2026-02-14

# Getting Started with cXML

# Content

# cXML License Agreement

IMPORTANT: PLEASE CAREFULLY READ THIS cXML LICENSE AGREEMENT ("LICENSE") BEFORE USING THE cXML SPECIFICATION ("SPECIFICATION"). BY USING THE SPECIFICATION, YOU AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, DO NOT USE OR ACCESS THE SPECIFICATION. Licensor may publish new versions (including revisions) of this Agreement from time to time on the cXML site (www.cxml.org). The rights granted under this license with respect to the Specification are subject to the version of the Agreement in effect at the time it was downloaded or accessed by you.

1. Openness. cXML is designed and intended to be an open standard to facilitate electronic commerce. You are welcome to use and adopt this standard, and to submit comments, recommendations, and suggestions to cXML.org. Once submitted, your comments go through an approval process - and your comments may ultimately be incorporated into cXML.
2. License. Subject to the terms and conditions herein, Licensor hereby grants to you a perpetual, nonexclusive, royalty-free, worldwide right and license to use the Specification under the Licensor intellectual property necessary to implement the Specification to (a) use, copy, publish, and distribute (including but not limited to distribution as part of a separate computer program) the unmodified Specification, and (b) to implement and use the Specification, including the cXML tags and schema guidelines included in the Specification for the purpose of creating, distributing, selling or otherwise transferring computer programs that adhere to such guidelines. If you use, publish, or distribute the unmodified Specification, you may call it "cXML".
3. Restrictions. Your rights under this License will terminate automatically without notice from Licensor if you fail to comply with any terms of this License.
4. Licensor expressly reserves all other rights it may have in the material and subject matter of the Specification, and you acknowledge and agree that Licensor owns all right, title, and interest in and to the Specification, however, Licensor does not own the computer programs or related documentation you create, nor does Licensor own the underlying XML or non-Ariba intellectual property from which cXML has been derived. You agree to not assert any intellectual property rights that would be necessarily infringed by implementation or other use of the Specification against Licensor or any other entity with respect to such implementation or other use of the Specification; provided that your agreement to not assert shall cease to apply to any entity including Licensor (except where Licensor or another entity is asserting intellectual property rights against you as part of an assertion that you have breached this Agreement) that asserts against you that its intellectual property rights are infringed by your implementation or other use of the Specification. If you publish, copy or distribute the Specification, then this License must be attached. If you submit any comments or suggestions to Licensor, and Licensor modifies the Specification based on your input, Licensor shall own the modified version of the Specification.
5. No Warranty. YOU ACKNOWLEDGE AND AGREE THAT ANY USE OF THE SPECIFICATION BY YOU IS AT YOUR OWN RISK. THE SPECIFICATION IS PROVIDED FOR USE "AS IS" WITHOUT WARRANTY OF ANY KIND. LICENSOR AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES OF ANY KIND, INCLUDING BUT NOT LIMITED TO ANY EXPRESS WARRANTIES, STATUTORY WARRANTIES, AND ANY IMPLIED WARRANTIES OF: MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMNT. YOUR SOLE AND EXCLUSIVE REMEDY RELATING TO YOUR USE OF THE SPECIFICATION SHALL BE TO DISCONTINUE USING THE SPECIFICATION.
6. Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY LAW, UNDER NO CIRCUMSTANCES SHALL LICENSOR BE LIABLE FOR ANY DAMAGES WHATSOEVER RELATING TO THIS LICENSE OR YOUR

USE OF THE SPECIFICATION (INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL, PUNITIVE, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES), REGARDLESS OF WHETHER A CLAIM IS BASED ON TORT, CONTRACT, OR OTHER THEORY OF LIABILITY, AND EVEN IF LICENSOR IS ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. To the extent your jurisdiction does not allow any of the above exclusions of damages, in such case you agree that Licensor's total liability to you for all damages under this License shall not exceed the amount of ten dollars ($10.00). 6. Government End Users. If the Specification is supplied to the United States Government, the Specification is classified as "restricted computer software" as defined in clause 52.227-19 of the FAR. The United States Government's rights to the Specification are as provided in clause 52.227-19 of the FAR.

7. This License shall be deemed to have been made in, and shall be construed pursuant to the laws of the State of California and the federal U.S. laws applicable therein, excluding its conflict of laws provisions. Any legal action or proceeding relating to this License shall be instituted in a state or federal court in San Francisco, Santa Clara or San Mateo County, California, and each party hereby consents to personal jurisdiction in such counties. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, the remainder of this License shall continue in full force and effect.
8. You assume the entire risk resulting from your use of the Specification.
9. Complete Agreement. This License is the complete and exclusive statement, and an absolute integration of the mutual understanding of the parties and supersedes and cancels all previous written and oral agreements and communications relating to the subject matter of this License. You acknowledge that any material breach by you of the provisions of the License will cause irreparable damage to Licensor and that a remedy at law will be inadequate. Therefore, in addition to any and all other legal or equitable remedies, Licensor will be entitled to seek injunctive relief necessary to remediate the breach of this License. Ariba, Inc. shall be deemed the Licensor.
10. Notices. Any notice directed to Licensor must be sent in writing to comments@cxml.org.

7-19-04

# 1 Preface

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



## 1.1 Audience and Prerequisites

This document is intended for application developers who design cXML-enabled applications.

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

- E-commerce network hubs
- Electronic product catalogs
- PunchOut catalogs
- Procurement applications
- Buyers
- Suppliers
- E-commerce service providers

Readers should have a working knowledge of e-commerce concepts, the HTTP Internet communication standard, and XML format.

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

## 1.2 Typography

cXML elements and attributes are denoted with a monotype font. cXML element and attribute names are case-sensitive. Both are a combination of lower and uppercase, with elements beginning with an uppercase letter, and attributes beginning with a lowercase letter. For example, `MyElement` is a cXML element, and `myAttribute` is a cXML attribute.

The following table describes the typographic conventions used in this book:

| Typeface or Symbol | Meaning | Example |
|---|---|---|
| *`AaBbCc123`* | Text you need to change is italicized. | http://*`server`*:*`port`*/inspector |
| **AaBbCc123** | The names of user interface controls, menus, and menu items | Choose **Edit** from the **File** menu. |

| Typeface or Symbol | Meaning | Example |
| --- | --- | --- |
| `AaBbCc123` | Files and directory names, parameters, fields in CSV files, command lines, and code examples | A `ProfileRequest` document is sent from a buyer to the network. |
| *AaBbCc123* | Book titles | For more information, see *Acme Configuration Overview*. |

# 2 Introduction to cXML

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



## 2.1 cXML, an XML Implementation

XML (eXtensible Markup Language) is a meta-markup language used to create syntaxes for languages. It is also a standard for passing data between applications, particularly those that communicate across the Internet.

XML documents contain data in the form of tag/value pairs, for example:

```
<DeliverTo>Joe Smith</DeliverTo>
```

XML has a structure similar to HTML (HyperText Markup Language), which is an implementation of SGML, XML's parent meta language. Applications can extract and use data from XML documents more easily than from HTML documents, however, because XML data is tagged according to its purpose. XML contains only data, while HTML contains both data and presentation information.

Each cXML document is constructed based on XML Document Type Definitions (DTDs). Acting as templates, DTDs define the content model of a cXML document, for example, the valid order and nesting of elements, and the data types of attributes.

The DTDs for cXML are files available on the www.cXML.org website.

## 2.2 cXML Capabilities

cXML allows buying organizations, suppliers, service providers, 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 a well-defined, robust language designed specifically for B2B e-commerce, and it is the choice of high volume buying organizations and suppliers.

cXML transactions consist of documents, which are simple text files containing values enclosed by predefined tags. Most types of cXML documents are analogous to hardcopy documents traditionally used in business.

The most commonly used types of cXML documents are:

- Catalogs [page 9]
- PunchOut [page 10]
- Purchase Orders [page 11]

The following subsections describe these cXML documents.

## 2.2.1 Catalogs

Catalogs are files that convey product and service content to buying organizations. They describe the products and services offered by a supplier and their prices, and they are the main communication channel from suppliers to their buyers.

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

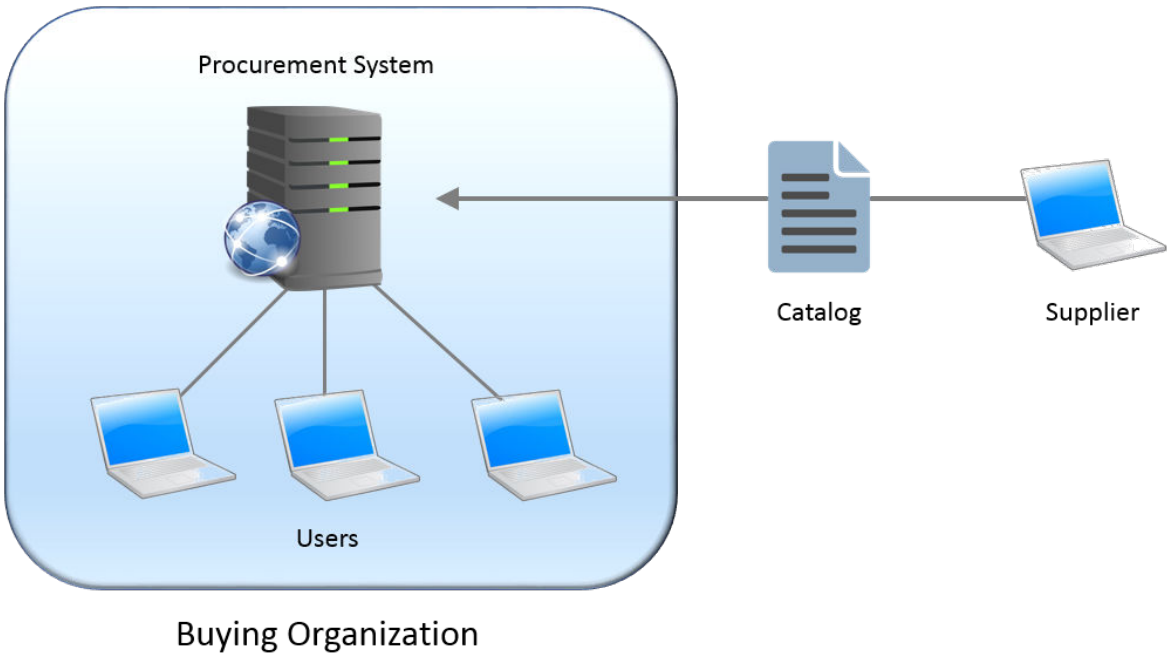



Figure 1: Sending Product and Service Content to a Buying Organization

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

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

## 2.2.2 PunchOut

PunchOut is an easy-to-implement protocol for interactive sessions managed across the Internet. Using real-time, synchronous cXML messages, PunchOut enables communication between applications, providing seamless user interaction at remote sites.

There are three types of PunchOut:

- Procurement PunchOut [page 10]
- PunchOut Chaining [page 11]
- Provider PunchOut [page 11]

### Procurement PunchOut

Procurement PunchOut gives suppliers an alternative to static catalog files. PunchOut sites are live, interactive catalogs running on a website.

Suppliers that have e-commerce websites can modify them to support PunchOut. PunchOut sites communicate with procurement systems over the Internet by using cXML.

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 the supplier's local website. Depending on how the supplier implements these 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.

The following diagram shows an interactive PunchOut session between a user and a supplier web site.

