IBM® WebSphere® Commerce Version 5.4 Business Edition Integration Guide

for the Ariba® Buyer System

Note: Before using this information and the product it supports, read the information in "Notices and trademarks"

First Edition (August 2002)

This edition applies to version 5, release 4 of IBM WebSphere Commerce Business Edition (Program 5724 - A18) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can send your comments by any one of the following methods:

1. Electronically to the E-mail address listed below. Be sure to include your entire network address if you wish a reply.

   `Internet:torrcf@ca.ibm.com`

2. By regular mail to the following address:

   IBM Canada Ltd. Laboratory
   B3/KB7/8200/MKM
   8200 Warden Avenue
   Markham, Ontario, Canada L6G 1C7

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. Introduction

About this book

This book provides information about the features and the major capabilities of the reference application for procurement integration between WebSphere Commerce version 5.4, Business Edition and the Ariba Buyer version 7.1. It talks about the business models enabled with this integration, creating and configuring buyers and suppliers, creating and configuring a business-to-business (B2B) store, and customizing procurement integration.

Procurement integration

The procurement integration subsystem is a generic framework that enables WebSphere Commerce Version 5.4, Business Edition to handle B2B transactions using industry-standard protocols. It provides an extensible and customizable functionality on WebSphere Commerce that allows you to extend the message, schema, or business logic.

Who should read this book?

This document is oriented towards WebSphere Commerce administrators, programmers, and other experts. The following knowledge and experience is assumed:

- Thorough knowledge of Java, and a working knowledge of VisualAge® for Java
- Experience with the Windows 2000® or Windows NT® user interfaces and the Internet Explorer V5.5 web browser.
- An understanding of creating content and programming for the Internet as well as the World Wide Web environment.
- An understanding of HTML 3.2 syntax, including tables and forms.
- Experience with database programming using SQL.
- Experience with the IBM WebSphere Application Server.
- Experience with Java Servlets, Java Server Pages (JSPs), and Enterprise Java Beans (EJBs).
- Familiarity with IBM WebSphere Commerce Version 5.4, Business Edition.
- Knowledge of e-procurement systems.

Software requirements

The following must be installed on your system:

- IBM WebSphere Commerce Version 5.4, Business Edition on Windows NT or Windows 2000 with the associated software stack.

- IBM DB2 Universal Database 7.2 (with Fixpack 5) or Oracle 8.1.7.2.

For more details on the software requirements, refer to the *IBM WebSphere Commerce Installation Guide.*

Note: The hardware requirements will be the same as those for WebSphere Commerce Version 5.4, Business Edition.

Conventions and terminology used in this book

Procurement integration

The procurement integration subsystem is a generic framework that enables WebSphere Commerce to handle B2B transactions using industry-standard protocols.

Reference application

A reference application is a set of system features combined or packaged together as an offering that demonstrates to users a unique capability of the system.

Member subsystem

The member subsystem is a component of WebSphere Commerce that includes data for users, groups of users, and organizational entities. Business logic provides registration, profile management, access control, authentication, and session management services.

Catalog subsystem

The catalog subsystem is a component of WebSphere Commerce that provides online catalog navigation, partitioning, categorization, and associations. In addition, the catalog subsystem includes support for personalized interest lists and customer display pages. The catalog subsystem contains all logic and data relevant to an online catalog. This includes categories, products, items, and any associations or relationships among them.

Order subsystem

The order subsystem is a component of WebSphere Commerce that provides shopping carts, order processing, and management functions support. Related services such as pricing, taxation, payment, and fulfillment are also part of the order subsystem. Order processing capabilities include quick order or buy, scheduled orders, multiple pending orders, and reorders.

Contract

A contract is an arrangement between a seller and one or more buyers. Through this contract, the buyers can purchase goods and services from the seller, based on mutually agreed terms and conditions, for the specified duration of time.

Note: A contract does not refer to a one-time purchase order. It is the agreement on the terms of orders that the buyer may place during the validity period of the contract.

Chapter 2. Features of procurement integration

This section details the features, major capabilities, and business models of procurement integration.

General overview

Procurement integration is a component of WebSphere Commerce that enables the functionality to integrate external buy-side systems. It allows registered buyer organizations to connect from their procurement systems to interact with the supplier’s catalog system and conduct B2B e-commerce transactions. This results in increased sales and enhances the organization’s B2B presence on the web.

Procurement integration provides the following benefits:

- Suppliers can maintain a single catalog within WebSphere Commerce and use that catalog to enable their own web presence and participate in the procurement system’s network.
- Reduces costs of order processing through WebSphere Commerce connectivity to Supply Chain Management, Retail Business System, and Order Management backend systems. These automate the flow of orders from the procurement system’s buyer to your business systems.
- Uses the updated B2B features of WebSphere Commerce to use and maintain buyer organizations, buyer-specific catalogs, price lists, and contract pricing.

