, or which need to be backordered.

Punchout Transaction

The punchout message definitions are request/response messages that are carried inside the Request and Response elements. All of the following messages must be implemented by suppliers to support punchout.

PunchOutSetupRequest

PunchOutSetupRequest and PunchOutSetupResponse are the request/response pair used to set up a punchout session to a remote system. The client uses them to identify the procurement application, send setup information, and receive a response indicating where to go to initiate an HTML browsing session on the remote Website.

A PunchOutSetupRequest element is carried within the Request element. The following example shows a PunchOutSetupRequest.

```
<PunchOutSetupRequest operation="create">
    <BuyerCookie>34234234ADFSDF234234</BuyerCookie>
    <Extrinsic name="department">Marketing</Extrinsic>
    <BrowserFormPost>
        <URL>http://orms.acme.com:1616/punchoutexit</URL>
    </BrowserFormPost>
    <SelectedItem>
        <ItemID>
            <SupplierPartID>54543</SupplierPartID>
        </ItemID>
    </SelectedItem>
    <SupplierSetup>
        <URL>http://workchairs.com/cxml</URL>
    </SupplierSetup>
</PunchOutSetupRequest>
```

PunchOutSetupRequest has the following attribute:

operation	Specifies the type of PunchOutSetupRequest: “create”, “inspect”, or “edit”.

This element also contains the following elements: BuyerCookie, Extrinsic, BrowserFormPost, Contact, ShipTo, SelectedItem, SupplierSetup and an ItemOut list. Only the BuyerCookie element is required. The structure of Extrinsic, Contact and ShipTo elements is discussed in more detail in “OrderRequestHeader” on page 65. The ItemOut element is discussed in “ItemOut” on page 70. In this context (outside of an OrderRequest), the Distribution and Comments elements and lineNumber, requisitionID, and requestedDeliveryDate attributes of an ItemOut add little or no value and should not be included. Because punchout sessions take place before ordering, this information is not relevant within a PunchOutSetupRequest.

An ItemOut list describes an existing shopping cart (items from a previous punchout session). The inspect operation initiates a read-only punchout session (enforced by both the client and the server) to view details about the listed items. The edit operation also starts from the previous shopping cart (described using the ItemOut list), but

allows changes. Support for the edit operation implies inspect support (see "PunchOutOrderMessageHeader" on page 78 and "Empty Shopping Carts" on page 78).

BuyerCookie

This element transmits information that is opaque to the remote Website, but must be returned to the originator for all subsequent punchout operations. This element allows the procurement application to match multiple outstanding punchout requests.

BrowserFormPost

This element is the destination for the data in the PunchOutOrderMessage. It contains a URL element whose use will be further explained in the PunchOutOrderMessage definition. If the URL-Form-Encoded method is not being used, this element does not have to be included.

Extrinsic

This optional element contains any additional data that the requestor wants to pass to the external Website. This example passes the department of the user initiating the punchout operation. The cXML specification does not define the content of Extrinsic elements—it is something that each requestor and remote Website must agree on and implement.

Extrinsic elements are intended to provide additional machine-readable information. They extend the cXML protocol to support features not required by all implementations. In this context, the new data further describes the user initiating the punchout request.

The Extrinsic element can also appear in the OrderRequestHeader, ItemDetail, and ContractItem elements. These contexts are described further elsewhere in this document.

SelectedItem

This optional element indicates the items users want to punchout to purchase. It contains a single, required ItemID. This item leads users from their local catalog to the supplier's Website.

This element is usually present in create operations. Procurement applications that allow users to punchout directly from a supplier listing should leave out SelectedItem.

For edit and inspect operations, SelectedItem should appear only if the user chose to return to the supplier's Website while viewing new information in the local catalog rather than items in an existing requisition. In either case, the current shopping cart must appear in the ItemOut list.

Suppliers can create their catalogs so that SelectedItem leads to store-, aisle-, or product-level punchout. The more specific the item is in the catalog, the less searching users have to do at the supplier's Website.

SupplierSetup

This optional element specifies the URL to which to post the PunchOutSetupRequest. This element is not needed if the e-commerce network hub knows the supplier's punchout URL.

PunchOutSetupResponse

After the remote Website has received a PunchOutSetupRequest, it responds with a PunchOutSetupResponse, as shown below:

```
<PunchOutSetupResponse>
    <StartPage>
        <URL>
            http://premier.workchairs.com/store?23423SDFSDF23
        </URL>
    </StartPage>
</PunchOutSetupResponse>
```

StartPage

This element contains a URL element that specifies the URL to pass to the browser to initiate the punchout browsing session requested in the PunchOutSetupRequest. This URL must contain enough state information to bind to a session context on the remote Website, such as the requestor identity and the appropriate BuyerCookie element.

At this point, the user who initiated the PunchOutSetupRequest browses the external Website, and selects items to be transferred back to the procurement application through a PunchOutOrderMessage.

PunchOutOrderMessage

This element sends the contents of the remote shopping basket to the originator of a PunchOutSetupMessage. It can contain much more data than the other messages because it needs to be able to fully express the contents of any conceivable shopping basket on the external Website. This message does not strictly follow the Request/Response model.

The remote Website generates a PunchOutOrderMessage when the user checks out. This message communicates the contents of the remote shopping basket to the procurement application; for example:

```
<PunchOutOrderMessage>
    <BuyerCookie>34234234ADFSDF234234</BuyerCookie>
    <PunchOutOrderMessageHeader operationAllowed="create">
        <Total>
            <Money currency="USD">100.23</Money>
        </Total>
    </PunchOutOrderMessageHeader>
    <ItemIn quantity="1">
        <ItemID>
            <SupplierPartID>1234</SupplierPartID>
            <SupplierPartAuxiliaryID>
                additional data about this item
            </SupplierPartAuxiliaryID>
        </ItemID>
        <ItemDetail>
            <UnitPrice>
                <Money currency="USD">10.23</Money>
            </UnitPrice>
            <Description xml:lang="en">
                Learn ASP in a Week!
            </Description>
            <UnitOfMeasure>EA</UnitOfMeasure>
            <Classification domain="SPSC">12345</Classification>
        </ItemDetail>
    </ItemIn>
</PunchOutOrderMessage>
```

The following sections discuss these elements.

BuyerCookie

This element is the same element that was passed in the original PunchOutSetupRequest. It must be returned here to allow the procurement application to match the PunchOutOrderMessage with an earlier PunchOutSetupRequest.

