cXML/1.0

Table of Contents

1 Introduction

This document describes the protocol and data formats of Commerce XML (cXML) version 1.0. It contains all the information you need to implement any of the supported transactions from either the client or the server system perspective. Both the protocol interactions and business documents contained in the transactions are discussed in depth.

cXML is designed to provide a simple XML-based protocol between entities engaged in Business-to-Business eCommerce transactions over the Internet. Ease of implementation has been a primary focus along with an emphasis on prototype implementations to discover remaining issues. Most of the feedback that has been incorporated into the specification is based on ideas and solutions found during actual implementations between interacting companies and their systems.

The rest of this document is structured as follows: first, the protocol specification that describes how cXML documents are exchanged between the parties; second, a section that describes the underlying shared elements; third, a series of sections that describe the details of specific business documents; last, a look at the complete cXML XML definitions.

Examples and discussions from actual implementations appear throughout to help clarify how cXML can be used.

2 Protocol specification

cXML transactions take place in one of two models: Request-Response and One-Way. These two models allow for simplicity in implementation from the client/requestor perspective because the operations required are strictly described. Both models are required because there are situations when one model cannot provide a good fit.

Additionally, Request-Response transactions are strongly bound to the HTTP transport mechanism—that is, they can be performed only over an HTTP connection. One-Way messages are not restricted to a transport; two will be discussed below—HTTP and URL-Form-Encoding.

Because the Request-Response model is simpler to explain and understand, we will examine it first.

2.1 Request-Response

The following figure describes the protocol steps involved in a Request-Response interaction:

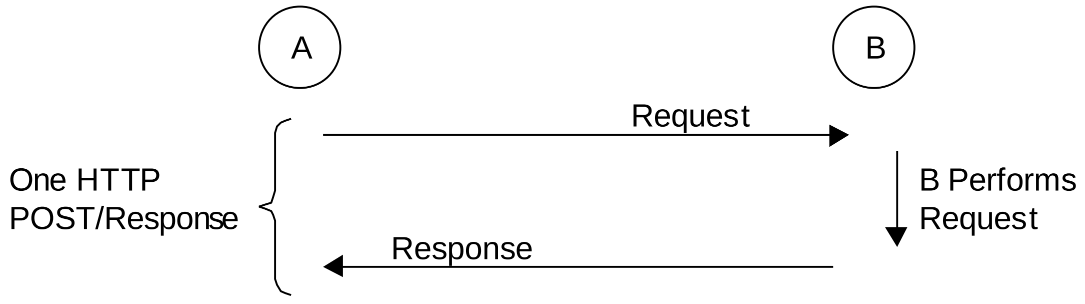


Figure 1 Example of a Single Request-Response Transaction

The above figure shows a cXML transaction between A and B. The transaction contains the following steps.

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

8. A reads the cXML Response and returns it to the process that initiated the Request.
9. A closes the HTTP connection established in Step 1.

This process is then repeated for further Request/Response cycles. To simplify the work required to accomplish the above steps, a cXML message is divided into two distinct parts:

- Header
- Request/Response data

The Header contains authentication information and addressing. The Request/Response bodies contain a specific request, the information to be passed, and the data that is expected as the response. Both of these elements are carried in a parent envelope element. The following example shows the document structure (incomplete to make the structure more evident):

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

The above example is a case where the body of the message is a Request. The following is an example of a Response:

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

The Response structure does not contain a Header element. It is not needed because the Response always travels in the same HTTP request that the Request traveled in.

The following sections discuss the various elements in more detail.

2.2 cXML Envelope

The envelope element is the root of the cXML message structure and it contains all the other elements. The `cXML` element is present in each cXML transaction. The following example shows a fully specified `cXML` element.

```
<cXML version="1.0" payloadID="1234567.4567.5678@test.ariba.com"
timestamp="1999-03-31T18:39:09-08:00">
```

A cXML element contains three attributes:

version	Specifies the version of the cXML protocol.
payloadID	A unique number with respect to space and time, used for logging purposes to identify messages that might have been lost or had problems. The recommended implementation is: datetime.process id.random number@hostname
timestamp	The date and time the message was sent, in ISO 8601 format.

2.3 Header

The `Header` contains information that is more transport-oriented than application-request/response-level-oriented. The same `Header` element is used regardless of which specific Request or Response is contained in the body of the cXML message. The application needs the requestor's identity, but does not need validation that the information provided for identity is correct. The following example shows the `Header` element in full detail.

```
<Header>
    <From>
        <Credential domain=“AribaNetworkUserId”>
            <Identity>aribaadmin@cisco.com</Identity>
        </Credential>
    </From>
    <To>
        <Credential domain=“DUNS”>
            <Identity>012345678</Identity>
        </Credential>
    </To>
    <Sender>
        <Credential domain=“AribaNetworkUserId”>
            <Identity>aribaadmin@cisco.com</Identity>
            <SharedSecret>welcome</SharedSecret>
        </Credential>
        <UserAgent>Ariba ORMS 6.0</UserAgent>
    </Sender>
</Header>
```

The `From` and `To` elements are synonymous with `From` and `To` in SMTP email messages; they are the logical source and destinations of the messages. `Sender` is the identity of A in Figure 1. A is the entity opening the HTTP connection and sending the cXML document. Usually, `Sender` and `From` are the same, with `Sender` containing the `Credential` element that authenticates the party to perform the Request. This allows strong authentication without requiring a public-key end-to-end digital certificate infrastructure. Only a user name and password needs to be issued by B to allow A to perform Requests.

The `Sender` element is the only one that changes as the cXML message is routed through the systems participating in delivering the message to its recipient.

The following sections discuss the elements in more detail.

2.3.1 From

This element contains a `Credential` element. It can optionally contain more than one `Credential` element, allowing the requestor to identify themselves in multiple ways. This is analogous to sending both SMTP and X.400 email addresses in an email message.

2.3.2 To

This element contains the destination of the cXML request. As in the `From` element case, more than one `Credential` can be specified to help identify the target.

2.3.3 Sender

This element contains the `Credential` of the entity on the other end of the HTTP connection performing a Request. It is a stronger authentication `Credential` than the ones contained in the `From` or `To` elements, which is necessary because the receiver on the other end of the HTTP connection must know who is asking it to perform work. See the `Credential` element for a description of the various ways of authenticating.

2.3.3.1 Credential

This element contains identification and authentication values used in cXML messages. It contains one attribute:

domain	Specifies the domain of the Credential. This allows multiple domains to co-exist in messages and allows multiple authentication domains to be used. For messages sent on Ariba Network, for instance, the domain will be “AribaNetworkUserId” or “DUNS”.

The elements that this element contains include an `Identity` element and optionally a `SharedSecret` or `DigitalSignature` element. The Identity element is used to state *who* is the Credential representing, while the optional Authentication elements then allow the Credential to assert that they are *who* they say they are.

The `SharedSecret` element would be used when the `Sender` has a username/password combination that the Requester at the other end of the HTTP connection can understand. That is the scenario used in the Header example above.

The `DigitalSignature` element could be used if the two parties agree on a common certificate format and authority. The `type` attribute on a `DigitalSignature` element would be used to coordinate this. Refer to the definitions in `Transport.mod` for more details.

2.4 Request

Requests are sent by clients to request 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 messages. Though the Request element can contain virtually any type of XML data, we will only look at the requests that are defined in cXML. These elements will be discussed later in the Business Document definition sections.

deploymentMode	Indicates whether the request is a test request or a production request.

2.5 Response

Responses are sent by the server to inform the client 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 contain any application-level data. In the PunchOut scenarios looked at later, that means a `PunchOutStartupResponse` element.

2.5.1 Status

This element conveys the success or failure of a Request operation. It has the following attributes:

code	The status code of the request. This follows the HTTP status code model very closely, with 200 being request successful, etc.
text	The text of the status code. This is here to aid human readability in logs. These will be canonical strings in English.

The attributes of the `Status` element indicate what happened to the request. The complete list of defined status code is forthcoming. The data in-between the `Status` tags can be anything that particular status code wants to deliver. For a `200/OK` status code, there might be no data. However, for a `500/ERROR` status code, it is strongly recommended that the actual XML parse error or application error be presented. This error allows better one-sided debugging and interoperability testing, because people on both sides of the operation do not have to be involved.

2.6 One-Way (Asynchronous)

One-Way messages are for situations when an HTTP channel (or really, 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.

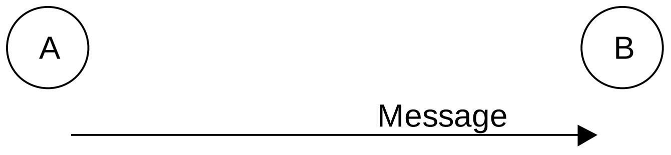


Figure 2 Diagram of a One-Way Message (Asynchronous)

In this case, a possible scenario would be:

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

It is important to realize that A and B do *not* have an explicit Request-Response cycle here. Messages from other parties might arrive; other conversations could take place, etc. 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. The best example of a transaction that uses One-Way is the PunchOutOrder* set of transactions.

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

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

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

2.6.1 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. The Message element itself has only one optional attribute:

InReplyTo (optional)	Used to specify 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 Request-Response like conversation over many Messages.

2.6.2 Transport Options

There are two well-known options for transporting One-Way Messages: HTTP and URL-Form-Encoding. It is important to note that these are just two of the well-defined transports today, more could become supported in the future, with SMTP being a prime candidate.