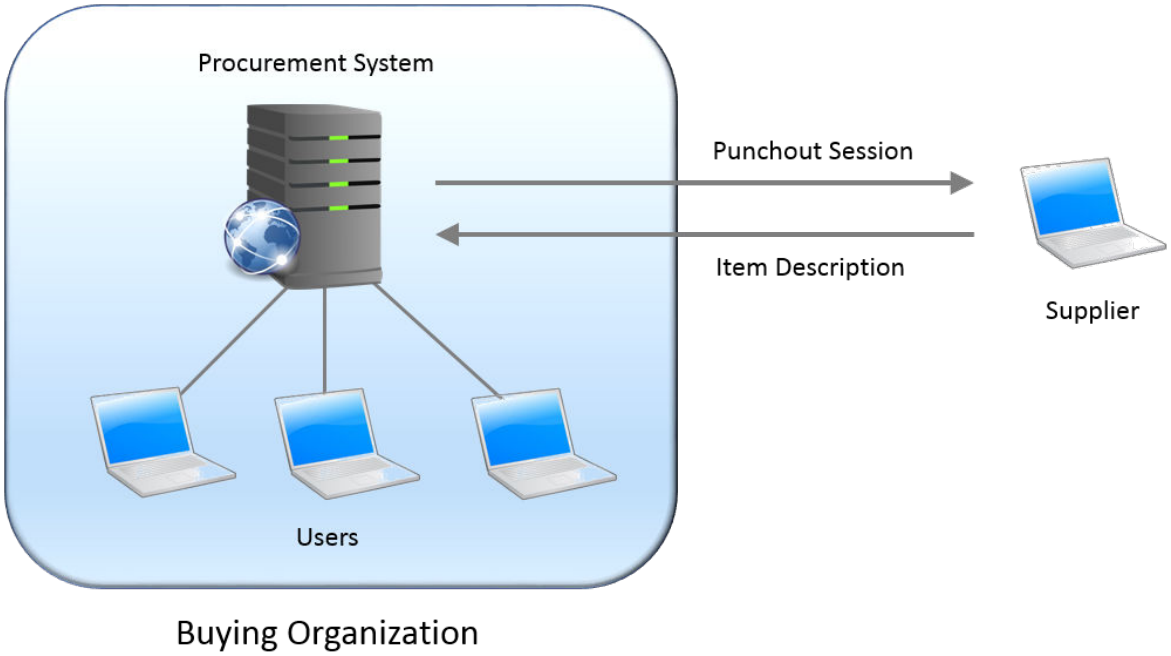



Figure 2: Interactive PunchOut Session Between a User and a Supplier Website

Suppliers' websites can offer previously agreed-upon contract products and prices.

### PunchOut Chaining

PunchOut chaining is Procurement PunchOut that involves more than one PunchOut. cXML Path Routing enables this functionality.


Figure 3: PunchOut Chaining

cXML Path Routing allows the order and other subsequent messages to return to the marketplaces and suppliers involved in producing the quote. Path Routing notifies all parties about the final order, and any subsequent PunchOut specifies to the procurement application how to split orders on behalf of the marketplace.

### Provider PunchOut

Provider PunchOut enables applications to punch out to a remote applications that supply services to the originating application, such as credit card validation, user authentication, or self-registration.

## 2.2.3 Purchase Orders

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

The following diagram shows a purchase order communicated to a supplier:

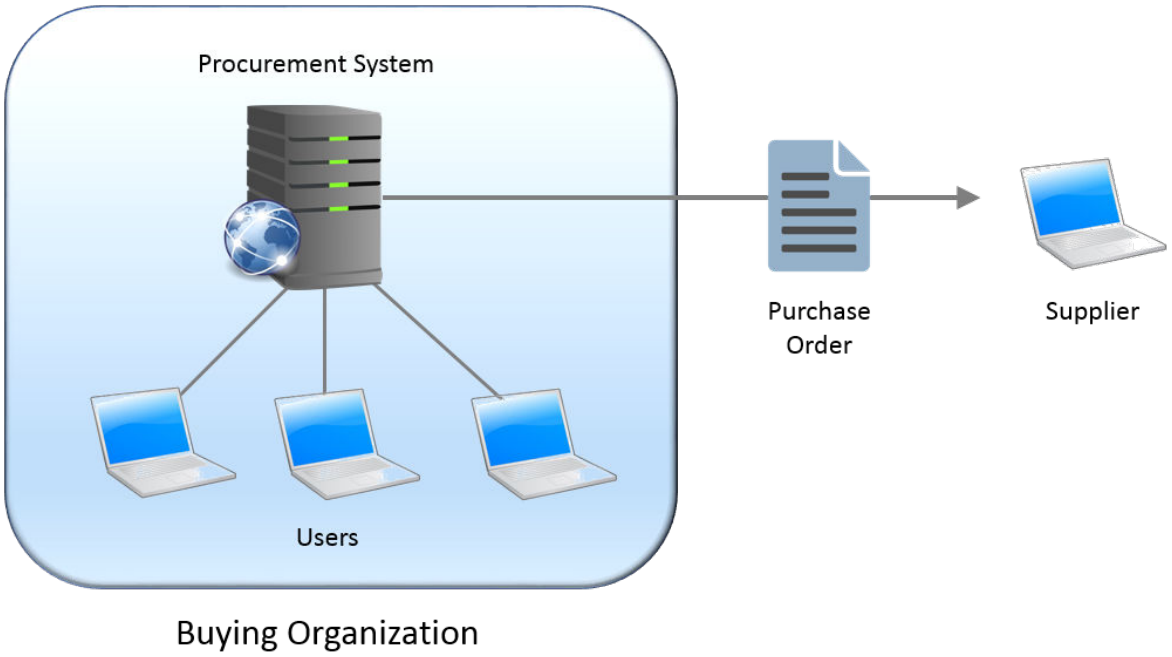



Figure 4: Purchase Order Communicated to a Supplier

cXML is better for communicating purchase orders than other formats (such as ANSI X12 EDI 850), because it is flexible, inexpensive to implement, and it supports the widest array of data and attachments.

## 2.3 Types of Applications that Use cXML

Any e-commerce application can use cXML. Buying organizations, vertical and horizontal buying communities, suppliers, and application vendors currently use cXML. The following subsections describe the main types of applications that currently use cXML.

### Procurement Applications

Procurement applications, such as SAP Ariba Buying, SAP Ariba Buying and Invoicing, and Ariba Buyer, use cXML for external transactions.

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

### Network Hubs

Network hubs, such as SAP Business Network, are Web-based services for connecting buyers and suppliers. These Web services provide features such as catalog validation and versioning, catalog publishing and subscription, automated purchase order routing, and purchase order history.

Network hubs can act as intermediaries that authenticate and route requests and responses to and from diverse organizations. Communication between these organizations can occur entirely through cXML over the Internet.

### PunchOut Catalogs

As described in the previous section, PunchOut catalogs are interactive catalogs, available at supplier websites. PunchOut catalogs are made possible by Web server applications, written in a programming language such as ASP (Active Server Pages), JavaScript, or CGI (Common Gateway Interface), that manage buyers' PunchOut sessions.

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.

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

## 2.4 Content Delivery Strategy

Procurement applications present product and service content to users. Suppliers want to control the way their buyers 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.

| Commodities | Properties | Content Delivery Method |
|---|---|---|
| Office Supplies,<br>Internal Supplies | Static content, stable pricing | Static catalogs |
| Lab Supplies,<br>MRO (Maintenance, Repair, and Operations),<br>Electronic Parts | Requires normalization to be useful | PunchOut to a vertical commodity portal |
| Books,<br>Chemicals | Large number of line Items | PunchOut to a supplier hosted site |
| Computers,<br>Network Equipment,<br>Peripherals | Many possible configurations | PunchOut to a supplier hosted configuration tool |
| Services,<br>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 this table indicates, PunchOut offers a flexible framework upon which suppliers, depending on their commodity or buyer, can provide customized content. The objective of this content strategy is to allow buyers and suppliers to exchange catalog data by the method that makes the most sense.

## 2.5 cXML DTDs

Because cXML is an XML language, it is thoroughly defined by a set of Document Type Definitions (DTDs). 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.

The header of each cXML document contains the URL to the DTD that defines the document. cXML applications can retrieve the DTD and use it to validate the document.

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. cXML applications are not required to validate cXML documents received, although it is recommended. However, all cXML documents must be valid and must refer to the cXML DTDs described in the following section.

### Getting cXML DTDs

DTDs for all versions of cXML are available on cXML.org. The various kinds of cXML documents are defined in multiple DTDs to reduce DTD size, which enables faster validation in some parsers.

| Document | DTD |
|---|---|
| Basic cXML documents | http://xml.cXML.org/schemas/cXML/*version*/cXML.dtd |
| Confirmation and Ship Notice | http://xml.cXML.org/schemas/cXML/*version*/Fulfill.dtd |
| Invoice | http://xml.cXML.org/schemas/cXML/*version*/InvoiceDetail.dtd |
| Type Definition | http://xml.cXML.org/schemas/cXML/*version*/Catalog.dtd |
| Payment Remittance | http://xml.cXML.org/schemas/cXML/*version*/PaymentRemittance.dtd |
| Request for Quotations | http://xml.cXML.org/schemas/cXML/*version*/Quote.dtd |
| Contracts | http://xml.cXML.org/schemas/cXML/*version*/Contract.dtd |
| Logistics | http://xml.cXML.org/schemas/cXML/*version*/Logistics.dtd |

where *version* is the full cXML version number.

cXML applications use these DTDs to validate all incoming and outgoing documents.

### Caching DTDs

For best performance, cXML applications should cache DTDs locally. After cXML DTD files are published, they never change, so you can cache them indefinitely. (Each new version of the DTDs has a new URL.) When cXML applications parse a cXML document, they should look at the SYSTEM identifier in the document header and retrieve that DTD if it has not already been stored locally.

Caching DTDs locally offers the advantages of faster document validation and less dependence on the cXML.org site.

In some environments, cXML applications might not be allowed to automatically retrieve DTDs as they receive new documents. In these environments, you must manually retrieve the DTDs, store them locally, and instruct your applications to look for them locally, not at cXML.org. However, generated cXML documents must point to the DTDs at cXML.org, not the local DTDs.

## 2.6 Profile Transaction

The Profile transaction communicates basic information about what transactions a particular cXML server can receive. All cXML servers must support this transaction. It is intended for backend integrations between applications, making the capabilities of cXML servers available to client systems.

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 and higher servers must accept the Profile transaction.

### 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 the cXML transactions it supports, their locations, and any named options with a string value.

## 2.7 Service Status Response

A response with a status code of 200 from an URL that accepts POSTed cXML is up and running. When an HTTP GET is sent to a service location, the service responds with a valid, dynamically generated cXML Response document. A service can be any HTTP URL at which cXML Request documents are received.

## 2.8 XML Utilities

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

- **Internet Explorer** from Microsoft. An XML-aware Web browser that can validate XML files against DTDs. www.microsoft.com/windows/ie/default.htm
- **Turbo XML** from TIBCO Software. An Integrated Development Environment (IDE) for creating, validating, converting and managing XML assets. www.tibco.com/software/metadata/turboxml.jsp
- **XML Spy** from Altova. A tool for maintaining DTDs and XML files with a grid, source and browser view. www.altova.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.xml.com
- http://www.ibm.com/developerworks/xml/

# 3 cXML Basics

This section describes the basic protocol and data formats of cXML. It contains information needed to implement all transactions.



## 3.1 Protocol Specification

There are two communication models for cXML transactions: Request-Response and One-Way. Because these two models strictly specify the operations, they enable simple implementation. Both models are required, because there are situations when one model would not be appropriate.