PunchOutOrderMessageHeader

This element contains information about the entire shopping basket contents being transferred. The only required element is Total, which is the overall cost of the items being added to the requisition. Additional elements that are allowed are Shipping and Tax, which are the amount and description of any shipping or tax charges computed on the remote Website. ShipTo is also optional, and it specifies the Ship-To addressing information the user selected on the remote site or that was passed in the original PunchOutSetupRequest. All monetary amounts are in a Money element that always specifies currency in a standardized format.

PunchOutOrderMessageHeader has the following attribute:

operationAllowed	Specifies the PunchOutSetupRequest operations allowed: create, inspect, or edit.

This attribute controls whether the user can initiate later PunchOutSetupRequest transactions containing data from this PunchOutOrderMessage. If operationAllowed="create", only a later OrderRequest can contain these items. Otherwise, the procurement application can inspect or edit the shopping cart later (initiating subsequent PunchOutSetupRequest transactions with the appropriate operations and the ItemID elements corresponding to the ItemIn list returned in this PunchOutOrderMessage). Support for edit implies support for inspect. The procurement application can always use the items in a subsequent OrderRequest.

Empty Shopping Carts

The PunchOutOrderMessage can contain a list of items corresponding to a shopping cart on the supplier Website. It always indicates the end of the interactive punchout session. The following paragraphs detail a few cases without a list of items in the PunchOutOrderMessage. These messages allow clients to resume immediately when the user leaves the supplier Website.

If the operation in the original PunchOutSetupRequest was inspect, the item list of the PunchOutOrderMessage must be ignored by the procurement application. The supplier site should return no ItemIn elements in this case. If a PunchOutOrderMessage contains no ItemIn elements and the operation was create, no items should be added to the requisition. The supplier site or the user has cancelled the punchout session without creating a shopping cart. If the operation was edit and the PunchOutOrderMessage contains no ItemIn elements, existing items from this punchout session must be deleted from the requisition in the procurement application.

The Status code “204/No Content” indicates the end of a session without change to the shopping cart. Again, the PunchOutOrderMessage (which is always needed for the BuyerCookie) should not contain ItemIn elements. This code would be handled

identically to the other "empty" cases detailed above unless the operation was edit. In that case, the user cancelled the session without making any change and no change should be made to the requisition in the procurement application.

ItemIn

This element adds an item from a shopping basket to a requisition in the procurement application. It can contain a variety of elements, only two of which are required: ItemID and ItemDetail.

ItemIn has the following attributes:

quantity	The number of items selected by the user on the remote Website. Because the supplier site can enforce rules for partial units, the protocol allows fractional quantities. Should never be negative.
lineNumber (optional)	The position of this item within an order. Because punchout sessions normally take place prior to ordering and the server cannot control placement of items within an order in any case, this attribute is not relevant within a PunchOutOrderMessage.

The optional elements are ShipTo, Shipping, and Tax, which are the same elements as those specified in PunchOutOrderMessage, above.

With the exception of the Distribution and Comments elements and requisitionID and requestedDeliveryDate attributes available in the ItemOut element, the ItemIn and ItemOut structures match one-to-one. The originating buying system can convert directly between ItemIn and ItemOut lists when initiating an inspect or edit operation. Suppliers can convert one to the other (dropping the listed extensions available in the ItemOut element) when executing an edit operation. The originating buying system can perform the direct conversion and add additional shipping and distribution information and comments when initiating an OrderRequest transaction. ItemDetail data (with the possible exception of Extrinsic elements) contained within ItemIn elements must not be removed when converting from ItemIn to ItemOut.

ItemID

This element uniquely identifies the item to the remote Website. It is the only element required to be returned to the remote Website to re-identity the item being transferred.

ItemID contains two elements: SupplierPartID and SupplierPartAuxiliaryID. Only SupplierPartID is required. SupplierPartAuxiliaryID helps the remote Website transport complex configuration or bill-of-goods information to re-identify the item when it is presented to the remote Website in the future.

If SupplierPartAuxiliaryID contains special characters (for example, if it contains additional XML elements not defined in the cXML protocol), they must be escaped properly. Due to the necessity to pass SupplierPartAuxiliaryID information through applications and back to the originating supplier, an internal subset containing any additional XML elements is insufficient.

ItemDetail

This element contains descriptive information about the item that procurement applications present to users. The contents of an ItemDetail element can be quite complex, but the minimum requirements are simple: UnitPrice, Description, UnitOfMeasure, and Classification.

In the context of an ItemIn element, the Extrinsic elements contained within an ItemDetail function identically to those found within an Index (specifically an IndexItemAdd).

Description

This element describes the item in a textual form. Because this text might exceed the limits of a short table of line items (or other constrained user interface) and random truncations could occur, the Description element contains an optional ShortName element. If provided, clients should present the ShortName instead of a truncation of the Description text in any restricted fields. Clients must continue to truncate the Description text if no ShortName is provided.

For example:

```
<Description xml:lang="en-US">
    <ShortName>Big Computer</ShortName>
    This wonder contains three really big disks, four CD-Rom drives, two Zip drives, an
ethernet card or two, much more memory than you could ever use, four CPUs on two
motherboards.  We'll throw in two monitors, a keyboard and the cheapest mouse we can
find lying around.
</Description>
```

might appear as “Big Computer” where space is tight, and “Big Computer: This wonder … lying around.” (or as two separate but complete fields) where there is space to display more text.

Later Status Changes

After the OrderRequest transaction has completed, suppliers and intermediate servers might need to communicate additional information back to the buying system. The transactions described in this section are used for that purpose. These transactions share some common semantics and elements.

Like the response to an OrderRequest (see "Response to an OrderRequest" on page 73), none of these transactions includes a specific Response element. Instead, the returned document contains a nearly empty Response (only a Status). Each returned document has the form:

```
<cXML version="1.1.007" payloadID="9949494@supplier.com"
        timestamp="2000-01-12T18:39:09-08:00" xml:lang="en-US">
    <Response>
        <Status code="200" text="OK"/>
    </Response>
</cXML>
```

The returned code is "200" only if the operation completed successfully.

DocumentReference

The DocumentReference element contains enough information to associate the update request with a particular document. It repeats a required attribute of the earlier document and adds one optional identifier generated by the supplier. For example:

```
<DocumentReference
    payloadID="0c300508b7863dcclb_14999"/>
```

DocumentReference contains no elements, but has the following attribute:

payloadID	A unique number with respect to space and time that is used for logging purposes to identify documents. This value should not change in the case of retry attempts. The recommended implementation is: datetime.process id.random number@hostname Taken directly from the cXML element of the OrderRequest document.