2.6.2.1 HTTP

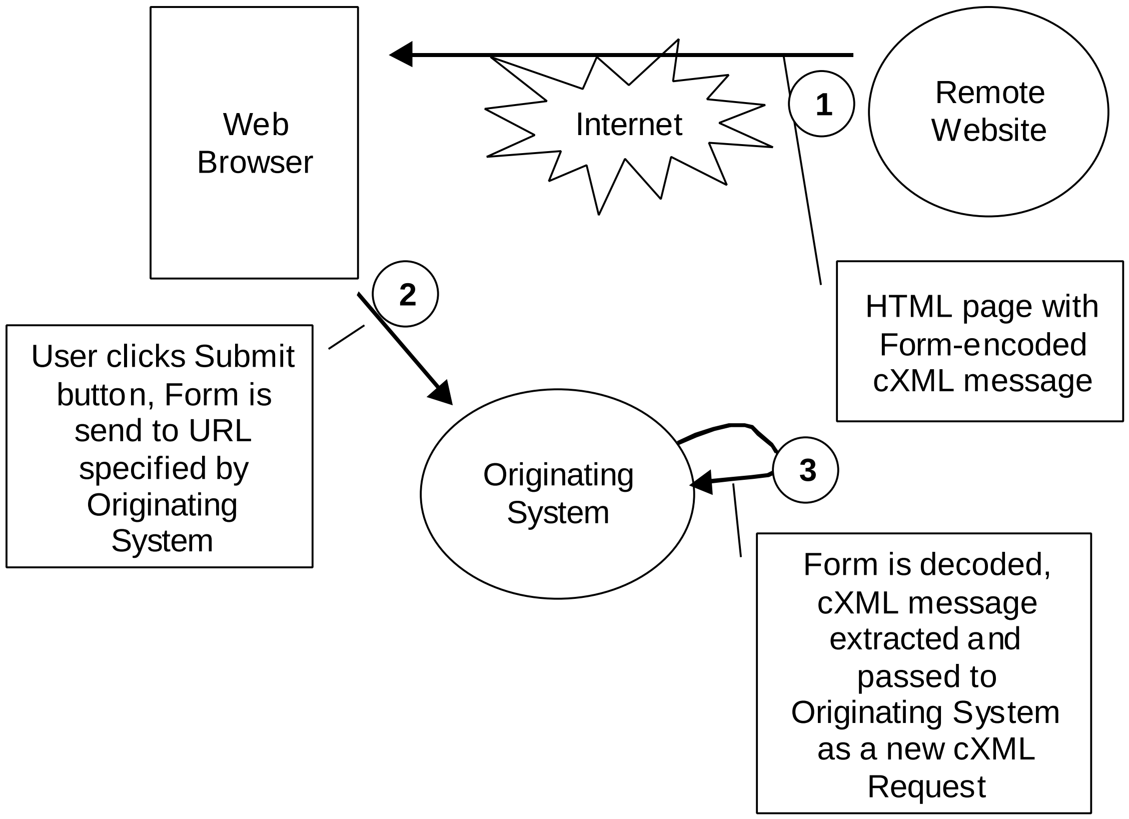
This transport is similar to the first part of the Request-Response cycle where the originator initiates an HTTP connection to the receiver. However, instead of waiting for a Response to come back in the HTTP channel, the originator simple transmits the cXML Message as a POST and closes the connection.

HTTP is the preferred channel for One-Way Message transmission if it is available.

2.6.2.2 URL-Form-Encoding

This transport is best understood by examining how the PunchOutOrderMessage transaction is performed. URL-Form-Encoding is used to enable integration between remote Web site and the originating system. It also serves as a way to bypass the requirements of having a listening server on the originating system that is directly accessible through the Internet.

The PunchOutOrderMessage cXML message is not directly sent to the originating system by the remote Web site, but is encoded as a hidden HTML Form field and then posted to the URL specified in the `BrowserFormPost` element of the PunchOutSetupRequest. This permits the remote Web site to display a check out Web page. When the user clicks on the check out button, the data is presented to the originating system as an HTML Form Submit. The following diagram shows what happens:


Web Browser
Internet
Remote Website
1
HTML page with Form-encoded cXML message
2
User clicks Submit button, Form is send to URL specified by Originating System
Originating System
3
Form is decoded, cXML message extracted and passed to Originating System as a new cXML Request

The exact packing and unpacking semantics are desscribed below.

2.6.2.2.1 Form Packing

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

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

There would of course be additional HTML tags on the page containing the above fragment, which might be describing the contents of the shopping basket in detail, etc.

2.6.2.2.2 Form Unpacking and Processing

The originating system, which had provided the appropriate URL previously, would receive an HTML Form POST containing the Form data as described above. The Form POST processor would look for the `cXML-urlencoded` variable, extract the URL-Encoded cXML message, URL-Decode it and then process it as if it had been received through a normal HTTP Request/Response cycle. The primary difference in this case is that there is no Response that can be generated, because there is no HTTP connection on which to send it.

3 Basic and Common Elements

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

3.1 Type Entities

These definitions are from the XML-Data note submission to the W3C (refer to `Common.mod` for the list and the URLs for the source XML-data definitions). A few higher-level type entities that are also defined here are not from XML-Data.

3.1.1 isoLangCode

This entity is an ISO Language Code from the ISO 639 standard.

3.1.2 unitOfMeasure

This entity is a unit of measure definition from the UN/CEFACT (Recommendation 20) standard.

3.1.3 URL

This entity is a URL as defined by the HTTP/1.1 standard.

3.2 Base elements

Elements used throughout the specification, which range from generic ones like `Name` and `Extrinsic`, to specific ones like `Money`.

Each is described with a detailed header comment in `Base.mod`. Refer to it for details.

4 Order Definitions

The cXML ordering documents are OrderRequest and OrderResponse. OrderRequest is analogous to a Purchase Order (PO). OrderResponse is the acknowledgement that the supplier received your PO. It is not a commitment to execute the PO, but confirmation that it was correctly received.

4.1 OrderRequest

An `OrderRequest` element looks like the following:

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

The details of the `OrderRequestHeader` and the `ItemOut` elements have been hidden to show the structure of an `OrderRequest`.

4.1.1 OrderRequestHeader

The following example shows an `OrderRequestHeader` in full detail:

```
<OrderRequestHeader orderID="DO1234" orderDate="1999-03-12"
requestedDeliveryDate="1999-03-24" type=”new”>
    <Total>
        <Money currency="USD">12.34</Money>
    </Total>
    <ShipTo>
        <Address>
            <Name xml:lang="en">Cisco Corporation</Name>
            <PostalAddress name="foo">
                <DeliverTo>Joe Smith</DeliverTo>
                <DeliverTo>Mailstop M-543</DeliverTo>
                <Street>123 Anystreet</Street>
                <City>Sunnyvale</City>
                <State>CA</State>
                <PostalCode>90489</PostalCode>
                <Country>US</Country>
            </PostalAddress>
        </Address>
    </ShipTo>
    <BillTo>
        <Address>
            <Name xml:lang="en">Cisco Corporation</Name>
            <PostalAddress name="foo">
                <Street>123 Anystreet</Street>
                <City>Sunnyvale</City>
                <State>CA</State>
                <PostalCode>90489</PostalCode>
                <Country>US</Country>
            </PostalAddress>
        </Address>
    </BillTo>
    <Shipping>
        <Money currency="USD">12.34</Money>
        <Description xml:lang="en-us">FedEx 2-day</Description>
    </Shipping>
    <Tax>
        <Money currency="USD">12.34</Money>
        <Description xml:lang="en">CA State Tax</Description>
    </Tax>
    <Payment>
        <PCard number="1234567890123456" expiration="1999-03-
12"/>
    </Payment>
    <Comments>Anything well formed in XML can go here.</Comments>
</OrderRequestHeader>
```

This element has the following attributes:

OrderID	The identifier for this order. What the PO Number is today.
OrderDate	The date and time this order was placed, in ISO 8601 format.
RequestedDeliveryDate	The date and time this order is requested for delivery, in ISO 8601 format.
Type	Type of the request – new or cancel.

4.1.1.1 Total

This element contains the total amount of the order. It is a container for the `Money` element defined in `Base.mod`.

4.1.1.2 ShipTo/BillTo

These elements contain the addresses of the Ship To and Bill To entities on the `OrderRequest`. These are `Address` elements as specified in the `Base.mod`.

4.1.1.3 Shipping

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

4.1.1.4 Tax

This element contains the tax associated with the order. This element is present if the buying organization computes tax.

4.1.1.5 Payment

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

4.1.2 ItemOut

The following example shows a fully valid minimal ItemOut element.

```
<ItemOut quantity="1" requestedDeliveryDate="1999-03-12">
    <ItemID>
        <SupplierPartID>5555</SupplierPartID>
    </ItemID>
</ItemOut>
```

The attributes on an ItemOut are the following:

quantity	The number of items desired.
requestedDeliveryDate	The time and date this item is requested for delivery, which allows item-level delivery dates in the `OrderRequest`. It must be in ISO 8601 format.

The following example shows a more complicated `ItemOut`.

```
<ItemOut quantity="2" requestedDeliveryDate="1999-03-12">
    <ItemID>
        <SupplierPartID>1233244</SupplierPartID>
    </ItemID>
    <ItemDetail>
        <UnitPrice>
            <Money currency="USD">1.34</Money>
        </UnitPrice>
        <Description xml:lang="en">hello</Description>
        <UnitOfMeasure>EA</UnitOfMeasure>
        <Classification domain="SPSC">12345</Classification>
        <ManufacturerPartID>234</ManufacturerPartID>
        <ManufacturerName>foobar</ManufacturerName>
        <ManufacturerURL>www.bar.com</ManufacturerURL>
        <SupplierURL>www.foo.com</SupplierURL>
    </ItemDetail>
    <ShipTo>
        <Address>
            <Name xml:lang="en">Cisco Corporation</Name>
            <PostalAddress name="foo">
                <Street>123 Anystreet</Street>
                <City>Sunnyvale</City>
                <State>CA</State>
                <PostalCode>90489</PostalCode>
                <Country>US</Country>
            </PostalAddress>
        </Address>
    </ShipTo>
    <Shipping>
        <Money currency="USD">1.34</Money>
        <Description xml:lang="en-us">FedEx 2-day</Description>
    </Shipping>
    <Tax>
        <Money currency="USD">1.34</Money>
        <Description xml:lang="en">foo</Description>
    </Tax>
    <Distribution>
        <Accounting name="DistributionCharge">
            <Segment type="G/L Account" id="23456"
description="Entertainment"/>
            <Segment type="Cost Center" id="2323"
description="Western Region Sales"/>
        </Accounting>
        <Charge>
            <Money currency="USD">.34</Money>
        </Charge>
    </Distribution>
    <Distribution>
        <Accounting name="DistributionCharge">
            <Segment type="G/L Account" id="456"
description="Travel"/>
            <Segment type="Cost Center" id="23"
description="Europe Implementation"/>
        </Accounting>
        <Charge>
            <Money currency="USD">1</Money>
        </Charge>
    </Distribution>
    <Comments>Anything well formed in XML can go here.</Comments>
</ItemOut>
```

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

The `ShipTo`, `Shipping` and `Tax` elements are identical to the ones that can be in the `OrderRequestHeader`. This allows per item shipping, shipping type and associated cost, and associated tax to be represented.