### 3.1.1 Request-Response Model

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

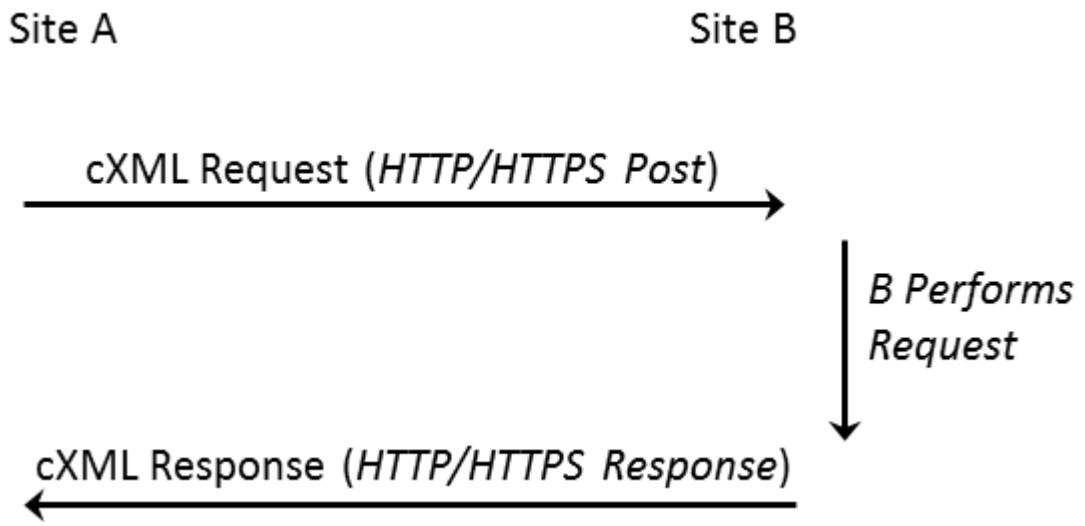



Figure 5: Request-Response Transaction

This transaction contains the following steps:

1. Site A initiates an HTTP/1.x connection with Site B on a predetermined URL that represents Site B's address.

2. Site A uses a POST operation to send the cXML document through the HTTP connection. Site A then waits for a response.
3. Site 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 Site B's HTTP server, for example, a CGI program or an ASP page.
4. Site B's resource identified in step 3 reads the cXML document contents and maps the `Request` to the appropriate handler for that request.
5. Site B's handler for the cXML `Request` performs the work that the `Request` specifies and generates a cXML `Response` document.
6. Site B sends the cXML `Response` to Site A through the HTTP connection established in step 1.
7. Site A reads the cXML `Response` and returns it to the process that initiated the `Request`.
8. Site 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
    </Header>
    <Request>
        Request information
    </Request>
</cXML>
```

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

```
<cXML>
    <Response>
        Response information
    </Response>
</cXML>
```

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

## 3.1.2 cXML Conventions

cXML uses elements to describe discrete items, which are properties in traditional business documents. Elements also describe information with obvious subdivisions and relationships between those subdivisions, such as an addresses, which are composed of street, city, and country.

cXML also uses attributes, which modify elements or provide context.

Element and attribute names are case-sensitive and 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:** `payloadID, lineNumber, domain`

If optional elements have no content (they are null), leave them out entirely. Avoid empty or whitespace elements, because missing values can affect some parsers.

In DTD files and in this document, symbols are used to indicate how many times an element can occur in a transaction. A '+' means the element can occur one or more times, a '?' means the element can occur 0 or once, and a '*' means the element can occur 0 or more times.

## 3.1.3 cXML Document

The `cXML` element is the body of a cXML document. A document might begin as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.2.014/cXML.dtd">
<cXML xml:lang="en-US"
    payloadID=”1234567.4567.5678@buyer.com"
    timestamp="2002-01-09T01:36:05-08:00">
```

The first characters in cXML documents must be `<?` or `<!`. Documents must not start with white space or tabs. For example, the HTML form that contains a `PunchOutOrderMessage` document must not insert any character between the opening quote and the left angle bracket.

The second line in cXML documents must contain the DOCTYPE document type declaration. This is the only external entity that can appear in cXML documents. This line references the cXML DTD.

cXML documents can have any one of the following top-level elements: `cXML`, `Supplier`, `Contract`, and `Index`. The `cXML` element is for "transactional" data. The other elements describe static content.

### Related Information

cXML DTDs [page 14]

## 3.1.4 Wrapping Layers

cXML documents are usually transmitted through HTTP with the HTTP header specifying a MIME (Multipurpose Internet Mail Extensions) media type of `text/xml` and a `charset` parameter matching the encoding in the cXML document.

Because HTTP is eight-bit clean, any character encoding supported by the receiving parser can be used without a content-transfer encoding such as `base64` or quoted-printable. All XML parsers support the `UTF-8`

(Universal Transformation Format) encoding, which includes all Unicode characters, including all of US-ASCII. Therefore, applications should use UTF-8 when transmitting cXML documents.

> **Note**
>
> According to IETF 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.` You can also use the application/xml media type, which does not override the XML declaration or affect the recipient's decoding notes, and which does not require the charset parameter.

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"?>
…
```

## 3.1.5 Attachments

The cXML protocol supports the attachment of external files of any type to cXML documents. For example, buyers sometimes need to clarify purchase orders with supporting memos, drawings, or faxes. Another example is the `CatalogUploadRequest` document, which includes catalog files as attachments.

Files referenced by cXML documents can reside either on a server accessible by the receiver or within an envelope that also includes the cXML documents themselves. To attach external files to a cXML document in a single envelope, use Multipurpose Internet Mail Extensions (MIME). The cXML document contains references to external parts sent within a multipart MIME envelope.

### Including Attachments

A cXML requirement for this envelope (over the requirements described in IETF RFC 2046 "Multipurpose Internet Mail Extensions Part Two: Media Types") is the inclusion of `Content-ID` headers with each attached file.

The contained URL must begin with `cid:`, which is the identifier for the referenced attachment within the larger transmission. The `cid:` identifier must match the Content-ID header of one (and only one) part of the MIME transmission containing the document being forwarded.

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"?>
…
   <Attachment>
      <URL>cid:uniqueCID@sender.com</URL>
   </Attachment>
…
--something unique
Content-type: image/jpeg
Content-ID: <uniqueCID@sender.com>
…
--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=<uniqueMainCID@sender.com>
--something unique
Content-type: text/xml; charset="UTF-8"
Content-ID: <uniqueMainCID@sender.com>
<?xml version="1.0" encoding="UTF-8"?>
…
   <Attachment>
      <URL>cid:uniqueAttachmentCID@sender.com</URL>
   </Attachment>
…
--something unique
Content-type: image/jpeg
Content-ID: <uniqueAttachmentCID@sender.com>
…
--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.

## Attachment Examples

The following example shows a `CatalogUploadRequest` with an attached catalog.

```
POST /cXML HTTP/1.0
Content-type: multipart/related; boundary=kdflkajfdksadjfk;
    type="text/xml"; start="<part1.PCO28.975@saturn.workchairs.com>"
<--! begin first MIME body part header -->
--kdflkajfdksadjfk
Content-type: text/xml; charset=UTF-8
Content-ID: <part1.PCO28.975@saturn.workchairs.com>
<--! end first MIME body part header -->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.2.014/cXML.dtd">
<cXML timestamp="2000-12-28T16:56:03-08:00" payloadID="12345666@10.10.83.39">
    <Header>
        <From>
```

```
            <Credential domain="DUNS">
                <Identity>123456789</Identity>
            </Credential>
        </From>
        <To>
            <Credential domain="NetworkID">
                <Identity>AN01000000001</Identity>
            </Credential>
        </To>
        <Sender>
            <Credential domain="DUNS">
                <Identity>123456789</Identity>
                <SharedSecret>abracadabra</SharedSecret>
            </Credential>
        </Sender>
    </Header>
    <Request>
        <CatalogUploadRequest operation="new">
            <CatalogName xml:lang="en">Winter Prices</CatalogName>
            <Description xml:lang="en">premiere-level prices</Description>
            <Attachment>
            <!-- ID of MIME attachment follows -->
                <URL>cid:part2.PCO28.975@saturn.workchairs.com</URL>
            </Attachment>
        </CatalogUploadRequest>
    </Request>
</cXML>
<--! begin second MIME body part header -->
--kdflkajfdksadjfk
Content-type: text/plain; charset=US-ASCII
Content-Disposition: attachment; filename=PremiereCatalog.cif
Content-ID: <part2.PCO28.975@saturn.workchairs.com>
Content-length: 364
<--! end second MIME body part header -->
CIF_I_V3.0
LOADMODE: F
CODEFORMAT: UNSPSC
CURRENCY: USD
SUPPLIERID_DOMAIN: DUNS
ITEMCOUNT: 3
TIMESTAMP: 2001-01-15 15:25:04
DATA
942888710,34A11,C11,"Eames Chair",11116767,400.00,EA,3,"Fast MFG",,,400.00
942888710,56A12,C12,"Eames Ottoman",11116767,100.00,EA,3,"Fast MFG",,,100.00
942888710,78A13,C13,"Folding Chair",11116767,25.95,EA,3,"Fast MFG",,,25.95
ENDOFDATA
<!-- MIME trailer follows -->
--kdflkajfdksadjfk--
```

Surround IDs in `Content-ID` or `Content-Type` headers with angle brackets (< >), but omit these brackets when referring to IDs in `URL` elements. Similarly, prepend message IDs with `cid:` in `URL` elements, but not in MIME headers.

Special characters in cid URLs must be hex encoded (in `%hh` format).

Use the `Attachment` element when attaching text files, PDFs, images, or other such documents to a cXML document. When attaching another cXML document, use `cXMLAttachment`, regardless of whether that cXML document contains attachments itself. The `cXMLAttachment` element serves to alert the receiving system that additional cXML processing might be required to handle the attachment.

The following example shows a `CopyRequest` forwarding a cXML document with attachments using `cXMLAttachment`.