StatusUpdateRequest

This transaction informs an earlier node about changes in the processing status of an order. One change is of particular significance: When an intermediate hub successfully transmits an OrderRequest onward, it can inform the original sender or a previous hub about that success. Transitions through various queues and processing steps at a supplier or hub might also be significant to the buyer.

This request updates the processing status of a single OrderRequest document. For example:

```
<cXML version="1.1.007" xml:lang="en-US"
        payloadID="0c30050@supplier.com"
        timestamp="2000-01-08T23:00:06-08:00">
    <Header>
        Routing, identification and authentication information.
    </Header>
    <Request>
        <StatusUpdateRequest>
            <DocumentReference
                payloadID="0c300508b7863dcclb_14999"/>
            <Status code="200" text="OK" xml:lang="en-US">Forwarded
                to supplier</Status>
        </StatusUpdateRequest>
    </Request>
</cXML>
```

This request contains only an DocumentReference and a Status element. Both are required. The Status can communicate a later transport error encountered by an intermediate hub. The semantics of this element are identical to a Status that might have been returned in the initial HTTP response to an OrderRequest document.

The 200/OK code is especially important when documents are stored and forwarded. This code indicates that a supplier has begun processing the OrderRequest or a hub has forwarded the document. The recipient should expect no further StatusUpdateRequest documents after 200/OK arrives.

Suppliers and hubs utilizing the StatusUpdate transaction must return code 201/Accepted when an OrderRequest is queued for later processing. After it sends 200/OK (in the immediate Response to an OrderRequest or a later StatusUpdateRequest), the server should send no further StatusUpdate transactions for that order. Errors later in processing might lead to exceptions to this rule.

Catalog Definitions

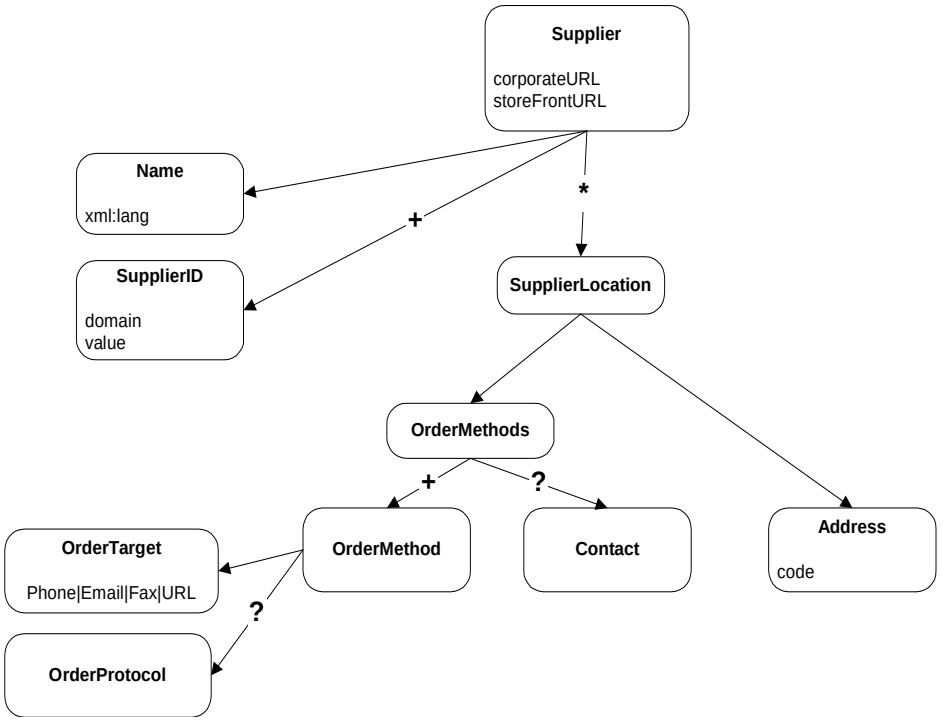
The cXML catalog definitions consist of three main elements: Supplier, Index, and Contract. All three elements describe data intended for persistent or cached use within a hub or a buying organization's procurement system.

- Supplier—Contains basic data about the supplier such as address, contact, and ordering information.
- Index—Describes data about the supplier's inventory of goods and services, such as description, part numbers, and classification codes.
- Contract—Describes data about flexible aspects of the inventory negotiated between the buyer and supplier, such as price.

Note that Index uses several sub-elements to describe line items in suppliers' inventories. Suppliers can send either price information for caching within buyers' systems, or punchout information to enable buyers to punch-out to remote Websites for pricing and other information.

Supplier

The Supplier element encapsulates a named supplier of goods or services. It must have a Name element and a SupplierID element. It additionally describes optional address and ordering information for the supplier:



Supplier has the following attributes:

corporateURL (optional)	URL for supplier's Website.
storeFrontURL (optional)	URL for Website for shopping or browsing.

The following example shows an outline of the Supplier element:

```
<Supplier>
    <SupplierID domain="InternalSupplierID">29</SupplierID>
    <SupplierID domain="DUNS">76554545</SupplierID>
    <SupplierLocation>
        <Address>
```

```
                    <Name xml:lang="en-US">Main Office</Name>
                    <PostalAddress>
                        …
                    </PostalAddress>
                <Email>bobw@workchairs.com</Email>
                <Phone name="Office">
                    …
                </Phone>
                <Fax name="Order">
                    …
                </Fax>
                <URL>http://www.workchairs.com/Support.htm</URL>
                </Address>
                <OrderMethods>
                    <OrderMethod>
                        <OrderTarget>
                            <URL>http://www.workchairs.com/cxmlorders</URL>
                        </OrderTarget>
                    </OrderMethod>
                    <Contact>
                        <Name xml:lang="en-US">Mr. Smart E. Pants</Name>
                        <Email>sepants@workchairs.com</Email>
                        <Phone name="Office">
                            …
                        </Phone>
                    </Contact>
                </OrderMethods>
            </SupplierLocation>
        </Supplier>
```

SupplierLocation

Some suppliers conduct business from more than one location. A SupplierLocation element can be used for each location. This element also encapsulates how that location does business, or the ways that it can accept orders. A SupplierLocation element contains an Address and a set of OrderMethods.

OrderMethods and OrderMethod

The OrderMethods element is a grouping of one or more OrderMethod elements for the given SupplierLocation element. The position of OrderMethods in the list is significant—the first element is the preferred ordering method, the second element is the next priority, and so on in decreasing order of preference.

OrderMethod encapsulates ordering information in the form of an order target (such as phone, fax, or URL) and an optional protocol to further clarify the ordering expectations at the given target; for example, “cxml” for a URL target.

Index

This element is the root element for updating catalogs within buying organization's procurement systems.