4.1.2.1 Distribution

A distribution represents a way to charge the cost of a given item in multiple ways. It models the notion of multiple entities buying the same set of items, but wanting to accurately represent who pays for what. The most likely use of the `Distribution` element would be for the supplier to return it on an Invoice to facilitate the buyer's reconciliation process.

4.1.2.1.1 Accounting

This element groups Segments to identify who will be charged. It has the following attribute:

name	The name for this accounting combination.

Segments have the following attributes:

type	An identifying name for this Segment with respect to the others in the `Accounting` element.
id	The unique identifier within this Segment type. This might be the actual account code if the type were "Cost Center".
description	A user level description of the id. This might be " North American Sales" in the "Cost Center" example above.

4.1.2.1.2 Charge

The amount to be charged to the entity represented by the `Accounting` element.

4.2 OrderResponse

This is the Response part of the synchronous Request-Response transaction. The following example shows an OrderResponse:

```
<cXML version="1.0" payloadID="9949494" timestamp="1999-03-
12T18:39:09-08:00">
    <Response>
        <Status code="200" text="OK"/>
    </Response>
</cXML>
```

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

The OrderResponse allows the Requestor to know their OrderRequest was successfully parsed and acted on by the remote part of HTTP connection. It does not communicate order level acknowledgement such as which items can be shipped, or which need to be backordered.

5 PunchOut Definitions

The PunchOut messages definitions are the Request/Response messages that are carried inside the `Request` and `Response` elements. All of the following messages must be implemented to support the PunchOut specification for cXML.

5.1 PunchOutSetup*

These elements are the Request/Response pair used to set up a PunchOut request to a remote system. They identify the originating system, send setup information, and receive a response indicating where to go to initiate an HTML browsing session on the remote Web site.

5.1.1 PunchOutSetupRequest

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

```
<PunchOutSetupRequest operation="create">
    <BuyerCookie>34234234ADFSDF234234</BuyerCookie>
    <Extrinsic name="department">Marketing</Extrinsic>
    <BrowserFormPost>
        <URL>http://ariba.cisco.com:1616/punchoutexit</URL>
    </BrowserFormPost>
</PunchOutSetupRequest>
```

The attribute on the PunchOutSetupRequest element is:

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

This element also contains the following elements: `BuyerCookie`, `Extrinsic`, and `BrowserFormPost`.

5.1.1.1 BuyerCookie

This element transmits information that is opaque to the remote Web site, but must be returned to the originator for all subsequent PunchOut operations. This element is used to allow the originating system to match multiple outstanding PunchOut requests.

5.1.1.2 BrowserFormPost

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

5.1.1.3 Extrinsic

This element is optional and is used for any additional data that the requestor wants to pass to the external Web site. This example passes the department of the user initiating the PunchOut operation. It is important to realize that the cXML specification says nothing about this Extrinsic element—it is something that each requestor and remote Web site would need to agree on and implement.

5.1.2 PunchOutSetupResponse

After the remote Web site has received a PunchOutSetupRequest, it needs to respond with a PunchOutSetupResponse, as shown below:

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

5.1.2.1 StartPage

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

At this point, the user who initiated the PunchOutSetupRequest browses the external Web site, and selects items to be transferred back to the originating system through a PunchOutOrderMessage.

5.2 PunchOutOrderMessage

This element sends the contents of the remote shopping basked to the originator of a PunchOutSetupMessage. It can contain much more data than the other messages because it needs to be able to fully express the contents of any conceivable shopping basket on the external Web site. This message does not strictly follow the Request/Response paradigm; how it is used will be explained in detail.

A PunchOutOrderMessage is generated when the user browsing the remote Web site is ready to check out or transfer the information obtained there to their originating system. The data present in the remote shopping basket is then transferred; for example:

```
<PunchOutOrderMessage>
    <BuyerCookie>34234234ADFSDF234234</BuyerCookie>
    <PunchOutOrderMessageHeader operationAllowed="none">
        <Total>
            <Money currency="USD">100.23</Money>
        </Total>
    </PunchOutOrderMessageHeader>
    <ItemIn quantity="1">
        <ItemID>
            <SupplierPartID>1234</SupplierPartID>
            <SupplierPartAuxiliaryID>
                additional data about this item
            </SupplierPartAuxiliaryID>
        </ItemID>
        <ItemDetail>
            <UnitPrice>
                <Money currency="USD">10.23</Money>
            </UnitPrice>
            <Description xml:lang="en">
                Learn ASP in a Week!
            </Description>
            <UnitOfMeasure>EA</UnitOfMeasure>
            <Classification domain="SPSC">12345</Classification>
</ItemDetail>
    </ItemIn>
</PunchOutOrderMessage>
```

There are many elements listed above. The following sections discuss them in detail.

5.2.1 BuyerCookie

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

5.2.2 PunchOutOrderMessageHeader

This element contains information about the entire shopping basket contents being transferred. The only required element is `Total`, which is the number of items being added to the requisition. Additional elements that are allowed are `Shipping` and `Tax`, which are the amount and description of any shipping or tax charges computed on the remote Web site. `ShipTo` is also optional, and it specifies the ShipTo addressing information the user selected on the remote site. All monetary amounts are in a special `Money` element that always specifies currency in a standardized format. See the `Base.mod` file for the exact definition.

The only attribute allowed is:

operationAllowed	Specifies the type of PunchOutSetupRequest operations that are allowed: “create”, “inspect”, and “edit”. Only “create” is examined in this document.

5.2.3 ItemIn

This element adds an item from a shopping basket to a requisition on the originating system. It can contain a variety of elements, only two of which are required: `ItemID` and `ItemDetail`. It defines the following attribute:

quantity	Number of item selected by the user on the remote Web site.

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

5.2.3.1 ItemID

This element uniquely identifies the item in a way that the remote Web site will understand. It is the only element that is required to be returned to the remote Web site to re-identity the item being transferred.

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

5.2.3.2 ItemDetail

This element contains descriptive information about the item that the originating system can present to the user. The contents of an ItemDetail element can be quite complex, but the minimum requirements are simple: `UnitPrice`, `Description`, `UnitOfMeasure`, and `Classification`. See `Item.mod` for details about the required and optional elements.

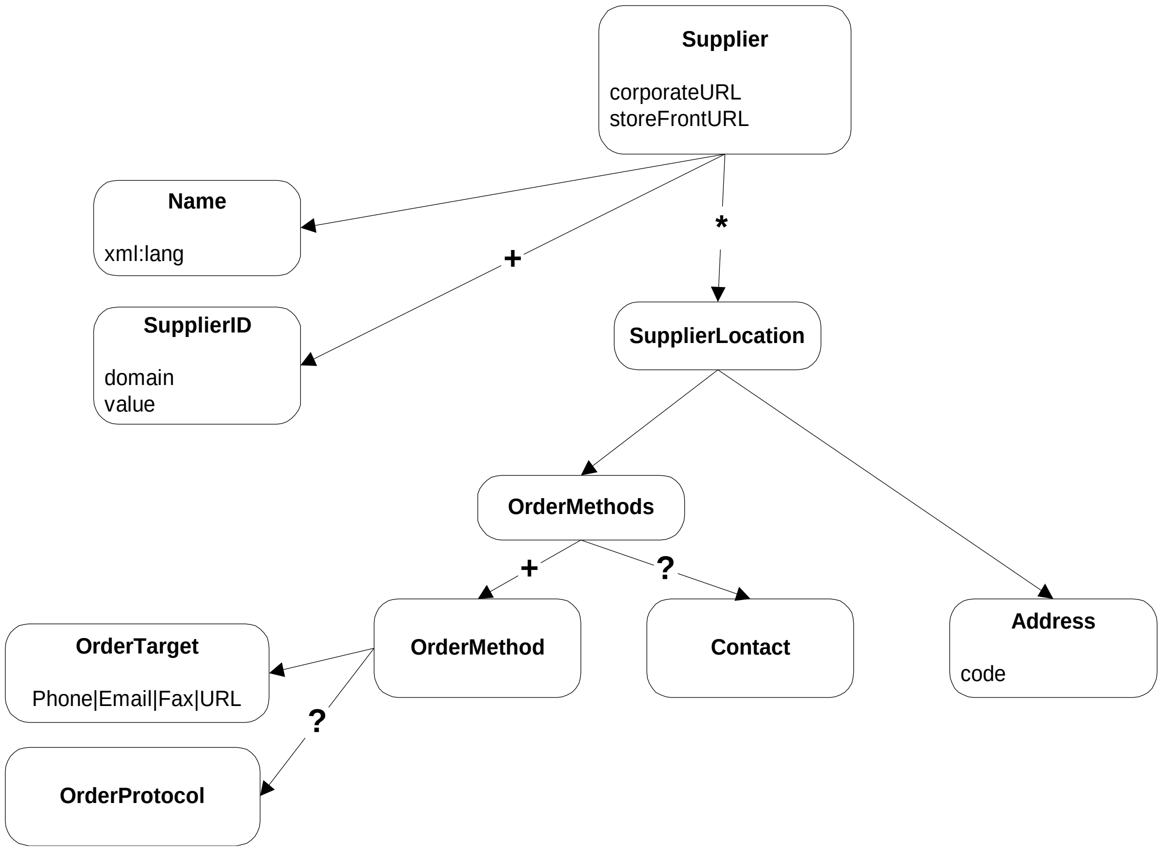
6 Catalog Definitions

The cXML Catalog definitions consist of three main elements: `Supplier`, `Index`, and `Contract`. All three elements describe data intended for persistent or cached use within a buyer's system.