```
Content-Type: Multipart/Related; boundary=outer-boundary
[Other headers]
```

```
--outer-boundary
Content-Type: text/xml; charset=UTF-8
Content-ID: <111@sendercompany.com>
[Other headers]
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cxml.org/schemas/cXML/1.2.014/cXML.dtd">
<cXML payloadID="123@sendercompany.com"
            timestamp="2003-11-20T23:59:45-07:00">
    <Header>
        <From>
            <!-- Sender -->
            <Credential domain="AribaNetworkUserId">
                <Identity>sender@sendercompany.com</Identity>
            </Credential>
        </From>
        <To>
            <!-- Recipient -->
            <Credential domain="AribaNetworkUserId">
                <Identity>recipient@recipientcompany.com</Identity>
            </Credential>
        </To>
        <Sender>
            <!-- Sender -->
            <Credential domain="AribaNetworkUserId">
                <Identity>sender@sendercompany.com</Identity>
                <SharedSecret>abracadabra</SharedSecret>
            </Credential>
            <UserAgent>Sender Application 1.0</UserAgent>
        </Sender>
    </Header>
    <Request deploymentMode="production">
        <CopyRequest>
            <cXMLAttachment>
                <Attachment>
                    <URL>cid:222@sendercompany.com</URL>
                </Attachment>
            </cXMLAttachment>
        </CopyRequest>
    </Request>
</cXML>
--outer-boundary
Content-Type: Multipart/Related; boundary=inner-boundary
Content-ID: <222@sendercompany.com>
[Other headers]
--inner-boundary
Content-Type: text/xml; charset=UTF-8
Content-ID: <333@sendercompany.com>
[Other headers]
[Forwarded cXML]
--inner-boundary
[Attachment 1 of the forwarded cXML]
--inner-boundary
[Attachment 2 of the forwarded cXML]
--inner-boundary--
--outer-boundary--
```

## More Information About MIME

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

- www.hunnysoft.com/mime
- www.ietf.org/rfc1341.txt

- www.ietf.org/rfc/rfc2046.txt
- www.ietf.org/rfc/rfc2387.txt

## 3.1.6 cXML Envelope

The `cXML` element is the root of cXML documents, and it contains all other elements. The `cXML` element is present in every cXML transaction. The following example shows a fully specified `cXML` element:

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

`cXML` has the following attributes:

| Attribute | Description |
|---|---|
| `version` (deprecated) | This attribute was deprecated in cXML 1.2.007; do not use it in new cXML documents.<br>Specifies the version of the cXML protocol. A validating XML parser could also determine the version attribute from the referenced DTD.<br>Because this version number also appears in the SYSTEM identifier in the cXML document, you should omit this attribute. |
| `xml:lang` | 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. Specify the most descriptive and specific locale possible. |
| `payloadID` (required) | 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.<br>The recommended implementation is:<br>datetime.process id.random number@hostname |
| `timestamp` (required) | The date and time the message was sent, in ISO 8601 format. This value should not change for retry attempts.<br>The format is YYYY-MM-DDThh:mm:ss-hh:mm (for example, 2015-07-14T19:20:30+01:00). |
| `signatureVersion` | If present, implies that the document is digitally signed, that is, that the document contains one or more valid `ds:Signature` elements immediately following the `Request`, `Response`, or `Message` element. The only valid value for the attribute is 1.0; other values are reserved for future use. |

### Related Information

cXML Digital Signatures [page 53]

### 3.1.6.1 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 most descriptive and specific locale known should be specified in this attribute.

The `xml:lang` attributes appearing throughout the cXML protocol have no effect on formatted data such as numbers, dates, and times. As described for the timestamp attribute in the following section, 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.

### 3.1.6.2 Date, 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. This is 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 should include 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, `2015-04-14T13:36:00-08:00` corresponds to April 14, 2015, 1:36 p.m., U.S. Pacific Standard Time.

> **Note**
>
> Although the `timestamp` attribute is required by the cXML DTD, validation of the value's format depends on your application.

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

- Microsoft's XML Data Types Reference, msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/b24aafc2-bf1b-4702-bf1c-b7ae3597eb0c.asp
- The original XML Data proposal to the Word Wide Web Consortium (W3C), www.w3c.org/TR/1998/NOTE-XML-data-0105

### 3.1.6.3 Special Characters

In cXML, as in XML, not all characters can be typed from the keyboard, such as the registered trademark symbol (®). Others, such as < and &, have special meaning to XML. These characters must be encoded using character entities.

XML defines the following built-in character entities:

| Entity | Character |
|---|---|
| `<` | < |
| `>` | > |
| `&` | & |
| `&quot;` | “ |
| `&apos;` | ‘ |

For characters outside of the encoding you use, use the Unicode number of the character (decimal or hexadecimal), preceded by pound (#). For example, `&#174;` and `&#xAE;` represent a registered trademark symbol, ®.

For example,

```
<Description xml:lang="en-US">The best prices for software®</Description>
```

could be encoded as

```
<Description xml:lang="en-US">The best prices for software &#174;</Description>
```

Single (') or double (") quotation marks must be escaped only within attribute values that are quoted using that delimiter. It is recommended that you use only single quotes to delimit attributes, unless the content will never contain quotes.

## 3.1.6.3.1 Handling Special Characters in Documents

1. Use a template that only uses single quotes to delimit attributes.
2. Add values to the template by doing one of the following:
   - If the document is a `PunchOutOrderMessage` to be transmitted by the cxml-urlencoded hidden field, fill the values in the template using US-ASCII encoding. This encoding requires XML character entities for all characters beyond that encoding. For example, enter the registered trademark symbol (®), which is not available in US-ASCII, as `&#174;`.
   - Otherwise, fill the values in the document using UTF-8 encoding. UTF-8 should be used for all documents sent by HTTP Post directly, or embedded in a cXML-base64 hidden field. UTF-8 includes all of US-ASCII.
3. XML escape attribute values and element content as you create the cXML document. Characters that must be escaped are &, ', < and >.
   The following steps are required if you are transmitting the document in a `PunchOutOrderMessage`.
4. Pay attention to all characters that browsers interpret:
   - If you are using a cxml-urlencoded hidden field, convert all double quotes to `&#34;`.
   - Further (for the cxml-urlencoded field), escape all ampersands that appear in contexts significant to HTML with `&`. To be safe, you can escape all ampersands. For example, escape ampersand (&) as `&` and apostrophe (') as `&apos;`. Escape the registered trademark symbol (®) as `&#174;`.
   - Otherwise, if you are using a cxml-base64 hidden field, base64 encode the entire cXML document.

5. Embed the document in the HTML form with double quotes around the string value. For example, to send a `Money` element with an attribute having the value `®®'"""&<>>` and containing the value `®®'"""&<>>"`, the XML document might appear as:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Money SYSTEM 'SpecialChars.dtd'>
<Money alternateAmount='&#174;&#xAE;&apos;"&#34;&quot;&amp;lt;&gt;&gt;'>
&#174;&#xAE;&apos;"&#34;&quot;&amp;lt;&gt;&gt;</Money>
```

which should be encoded as follows:

```
<!-- Recommendation for cXML-urlencoding: Uses double quotes to delimit the -->
<!-- field value and single quotes for the contained attributes. -->
<Input type="Hidden" name="cXML-urlencoded" value="<?xml version='1.0'
encoding='UTF-8'?>
<!DOCTYPE Money SYSTEM 'SpecialChars.dtd'>
<Money
alternateAmount='MoneyalternateAmount='&amp;#174;&amp;#xAE;&amp;apos;&#34;&amp;#3
4;
&amp;quot;&amp;amp;&amp;lt;>&amp;gt;'>&amp;#174;&amp;#xAE;'&amp;apos;
&#34;&amp;#34;&amp;quot;&amp;amp;&amp;lt;&amp;gt;&amp;gt;'</Money>">
<!-- Best choice: Base64 encode the value. Don't have to worry about what -->
<!-- the browser interprets. -->
<Input type="Hidden" name="cXML-
base64"value="PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nVVRGLTgnPz4K
PCFET0NUWVBFIE1vbmV5IFNZU1RFTSAnU3BlY2lhbENoYXJzLmR0ZCc+CjxNb
25leSBhbHRlcm5hdGVBbW91bnQ9JyYjMTc0OyYjeEFFOyZhcG9zOyImIzM0OyZxd
W90OyZhbXA7Jmx0Oz4mZ3Q7Jz4KJiMxNzQ7JiN4QUU7JyZhcG9zOyImIzM0OyZx
dW90OyZhbXA7Jmx0Oz4mZ3Q7PC9Nb25leT4K">
```

The preceding examples illustrate alternatives for encoding the cXML-urlencoded field. They avoid XML escaping a few characters, such as angle brackets, that are not special to XML in all contexts. A direct implementation of the previous steps would result in an HTML field such as:

```
<Input type="Hidden" name="cXML-urlencoded" value="<?xml version='1.0'
encoding='UTF-8'?>
<!DOCTYPE Money SYSTEM 'SpecialChars.dtd'>
<Money alternateAmount='&#174;&#174;&apos;"""
&amp;&lt;&gt;&gt;'>&#174;&#174;'"""
&amp;&lt;&gt;&gt;</Money>">
```

or the XML document:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Money SYSTEM 'SpecialChars.dtd'>
<Money alternateAmount='&#174;&#174;&apos;"""&amp;&lt;&gt;&gt;'>
&#174;&#174;'"""&amp;&lt;&gt;&gt;</Money>
```

## 3.1.7 Header

The `Header` element contains addressing and authentication information. The `Header` element is the same regardless of the specific `Request` or `Response` within 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@buyer.com</Identity>
            <SharedSecret>abracadabra</SharedSecret>
        </Credential>
        <UserAgent>Network Hub 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 credential 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`.

When the document is initially sent, `Sender` and `From` are the same, However, if the cXML document travels through network hubs, the `Sender` element changes to indicate current sending party.

### 3.1.7.1 From

This element identifies the originator of the cXML request.

### 3.1.7.2 To

This element identifies the destination of the cXML request.

### 3.1.7.3 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.

### 3.1.7.4 UserAgent

A textual string representing the `UserAgent` who is conducting the cXML conversation. This should be a unique per-product string, and ideally, per-version. Analogous to `UserAgent` for HTTP conversations.

### 3.1.7.5 Credential

This element contains identification and authentication values.

`Credential` has the following attributes:

| Attribute | Description |
|---|---|
| `domain` (required) | Specifies the type of credential. This attribute allows documents to contain multiple types of credentials for multiple authentication domains.<br>For messages sent on SAP Business Network, for instance, the domain can be `AribaNetworkUserId` to indicate an email address, `DUNS` for a D-U-N-S number, or `NetworkId` for a preassigned ID. |
| `type` | 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 a `CredentialMac` element. The `Identity` element states who the `Credential` represents, while the optional authentication elements verify the identity of the party.

#### SharedSecret

The `SharedSecret` element is used when the `Sender` has a password that the requester recognizes.

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

#### CredentialMac

The `CredentialMac` element is used for the Message Authentication Code (MAC) authentication method. This authentication method is used in situations where the sender must prove to the receiver that it has been authenticated by shared secret by a trusted third party. For example, a direct PunchOut request can travel directly from a buyer to a supplier without going through a network hub, because it contains a MAC (generated by the network hub) that allows the supplier to authenticate it.

The trusted third party computes the MAC and transfers it to the sender through the Profile transaction. The MAC is opaque to the sender (it is secure and non-reversible). The MAC is transmitted from the trusted third party to the sender with a `ProfileResponse` object.

The receiver computes the MAC using the same inputs as the trusted third party and compares it with the MAC received in the cXML document. If the two values match, the document is authentic.

To learn how to compute the MAC value, see Message Authentication Code (MAC) [page 45].

`CredentialMac` has the following attributes:

| Attribute | Description |
|---|---|
| `type` (required) | Identifies the data being authenticated and the method in which it is formatted for authentication. The only supported value is `"FromSenderCredentials"`. |
| `algorithm` (required) | Identifies for the MAC algorithm used on the data. The only supported value is `"HMAC-SHA1-96"`. |
| `creationDate` (required) | Specifies the date and time the MAC was generated. |
| `expirationDate` (required) | Specifies the date and time after which this MAC is no longer valid. Receivers must reject MACs that are received after the expirationDate. Receivers can optionally reject unexpired MACs. For example, a receiver might reject MACs that are scheduled to expire in less than an hour. |

The following example shows a `Credential` element that contains a `CredentialMac` element:

```
<Sender>
    <Credential domain="NetworkId">
        <Identity>AN9900000100</Identity>
        <CredentialMac type="FromSenderCredentials"
                    algorithm="HMAC-SHA1-96"
                    creationDate="2003-01-15T08:42:46-0800"
                    expirationDate="2003-01-15T11:42:46-0800">
            MnXkusp8Jj0lw3mf
        </CredentialMac>
        <UserAgent>Procurement Application 8.1</UserAgent>
    </Credential>
</Sender>
```

## Multiple Credentials

The `From`, `To`, and `Sender` elements can each optionally contain multiple `Credential` elements. The purpose of supplying multiple credentials is to identify a single organization using different domains. For example, an organization might be identified by including both a DUNS number and a NetworkId number.

The receiver should validate all credentials with domains it recognizes and it should reject the document if any credentials with recognized domains do not match an organization it knows. It should also reject the document if any two credentials in the same `From`, `To`, or `Sender` section appear to refer to different entities.

The receiver should reject the document if there are multiple credentials in a `To`, `From`, or `Sender` section that use different values but use the same domain.

## 3.1.7.6 Correspondent

The `From` and `To` elements can each optionally contain a `Correspondent` element. `Correspondent` elements are used in cases where a party or a connecting hub does not know the originating or receiving organization. The sender, receiver, or connecting hub can use the information in the `Correspondent` element to identify the unknown organization.

`Correspondent` has the following attribute:

| Attribute | Description |
|---|---|
| `preferredLanguage` | The preferred language of the organization, if it is known. |

`Correspondent` has the following elements:

| Element | Description |
|---|---|
| `Contact` (required) | Contains contact information for following up on an order. See Contact [page 44]. |
| `Routing` | Defines the corresponding routing destination. See Routing [page 31]. |
| `Extrinsic` | Contains any additional information related to this organization. |

## 3.1.7.6.1 Routing

Defines the corresponding routing destination of an external business partner. `Routing` has the following attribute:

| Attribute | Description |
|---|---|
| `destination` (required) | The name of the routing destination. Possible values:<br>• `peppol`<br>• `fieldglass` |

The following example shows a `Routing` element for an external business partner:

```
<Header>
  <From>
    <Credential domain="BusinessPartnerId">
      <Identity>
        <IdReference domain="iso6523" identifier="9925:BE12345678"/>
      </Identity>
    </Credential>
  </From>
  <To>
    <Credential domain="BusinessPartnerId">
      <Identity>
        <IdReference domain="iso6523" identifier="9925:BE3456789" />
      </Identity>
    </Credential>
    <Correspondent preferredLanguage="de">
      <Contact role="correspondent">
        <Name xml:lang="en-US">SupplierTradingName Ltd.</Name>
        <PostalAddress>
```

```
          <Street>Street</Street>
          <City>City</City>
          <State>State</State>
          <PostalCode>04726010</PostalCode>
          <Country isoCountryCode="BE" />
        </PostalAddress>
        <Phone name="work">
          <TelephoneNumber>
            <CountryCode isoCountryCode="BE" />
            <AreaOrCityCode />
            <Number>1151869655</Number>
          </TelephoneNumber>
        </Phone>
      </Contact>
      <Routing destination="peppol" />
    </Correspondent>
  </To>
  <Sender>
    <Credential domain="NetworkID">
      <Identity>AN01000000001</Identity>
    </Credential>
    <UserAgent>Ariba Network</UserAgent>
  </Sender>
</Header>
```

## 3.1.8 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.

Typical `Request` elements are:

- `OrderRequest`
- `ProfileRequest`
- `PunchOutSetupRequest`
- `StatusUpdateRequest`
- `GetPendingRequest`
- `ConfirmationRequest`
- `ShipNoticeRequest`
- `ProviderSetupRequest`
- `PaymentRemittanceRequest`

`Request` has the following attributes:

| Attribute | Description |
|---|---|
| `deploymentMode` | Indicates whether the request is a test request or a production request. Allowed values are “production” (default) or “test”. |
| `Id` | This attribute can be used to call out an element and all its children as a target for a digital signing. |

### Related Information

cXML Digital Signatures [page 53]

## 3.1.9 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. During PunchOut for example, the application-level data is contained in a `PunchOutSetupResponse` element.

The typical `Response` elements are:

- `ProfileResponse`
- `PunchOutSetupResponse`
- `GetPendingResponse`

`Response` has the following attribute:

| Attribute | Description |
|---|---|
| `Id` | This attribute can be used to call out an element and all its children as a target for a digital signing. |

### Related Information

cXML Digital Signatures [page 53]

## 3.1.9.1 Status

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

`Status` has the following attributes:

| Attribute | Description |
|---|---|
| `code` (required) | The status code of the request. For example, 200 represents a successful request. See the table of codes below. |
| `text` (required) | The text of the status. This text aids user readability in logs, and is a canonical string for the error in English. |
| `xml:lang` | The language of the data in the `Status` element. Optional for compatibility with cXML 1.0. Might be required in future versions of cXML. |

The attributes of the `Status` element indicate what happened to the request. For example:

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

The content of the `Status` element can be any data needed by the requestor and should describe the error. For a cXML `200/OK` status code, there might be no data. However, for a cXML `500/Internal Server Error` status code, or other similar 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. For example:

```
<Status code="406" text="Not Acceptable">cXML did not validate. Big Problem!</
Status>
```

The following table describes the cXML status code ranges:

| Range | Meaning |
|---|---|
| 2xx | Success |
| 4xx | Permanent error. Client should not retry. The error prevents the request from being accepted. |
| 5xx | Transient error. Typically a transport error. Client should retry. The recommended number of retries is 10, with a frequency of one hour. At a minimum a six hour retry window is recommended. For high priority requests, such as rush orders, you might want to increase the retry frequency. |

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

Because cXML is layered above HTTP in most cases, many errors (such as HTTP `404/Not Found`) are handled by the transport. All transport errors should be treated as transient and the client should retry, as if a cXML 500 range status code had been received. All HTTP replies that don't include valid cXML content, including HTTP 404/Not found and HTTP 500/Internal Server Error status codes, are considered transport errors. Other common transport problems include timeouts, TCP errors (such as "`connection refused`"), and DNS errors (such as "`host unknown`"). Validation errors in parsing a `Request` document would normally result in a cXML permanent error in the `400` range, preferably `406/Not Acceptable`.

The following table includes possible cXML status codes:

| Status | Text | Meaning |
|---|---|---|
| 200 | OK | The server was able to execute the request or deliver it to the final recipient. The returned `Response` might contain application warnings or errors: the cXML `Request` itself generated no errors or warnings, however, this status does not reflect any errors or warnings that might be generated afterward by the application itself. You will receive no further status updates, unless an error occurs during later processing. |
| 201 | Accepted | The request has been accepted for forwarding by an intermediate hub, or has been accepted by its ultimate destination and not yet been examined. You will receive updates on the status of the request, if a mechanism to deliver them is available.<br><br>The client should expect later `StatusUpdate` transactions. |

| Status | Text | Meaning |
|---|---|---|
| 204 | No Content | All `Request` information was valid and recognized. The server has no `Response` data of the type requested.<br><br>In a `PunchOutOrderMessage`, this status indicates that the PunchOut session ended without change to the shopping cart (or client requisition). |
| 211 | OK | Buyers can use this status code to send a broadcast message to suppliers to inform them about any events they need to know, such as holiday schedules, production facility closure, or completion of certain activities such as planning run completion. |
| 280 | | The request has been forwarded by an intermediate hub. You will receive at least one more status update. This status could mean that the request was delivered to another intermediary or to the final recipient with 201 status, or that it was forwarded via a reliable non-cXML transport. |
| 281 | | The request has been forwarded by an intermediate hub using an unreliable transport (such as email). You might receive status updates; however, if you do not received status updates, there is not necessarily a problem. |
| 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 | `Request` unacceptable to the server, likely due to a parsing failure. |
| 409 | Conflict | The current state of the server or its internal data prevented the (update) operation request. An identical `Request` is unlikely to succeed in the future, but only after another operation has executed, if at all. |
| 412 | Precondition Failed | A precondition of the `Request` (for example, a PunchOut session appropriate for a `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. |
| 450 | Not Implemented | The server does not implement the particular `Request`. For example, `PunchOutSetupRequest` or the requested operation might not be supported. This status normally implies the client has ignored the server's profile. |
| 475 | Signature Required | The receiver is unwilling to accept the document because it does not have a digital signature. |
| 476 | Signature Verification Failed | The receiver is unable to validate the signature, possibly because the document was altered in transit, or the receiver does not support one or more algorithms used in the signature. |

| Status | Text | Meaning |
|---|---|---|
| 477 | Signature Unacceptable | The signature is technically valid, but is not acceptable to the receiver for some other reason. The signature policies or certificate policies may be unacceptable, the type of certificate used may be unacceptable, or there may be some other problem. |
| 500 | Internal Server Error | Server was unable to complete the `Request`. |
| 550 | Unable to reach cXML server | 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 | Unable to forward request because of 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. The client should retry later. |

The following table lists possible status codes for catalog-upload requests:

| Status | Text | Meaning |
|---|---|---|
| 200 | Success | The catalog-upload request succeeded. |
| 201 | Accepted | The catalog-upload request is processing. |
| 461 | Bad Commodity Code | The commodity code you assigned to the catalog is invalid. |
| 462 | Notification Error | No notification method (email or URL) provided. |
| 463 | Bad Catalog Format | The zip file is invalid. |
| 464 | Bad Catalog | No catalog is attached, or more than one is attached. |
| 465 | Duplicate Catalog Name | The name of the catalog exists. |
| 466 | No Catalog to Update | The catalog to be updated does not exist. |
| 467 | Publish Not Allowed | You attempted to publish a catalog that was not previously published. |
| 468 | Catalog Too Large | The size of the uploaded file exceeds the 4 MB limit. Zip the catalog to compress it before uploading it. |
| 469 | Bad Catalog Extension | The file name of the catalog must have .cif, .xml, or .zip extensions. |
| 470 | Catalog Has Errors | The message is the status of the catalog. (HasErrors) |
| 499 | Document Size Error | The cXML document is too large. |
| 561 | Too Many Catalogs | You cannot upload more than a specific number of catalogs per hour. |
| 562 | Publish Disabled | Catalog publishing is temporarily unavailable due to scheduled maintenance. It will be back online by the specified date and time. |
| 563 | Catalog Validating | You attempted to update a catalog before validation finished on a previous version of the catalog. |

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

## 3.1.10 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.


Figure 6: One-Way Message (Asynchronous)

In this case, a possible scenario would be:

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

In the One-Way model, Site A and Site 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 in cXML Envelope [page 24]. 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.

The Header element in a one-way message should not contain shared secret information in the sender credential. Authentication is done using the `BuyerCookie`. This is different from Request-Response Header.

## 3.1.11 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:

| Attribute | Description |
|---|---|
| `deploymentMode` | Indicates whether the request is a test request or a production request. Allowed values are "production" (default) or "test". |
| `inReplyTo` | 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. |
| `Id` | This attribute can be used to call out an element and all its children as a target for a digital signing. |

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 that of the `inReplyTo` attribute.

### Related Information

cXML Digital Signatures [page 53]

## 3.1.12 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

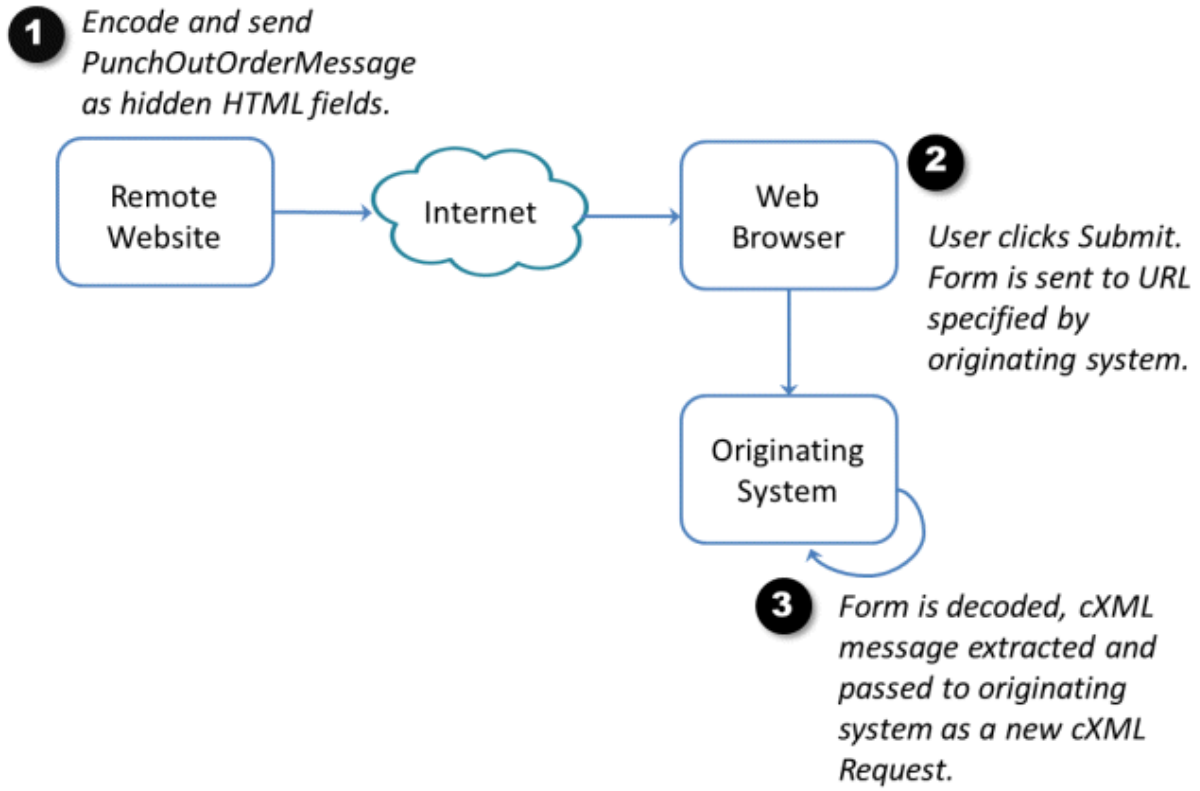
Procurement applications pull information using one-way HTTP communication. The one type of transaction that uses one-way HTTP communication is `GetPendingRequest`.

HTTPS is preferred, because it encrypts transmitted data for security.

## URL-Form-Encoding

URL-Form-Encoding enables integration between remote websites 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. This transport is best understood by examining how the `PunchOutOrderMessage` transaction works.

Remote websites do not directly send cXML `PunchOutOrderMessage` documents to procurement applications; instead, they encode them as hidden HTML Form fields and post them to the URL specified in the `BrowserFormPost` element of the `PunchOutSetupRequest`. When the user clicks a Check Out button on the website after shopping, 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

Remote websites assign each `PunchOutOrderMessage` document to a hidden field on the Form named `cXML-urlencoded` or `cXML-base64`. They assign the HTML Form element 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="Entire 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 names `cXML-urlencoded and cXML-base64` are case insensitive.

## cXML-urlencoded

The `cXML-urlencoded` field is URL encoded (per the HTTP specification) by the Web browser, not by the Web server or the supplier. This is because the encoding is required only when the form is submitted by a Web browser, such as when a user clicks Check Out in the previous example. However, the Web server must HTML-encode the field value, escaping quotation marks and other special characters, so that the form will display correctly.

> **Note**
>
> Suppliers should never URL encode the cXML-urlencoded field. This field is automatically URL-encoded by the web browser.

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.

## cXML-Base64

The `cXML-base64` hidden field supports 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.”

Base64-encoding from the remote website 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 default character encoding is `UTF-8,` as for any `application/xml` documents.

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.

## 3.1.13 Service Status Response

This transaction determines whether a particular service is currently available. When an HTTP `GET` is sent to a service location, the service responds with a valid, dynamically generated cXML `Response` document. A service can be any HTTP URL at which cXML `Request` documents are received.

For example, an HTTP `GET` sent to `https://service.ariba.com/service/transaction/cxml.asp` yields the following response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE cXML "http://xml.cXML.org/schemas/cXML/1.2.014/cXML.dtd">
<cXML timestamp="2001-01-08T10:47:01-08:00"
payloadID="978979621537--4882920031100014936@206.251.25.169">
    <Response>
        <Status code="200" text="OK">Ping Response Message</Status>
    </Response>
</cXML>
```