An Index element is associated with a single supplier. The Index element allows for a list of supplier IDs, where each ID is considered a synonym for that supplier.

The Index contains one or more IndexItem elements as well as an optional set of SearchGroup elements for defining parametric search data for items. The IndexItem element contains elements that add or delete from the buying organization's cached catalog. The following example shows an outline of an Index element:

```
<Index>
    <SupplierID> … </SupplierID>
    ...
    <IndexItem>
        <IndexItemAdd>
            <IndexItemDetail>
                …
            </IndexItemDetail>
        </IndexItemAdd>
        …
        <IndexItemDelete>
            …
        </IndexItemDelete>
        …
        <IndexItemPunchout>
            …
        </IndexItemPunchout>
    </IndexItem>
</Index>
```

IndexItem, IndexItemAdd, IndexItemDelete and IndexItemPunchout

The IndexItem element is a container for the list of items in an index. It contains three types of elements:

- IndexItemAdd—Inserts a new item or updates an existing item in the index. It contains an ItemID element, a ItemDetail element, and a IndexItemDetail element.
- IndexItemDelete—Removes an item from the index. It contains an ItemID element identifying the item.
- IndexItemPunchout—Inserts an item for initiating puchout to the supplier's Website. It contains a PunchoutDetail element and an ItemID element. It is similar to an IndexItemAdd element except that it does not require price information. Buyers acquire item details in real-time from the supplier's Website.

ItemID

The ItemID element uniquely identifies a supplier's items. It contains a SupplierPartID element and an optional SupplierPartAuxiliaryID element.

SupplierPartAuxiliaryID

If SupplierPartID does not uniquely identify the item, the supplier should use SupplierPartAuxiliaryID to specify an "auxiliary" key that identifies the part uniquely when combined with the SupplierID and SupplierPartID. For example, a supplier might use the same SupplierPartID for an item, but have a different price for units of "EA" and "BOX". In this case, a reasonable SupplierPartAuxiliaryID for the two items might be "EA" and "BOX."

SupplierPartAuxiliaryID could also be used as a supplier cookie, enabling the supplier to refer to complex configuration or part data. It could contain all the data necessary for the supplier to reconstruct what the item in question is in their computer system (a basket or cookie of data that makes sense only to the supplier). For more information, see "Buyer and Supplier Cookies" on page 36.

ItemDetail

ItemDetail contains detailed information about an item, or all the data that a user might want to see about an item beyond the essentials represented in the ItemID. It must contain a UnitPrice, a UnitOfMeasure, one or more Description elements, and a Classification, and it can optionally contain a ManufacturerPartID, a ManufacturerName, a URL, and any number of Extrinsic elements. For more information, see "ItemDetail" on page 80.

In the context of an IndexItemAdd, Extrinsic elements extend information about a particular item. These extensions should not be transmitted to a supplier within an OrderRequest, because the supplier can retrieve the same data using the unique ItemID.

IndexItemDetail

The IndexItemDetail element contains index-specific elements that define additional aspects of an item, such as LeadTime, ExpirationDate, EffectiveDate, SearchGroupData, or TerritoryAvailable.

PunchoutDetail

PunchoutDetail is similar to ItemDetail, except it requires only one or more Description elements and a Classification. It can also contain URL, ManufacturerName, ManufacturerPartID, ExpirationDate, EffectiveDate, SearchGroupData, TerritoryAvailable, and Extrinsic elements. It does not contain pricing, lead time, or unit of measure information.

Contract

A Contract element represents a contract between a supplier and buyer for goods or services described in the supplier's index. It allows the supplier to "overlay" item attributes (such as price) in the index with values negotiated with the buyer. It further allows suppliers and buyers to segment these overlays based on an agreed-upon "segment key", meaningful within a buying organization, such as the name of a plant or a cost center.

Contract has the following attributes:

effectiveDate	Effective date and time of the contract, in ISO 8601 format.
expirationDate	Expiration date and time of the contract, in ISO 8601 format.

Contract contains one or more ItemSegment elements, for example:

```
<Contract effectiveDate="2000-01-03T18:39:09-08:00"
        expirationDate="2000-07-03T18:39:09-08:00">
    <SupplierID domain="InternalSupplierID">29</SupplierID>
    <ItemSegment segmentKey=Plant12>
        <ContractItem>
            <ItemID>
                <SupplierPartID>pn12345</SupplierPartID>
            </ItemID>
            <UnitPrice>
                <Money currency=USD>40.00</Money>
            </UnitPrice>
        </ContractItem>
        …
    </ItemSegment>
</Contract>
```

ItemSegment

The ItemSegment element is a container for a list of ContractItem elements for a given "segment," where a segment represents an arbitrary partitioning of contract items based on a segment key agreed upon between supplier and buyer.

ItemSegment has the following attribute:

segmentKey (optional)	Agreed-upon string used to segment custom prices.

ContractItem

A contract item element is a particular item overlay for an index item. It contains an ItemID that uniquely identifies the index item within the procurement system to overlay. It can contain any number of Extrinsic elements containing the overlaid value for the named index item attribute.

Subscription Management Definitions

Intermediaries, such as e-commerce network hubs, can manage the suppliers and supplier catalogs used by buying organizations' procurement systems. These intermediaries can provide direct links between procurement systems and supplier systems. This section contains element definitions for managing supplier data and catalog contents. These definitions build on many of the previous definitions for cXML request/responses, one-way messages, and catalog definitions.

Supplier Data

The definitions for supplier data management consist mainly of the elements SupplierListRequest, SupplierListResponse, SupplierDataRequest, SupplierDataResponse, and SupplierChangeMessage. These elements are described below with examples where the intermediary is Ariba Network.