The `Supplier` element describes data about the supplier the buyer might need to know (address, contact, and ordering information). The `Index` element describes data about the supplier's inventory of goods and services (such as description, part numbers, and classification codes). And, the `Contract` element describes data about flexible aspects of the inventory negotiated between the buyer and supplier, such as price. Note that the `Index` uses several sub-elements to describe aspects of line items in the supplier's inventory. The supplier can send either price information for caching within the buyers system or PunchOut information to enable the buyer to punch-out to the supplier's web-site for pricing and other information.

6.1 Supplier

The `Supplier` element encapsulates a named supplier of goods and services. It must have a `Name` element and at least one `SupplierID` that specifies how this supplier is known. It additionally describes optional address and ordering information for the supplier:



Its attributes are described in the following table.

corporateURL	URL for supplier's Web site.
storeFrontURL	URL for Web site for shopping or browsing.

The following example shows `Supplier` (with details omitted for clarity):

```
<Supplier>
    <SupplierID domain="InternalSupplierID">29</SupplierID>
    <SupplierLocation>
        <Address>
            <Name xml:lang="en-US">Main Office</Name>
            <PostalAddress>
                …
            </PostalAddress>
            <Email>bobw@globalcorp.com</Email>
            <Phone name="Office">
                …
            </Phone>
            <Fax name="Order">
                …
            </Fax>
            <URL>http://www.bigcorp.com/Support.htm</URL>
        </Address>
       <OrderMethods>
            <OrderMethod>
                <OrderTarget>
                    <URL>http://www.bigcorp.com/cxmlorders</URL>
                </OrderTarget>
            </OrderMethod>
            <Contact>
                <Name xml:lang="en-US">Mr. Smart E. Pants</Name>
                <Email>sepants@bigcorp.com</Email>
                <Phone name="Office">
                     …
                </Phone>
            </Contact>
        </OrderMethods>
    </SupplierLocation>
</Supplier>
```

6.1.1 SupplierLocation

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

6.1.1.1 OrderMethods and OrderMethod

The `OrderMethods` element is a grouping of one or more `OrderMethod` elements for the given `SupplierLocation` element. The ordering of the `OrderMethods` in the list is significant – the first element is considered the preferred ordering method and so on in decreasing order of preference.

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

6.2 Index

This element is the root element used for updating the inventory of goods and services in the buyers system.

An Index element is associated with a *single* supplier. The Index element allows for a list of one or more (at least one is required) supplier Ids, where each ID is considered a synonym for the same supplier.

The Index contains one or more IndexItem elements as well as an optional set of SearchGroup elements describing parametric searches for the index. The IndexItem element contains elements indicating what to add or delete from the buyer's cached representation of the supplier's inventory. The general outline of an Index element follows:

```
<Index>
    <SupplierID> … </SupplierID>
    <IndexItem>
        <IndexItemAdd>
            <IndexItemDetail>
                …
            </IndexItemDetail>
        </IndexItemAdd>
            …
        <IndexItemDelete>
            …
        </IndexItemDelete>
            …
        <IndexItemPunchOut>
            …
        </IndexItemPunchOut>
    </IndexItem>
</Index>
```

6.2.1 IndexItem, IndexItemAdd, IndexItemDelete, and IndexItemPunchOut

The `IndexItem` element is a container for the list of items in an index. There are three types of child item elements, one or more of which can be specified within an `IndexItem`—

`IndexItemAdd`, `IndexItemDelete`, and `IndexItemPunchOut`. These children contain specific detail elements depending on what data is required for add, delete, or punchout:

- The `IndexItemAdd` element inserts a new item or updates an existing item in the index. It contains exactly one `ItemID` element, one `ItemDetail` element, and one `IndexItemDetail` element.
- The `IndexItemDelete` element removes an item from the index, and it contains a single `ItemID` element identifying the item.
- The `IndexItemPunchout` element identifies an item that can be used to dynamically connect an index item to the supplier's resource for that item. It consists of a `PunchOutDetail` element and an `ItemID` element. It is similar to an IndexItemAdd element except that it does not require price information—the intent is that the buyer gets that information real time through the supplier's Web site.

6.2.1.1 ItemID

The `ItemID` element enables suppliers to uniquely identify the items they sell. It is composed of a `SupplierPartID` element and an optional `SupplierPartAuxiliaryID` element.

6.2.1.1.1 SupplierPartAuxiliaryID

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

`SupplierPartAuxiliaryID` could also be an opaque way for the supplier to refer to complex configuration or part data. It could contain all the data necessary for the supplier to reconstruct what the item in question is in their computer system (a basket or cookie of data that makes sense only to the supplier). This is the use referred to above in the discussion of PunchOut.

6.2.1.2 ItemDetail

`ItemDetail` contains detailed information about an item, or all the data that a user might want to see about an item beyond the essentials that are represented in the `ItemID`. It must contain a `UnitPrice`, a `UnitOfMeasure`, a `Description`, and a `Classification`, and it can optionally contain a `ManufacturerPartID`, a `ManufacturerName`, a `ManufacturerURL`, a `SupplierURL`, and any number of `Extrinsic` elements.

6.2.1.3 IndexItemDetail

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

6.3 Contract

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

The attributes that describe the effective and expiration dates for the contract are:

effectiveDate	Effective date of the contract.
expirationDate	Expiration date of the contract.

The contract element contains one or more ItemSegment elements. An example follows:

```
<Contract>
    <SupplierID domain="InternalSupplierID">29</SupplierID>
    <ItemSegment segmentKey=Plant12>
        <ContractItem>
            <ItemID>
                <SupplierPartID>pn12345</SupplierPartID>
            </ItemID>
            <UnitPrice>
                <Money currency=USD>40.00</Money>
            </UnitPrice>
        <ContractItem>
        …
     </ItemSegment>
 </Contract>
```

6.3.1 ItemSegment

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

segmentKey	Agreed-upon string used to segment custom prices.

6.3.2 ContractItem

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

7 Subscription Management Definitions

An intermediary, such as Ariba Network, can manage the suppliers and supplier catalogs that a buyer's system uses. Such an intermediary can provide a direct link between the buyer's system and the suppliers and/or their systems. This section contains element definitions for managing supplier data and catalog contents. These definitions build on many of the previous definitions for cXML request/responses, one-way messages and catalog definitions.

7.1 Supplier Data

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

7.1.1 SupplierListRequest

SupplierListRequest gets a list of the suppliers with whom the buyer has enabled to have an active relationship.

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

7.1.2 SupplierListResponse

SupplierListResponse contains a list of the buyer's currently enabled suppliers.

```
<Response>
  <Status code="200" text="OK"/>
  <SupplierListResponse>
     <Supplier corporateURL="http://www.dummy.com"
storeFrontURL="http://www.buydummy.com">
        <Name xml:lang="en-US">A Dummy Company </Name>
        <Comments xml:lang="en-US">this is a cool company</Comments>
        <SupplierID domain="DUNS">123456</SupplierID>
     </Supplier>
     <Supplier corporateURL="http://www.dummy2.com"
storeFrontURL="http://www.buydummy2.com">
         <Name xml:lang="en-US">A Dummy Company - 2 </Name>
         <Comments xml:lang="en-US">this is a cool company</Comments>
         <SupplierID domain="DUNS">123456789</SupplierID>
      </Supplier>
   </SupplierListResponse>
</Response>
```

7.1.3 SupplierDataRequest

SupplierDataRequest requests data about a supplier.

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

7.1.4 SupplierDataResponse

SupplierDataResponse contains data about a supplier.

```
<Response>
    <Status code="200" text="OK"/>
    <SupplierDataResponse>
        <Supplier corporateURL="http://www.dummy.com"
storeFrontURL="http://www.buydummy.com">
            <Name xml:lang="en-US">A Dummy Company </Name>
            <Comments xml:lang="en-US">this is a cool company</Comments>
            <SupplierID domain="DUNS">123456</SupplierID>
            <SupplierLocation>
               <Address>
                   <Name xml:lang="en-US">Main Office</Name>
                   <PostalAddress>
                        <DeliverTo>Bob A. Worker</DeliverTo>
                        <Street>123 Front Street</Street>
                        <City>Toosunny</City>
                        <State>CA</State>
                        <PostalCode>95000</PostalCode>
                        <Country>US</Country>
                   </PostalAddress>
                   <Email>bobw@dummy.com</Email>
                   <Phone name="Office">
                         <TelephoneNumber>
                             <CountryCode isoLangCode="US">011</CountryCode>
                             <AreaOrCityCode>800</AreaOrCityCode>
                             <Number>555-1212</Number>
                         </TelephoneNumber>
                   </Phone>
                   <Fax name="Order">
                        <TelephoneNumber>
                            <CountryCode isoLangCode="US">011</CountryCode>
                            <AreaOrCityCode>408</AreaOrCityCode>
                            <Number>555-1234</Number>
                        </TelephoneNumber>
                   </Fax>
                   <URL>http://www.dummy.com/Support.htm</URL>
               </Address>
               <OrderMethods>
                   <OrderMethod>
                        <OrderTarget>
                             <URL>http://www.dummy.com/cxmlorders</URL>
                        </OrderTarget>
                        <OrderProtocol>cXML</OrderProtocol>
                   </OrderMethod>
               </OrderMethods>
            </SupplierLocation>
        </Supplier>
    </SupplierDataResponse>
</Response>
```