> **Note**
>
> This combination of transport (HTTP) and protocol (cXML) levels should be used only for the case described above.

## 3.2 Basic Elements

The following entities and elements are used throughout the cXML specification. Most of the definitions listed 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.

### 3.2.1 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.

#### isoLangCode

An ISO Language Code from the ISO 639 standard.

#### isoStateCode

An ISO 3166-2:2013 country subdivision code that identifies a state or province. It is used in conjunction with the country codes listed in ISO 3166-1.

#### 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. See www.unece.org/cefact/codesfortrade/codes_index.html.

### URL

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

### Related Information

cXML Envelope [page 24]

## 3.2.2 Base Elements

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

### Money

The `Money` element has three possible attributes: `currency`, `alternateAmount`, `alternateCurrency.` The attributes `currency` and `alternateCurrecy` must be a three-letter ISO 4217 currency code. The content of the `Money` element and of the `alternateAmount` attribute should be a numeric value. For example:

```
<Money currency="USD">12.34</Money>
```

The optional `alternateCurrency` and `alternateAmount` attributes are used together to specify an amount in an alternate currency. These can be used to support dual-currency requirements such as the euro. For example:

```
<Money currency="USD" alternateCurrency="EUR"
  alternateAmount="14.28">12.34</Money>
```

> **Note**
>
> You can optionally use commas as thousands separators. Don't use commas as decimal separators.

## State

Contains the state or country subdivision identifier. Contained by the `PostalAddress` element. It has an optional isoStateCode [page 42] attribute.

```
<State isoStateCode="US-CA">CA</State>
```

## Country

Contains the name of the country in a location. Contained by the `PostalAddress` element. It has an optional isoCountryCode [page 42] attribute.

```
<Country isoCountryCode="US">United States</Country>
```

## CountryCode

Contains the International ITU dial code for the country code. It can be entered onto a telephone keypad after the escape code to reach the country. Used by the `TelephoneNumber` element.

```
<TelephoneNumber>
  <CountryCode isoCountryCode="US">1</CountryCode>
  <AreaOrCityCode>800</AreaOrCityCode>
  <Number>5551212</Number>
</TelephoneNumber>
```

## Contact

The Contact element contains information about any contact important to the current transaction. For example:

```
<Contact>
  <Name xml:lang="en-US">Mr. Smart E. Pants</Name>
  <Email>sepants@workchairs.com</Email>
  <Phone name="Office">
    …
  </Phone>
</Contact>
```

# 4 Alternative Authentication Methods

cXML supports alternatives to the shared secret authentication method for verifying the sender of cXML documents.



## 4.1 Message Authentication Code (MAC)

Message Authentication Code (MAC) authentication allows the authentication of documents sent directly from a client to a server without passing through a trusted third party (such as a network hub) for authentication. These documents contain a credential with an authentication code that can be interpreted only by the trusted third party and the receiving server, not by the sender.

The format of the `Credential` element containing the MAC is described in Credential [page 29].

### 4.1.1 Overview of MACs

The primary purpose of MACs is to convey receivers' shared secrets without revealing them to senders. MACs keep shared secrets secure by encoding them through a hash.

MACs are as secure as shared secrets. Senders must guard MACs as carefully as shared secrets. Compromising either piece of information could make trading partners vulnerable.

To use MAC authentication, both the trusted third party and the receiver must be able to compute MACs.

### 4.1.2 Computation Algorithm

MACs are created by an algorithm that combines data known by both the trusted third party and the receiver.

cXML specifies the use of the HMAC-SHA1 algorithm described in IETF RFC 2104, "HMAC: Keyed-Hashing for Message Authentication".

The HMAC-SHA1 algorithm provide the security required for cXML, and it has been formally proven to be as secure as the underlying hash algorithm.

For more information about IETF RFC 2104, see www.ietf.org/rfc/rfc2104.txt.

## 4.1.3 Creation and Expiration Dates

Creation and expiration dates add additional security to MACs.

If a MAC is stolen, changing the sender's shared secret has no effect. It is impractical to expect the sender to contact the receiver out-of-band to invalidate the MAC, because they might not have an established relationship. To address this problem, a creation date (`creationDate`) and an expiration date (`expirationDate`) are embedded in MACs. The expiration date limits the damage that can be result from a stolen MAC, because MACs eventually expire. The shorter the expiration period, the greater the security afforded. Receivers must reject MACs that are received after their expiration date.

Receivers can also reject unexpired MACs based on the amount of time that has elapsed since the creation date. For example, if a receiver receives a MAC that was created several years ago, but expires tomorrow, the receiver might not wish to accept the MAC. This decision is left with the implementors of the receiving systems.

It is mandatory for receivers to check that the creation date is in the past and the expiration date is in the future, and to reject it if either is not the case. However, it is optional for receivers to check whether the creation date is too long in the past.

Receivers must not only check that MACs are valid, but also that the data authenticated by MACs is acceptable. Specifically, receivers must validate that they wish to accept messages from the entities identified by the From and Sender credentials.

## 4.1.4 Computation Process

This section describes how to compute a MAC of `type="FromSenderCredentials"`. The inputs for this MAC type are known only by the trusted third party and the receiver.

The trusted third party uses this computation to generate `ProfileResponse Option` elements and the receiving server uses it to validate the `CredentialMac` element.

### 4.1.4.1 Assembling the Hash Inputs

The MAC function takes two inputs, the data input and the secret key input:

- The data input is the UTF-8-encoded byte representation of each value listed below, in order, after normalization, with each value terminated by a single null byte (0x00):

```
From/Credential@domain
From/Credential/Identity
Sender/Credential@domain
Sender/Credential/Identity
Sender/Credential/CredentialMac@creationDate
Sender/Credential/CredentialMac@expirationDate
```

- The secret key input is the cXML shared secret used between the receiver and the third party.

### 4.1.4.2 Normalizing the Inputs

Normalize the hash input values to remove differences in case and formatting before computation:

| Value | Normalize by... | Normalized Example |
|---|---|---|
| `domain` | Use the lowercase version of the string, unless it is known to be case sensitive, for example, “AribaNetworkUserId”. Note that “NetworkId” and “DUNS” are not case-sensitive. | `networkid` |
| `Identity` | Discard leading or trailing whitespace and use the lowercase version of the string. | `an9900000100` |
| `creationDate`<br>`expirationDate` | No normalization needed, because they are in ISO8601 format described in Date, Time, and Other Data Types [page 25]. | `2003-01-15T11:42:46-08:00` |

Do not normalize the shared secret.

### 4.1.4.3 MAC Algorithm

The only supported MAC algorithm value is `"HMAC-SHA1-96"`, which corresponds to the HMAC-SHA1 algorithm, which produces a 160 bit (20 byte) output, and retaining only the left-most 96 bits (12 bytes). The 12 bytes are then base-64 encoded, yielding a 16-byte character string consisting only of characters in the set [A-Z a-z 0-9 +/].