SupplierListRequest

SupplierListRequest requests a list of the suppliers with whom the buyer has established trading relationships.

```
<Request>
    <SupplierListRequest/>
</Request>
```

SupplierListResponse

SupplierListResponse lists the suppliers with whom the buyer has established trading relationships.

```
<Response>
    <Status code="200" text="OK"/>
    <SupplierListResponse>
        <Supplier corporateURL=http://www.workchairs.com
                storeFrontURL="http://www.workchairs.com">
            <Name xml:lang="en-US">Workchairs, Inc.</Name>
```

```
            <Comments xml:lang="en-US">this is a cool company</Comments>
            <SupplierID domain="DUNS">123456</SupplierID>
        </Supplier>
        <Supplier corporateURL=http://www.computersRus.com
                storeFrontURL="http://www.computersRus.com">
            <Name xml:lang="en-US">Computers R us</Name>
            <Comments xml:lang="en-US">another cool company</Comments>
            <SupplierID domain="DUNS">123456789</SupplierID>
        </Supplier>
    </SupplierListResponse>
</Response>
```

SupplierDataRequest

SupplierDataRequest requests data about a supplier.

```
<Request>
    <SupplierDataRequest>
        <SupplierID domain="DUNS">123456789</SupplierID>
    </SupplierDataRequest>
</Request>
```

SupplierDataResponse

SupplierDataResponse contains data about a supplier.

```
<Response>
    <Status code="200" text="OK"/>
    <SupplierDataResponse>
        <Supplier corporateURL=http://www.workchairs.com
                storeFrontURL="http://www.workchairs.com">
            <Name xml:lang="en-US">Workchairs, Inc.</Name>
            <Comments xml:lang="en-US">this is a cool company</Comments>
            <SupplierID domain="DUNS">123456</SupplierID>
            <SupplierLocation>
                <Address>
                    <Name xml:lang="en-US">Main Office</Name>
                    <PostalAddress>
                        <DeliverTo>Bob A. Worker</DeliverTo>
                        <Street>123 Front Street</Street>
                        <City>Toosunny</City>
                        <State>CA</State>
                        <PostalCode>95000</PostalCode>
                        <Country isoCountryCode="US">USA</Country>
                     </PostalAddress>
                    <Email>bobw@workchairs.com</Email>
                    <Phone name="Office">
                        <TelephoneNumber>
                            <CountryCode
```

```
                                        isoCountryCode="US">1</CountryCode>
                                    <AreaOrCityCode>800</AreaOrCityCode>
                                    <Number>5551212</Number>
                                </TelephoneNumber>
                            </Phone>
                            <Fax name="Order">
                                <TelephoneNumber>
                                    <CountryCode
                                        isoCountryCode="US">1</CountryCode>
                                    <AreaOrCityCode>408</AreaOrCityCode>
                                    <Number>5551234</Number>
                                </TelephoneNumber>
                            </Fax>
                            <URL>http://www.workchairs.com/Support.htm</URL>
                        </Address>
                        <OrderMethods>
                            <OrderMethod>
                                <OrderTarget>
                                    <URL>http://www.workchairs.com/cxmlorder</URL>
                                </OrderTarget>
                                <OrderProtocol>cXML</OrderProtocol>
                            </OrderMethod>
                        </OrderMethods>
                    </SupplierLocation>
                </Supplier>
            </SupplierDataResponse>
        </Response>
```

SupplierChangeMessage

This element is for notification of changes to supplier data.

```
<Message>
    <SupplierChangeMessage type="new">
        <Supplier corporateURL=http://www.workchairs.com
                storeFrontURL="http://www.workchairs.com">
            <Name xml:lang="en-US">Workchairs, Inc.</Name>
            <Comments xml:lang="en-US">this is a cool company</Comments>
            <SupplierID domain="DUNS">123456</SupplierID>
            <SupplierLocation>
                <Address>
                    <Name xml:lang="en-US">Main Office</Name>
                    <PostalAddress>
                        <DeliverTo>Bob A. Worker</DeliverTo>
                        <Street>123 Front Street</Street>
                        <City>Toosunny</City>
                        <State>CA</State>
                        <PostalCode>95000</PostalCode>
                        <Country isoCountryCode="US">USA</Country>
```

```
                                    </PostalAddress>
                                    <Email>bobw@workchairs.com</Email>
                                    <Phone name="Office">
                                        <TelephoneNumber>
                                            <CountryCode
                                            isoCountryCode="US">1</CountryCode>
                                            <AreaOrCityCode>800</AreaOrCityCode>
                                            <Number>5551212</Number>
                                        </TelephoneNumber>
                                    </Phone>
                                    <Fax name="Order">
                                        <TelephoneNumber>
                                            <CountryCode
                                            isoCountryCode="US">1</CountryCode>
                                            <AreaOrCityCode>408</AreaOrCityCode>
                                            <Number>5551234</Number>
                                        </TelephoneNumber>
                                    </Fax>
                                    <URL>http://www.workchairs.com/Support.htm</URL>
                                </Address>
                                <OrderMethods>
                                    <OrderMethod>
                                        <OrderTarget>
                                            <URL>http://www.workchairs.com/cxmlorder</URL>
                                        </OrderTarget>
                                        <OrderProtocol>cXML</OrderProtocol>
                                    </OrderMethod>
                                </OrderMethods>
                            </SupplierLocation>
                        </Supplier>
                    </SupplierChangeMessage>
                </Message>
```

Catalog Subscriptions

The definitions for catalog-subscription management are described below. The examples show the intermediary as Ariba Network.

Subscription

This element captures metadata about a single catalog subscription. Its sub-elements include:

- InternalID – a unique ID internal to the intermediary
- Name – the name of the subscription
- ChangeTime – the date and time when anything about the subscription last changed
- SupplierID – the ID of the supplier who is supplying the catalog
- Format – the format of the catalog
- Description – a description of the catalog

```
<Subscription>
    <InternalID>1234</InternalID>
    <Name xml:lang="en-US">Q2 Prices</Name>
    <Changetime>1999-03-12T18:39:09-08:00</Changetime>
    <SupplierID domain="DUNS">123456789</SupplierID>
    <Format version="2.1">CIF</Format>
    <Description xml:lang="en-US">The best prices for software</Description>
</Subscription>
```

SubscriptionListRequest

This element requests the buyer's current list of catalog subscriptions.

```
<Request>
    <SubscriptionListRequest/>
</Request>
```

SubscriptionListResponse

This element lists the buyer's current list of catalog subscriptions.

```
<Response>
    <Status code="200" text="OK"/>
    <SubscriptionListResponse>
        <Subscription>
            <InternalID>1234</InternalID>
            <Name xml:lang="en-US">Q2 Prices</Name>
```

```
                <Changetime>1999-03-12T18:39:09-08:00</Changetime>
                <SupplierID domain="DUNS">123456789</SupplierID>
                <Format version="2.1">CIF</Format>
                <Description xml:lang="en-US">The best prices for software
                </Description>
            </Subscription>
            <Subscription>
                <InternalID>1235</InternalID>
                <Name xml:lang="en-US">Q2 Software Prices</Name>
                <Changetime>1999-03-12T18:15:00-08:00</Changetime>
                <SupplierID domain="DUNS">555555555</SupplierID>
                <Format version="2.1">CIF</Format>
                <Description xml:lang="en-US">The best prices for software
                </Description>
            </Subscription>
        </SubscriptionListResponse>
</Response>
```