7.1.5 SupplierChangeMessage

This element contains a message used to notify of changes in supplier data.

```
<Message>
    <SupplierChangeMessage type="new">
         <Supplier corporateURL="http://www.dummy.com"
storeFrontURL="http://www.buydummy.com">
            <Name xml:lang="en-US">A Dummy Company </Name>
            <Comments xml:lang="en-US">this is a cool company</Comments>
            <SupplierID domain="DUNS">123456</SupplierID>
            <SupplierLocation>
               <Address>
                   <Name xml:lang="en-US">Main Office</Name>
                   <PostalAddress>
                        <DeliverTo>Bob A. Worker</DeliverTo>
                        <Street>123 Front Street</Street>
                        <City>Toosunny</City>
                        <State>CA</State>
                        <PostalCode>95000</PostalCode>
                        <Country>US</Country>
                   </PostalAddress>
                   <Email>bobw@dummy.com</Email>
                   <Phone name="Office">
                         <TelephoneNumber>
                             <CountryCode isoLangCode="US">011</CountryCode>
                             <AreaOrCityCode>800</AreaOrCityCode>
                             <Number>555-1212</Number>
                         </TelephoneNumber>
                   </Phone>
                   <Fax name="Order">
                        <TelephoneNumber>
                            <CountryCode isoLangCode="US">011</CountryCode>
                            <AreaOrCityCode>408</AreaOrCityCode>
                            <Number>555-1234</Number>
                        </TelephoneNumber>
                   </Fax>
                   <URL>http://www.dummy.com/Support.htm</URL>
               </Address>
               <OrderMethods>
                   <OrderMethod>
                        <OrderTarget>
                             <URL>http://www.dummy.com/cxmlorders</URL>
                        </OrderTarget>
                        <OrderProtocol>cXML</OrderProtocol>
                   </OrderMethod>
               </OrderMethods>
            </SupplierLocation>
        </Supplier>
    </SupplierChangeMessage>
</Message>
```

7.2 Catalog Subscriptions

The definitions for Catalog-subscription management consist mainly of the elements Subscription, SubscriptionListRequest, SubscriptionListResponse, SubscriptionContentRequest, SubscriptionContentResponse, and SubscriptionChangeMessage. These are described below with examples where the intermediary is Ariba Network.

7.2.1 Subscription

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

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

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

7.2.2 SubscriptionListRequest

This element requests the current list of catalog subscriptions created at an intermediary for a buyer.

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

7.2.3 SubscriptionListResponse

This element contains the current set of subscriptions for the buyer.

```
<Response>
        <Status code="200" text="OK"/>
        <SubscriptionListResponse>
            <Subscription>
                <InternalID>1234</InternalID>
                <Name xml:lang="en-US">Q2 Prices</Name>
                <Changetime>1999-03-12T18:39:09-08:00</Changetime>
                <SupplierID domain="DUNS">123456789</SupplierID>
                <Format version="2.1">CIF</Format>
                <Description xml:lang="en-US">The best prices for
software</Description>
            </Subscription>
            <Subscription>
                <InternalID>1235</InternalID>
                <Name xml:lang="en-US">Q2 Software Prices</Name>
                <Changetime>1999-03-12T18:15:00-08:00</Changetime>
                <SupplierID domain="DUNS">555555555</SupplierID>
                <Format version="2.1">CIF</Format>
                <Description xml:lang="en-US">The best prices for
software</Description>
            </Subscription>
        </SubscriptionListResponse>
</Response>
```

7.2.4 SubscriptionContentRequest

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

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

7.2.5 SubscriptionContentResponse

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

```
<Response>
        <Status code="200" text="OK"/>
        <SubscriptionContentResponse>
            <Subscription>
                <InternalID>1234</InternalID>
                <Name xml:lang="en-US">Q2 Prices</Name>
                <Changetime>1999-03-12T18:39:09-08:00</Changetime>
                <SupplierID domain="DUNS">123456789</SupplierID>
                <Format version="3.0">CIF</Format>
                <Description xml:lang="en-US">The best prices for
software</Description>
            </Subscription>
            <SubscriptionContent filename="foobar.cif">
                <CIFContent>
                    <!-- base64 encoded data -->
                    ABCDBBDBDBDBDB
                </CIFContent>
            </SubscriptionContent>
         </SubscriptionContentResponse>
     </Response>
```

7.2.6 SubscriptionChangeMessage

This element signals the buyer's system that a catalog it has subscribed to has changed.

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

It has the following attribute:

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

8 Message Retrieval Definitions

Buyer systems are sometimes designed so they do not have an HTTP entry point residing outside the firewall for receiving cXML messages. The cXML specifications allow for this limitation; for example, PunchOut definitions consider this.

This section introduces definitions that allow the source system to queue messages when the target is unable to directly accept an HTTP post and let the target pull the messages at its convenience.

8.1 GetPendingRequest

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

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

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

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

8.2 GetPendingResponse

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

```
<Response>
        <Status code="200" text="OK"/>
        <GetPendingResponse>
                <cXML version="1.0"
                      payloadID="456778@ariba.com"
                      timestamp="1999-03-12T18:39:09-08:00">
                   <Header>
                        <From>
                                <Credential domain="AribaNetworkUserId">
                                        <Identity>admin@ariba.com</Identity>
                                </Credential>
                        </From>
                        <To>
                                <Credential domain="AribaNetworkUserId">

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

9 cXML XML Documents

This section contains all the XML documents that comprise the cXML definition.

9.1 cXML.dtd

```
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/cXML.dtd#4 $
-->

<!-- Imports -->
<!ENTITY % defineCommonModule SYSTEM "Common.mod">
%defineCommonModule;

<!ENTITY % defineBaseModule SYSTEM "Base.mod">
%defineBaseModule;

<!ENTITY % defineSupplierModule SYSTEM "Supplier.mod">
%defineSupplierModule;

<!ENTITY % defineItemModule SYSTEM "Item.mod">
%defineItemModule;

<!ENTITY % defineTransactionModule SYSTEM "Transaction.mod">
%defineTransactionModule;

<!ENTITY % defineTransportModule SYSTEM "Transport.mod">
%defineTransportModule;

<!ENTITY % defineContractModule SYSTEM "Contract.mod">
%defineContractModule;

<!ENTITY % defineIndexModule SYSTEM "Index.mod">
%defineIndexModule;

<!ENTITY % definePendingModule SYSTEM "Pending.mod">
%definePendingModule;

<!ENTITY % defineSubscriptionModule SYSTEM "Subscription.mod">
%defineSubscriptionModule;

<!-- Done with Imports all Elements/Entities are now defined -->
```

9.2 Common.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Common.mod#10 $
-->

<!--
    Common types used throughout the cXML definiton.

    The types try to follow the XML DATA definition submitted to the W3C. See
    the following for more information,

        http://msdn.microsoft.com/xml/reference/schema/datatypes.asp
        http://www.w3c.org/TR/1998/NOTE-XML-data-0105/

-->

<!-- Imports are NOT allowed in .mod files -->

<!-- Atomic-level Types -->
<!ENTITY % bin.base64 "CDATA">
<!ENTITY % bin.hex "CDATA">
<!ENTITY % boolean "(0 | 1)"> <!-- 0 is false, 1 is true -->
<!ENTITY % char "CDATA">
<!ENTITY % date "CDATA">
<!ENTITY % datetime.tz "CDATA">
<!ENTITY % fixed.14.4 "CDATA">
<!ENTITY % i8 "CDATA">
<!ENTITY % int "%i8;">
<!ENTITY % r8 "CDATA">
<!ENTITY % number "%r8;">
<!ENTITY % string "CDATA">
<!ENTITY % time.tz "CDATA">
<!ENTITY % ui8 "CDATA">
<!ENTITY % uint "%ui8;">
<!ENTITY % uri "CDATA">
<!ENTITY % uuid "CDATA">

<!-- Higher-level Types -->
<!ENTITY % isoLangCode "CDATA">
<!ENTITY % isoCountryCode "CDATA">
<!ENTITY % URL "%uri;">
```

9.3 Base.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Base.mod#19 $
-->

<!-- Imports are NOT allowed in .mod files -->

<!--
    This file defines the basic elements used to build higher level
    constructs in cXML.
-->

<!-- Basic Name/Data Elements -->
<!--
    Name is used to provide an identifier for other elements.

    xml:lang
        ISO 639 language identifiers
-->
<!ELEMENT Name (#PCDATA)> <!-- string -->
<!ATTLIST Name
    xml:lang  %isoLangCode;  #REQUIRED
>

<!--
    Value is used to represent #PCDATA values in other elements
-->
<!ELEMENT Value (#PCDATA)> <!-- string -->

<!--
    An Extrisic is an element which can be used to extend the data
    associated with known elements.