To computer the MAC:

1. Concatenate the UTF-8-encoded byte representation of the following strings, each followed by a null byte (0x00). (The strings have been normalized as described above):
   “networkid”, “an9900000100”, “networkid”, “an9900000100”,
   “2003-01-15T08:42:46-08:00”, “2003-01-15T11:42:46-08:00”
   The concatenation yields the following byte sequence:

   ```
   6e 65 74 77 6f 72 6b 69 64 00 61 6e 39 39 30 30
   30 30 30 31 30 30 00 6e 65 74 77 6f 72 6b 69 64
   00 61 6e 39 39 30 30 30 30 30 31 30 30 00 32 30
   30 33 2d 30 31 2d 31 35 54 30 38 3a 34 32 3a 34
   36 2d 30 38 3a 30 30 00 32 30 30 33 2d 30 31 2d
   31 35 54 31 31 3a 34 32 3a 34 36 2d 30 38 3a 30
   30 00
   ```

2. Use HMAC-SHA1 to hash the above sequence with the receiver’s shared secret, for example, “abracadabra” (61 62 72 61 63 61 64 61 62 72 61), which yields:

   ```
   71 1e 89 a7 3e 7c 9e b8 97 11 10 cd 78 57 fd a0 94 da fd
   ```

   Do not terminate or normalize the shared secret.
3. Truncate the above result to 96 bits (12 bytes):

   ```
   71 1e 89 a7 3e 7c 9e b8 97 11 10 cd
   ```

   Truncation helps increase the security of the hash.

4. Base-64 encode the above result to yield the final result:

```
cR6Jpz58nriXERDN
```

The trusted third party inserts the final result in `ProfileResponse` documents it sends to the entity that will be the client (document sender), and the client inserts it in a `CredentialMac` element in all direct communication to the server (document receiver).

## 4.1.5 ProfileResponse

The following cXML example shows a `ProfileResponse` sent from a trusted third party (such as a network hub) to a client (such as a procurement application) so the client can send direct requests to the receiving server.

```
<cXML payloadID="1234567890@bighub.com"
  timestamp="2003-01-15T09:39:09-08:00" xml:lang="en-US">
  <Response>
    <Status code="200" text="OK"/>
    <ProfileResponse>
      <Option name="CredentialMac.type">FromSenderCredentials</Option>
      <Option name="CredentialMac.algorithm">HMAC-SHA1-96</Option>
      <Option name="CredentialMac.creationDate">2003-01-15T08:42:46
      -0800</Option>
      <Option name="CredentialMac.expirationDate">2003-01-15T11:42:46
      -0800</Option>
      <Option name="CredentialMac.value">cR6Jpz58nriXERDN</Option>
      <Transaction requestName="OrderRequest">
        <URL>https://service.hub.com/ANCXMLDispatcher.aw/ad/cxml</URL>
      </Transaction>
      <Transaction requestName="PunchOutSetupRequest">
        <URL>https://service.hub.com/AN/cxml</URL>
        <Option name="Direct.URL">https://bigsupplier.com/punchout</Option>
        <Option name="Direct.AuthenticationMethod.CredentialMac">Yes
        </Option>
        <Option name="Direct.AuthenticationMethod.Certificate">Yes</Option>
      </Transaction>
    </ProfileResponse>
  </Response>
</cXML>
```

## 4.1.6 CredentialMac

The following cXML document fragment shows an example `CredentialMac` element as it would be inserted by the client in documents sent directly to the server.

```
<cXML>
  <Header>
    <To>
      <Credential domain="DUNS">
        <Identity>049329048</Identity>
      </Credential>
    </To>
    <From>
      <Credential domain="NetworkId">
        <Identity>AN9900000100</Identity>
```

```
      </Credential>
    </From>
    <Sender>
      <Credential domain="NetworkId">
        <Identity>AN9900000100</Identity>
        <CredentialMac type="FromSenderCredentials"
            algorithm="HMAC-SHA1-96"
            creationDate="2016-01-15T08:42:46-0800">
            expirationDate="2016-01-15T11:42:46-0800">
          cR6Jpz58nriXERDN
        </CredentialMac>
        <UserAgent>Procure System 3.0</UserAgent>
      </Credential>
    </Sender>
   </Header>
  [. . .]
</cXML>
```

### Related Information

Credential [page 29]

## 4.2 Auth Transaction

The Auth transaction allows receivers to validate organizations' credentials through a mutually trusted third party. It should be used to authenticate received documents that do not contain either a shared secret or a MAC.

The receiver encloses the credential of the sender (the principal) in an `AuthRequest` document and sends it to the trusted third party for validation.

If the principal attempts to authenticate using a client digital certificate, the receiver includes both the principal's credential and information about the principal's certificate in the `AuthRequest` document. (The receiver obtains this certificate information from its Webserver or TLS implementation.)

The trusted third party receives the `AuthRequest` and looks up the principal's credential to see if it is a recognized organization. If the principal's certificate information was included, the trusted third party makes sure the certificate is valid and that it matches the organization associated with the credential.

If the credential (and optional certificate) authenticates, the trusted third party responds with a positive `AuthResponse` that contains the validated credential. If the credential is invalid, the trusted third party responds with an empty cXML response of status 403 (Forbidden).

The receiver can cache the results of the Auth transaction until the expiration date indicated in the `AuthResponse`. During this period, if the principal presents the same credential and certificate, the receiver need not send another `AuthRequest`.

## 4.2.1 AuthRequest

A request sent to a mutually trusted third party to authenticate an entity.

The following example includes X509 certificate information, which comes from the requesting entity's client digital certificate.

```
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.2.014/cXML.dtd">
<cXML timestamp="2000-12-28T16:56:03-08:00" payloadID="foo123@bigsupplier.com">
  <Header>
    <From>
      <Credential domain="NetworkId">
        <Identity>AN99000000092</Identity>
      </Credential>
     </From>
    <To>
      <Credential domain="NetworkId">
        <Identity>AN99000000092</Identity>
      </Credential>
    </To>
    <Sender>
      <Credential domain="NetworkId">
        <Identity>AN99000000092</Identity>
        <SharedSecret>abracadabra</SharedSecret>
      </Credential>
      <UserAgent>cXML application 2.0</UserAgent>
    </Sender>
  </Header>
  <Request>
    <AuthRequest>
      <Credential domain="DUNS">
        <Identity>12345</Identity>
      </Credential>
      <X509Data>
        <X509IssuerSerial>
          <X509IssuerName>Verisign</X509IssuerName>
          <X509SerialNumber>12345</X509SerialNumber>
        </X509IssuerSerial>
      </X509Data>
    </AuthRequest>
  </Request>
</cXML>
```

### 4.2.1.1 Credential

A cXML credential. See Credential [page 29].

### 4.2.1.2 X509Data

Describes the X.509 client certificate being used for authentication.

### X509IssuerSerial

A container for the serial number and issuer name of the X.509 certificate.

`X509IssuerSerialChild` has the following elements:

- `X509IssuerName`
  The distinguished name of the issuer of the X.509 certificate. The distinguished name should be a string representation of an LDAP Distinguished Name, as described in RFC 2253. For example,
  `C=US, O="Mega Data Security, Inc.", OU=Secure Server CA`
- `X509SerialNumber`
  The serial number of the X.509 certificate.

### X509SKI

The Subject Key Identifier of the X.509 certificate.

### X509 SubjectName

The distinguished name of the subject of the X.509 certificate. This should be a string representation of an LDAP distinguished name, as described in RFC 2253.

### X509Certificate

Contains the Base-64-encoded X.509v3 certificate.

### X509CRL

Contains a Base-64-encoded X.509v3 Certificate Revocation List.

## 4.2.2 AuthResponse

Returns a list of valid credentials of the person entity in the `AuthRequest` document. Note that this response is for successful authentications only.

`AuthResponse` has the following attribute:

| Attribute | Description |
| --- | --- |
| `expirationDate` | Specifies the time beyond which the information contained in the `AuthResponse` must be discarded. The inclusion of this attribute specifies that the receiver can cache the AuthResponse information until the `expirationDate`. |

The absence of an `expirationDate` should be interpreted to forbid caching.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.2.014/cXML.dtd">
<cXML payloadID="234234@hub.com" timestamp="2001-01-25T15:19:07-08:00">
    <Response>
        <Status code="200" text="OK"/>
        <AuthResponse expirationDate="2002-12-31T09:00:00-08:00">
            <Credential domain="DUNS">
                <Identity>12345</Identity>
            </Credential>
        </AuthResponse>
    </Response>
</cXML>
```

# 5 cXML Digital Signatures

Any cXML request, response, or message can be signed using World Wide Web Consortium (W3C) XML Digital Signatures. Support for the XML Advanced Electronic Signature (XAdES) standard is also included.

Readers of this section should be familiar with electronic signature terminology and concepts such as asymmetric key pairs, certificates, and smart cards.

Digital Signature Overview [page 53]

Signing cXML Documents [page 54]

## 5.1 Digital Signature Overview

Digital signatures confirm the identity of the sender of an electronic document, and ensure that the document was not modified after it was generated by the signer. They consist of a series of bytes that contain cryptographic information, including the sender's public key and detailed information about the contents of the document being signed.

An XML digital signature—which is a specific arrangement of a digital signature—is an element that contains other information besides the cryptographic signature itself, including a list of what was signed, the signer's public key, and other attributes. A cXML signature is an XML digital signature of a certain form, as described later in this chapter.

XML Advanced Electronic Signature (XAdES) provides basic authentication and integrity protection.

W3C XML signatures and XAdES have many options designed to allow for flexibility.

For information about W3C XML digital signatures, see the following resources:

- www.w3.org
- XML Signature Syntax and Processing Version 1.1

For information about XAdES, see the following resources:

- XML Advanced Electronic Signatures (XAdES)
- uri.etsi.org/01903/v1.3.2

### 5.1.1 Options for Signing

You can use a service to sign documents on your behalf, or you can implement the necessary hardware or software systems to sign the documents yourself. If you implement your own signing system, you must obtain a certificate signed by a Certificate Authority (CA) trusted by the receiver. Meeting receiver requirements might mean obtaining hardware that keeps the private key secret, such as a smart card or Hardware Security Module.

Note that signature and certificate requirements vary according to local laws and regulations. Prior to implementing a signing system, be sure you learn the requirements of the relevant locale.

## 5.2 Signing cXML Documents

A valid cXML digital signature is not just an XML signature, but an XML signature that uses particular options, has particular elements present, and signs (or does not sign) certain portions of the document.

### 5.2.1 cXML Digital Signatures

Note that namespace prefix conventions are used here when referring to elements that come from other specifications. All W3C XML Digital Signature elements use the `ds` prefix, and all XAdES elements use the `xades` prefix.

#### 5.2.1.1 ds:Signature Element

The `cXML` element contains a space for the `ds:Signature` element after the `Request`, `Response`, or `Message` element. The `ds:Signature` element holds information about what is being signed, one or more signatures, and the keys used to create the signature or signatures. It also has a place to store additional information such as XAdES extensions or attachment manifests.

The `cXML` element also contains a space for the `signatureVersion` attribute.

| Attribute | Description |
|---|---|
| `signatureVersion` | If present, `signatureVersion` implies that the document is digitally signed, that is, that the document contains a valid `ds:Signature` element immediately following the `Request`, `Response`, or `Message` element. If the document is signed, this attribute must be present. The only valid value for the attribute is `1.0`; other values are reserved for future use. |
| `Id` | This attribute can be used to call out an element and all its children as a target for signing. For example, if a document contains `<Request Id="foo">`, then in the digital signature `<Reference URI="#foo">` will refer to the `Request` element and all its children. If the document is signed, this attribute must be present. |

The `Message`, `Request`, and `Response` elements contain an `Id` attribute.

### Related Information

cXML Envelope [page 24]

cXML Basics [page 17]

### 5.2.1.2 cXMLSignedInfo

The `cXMLSignedInfo` element includes cXML-specific details about the signature, and has the following attributes:

| Attribute | Description |
|---|---|
| `signatureVersion` (required) | Implies that the document is digitally signed, that is, that the document contains a valid `ds:Signature` element immediately following the `Request`, `Response`, or `Message` element. The only valid value for the attribute is `1.0`; other values are reserved for future use. |
| `payloadID` (required) | Used to establish links between documents. The `payloadID` in the `cXMLSignedInfo` element must be the same as the `payloadID` in the document's main `cXML` element. |
| `Id` (required) | Identifies this `cXMLSignedInfo` element for purposes of the signature. This attribute must always be present and should always have the value "`cXMLSignedInfo`". |

### 5.2.1.3 Signing Essentials

Because some information from the cXML header is significant, it must be signed. To sign these attributes from the header, repeat the information in a `cXMLSignedInfo` element placed within a `ds:Object` element. The `ds:Object` must be the first `ds:Object` in the signature. For example:

```
<ds:Object>
<cXMLSignedInfo Id="cXMLSignedInfo"
            signatureVersion="1.0"
            payloadID="xxx"/>
</ds:Object>
```

The value of the `Id` attribute must be `"cXMLSignedInfo"`. The values of the `signatureVersion` and `payloadID` attributes must exactly match the values specified in the `cXML` element, and the receiver of the document must verify this match. No transforms should be used in this `ds:Reference`. This element must be signed via the first `ds:Reference` object in the `ds:SignedInfo,` as follows:

```
<ds:Reference URI="#cXMLSignedInfo">
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>xxxxxxxxxxxxxxxxxxxxxxxxxxxx</ds:DigestValue>
</ds:Reference>
```

The `Request`, `Response`, or `Message` element should be signed in its entirety. To do this, specify the string `"cXMLData"` as the value of the `Id` attribute on the `Request`, `Response`, or `Message` element and include a `ds:Reference` element with the URI `"#cXMLData"` in the `ds:SignedInfo`. No transforms should be applied to this reference. This `ds:Reference` must be the second `ds:Reference` in the `ds:SignedInfo`.

The `ds:KeyInfo` element should be present with a single `ds:X509Certificate` element. This should include the Base64 encoding of the DER representation of an X.509 certificate containing the public key corresponding to the private key used to sign the document.

### 5.2.1.4 Using XAdES

The use of XAdES is required for digital signatures. In the signature, `xades:QualifyingProperties` should be the second `ds:Object`. The `xades:SignedProperties` element and all its children must be signed by specifying "`XAdESSignedProps`" as the value for the Id attribute of `xades:SignedProperties` and including a `ds:Reference` with the URI "`#XAdESSignedProps`" and no transforms in the `ds:SignedInfo`. When using XAdES, the certificate referred to in the `xades:Cert` element must be the same as that contained in the `ds:KeyInfo` element, the Id attribute of the `ds:Signature` element must be set to `cXMLSignature` and the `Target` attribute of `xades:QualifyingProperties` must be `#cXMLSignature`.

### 5.2.1.5 Signing Attachments

If the document in question includes attachments, digital signatures can be used to sign just the document, or both the document and its attachments. Signatures are structured in such a way that if the attachments are discarded, the signature on the document itself can still be validated.

The attachments should be signed using `ds:Reference` elements in a `ds:Manifest` element included under a `ds:Object` contained in the signature. The `Id` attribute of the `ds:Manifest` element must be "AttachmentManifest". The `ds:Object` should occur immediately after the `ds:Object` containing the `xades:QualifyingProperties` element, if it is present. Otherwise, it should occur immediately after the `ds:Object` containing the `cXMLSignedInfo element`.

Each `ds:Reference` in the manifest should use a URI with the `"cid:"` scheme to refer to the attachments through their MIME `Content-Id`. The `ds:Manifest` element itself should be signed using a fragment URI reference included in the `ds:SignedInfo.` This requirement exists because a compliant XML signature implementation must validate all the `ds:Reference` elements under `ds:SignedInfo`. Base validation ensures that the manifest itself has not been corrupted, but will not validate the objects referred to in the manifest. This approach makes it possible to validate the document on its own if the attachments have been discarded. For example:

```
<ds:Object>
    <ds:Manifest Id="AttachmentManifest">
        <ds:Reference URI="cid:23482390498.34284203.part1@some.host.com">
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>P6ua59kKBLLtMBFE+IwPUgp2xqc=</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="cid:23482390498.34284203.part2@some.host.com">
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>P6ua59kKBLLtMBFE+IwPUgp2xqc=</ds:DigestValue>
        </ds:Reference>
    </ds:Manifest>
</ds:Object>
```

## 5.2.2 Error Status Codes for Digital Signatures

The following table lists cXML digital signature status codes:

| Status | Text | Meaning |
|---|---|---|
| 475 | Signature Required | The receiver is unwilling to accept the document because it does not have a digital signature. |
| 476 | Signature Verification Failed | The receiver is unable to validate the signature, possibly because the document was altered in transit, or the receiver does not support one or more algorithms used in the signature. |
| 477 | Signature Unacceptable | The signature is technically valid, but is not acceptable to the receiver for some other reason. The signature policies or certificate policies might be unacceptable, the type of certificate used might be unacceptable, or there might be some other problem. |

## 5.2.3 Digital Signature Example

The following example shows a signed invoice. Note that the digest values and signature value are not correct, because parts of the invoice document have been abbreviated for this example.

```
<?xml version="1.0" ?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.2.0.11/
InvoiceDetail.dtd">
<cXML payloadID="20030912.jdoe004@live.company.com" signatureVersion="1.0"
timestamp="200104-20T23:59:45-07:00">
  <Header>
    <From>
      <Credential domain="AribaNetworkUserId">
        <Identity>jdoe@company.com</Identity>
      </Credential>
      </From>
      <To>
        <Credential domain="AribaNetworkUserId">
          <Identity>smistry@company.com</Identity>
        </Credential>
      </To>
      <Sender>
        <Credential domain="AribaNetworkUserId">
          <Identity>jdoe@company.com</Identity>
          <SharedSecret>abracadabra</SharedSecret>
        </Credential>
        <UserAgent>Our Invoice Application 4.0</UserAgent>
      </Sender>
  </Header>
  <Request Id="cXMLData" deploymentMode="production">
    <InvoiceDetailRequest>
      <InvoiceDetailRequestHeader invoiceDate="2001-04-20T23:59:20-07:00"
        invoiceID="123456-004" operation="new"
        purpose="standard">
        ...
      </InvoiceDetailRequestHeader>
      <InvoiceDetailOrder>
        ...
      </InvoiceDetailOrder>
      <InvoiceDetailSummary>
        ...
      </InvoiceDetailSummary>
```

```
    </InvoiceDetailRequest>
  </Request>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="cXMLSignature">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
    REC-xml-c14n20010315"></ds:CanonicalizationMethod>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
    </ds:SignatureMethod>
    <ds:Reference URI="#cXMLSignedInfo">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
      </ds:DigestMethod>
      <ds:DigestValue>mxtVp6Rg9K5wo/c5BO88g7sZYEg=</ds:DigestValue>
      </ds:Reference>
      <ds:Reference URI="#cXMLData">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
      </ds:DigestMethod>
        <ds:DigestValue>1uBJgSa3BXewh/1wsPDWCzn8Sgk=</ds:DigestValue>
      </ds:Reference>
      <ds:Reference URI="#XAdESSignedProps">
        <ds:DigestMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
        </ds:DigestMethod>
        <ds:DigestValue>XIasOHckorH8fz/thdyZIZvV2yI=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
nNfsBpc22u9aypYLvgE5cuiHVO077vnaolS76LoAuks9bAwLO0kz/nkTQfb2zKSQTy8jj6W/
TJGCQj691PlKBnIqaMPPN3k+hbi6A5cJHPRd3HNPexU5sSi4StTuxlWAiHe/
XEeBEeclu7K6sR4Rh1gzzELg05v21aRX4oVGbjk=</ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICgDCCAekCAw7cUTANBgkqhkiG9w0BAQQFADCBijELMAkGA1UEBhMCVV
w7cUTANBgkqhkiG9w0BAQQFADCBijELMAkGA1UEBhMCVVMxEzARBgNV
MIICgDCCAekCAw7cUTANBgkqhkiG9w0BAQQFADCBijELMAkGA1UEBhMCVV
w7cUTANBgkqhkiG9w0BAQQFADCBijELMAkGA1UEBhMCVVMxEzARBgNVBA
MIICgDCCAekCAw7cUTANBgkqhkiG9w0BAQQFADCBijELMAkGA1UEBhMCVV
zuRel/9tb8M95FuN5yR9GUGl5PgkzwuCQYobJqIcAs=</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
        <cXMLSignedInfo Id="cXMLSignedInfo"
          payloadID="20030912.rsmith004@live.hub.com" signatureVersion="1.0">
        </cXMLSignedInfo>
      </ds:Object>
      <ds:Object>
        <xades:QualifyingProperties xmlns:xades=
        "http://uri.etsi.org/01903/v1.1.1#"
        Target="#cXMLSignature">
        <xades:SignedProperties Id="XAdESSignedProps">
           <xades:SignedSignatureProperties>
             <xades:SigningTime>2003-09-30T18:32:27Z</xades:SigningTime>
               <xades:SigningCertificate>
                 <xades:Cert>
                   <xades:CertDigest>
                     <ds:DigestMethod
                         Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
                     </ds:DigestMethod>
                     <ds:DigestValue>LETnT8c7gvZqp3oVt8/BLOJpeeA=
                     </ds:DigestValue>
                   </xades:CertDigest>
                   <xades:IssuerSerial>
                     <ds:X509IssuerName>EMAILADDRESS=an_ops@company.com,
                       CN=anrc.hub.com, O="Hub, Inc.", L=Mountain View,
                       ST=California, C=US</ds:X509IssuerName>
                     <ds:X509SerialNumber>973905</ds:X509SerialNumber>
                   </xades:IssuerSerial>
                 </xades:Cert>
```

```
              </xades:SigningCertificate>
            <xades:SignaturePolicyIdentifier>
              <xades:SignaturePolicyImplied>
              </xades:SignaturePolicyImplied>
            </xades:SignaturePolicyIdentifier>
          </xades:SignedSignatureProperties>
        </xades:SignedProperties>
      </xades:QualifyingProperties>
    </ds:Object>
  </ds:Signature>
</cXML>
```

# 6 Revision History

The following table provides a brief history of the updates to this guide.

| Month/year of update | Updated topics | Short description of change |
|---|---|---|
| January 2021 | cXML Basics | Added or updated "Correspondent" and "Routing" topics. |
| October 2020 | cXML Basics | Updated "Handling Special Characters in Documents" topic. |
| January 2020 | cXML Basics | Updated "Base Elements" topic, adding `State` element. |
| October 2019 | cXML Basics | Updated "Type Entities" topic, adding `isoStateCode`. |
| | Multiple pages | Updated references to network hubs for consistency. |
| July 2018 | Title page | Removed the subtitle, which incorrectly specified a cXML version. This book applies to all cXML versions. |
| April 2018 | n/a | Initial version |