SubscriptionContentRequest

This element requests the contents of a subscribed catalog. The request includes the InternalID and SupplierID for the catalog.

```
<Request>
    <SubscriptionContentRequest>
        <InternalID>1234</InternalID>
        <SupplierID domain="DUNS">123456789</SupplierID>
    </SubscriptionContentRequest>
</Request>
```

SubscriptionContentResponse

This element contains the contents of a catalog. The catalog format can be either CIF (Catalog Interchange Format) or cXML. If it is CIF, it is encoded using base64 and included as the content of a CIFContent element. If it is cXML, the Index and Contract elements are directly included.

```
<Response>
    <Status code="200" text="OK"/>
    <SubscriptionContentResponse>
        <Subscription>
            <InternalID>1234</InternalID>
            <Name xml:lang="en-US">Q2 Prices</Name>
            <Changetime>1999-03-12T18:39:09-08:00</Changetime>
            <SupplierID domain="DUNS">123456789</SupplierID>
            <Format version="3.0">CIF</Format>
            <Description xml:lang="en-US">The best prices for software
            </Description>
```

```
            </Subscription>
            <SubscriptionContent filename="foobar.cif">
                <CIFContent>
                    <!-- base64 encoded data -->
                    ABCDBBDBDBDBDB
                </CIFContent>
            </SubscriptionContent>
        </SubscriptionContentResponse>
</Response>
```

SubscriptionChangeMessage

This element signals the buying organization's procurement system that a subscribed catalog has changed.

```
<Message>
    <SubscriptionChangeMessage type="new">
        <Subscription>
            <InternalID>1234</InternalID>
            <Name xml:lang="en-US">Q2 Prices</Name>
            <Changetime>1999-03-12T18:39:09-08:00</Changetime>
            <SupplierID domain="DUNS">123456789</SupplierID>
            <Format version="2.1">CIF</Format>
        </Subscription>
    </SubscriptionChangeMessage>
</Message>
```

SubscriptionChangeMessage has the following attribute:

type	The type of the change: new, delete, or update.

Message Retrieval Definitions

Some buying organizations do not have HTTP entry points for receiving cXML messages from outside their corporate firewalls. The cXML specification allows for these environments.

This section introduces definitions that allow source systems to queue messages when targets are unable to directly accept HTTP posts. Targets instead pull messages at their convenience.

GetPendingRequest

This element pulls a set of messages waiting for the requester. The MessageType element and the lastReceivedTimestamp and maxMessages attributes control the type and count of the fetched messages.

lastReceivedTimestamp (optional)	The timestamp of the most recent message received.
maxMessages (optional)	Maximum number of messages in a single response that the requester can handle.

Upon receiving the request, the receiver returns the oldest messages, of the specified types, with timestamps equal to or later than the specified timestamp. If there are multiple messages meeting this criterion, multiple messages can be returned, subject to the maxMessages attribute. The queuing system discards all pending messages of the specified message types with timestamps earlier than the specified timestamp.

```
<Request>
    <GetPendingRequest lastReceivedTimestamp="1999-03-12T18:39:09-08:00"
            maxMessages="5">
        <MessageType>SubscriptionChangedMessage</MessageType>
    </GetPendingRequest>
</Request>
```

GetPendingResponse

This element contains one or more messages waiting for the requester.

```
<Response>
    <Status code="200" text="OK"/>
    <GetPendingResponse>
        <cXML version="1.1.007" xml:lang="en-US"
                payloadID="456778@ariba.com"
                timestamp="1999-03-12T18:39:09-08:00">
```

```
                    <Header>
                        <From>
                            <Credential domain="AribaNetworkUserId">
                                <Identity>admin@ariba.com</Identity>
                            </Credential>
                        </From>
                        <To>
                            <Credential domain="AribaNetworkUserId">
                                <Identity>admin@acme.com</Identity>
                            </Credential>
                        </To>
                        <Sender>
                            <Credential domain="AribaNetworkUserId">
                                <Identity>admin@ariba.com</Identity>
                            </Credential>
                        <UserAgent>Ariba.com</UserAgent>
                        </Sender>
                    </Header>
                    <Message>
                        <SubscriptionChangeMessage type="new">
                            <Subscription>
                                <InternalID>1234</InternalID>
                                <Name xml:lang="en-US">Q2 Prices</Name>
                                <Changetime>1999-03-12T18:39:09-08:00
                                </Changetime>
                                <SupplierID domain="DUNS">123456789
                                </SupplierID>
                                <Format version="2.1">CIF</Format>
                            </Subscription>
                        </SubscriptionChangeMessage>
                    </Message>
                </cXML>
            </GetPendingResponse>
        </Response>
```

Appendix B
New Features in cXML 1.1

cXML 1.1 contains the following categories of new features:

- General Changes to cXML
- Changes to Extrinsics
- Punchout Transaction Improvements
- New Purchase Order Features
- New Purchase Order Status Transaction

For in-depth discussions of any element or attribute mentioned here, see Appendix A, “cXML Language Specification.”

General Changes to cXML

The following changes affect large areas of the cXML language. These improvements make cXML easier to use internationally, and address cXML version compatibility. In addition, a new Profile transaction allows cXML clients to query cXML server capabilities.

Improved Multi-Language Support

For consistency and better multi-language support, the cXML, Status and ManufacturerName elements now have an optional xml:lang attribute.

For more information:

“Status” on page 54.

Example:

```
<Status
      xml:lang="en-US"
      code="200
      text="OK">
</Status>
```

This attribute specifies the language that cXML clients should use in responses, and that Punchout Websites should display to users.

Centralized DTDs

Previously, there was no declared central location for cXML DTDs, so cXML parsers could not automatically retrieve them. Now, DTDs for all versions of cXML are available at consistent locations on cxml.org.

For cXML DTDs, go to:

http://xml.cXML.org/schemas/cXML/<*version*>/cXML.dtd

For more information:

"Validation Against DTDs" on page 7.

where <*version*> is the full cXML version number, such as 1.1.007.

For best performance, cXML clients should not fetch DTDs each time they parse cXML documents. Instead, they should cache them locally. After populating a URL below //xml.cxml.org/schemas/cXML, there is no need to change that location. DTDs will never be changed in-place; instead, new branches will be added.

New Profile Transaction

A new Profile transaction communicates basic information about cXML servers. This transaction consists of two new documents named ProfileRequest and ProfileResponse. This transaction retrieves server capabilities, including supported cXML version, supported transactions, and options to those transactions.

For more information:

"Profile Transaction" on page 63

Note: All cXML 1.1 servers **must** support this transaction.

Clients can also use the Profile transaction to "ping" servers to verify that they are available.