    Since this Element is of type ANY, it could contain any arbitrary XML
    document within itself, or a binary ![CDATA[]] document.

    name
        Name used to identify this extrinsic.
-->
<!ELEMENT Extrinsic ANY>
<!ATTLIST Extrinsic
    name  %string;  #REQUIRED
>

<!--
    Description is a string which describes something.

    xml:lang
        an ISO 639 code representing the language in which the description is
        written
-->
<!ELEMENT Description (#PCDATA)> <!-- string -->
<!ATTLIST Description
    xml:lang  %isoLangCode;  #REQUIRED
```

```
>

<!-- Telephone Number Elements -->
<!--
    International ITU dial code for the country code in question.

    isoCountryCode
        The ISO 3166 country code for the dial code in question
-->
<!ELEMENT CountryCode (#PCDATA)> <!-- uint -->
<!ATTLIST CountryCode
    isoCountryCode  %isoCountryCode;  #REQUIRED
>

<!--
    The areacode or city code within a CountryCode.
-->
<!ELEMENT AreaOrCityCode (#PCDATA)> <!-- uint -->

<!--
    The local number part of a telephone number.
-->
<!ELEMENT Number (#PCDATA)> <!-- uint -->

<!--
    An extension within relative to the Number element. This element has no
    meaning without an associated Number element.
-->
<!ELEMENT Extension (#PCDATA)> <!-- uint -->

<!--
    TelephoneNumber represents international telephone numbers.a
-->
<!ELEMENT TelephoneNumber (CountryCode, AreaOrCityCode, Number, Extension?)>

<!--
     Phone is a "named" TelephoneNumber.

     name
          specifies an identifier which indicates the type of phone number. US
          examples would include "work","home", etc.
-->
<!ELEMENT Phone (TelephoneNumber)>
<!ATTLIST Phone
    name  %string;  #IMPLIED
>

<!--
    Fax number.
-->
<!ELEMENT Fax (TelephoneNumber | URL | Email)>
<!ATTLIST Fax
    name  %string;  #IMPLIED
>

<!-- Addressing Elements -->
<!--
    URL. A string which represents a URL
-->
<!ELEMENT URL (#PCDATA)> <!-- uri -->
```

```
<!ATTLIST URL
    name  %string;  #IMPLIED
>

<!--
    An email address. Address must conform to RFC 821 (SMTP Standard).
-->
<!ELEMENT Email (#PCDATA)> <!-- string -->
<!ATTLIST Email
    name  %string;  #IMPLIED
>

<!--
    Contact represents and entity at a location. The nature of this
    element is that it represents a communication "end point" for a
    location.
-->
<!ELEMENT Contact (Name, PostalAddress*, Email*, Phone*, Fax*, URL*)>

<!--
    The DeliverTo part of an Address. This would be internal to the actual
    address know to the outside world. Similar to what an extension is to a
    TelephoneNumber.
-->
<!ELEMENT DeliverTo (#PCDATA)> <!-- string -->

<!--
    Street is a single line of an Address' location.
-->
<!ELEMENT Street (#PCDATA)> <!-- string -->

<!--
    City is the name of the city in an Address' location.
-->
<!ELEMENT City (#PCDATA)> <!-- string -->

<!--
    State is an optional state identifier in an Address' location.
-->
<!ELEMENT State (#PCDATA)> <!-- string -->

<!--
    PostalCode (I have no idea how to describe it)
-->
<!ELEMENT PostalCode (#PCDATA)> <!-- string -->

<!--
    Country is the name of the country in an Address' location

    isoCountryCode
        The ISO 3166 2-letter country code.
-->
<!ELEMENT Country (#PCDATA)> <!-- string -->
<!ATTLIST Country
    isoCountryCode  %isoCountryCode;  #REQUIRED
>

<!--
    PostalAddress is a real-world location for a business or person.
-->
```

```
<!ELEMENT PostalAddress (DeliverTo*, Street+, City, State?, PostalCode?,
Country)>
<!ATTLIST PostalAddress
    name  %string;  #IMPLIED
>

<!--
    Address is the association of a Contact and an Location.

    isoCountryCode
        The ISO 3166 country code for the dial code in question

    addressID
        An id for the address. Needed to support address codes for
        relationships that require id references.
-->
<!ELEMENT Address (Name, PostalAddress?, Email?, Phone?, Fax?, URL?)>
<!ATTLIST Address
    isoCountryCode  %isoCountryCode;  #IMPLIED
    addressID       %string;          #IMPLIED
>

<!-- Financial Elements -->
<!--
    Money is the representation of the object used to pay for items.

    currency
        specifies the currency in which amount is stated, must conform to ISO
        4217 currency codes.

    alternateAmount
        the amount of money in the alternateCurrency. Optional and used to
        support dual-currency requirements such as the Euro.

    alternateCurrency
        specifies the currency in which the aleternateAmount is stated, must
        conform to ISO 4217 currency codes.
-->
<!ELEMENT Money (#PCDATA)> <!-- number -->
<!ATTLIST Money
    currency           %string;  #REQUIRED
    alternateAmount    %number;  #IMPLIED
    alternateCurrency  %string;  #IMPLIED
>

<!--
    Optional textual child for communicating arbitrary comments or
    description along with the parent.

    xml:lang
        an ISO 639 code representing the language in which the description is
        written
-->
<!ELEMENT Comments ANY>
<!ATTLIST Comments
    xml:lang  %isoLangCode;  #IMPLIED
>

<!--
    Price per unit of item.
```

```
-->
<!ELEMENT UnitPrice (Money)>
```

9.4 Supplier.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Supplier.mod#9 $
-->

<!-- Imports are NOT allowed in .mod files -->

<!--
    Supplier of goods and services. Includes a list of SupplierIDs which
    indentify the Supplier.

    corporateURL
        URL to web site about the supplier

    storeFrontURL
        URL to web site where a user can shop or browse
-->
<!ELEMENT Supplier (Name, Comments?, SupplierID+, SupplierLocation*)>
<!ATTLIST Supplier
    corporateURL   %URL;  #IMPLIED
    storeFrontURL  %URL;  #IMPLIED
>

<!--
    One of the locations for a supplier. Supplier location is
    generally a physical location.
-->
<!ELEMENT SupplierLocation (Address, OrderMethods)>

<!--
    OrderMethods is the list of methods by which one can order
    from a supplier. The contact element is the technical contact
    who should be able to assist with order processing issues.
    The list is to be ordered by supplier preference, the first
    element having the highest degree of preference.
-->
<!ELEMENT OrderMethods (OrderMethod+, Contact?)>

<!--
    OrderMethod is a method for ordering. It is comprised of a
    target address for the order and the protocol expected by
    the address.
-->
<!ELEMENT OrderMethod (OrderTarget, OrderProtocol?)>

<!--
    OrderTarget represents an address to which orders can be
    sent.
-->
<!ELEMENT OrderTarget (Phone | Email | Fax | URL | OtherOrderTarget)>

<!--
    OrderProtocol is the communication method to be used when
    communicating an order to a supplier. An example would be "cXML".
```

```
-->
<!ELEMENT OrderProtocol (#PCDATA)> <!-- string -->

<!--
    OtherOrderTarget represents an address which is not enumerated by
    default in the OrderTarget Element. This may contain address targets
    beyond the ability of this document to describe.

    name
        Optional name for target.
-->
<!ELEMENT OtherOrderTarget ANY>
<!ATTLIST OtherOrderTarget
    name  %string;  #IMPLIED
>

<!--
    Definition of a supplier id.  A supplier id is a (domain, value)
    pair so that suppliers have the flexibility to define their id's
    according to an arbitrary convention (e.g., (DUNS, 12345),
    (TaxID, 88888888)).

    domain
        the domain of the id
-->

<!ELEMENT SupplierID (#PCDATA)> <!-- string -->
<!ATTLIST SupplierID
    domain  %string;  #REQUIRED
>
```

9.5 Item.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Item.mod#14 $
-->

<!-- Imports are NOT allowed in .mod files -->

<!--
    Must be a UN/CEFACT (Recommendation 20) unit of measure code.
-->
<!ELEMENT UnitOfMeasure (#PCDATA)> <!-- string -->

<!--
    ID with which the item's manufacturer identifies the item.
-->
<!ELEMENT ManufacturerPartID (#PCDATA)> <!-- string -->

<!--
    Name of the item's manufacturer.
-->
<!ELEMENT ManufacturerName (#PCDATA)> <!-- string -->

<!--
    Classification is used to group items into similar categories.

    domain
        "name" of classification, ie., SPSC
-->
<!ELEMENT Classification (#PCDATA)> <!-- string -->
<!ATTLIST Classification
       domain %string;  #REQUIRED
>

<!--
    How the supplier identifies an item they sell.

    If SupplierPartID does not provide a unique key to identify the item,
    then the supplier should generate a key which identifies the part
    uniquely when combined with the SupplierID and SupplierPartID. The
    key is call SupplierPartAuxiliaryID.

    An example is where a Supplier would use the same PartID for an
    item but have a different price for units of "EA" versus "BOX".
    In this case, the ItemIDs should be:
    <ItemID>
       <SupplierPartID>pn12345</SupplierPartID>
       <SupplierPartAuxiliaryID>EA</SupplierPartAuxiliaryID>
    </ItemID>
    <ItemID>
       <SupplierPartID>pn12345</SupplierPartID>
       <SupplierPartAuxiliaryID><foo>well formed XML
here</foo></SupplierPartAuxiliaryID>
    </ItemID>
-->
```

```
<!ELEMENT SupplierPartID (#PCDATA)> <!-- string -->

<!ELEMENT SupplierPartAuxiliaryID ANY>

<!--
    A unique identification of an item. SupplierID is not required since
    ItemIDs never travel alone.
-->
<!ELEMENT ItemID (SupplierPartID, SupplierPartAuxiliaryID?)>