ProfileRequest

The ProfileRequest document has no content. It is simply routed to the appropriate cXML server using the Header credentials.

```
<cXML version="1.1.007" payloadID="9949494"
      xml:lang="en-US" timestamp="2000-03-12T18:39:09-08:00">
      <Header>
            Routing, identification, and authentication information.
      </Header>
      <ProfileRequest />
</cXML>
```

The server responds with a ProfileResponse document, described below.

ProfileResponse

The ProfileResponse document lists transactions supported by the cXML server, their locations, and any named options with a string value.

```
<ProfileResponse effectiveDate="2000-01-01T05:24:29-08:00">
    <Transaction requestName="OrderRequest">
        <URL>http://workchairs.com/cgi/orders.cgi</URL>
    </Transaction>
    <Transaction requestName="PunchOutSetupRequest">
        <URL>http://workchairs.com/cgi/PunchOut.cgi</URL>
    </Transaction>
</ProfileResponse>
```

New Status Codes

For more information:

"Status" on page 54

The cXML 1.1 specification includes new transaction status codes for more accurate client-server communication. In addition, the specification contains better descriptions of existing HTTP and cXML status codes.

New type Attribute for Marketplace Members

The Credential element has a new type attribute, which specifies whether the sender or receiver is a member of a marketplace. There can be multiple marketplaces, and each marketplace can have different requirements for credentials.

For more information:

"Credential" on page 53

The new type attribute has one possible value: marketplace. Use it to differentiate credentials for marketplace members from credentials for regular buyer or supplier companies. Credential elements without a type attribute identify companies that are not associated with a marketplace.

Requests to or from a marketplace must identify both the marketplace and the member company in To or From Credential elements.

Changes to Extrinsics

cXML 1.1 introduces additional elements and attributes for data previously contained in Extrinsic elements. These additions move information that was previously sent using Extrinsic into the base specification.

cXML 1.1 also introduces support for Extrinsic at the header level.

New Contact Element

The OrderRequestHeader, PunchOutSetupRequest, and ItemOut elements can now contain optional Contact elements, which list a person or group to contact for additional information.

For more information:

“Contact” on page 68

```
<ItemOut quantity="1">
    <ItemID>
        <SupplierPartID>5555</SupplierPartID>
    </ItemID>
    <ItemDetail>
        <UnitPrice>
            <Money currency="USD">134.00</Money>
        </UnitPrice>
        <Description xml:lang="en">
            <ShortName>Office Chair</ShortName>
            Black leather, with adjustable arms, adjustable height and back angle.
        </Description>
        <UnitOfMeasure>EA</UnitOfMeasure>
        <Classification domain="UNSPSC">12345</Classification>
    </ItemDetail>
    <Contact role="customerService">
        <Address>
            <Name xml:lang="en-US">Joe Bob Emmet</Name>
            <Email>joebob@workchairs.com</Email>
            <Phone name="Office">
                <TelephoneNumber>
                    <CountryCode isoCountryCode="US">1</CountryCode>
                    <AreaOrCityCode>800</AreaOrCityCode>
                    <Number>5551212</Number>
                </TelephoneNumber>
            </Phone>
            <Fax name="Order">
                <TelephoneNumber>
                    <CountryCode isoCountryCode="US">1</CountryCode>
                    <AreaOrCityCode>408</AreaOrCityCode>
                    <Number>5551234</Number>
                </TelephoneNumber>
            </Fax>
        </Address>
    </Contact>
</ItemOut>
```

The role attribute specifies the position or title of the contact person. Allowed roles are endUser, administrator, purchasingAgent, technicalSupport, customerService or sales.

requisitionID Attribute Supported

requisitionID is now fully supported. It is an optional attribute to the OrderRequest and ItemOut elements that identifies the buyer's requisition for a line item.

For more information:

"OrderRequestHeader" on page 65

```
<OrderRequest>
    <OrderRequestHeader
            orderID="DO1234"
            orderDate="2000-03-12T13:30:23+8.00"
            type="new"
            requisitionID="R4321">
        <Total>
            <Money currency="USD">12.34</Money>
        </Total>
        <ShipTo>
        . . .
        </ShipTo>
    </OrderRequestHeader>
    <ItemOut>
    . . .
    </ItemOut>
</OrderRequest>
```

Do not include requisitionID at the ItemOut level if you use it at the OrderRequestHeader level.

Summary of Moved Extrinsic Information

The following table lists the changes made to commonly used Extrinsic elements.

Old Extrinsic	Existing or new cXML element or attribute
Requested Ship Date	ItemOut requestedDeliveryDate attribute
ship complete	new shipComplete attribute
ReqNumber	existing requisitionID attribute
Requisition #	existing requisitionID attribute
Name	new Contact element
Phone	new Contact element
E-mailAddress	new Contact element
Buyer Name	new Contact element

Old Extrinsic	Existing or new cXML element or attribute
Buyer Phone	new Contact element
OriginalRequester	new Contact element
Requester Phone Number	new Contact element
ETA	existing requestedDeliveryDate attribute

Header-Level Extrinsics

Previously, purchase order Extrinsic elements could appear only at the product line-item level. Now, they can appear anywhere within the OrderRequest document.

Use this new capability for Extrinsic data that applies to the entire purchase order.

Extrinic elements with the same name cannot appear at both the header level and the line-item level in an OrderRequest document.

Punchout Transaction Improvements

Punchout transactions have been improved for better support of aisle-level and product-level shopping. cXML also now supports cancelled punchout sessions.

Improved PunchOutSetupRequest

The PunchOutSetupRequest document sent by buyers has been changed to improve the flexibility of punchout transactions. The URL specified in the PunchOutSetupRequest has been deprecated; cXML servers will ignore this element in the future.

The new methodology uses the identity (from the Credential) of the supplier. E-commerce network hubs receive the PunchOutSetupRequest document, read the supplier's ID, find the URL of the punchout Website from the supplier's account information, and send the PunchOutSetupRequest document to that URL. The e-commerce network hub, not the buyer, specifies the URL of the punchout Website, which is more flexible.

E-commerce network hubs allow suppliers to store the URLs of their punchout Websites.

SelectedItem Element

As part of the PunchOutSetupRequest enhancement, cXML now has better support for store-, aisle-, and product-level punchout. A new optional SelectedItem element in this document allows suppliers to specify punchout for an entire store or any subset of product offerings. Procurement applications can include the SelectedItem element in PunchOutSetupRequest documents, and punchout sites can use it to determine which products to display to users. If there is no SelectedItem, suppliers should present their entire (store-level) product offerings.

For more information:

"SelectedItem" on page 75

A SelectedItem contains an ItemOut element, which contains an ItemID, for example:

```
<SelectedItem>
    <ItemID>
        <SupplierPartID>5555</SupplierPartID>
    </ItemID>
</SelectedItem>
```

For the contents of the SelectedItem element, procurement applications use the ItemID (SupplierPartID and SupplierPartAuxiliaryID) from the punchout index catalog. No catalog changes are required.

Procurement applications should initially send both the new SelectedItem element and the old punchout URL in the PunchOutSetupRequest. E-commerce network hubs use the old URL only for suppliers that have not yet stored their punchout URL destinations.

Empty PunchOutOrderMessage

cXML now allows empty PunchOutOrderMessage documents, which allows users to end punchout shopping sessions without picking any items. Previously, the returned PunchOutOrderMessage had to contain at least one item.

Suppliers can implement a "Cancel" button that generates an empty PunchOutOrderMessage document. Then, both the punchout site and the procurement application know when a user has canceled a shopping session, and they can delete the shopping cart, delete items from the requisition, and perform other housekeeping tasks.

New cXML-base64 Hidden Field

The new cXML-base64 hidden field supports international documents within the FORM POST returned by punchout sites. cXML documents containing symbols outside of "us-ascii" should use this field instead of the cXML-urlencoded hidden field. This

alternative has almost identical semantics, but the entire document is base64 encoded throughout transport and not HTML encoded to the browser, nor URL encoded to the receiving Web server.

For more information:

"URL-Form-Encoding" on page 59

Base64 encoding maintains the original character encoding of a cXML document. Although no "charset" parameter arrives with the posted information, the decoded document (after the transfer encoding is removed) can be treated as the media type "application/xml". This treatment allows the receiving parser to honor any "encoding" attribute specified in the XML declaration. For this field, and all "application/xml" documents, the default character encoding is UTF-8.

New Purchase Order Features

cXML purchase order documents have been enhanced to support requested features.

New lineNumber Attribute

lineNumber is a new optional attribute to the ItemOut element. It specifies the location of an item within the buyer's purchase order. It is chosen as the order is placed (so, it is usually not relevant to punchout sessions).

For more information:

"ItemOut" on page 70

```
<ItemOut quantity="2" lineNumber="1"
        requestedDeliveryDate="2000-03-12">
    <ItemID>
        <SupplierPartID>11223344</SupplierPartID>
    </ItemID>
    <ItemDetail>
    . . .
    </ItemDetail>
</ItemOut>
```

Line numbers for items must remain constant through change orders, to identify the change.

Purchase Order Attachments

Buyers often need to clarify purchase orders with memos, drawings, or faxes. Procurement applications can now attach files of any type to cXML purchase orders by using MIME (Multipurpose Internet Mail Extensions).

cXML contains only references to external MIME parts sent within one multipart MIME envelope (with the cXML document, in an e-mail or faxed together).

A new Attachment element contains references to the attachments:

For more information:

"Attachment Transmission" on page 51

```
<Comments>
    <Attachment><URL>cid: uniqueCID@buyer.com</Attachment>
    Please see attached image for my idea of what this should look like
</Comments>
```

E-commerce network hubs receive the attachments, and they can forward them to the supplier or store them for online retrieval.

For more information about the MIME standard, see the following Websites:

www.hunnysoft.com/mime
www.rad.com/networks/1995/mime/mime.htm

New shipComplete Attribute

shipComplete is a new optional attribute to the OrderRequestHeader element that instructs suppliers to fill an order only when all items are available. This attribute prevents partial item shipments.

For more information:

"OrderRequestHeader" on page 65

```
<OrderRequest>
    <OrderRequestHeader
            orderID="DO1234"
            orderDate="2000-03-12T13:30:23+8.00"
            type="new"
            requisitionID="R4321"
            shipComplete="yes">
        <Total>
            <Money currency="USD">12.34</Money>
        </Total>
        <ShipTo>
        . . .
        </ShipTo>
    </OrderRequestHeader>
    <ItemOut>
    . . .
    </ItemOut>
</OrderRequest>
```

New ShortName Element

ShortName is a new, optional element within Item Description elements.

```
<Description xml:lang="en-US">
    <ShortName>Big Computer</ShortName>
```

```
        This wonder contains three really big disks, four CD-ROM drives, two Zip drives, an
        Ethernet card, much more memory than you could ever use, four CPUs on two
        motherboards.  We'll throw in two monitors, a keyboard and the cheapest mouse we
        can find lying around.
    </Description>
```

For more information:

"ItemDetail" on page 80

ShortName is a short (30-character recommended, 50-character maximum) name for the item, which fits product lists presented to users. Previously, because there was only a Description element, long descriptions were truncated at unknown points.

Procurement applications and other cXML clients should display ShortName instead of a truncation of the Description text in any space-restricted fields. If no ShortName is provided, cXML clients can continue to truncate the Description text.

Catalog creators should not use ShortName to duplicate the information in Description. Instead, they should use ShortName to name the product, and Description to describe product details.

CIF 3.0 catalog format has also been enhanced to support ShortName. The CIF field name is Short Name.

New Purchase Order Status Transaction

cXML 1.1 introduces a new transaction for sending the status of purchase orders to e-commerce network hubs.

This transaction relies upon the new DocumentReference element, which associates the status update with the OrderRequest most recently received from the buyer.

New DocumentReference Element

The new DocumentReference element associates a status update with a particular OrderRequest document.

For more information:

"DocumentReference" on page 81

```
    <DocumentReference
        payloadID="0c300508b7863dcclb_14999"
    </DocumentReference>
```

The new StatusUpdateRequest transaction uses this element.

New StatusUpdateRequest Transaction

Order-processing partners (such as fax or EDI service providers) send the new StatusUpdateRequest transaction to e-commerce network hubs to set purchase order status. It affects the order status indicator on the hub, which is visible to both buyers and suppliers. Additionally, suppliers can send this transaction to allow buyers to see the status of document processing within the supplier's organization.

For more information:

"StatusUpdateRequest" on page 82

> **Note:** This transaction informs interested parties about changes in the delivery and processing status of *purchase order documents*, not the shipping status of actual items.

One change is of particular significance: When an intermediate hub successfully forwards an OrderRequest document, it can inform the original sender or a previous hub about that success. Transitions through various queues and processing steps at a supplier or hub might also be significant to the buyer.

New Followup Element

Followup is an element within OrderRequestHeader that specifies the URL to which future StatusUpdateRequest documents should be posted. This location is the input location for any later documents which reference the current OrderRequest document.

For more information:

"Followup" on page 69

Index

L

M

O

P

Q

R

S

T

U

V

X

www.cxml.org