<!--
    ItemDetail contains detailed information about an item. All the data that
    a user would want to see about an item instead of the bare essentials that
    are represented in the ItemID.
-->
<!ELEMENT ItemDetail (UnitPrice, Description+, UnitOfMeasure,
                      Classification+, ManufacturerPartID?,
                      ManufacturerName?, URL?, Extrinsic*)>
```

9.6 Transaction.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Transaction.mod#21 $
-->

<!--
    For better definitions of these Elements/Entities, refer to the cXML
    Transaction Specification documents.
-->

<!-- Imports are NOT allowed in .mod files -->

<!-- Basic transactional elements used throughout -->
<!--
    The total for something.
-->
<!ELEMENT Total (Money)>

<!--
    The bill to for an item.
-->
<!ELEMENT BillTo (Address)>

<!--
    The ship to for a item.
-->
<!ELEMENT ShipTo (Address)>

<!--
    Definition of a cXML Shipping item. Represents a shipping cost in the
    shopping basket (PunchOutResponse) or an order to the supplier
    (SupplierOrder). There could be one of these for the entire order, or one
    per lineitem.

    trackingDomain
        represents the logistics supplier, I.E., "FedEx", "UPS", etc.

    trackingId
        an optional element value that represents the logistics supplier
        tracking number

    tracking
        Deprecated - Do Not Use
-->
<!ELEMENT Shipping (Money, Description)>
<!ATTLIST Shipping
    trackingDomain  %string;  #IMPLIED
    trackingId      %string;  #IMPLIED
    tracking        %string;  #IMPLIED
>

<!--
    Defines a Purchasing Card element used for payment
-->
```

```
<!ELEMENT PCard (PostalAddress?)>
<!ATTLIST PCard
    number      %uint;    #REQUIRED
    expiration  %date;    #REQUIRED
    name        %string;  #IMPLIED
>

<!--
    The list of valid payment types.
-->
<!ENTITY % cxml.payment  "PCard">
<!ELEMENT Payment (%cxml.payment;)>

<!--
    Defines an accounting segment.

    type
        The accounting type of this segment.

    id
        The unique key of this Segment against the type.

    description
        Textual description of the Segment. For human readability.
-->
<!ELEMENT Segment EMPTY>
<!ATTLIST Segment
    type         %string;  #REQUIRED
    id           %string;  #REQUIRED
    description  %string;  #REQUIRED
>

<!--
    An accounting object.

    name
        The name of the object containing the specified accounting segments.
-->
<!ENTITY % cxml.accounting  "Segment+">
<!ELEMENT Accounting (%cxml.accounting;)>
<!ATTLIST Accounting
    name  %string;  #REQUIRED
>

<!--
    A charge against an Accounting element.
-->
<!ELEMENT Charge (Money)>

<!--
    The combination of a Charge against an Accounting Element. A distribution
    represents the breakdown of one overall amount into sub-amounts.
-->
<!ELEMENT Distribution (Accounting, Charge)>

<!--
    Definition of a cXML Tax item. This represents what a Tax element should
    be in the classic notion of a line on a PO or Invoice. It can also
    represent a per-lineitem tax element depending on where it appears (inside
    of a item ELEMENT or inside of a something like a supplierOrder ELEMENT).
    Represents a tax item in the shopping basket. There could be one of these
    for the entire order, or one per lineitem.
-->
<!ELEMENT Tax (Money, Description)>

<!-- Item Elements -->
<!--
    The representation of a line item as it needs to be for sending to a
    supplier.

    quantity
        How many items are desired.

    requisitionID
        The buyers system requisition id for this line item. It might be the
        same as orderID, and it might not be included at all.

-->
<!ELEMENT ItemOut (ItemID, ItemDetail?, SupplierID?, ShipTo?, Shipping?, Tax?,
Distribution*, Comments?)>
<!ATTLIST ItemOut
    quantity              %uint;    #REQUIRED
    requisitionID         %string;  #IMPLIED
    requestedDeliveryDate  %date;    #IMPLIED
>

<!--
    The representation of a line item as it needs to be for sending to a
    buyer.

    quantity
        How many items are desired.
-->
<!ELEMENT ItemIn (ItemID, ItemDetail, SupplierID?, ShipTo?, Shipping?, Tax?)>
<!ATTLIST ItemIn
    quantity  %int;  #REQUIRED
>

<!-- OrderRequest* Elements -->
<!--
    Definition of an order.  This is the data that is send the the supplier
    to have them place an order in their order management system. The new
    world order equivalent of a PO.
-->
<!ELEMENT OrderRequest (OrderRequestHeader, ItemOut+)>

<!--
    Header of an order.  This is the data that is send the the supplier
    to have them place an order in their order management system. Money
    represents the total amount of this order.

    orderID
        The buyer system orderID for this request. Basically, what the PO
        number is today.

    orderDate
        The date the order request was created.

    type
```

```
        The type of the order request. Defaults to "new".
-->
<!ELEMENT OrderRequestHeader (Total, ShipTo?, BillTo, Shipping?, Tax?,
Payment?, Comments?)>
<!ATTLIST OrderRequestHeader
    orderID    %string;        #REQUIRED
    orderDate  %datetime.tz;   #REQUIRED
    type       (new | update | delete)  "new"
>

<!--
     The response to an OrderRequest. This is how the supplier confirms they
     have received and are going to act on an OrderRequest.

     orderID
         The buyer system orderID for this request. Basically, what the PO
         number is today.

     orderDate
         The date the order request was created. Should be the same date send
         in the SupplierOrderRequest.
-->
<!ELEMENT OrderResponse EMPTY>
<!ATTLIST OrderResponse
    orderID    %string;       #REQUIRED
    orderDate  %datetime.tz;  #REQUIRED
>

<!-- PunchOut* Elements -->
<!--
     Definition of a PunchOut Setup Request.  This is the data that is send to
     the the external system that the Ariba ORMS is going to extract catalog
     data from.

     The URL element is the URL we would like the browser re-directed to when
     the PunchOut shopping experience is finished (after the
     PunchOutOrderRequest messages have been exchanged).
-->
<!ELEMENT PunchOutSetupRequest (BuyerCookie, Extrinsic*, BrowserFormPost?,
                                SupplierSetup, ShipTo?, ItemOut*)>
<!ATTLIST PunchOutSetupRequest
    operation  (create | inspect | edit)  #REQUIRED
>

<!ELEMENT BuyerCookie ANY> <!-- any valid XML data -->

<!ELEMENT BrowserFormPost (URL)>
<!ELEMENT SupplierSetup (URL)>
<!ELEMENT PunchOutSetupResponse (StartPage)>
<!ELEMENT StartPage (URL)>

<!--
     Definition of a PunchOut Order Message.  This is the data that is send
     back to the Ariba ORMS system from the external system that the PunchOut
     Request was targeted at.
-->
<!ELEMENT PunchOutOrderMessage (BuyerCookie, PunchOutOrderMessageHeader,
ItemIn+)>

<!--
```

```
     Header of a PunchOut Order Request.  This is the data that is send from
the
     supplier to transfer the supplier aquired shopping basket back to the
     buyer system.

     operationAllowed
          list of operations that are allowed on the PunchOut shopping basket.
-->
<!ELEMENT PunchOutOrderMessageHeader (Total, ShipTo?, Shipping?, Tax?)>
<!ATTLIST PunchOutOrderMessageHeader
    operationAllowed  (create | inspect | edit)  #REQUIRED
>

<!ELEMENT PunchOutOrderAckMessage (BuyerCookie, PunchOutOrderMessageHeader,
ItemIn+)>
```

9.7 Transport.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Transport.mod#23 $
-->

<!--
    For better definitions of these Elements/Entities, refer to the cXML
    Protocol Specification documents.
-->

<!-- Imports are NOT allowed in .mod files -->

<!-- envelope -->
<!ELEMENT cXML ((Header, Message) |
                (Header, Request) |
                (Response))>

<!ATTLIST cXML
    version    %uint;         "1.0"
    payloadID  %string;       #REQUIRED
    timestamp  %datetime.tz;  #REQUIRED
>

<!-- header -->
<!ELEMENT Header (From, To, Sender)>

<!ELEMENT From (Credential+)>
<!ELEMENT To (Credential+)>
<!ELEMENT Sender (Credential, UserAgent)>

<!--
    A textual string representing who the UserAgent is conducting the cXML
    conversation. Analagous to UserAgent for HTTP conversations.
-->
<!ELEMENT UserAgent (#PCDATA)>

<!--
    A digital signature.  The recommended format is self-contained PK7. The
    exact signed content is not that significant but current timestamp would
    be used just as a convention.

    type
        The type of digital signature used.

    encoding
        How is the signature encoded in the XML stream.
-->
<!ELEMENT DigitalSignature ANY>
<!ATTLIST DigitalSignature
    type      %string;  "PK7 self-contained"
    encoding  %string;  "Base64"
>

<!--
```

```
    A shared secret. Typically, this is a username/password type of secret
    exchanged through a secure transport before communication takes place.
-->
<!ELEMENT SharedSecret ANY>

<!--
    Represents an identity for a credential.
-->
<!ELEMENT Identity ANY>

<!--
    A combination of an Identity and authentication element. If the
    authentication element is present, it strongly authenticates who/what
    someone is.

    domain
        In what domain is this Credendial represented?
-->
<!ENTITY % cxml.authentication  "SharedSecret |
                                 DigitalSignature"
>
<!ELEMENT Credential (Identity, (%cxml.authentication;)?)>
<!ATTLIST Credential
    domain  %string;  #REQUIRED
>

<!-- status -->
<!ELEMENT Status (#PCDATA)>
<!ATTLIST Status
    code  %uint;    #REQUIRED
    text  %string;  #REQUIRED
>

<!-- message -->
<!ENTITY % cxml.messages  "PunchOutOrderMessage |
                           PunchOutOrderAckMessage |
                           SubscriptionChangeMessage |
                           SupplierChangeMessage"
>

<!ELEMENT Message (Status?, (%cxml.messages;))>
<!ATTLIST Message
    deploymentMode  (production | test)  "production"
    inReplyTo       %string;  #IMPLIED
>

<!-- request -->
<!ENTITY % cxml.requests  "OrderRequest |
                           PunchOutSetupRequest |
                           GetPendingRequest |
                           SubscriptionListRequest |
                           SubscriptionContentRequest |
                           SupplierListRequest |
                           SupplierDataRequest"
>

<!ELEMENT Request (%cxml.requests;)>
<!ATTLIST Request
    deploymentMode  (production | test)  "production"
>
```

```
<!-- response -->
<!ENTITY % cxml.responses  "OrderResponse |
                            PunchOutSetupResponse |
                            GetPendingResponse |
                            SubscriptionListResponse |
                            SubscriptionContentResponse |
                            SupplierListResponse |
                            SupplierDataResponse"
>

<!ELEMENT Response (Status, (%cxml.responses;)?)>
```

9.8 Contract.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Contract.mod#11 $
-->

<!-- Imports are NOT allowed in .mod files -->

<!ELEMENT Contract (SupplierID+, Comments?, ItemSegment+)>
<!ATTLIST Contract
    effectiveDate   %datetime.tz;  #REQUIRED
    expirationDate  %datetime.tz;  #REQUIRED
>

<!--
    Defines an item segment for the index.  An item segment is an
    overlay for index items, allowing suppliers to override certain
    item attributes on a per-contract basis.

    Items may be segmented by some agreed-upon user-specific key that
    is used to determine who is eligible for these particular overlaid
    attributes (such as reduced or different prices).  Omitting the
    segmentKey indicates that the supplier wishes to set the given
    contract price system wide (for all users).

    segmentKey      - optional agreed-upon string used to segment
                      custom prices
-->
<!ELEMENT ItemSegment (ContractItem+)>
<!ATTLIST ItemSegment
    segmentKey  %string;  #IMPLIED
>

<!--
    A particular (custom) item overlay for a index item.  The item is
    referenced by the supplierPartID.

    ItemID - ID for the part to be overlaid.
    UnitPrice - Contract price for item
    Extrinsic - Named overlay. The Extrinsic should be named with the
    item field name it is to overlay. The Extrinsic must contain a
    <value> element which supplies the replacement value for the item
    field.
    For example:
    <ContractItem>
      <ItemId>
           <SupplierPartID>123456</SupplierPartID>
      </ItemId>
      <Extrinsic name="URL">http://www.newaddress.com</Extrinsic>
    </ContractItem>
-->
<!ELEMENT ContractItem (ItemID, UnitPrice?, Extrinsic*)>
```

9.9 Index.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Index.mod#10 $
-->

<!-- Imports are NOT allowed in .mod files -->

<!--
     IndexItemAdd is the element used to insert an item in an index.

     ItemID          - uniquely identifies the item
     ItemDetail      - general information about the item
     IndexItemDetail - Index specific item detail

-->
<!ELEMENT IndexItemAdd (ItemID, ItemDetail, IndexItemDetail)>
<!--
     IndexItemDelete is the element used to remove an item from the
     index.
     ItemID          - uniquely identifies the item

-->
<!ELEMENT IndexItemDelete (ItemID) >

<!--
     IndexItemPunchout is the element used to dynamically connect an
     index item to the supplier's resource for that item.

     ItemID          - uniquely identifies the item
     PunchoutDetail  - Describes the item being accessed
-->
<!ELEMENT IndexItemPunchout (ItemID, PunchoutDetail)>

<!--
     IndexItem is the general ELEMENT for the list of items in an
     index.

     IndexItemAdd      - Item(s) to be added to the index
     IndexItemDelete   - Item(s) to be removed from the index
     IndexItemPunchout - PunchoutItem(s) to be added to the index

-->
<!ELEMENT IndexItem (IndexItemAdd+ | IndexItemDelete+ | IndexItemPunchout+)>

<!--
     PunchoutDetail is the description of an item which is referenced
     in the index.

-->
<!ELEMENT PunchoutDetail (Description, URL, Classification+,
                          ManufacturerName?, ManufacturerPartID?,
                          ExpirationDate?, EffectiveDate?,
                          SearchGroupData*, TerritoryAvailable*)>

<!--
     Index is the element used to update the list of goods and/or
     services which are being handled by the system.

     SupplierID  - One or more identities by which this supplier is
                   known. NOTE: These are to be considered synonyms
                   for the same Supplier.
     SearchGroup - Description(s) of parametric search(es) for this
                   index
     IndexItem   - The list of items with which to modify the index

-->
<!ELEMENT Index (SupplierID+, Comments?, SearchGroup*, IndexItem+)>

<!--
     SearchGroup is a grouping of attributes which constitue a search
     which can be performed against an index.

     Name            - Name of the search
     SearchAttribute - List of searchable index fields.
-->
<!ELEMENT SearchGroup (Name, SearchAttribute+)>

<!--
     An attribute that can searched parametrically.

     name - name of the attribute.
     type - the type of the attribute
-->
<!ELEMENT SearchAttribute EMPTY>
<!ATTLIST SearchAttribute
    name  %string;  #REQUIRED
    type  %string;  #IMPLIED
>

<!--
     LeadTime specifies, in days, the amount of time required to
     receive the item.
-->
<!ELEMENT LeadTime (#PCDATA)>

<!--
     ExpirationDate is the date after which the element is no longer valid.
     Date must be specified in ISO 8601 format.

-->
<!ELEMENT ExpirationDate (#PCDATA)>
<!--
     EffectiveDate date at which the element becomes valid.
     Date must be specified in ISO 8601 format.
-->
<!ELEMENT EffectiveDate (#PCDATA)>

<!--
     IndexItemDetail contains various index specific elements which
     help to define an index item.
     LeadTime            - time in days to receive the item
     ExpirationDate      - Expiration date for the item in this index
     EffectiveDate       - Effective date for the item in this index
     SearchGroupData     - Parametric search data
     TerritoryAvailable - Country codes
-->
<!ELEMENT IndexItemDetail (LeadTime, ExpirationDate?, EffectiveDate?,
                           SearchGroupData*, TerritoryAvailable*)>

<!--
     Specification of a territory (using ISO country and/or region codes)
     in which the particular index item is available.
-->
<!ELEMENT TerritoryAvailable (#PCDATA)>

<!--
     SearchGroupData specifies the data which should be used to identify
     this item in a search.
-->
<!ELEMENT SearchGroupData (Name, SearchDataElement+)>

<!--
     SearchDataElement is a field and value which are used to provide the
     parametric data to a search.
-->
<!ELEMENT SearchDataElement EMPTY>
<!ATTLIST SearchDataElement
     name  %string; #REQUIRED
     value %string; #REQUIRED
>
```

9.10 Pending.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Pending.mod#4 $
-->

<!--
    For better definitions of these Elements/Entities, refer to the cXML
    Specification documents.
-->

<!-- Imports are NOT allowed in .mod files -->

<!--
    A request used for polling for waiting messages. A waiting message, if
    any, will be included in the returned stream. The lastReceivedTimestamp
    attribute, if present, provides the timestamp of the last received
    message. When the Receiver sees this, it can remove messages with earlier
    timestamps from the pending queue.

    The maxMessages attribute is used to indicate the maximum number of
pending
    messages that can be included in the response.
-->
<!ELEMENT GetPendingRequest (MessageType+)>
<!ATTLIST GetPendingRequest
    maxMessages            %int;          #IMPLIED
    lastReceivedTimestamp  %datetime.tz;  #IMPLIED
>

<!--
    Indicates the type of message(s) being polled for. The valid values are
    the corresponding element names e.g. SubscriptionChangeMessage.
-->
<!ELEMENT MessageType (#PCDATA)> <!-- string -->


<!--
    The data elements being carried back in the response. These are fully
    formed cXML messages being carried through the Request/Response channel.
-->
<!ELEMENT GetPendingResponse (cXML+)>
```

9.11 Subscription.mod

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 1996-1999 Ariba, Inc.
    All rights reserved. Patents pending.

    $Id: //ariba/specs/cXML/Subscription.mod#5 $
-->

<!-- Imports are NOT allowed in .mod files -->

<!--
    Indicates that something changed in a buyer's content subscription.  Since
    this is a Message, it can come at any time--no explicit Request needs to
    be send first.
-->
<!ELEMENT SubscriptionChangeMessage (Subscription+)>
<!ATTLIST SubscriptionChangeMessage
    type  (new | update | delete)  #REQUIRED
>

<!--
    A content subscription.
-->
<!ELEMENT Subscription (InternalID, Name, Changetime, SupplierID+, Format?,
Description?)>

<!ELEMENT InternalID (#PCDATA)> <!-- string -->
<!ELEMENT Changetime (#PCDATA)> <!-- datetime.tz -->
<!ELEMENT Format (#PCDATA)> <!-- string -->
<!ATTLIST Format
    version  %string;  #REQUIRED
>

<!--
    Requests a complete list of catalog subscriptions for a buyer.
-->
<!ELEMENT SubscriptionListRequest EMPTY>

<!--
    The list of Subscriptions for the given buyer.
-->
<!ELEMENT SubscriptionListResponse (Subscription+)>

<!--
    Requests the contents of a catalog that the buyer is subscribed to.
-->
<!ELEMENT SubscriptionContentRequest (InternalID, SupplierID+)>

<!--
    The data associated with a particular subscription.
-->
<!ELEMENT SubscriptionContentResponse (Subscription, SubscriptionContent+)>

<!--
    The actual content associated with a particular subscription.
-->
<!ELEMENT SubscriptionContent (CIFContent | Index | Contract)>
```

```
<!ATTLIST SubscriptionContent
    filename  %string;  #IMPLIED
>

<!--
    Contents of CIF file in base64 encoding.
-->
<!ELEMENT CIFContent (#PCDATA)> <!-- bin.base64 -->

<!--
    Indicates that something has changed in the supplier data for
    a supplier the buyer has a relationship with. Since this is a message, no
    Request needs to be sent to receive this Message.
-->
<!ELEMENT SupplierChangeMessage (Supplier+)>
<!ATTLIST SupplierChangeMessage
    type  (new | update | delete)  #REQUIRED
>

<!--
    Requests for a complete list of suppliers the buyer currently has
    relationships with.
-->
<!ELEMENT SupplierListRequest EMPTY>

<!--
    The list of suppliers requested by SupplierListRequest.
-->
<!ELEMENT SupplierListResponse (Supplier+)>

<!--
    Requests for a data associated with a particular supplier identified by
    SupplierID.
-->
<!ELEMENT SupplierDataRequest (SupplierID+)>

<!--
    The data associated with the desired supplier.
-->
<!ELEMENT SupplierDataResponse (Supplier)>
```