Procurement integration gives connective capabilities to the Ariba procurement system. It has out-of-the-box Commerce XML (cXML) capability that enables two different ways of shopping, through local catalogs and Internet catalogs.

Local catalog orders

In the local catalog mode, suppliers have their catalogs replicated on the procurement systems. Buyers browse the catalog and construct their shopping carts without connecting to WebSphere Commerce. When the requisitioning buyer submits the order, an `OrderRequest` message is sent to WebSphere Commerce (supplier) and batch processing of the order is performed. In this mode, no approval function is executed as it is assumed that the incoming order request is already approved. WebSphere Commerce sends a response (OrderResponse) indicating the success or failure of the order request. Managing the replication of the catalog is not within the scope of procurement integration.

Note: If there is a difference in the price of an item in the local catalog and the supplier’s catalog, the price in the supplier’s catalog is used.

Internet catalog orders

In this mode, suppliers maintain a single catalog within WebSphere Commerce and that catalog is used to enable their web presence and participation in the procurement system’s network. When the buyer selects the supplier on the procurement system, a connection is made to WebSphere Commerce through a message, for example, `PunchOutSetupRequest`. After successful authentication of the parties involved, WebSphere Commerce sends the appropriate CatalogDisplay URL and related information to bind the session back to the procurement system.

The procurement system uses the URL to display the WebSphere Commerce catalog in the browser. From this point until the shopping cart is prepared, the requisitioning buyer uses normal browser-based shopping.

When the requisitioning buyer prepares the shopping cart, it is placed in an XML message for example, `PunchOutOrderMessage` and is sent to the procurement system for approval. When the approver on the procurement system approves the order, an `OrderRequest` message (the same as in local catalog mode) is sent to WebSphere Commerce to create the order.

Major capabilities

To enable procurement integration the different subsystems of WebSphere Commerce such as the member subsystem, catalog subsystem, order subsystem, and so on are modified. To support XML messaging in WebSphere Commerce, XML messaging over HTTP is introduced. New controller commands, task commands, and view commands are introduced to support B2B functionality. The database schema is enhanced with new tables and corresponding EJBs.

The functionality included in the B2B sell-side extensions includes the following:

- Buyer organization registration.
- Buyer organization profile.
- Support for buyer and seller organization identification numbers, such as DUNS.
- Product classification code support for catalog entries and products, such as SPSC or UNSPSC.
- Unit of measure support for products, such as UNUOM.
- Contract support between buyers and sellers, where buyers can get specific views and prices of products based on the contract.
- The ability of the shopping cart functionality to send the shopping cart to the buyer systems for approval.
- The ability to process purchase orders sent from an external system.
- The buyer’s ability to change their shopping carts.
- The buyer’s ability to inspect the availability of a product before placing an order.
- The buyer’s ability to obtain more information about a product from their external procurement systems.
- The ability to process large orders from the external buyers.

- The ability to register requisitioning buyers from the external buyer organizations during runtime.

- The ability to authenticate and validate requests from buyers from external systems.

Business models enabled

The following business models are enabled by procurement integration::

- Suppliers can maintain a single catalog at the supplier’s site, and buyers can do Internet catalog shopping at the supplier’s site.

- Suppliers can receive messages from external procurement systems like Ariba, Commerce One®, and so on.

- Many buyers can access a single supplier’s store.

- The shopping cart can be sent in the procurement system’s specific.

- Buyers can receive Internet catalog orders.

- You can receive unsolicited orders through local catalog shopping.

Chapter 3. Supported cXML messages

Procurement integration uses five cXML messages (XML over HTTP) to integrate with Ariba. There are two inbound messages to WebSphere Commerce from Ariba, and three outbound response messages from WebSphere Commerce to Ariba.

The following are the messages and commands directly invoked by the messages (for inbound messages) or commands sending out the messages (for outbound messages):

Message Name	**Command (ME)**	**Direction**	**Response**
PunchOutSetupRequest	PunchOutSetup	IN	PunchOutSetupResponse
OrderRequest	BatchOrderRequest	IN	OrderResponse
PunchOutSetupResponse	PunchOutSetup	OUT	
PunchOutOrderMessage	PrepareOrder	OUT	
OrderResponse	BatchOrderRequest	OUT	

Here, `PunchOutOrderMessage` is the shopping cart sent to Ariba for approval. For sample messages see, “Appendix B: Sample XML messages”.

Chapter 4. Configuring the reference application

This section covers how to configure the buyer and supplier settings. It also provides information on creating a new store and enabling an existing store with procurement integration.

Configuring the buyer and supplier

Configuring supplier settings

In the `PISupplierConfig.xml` file present in the `procurementintegration.zip` modify the supplier information to configure a new PISupplier as a member of WebSphere Commerce. A `config.bat` file is provided to load the configuration data for the Windows environment only. This file resides in the same directory as the configuration XML files. The `Config.txt` file in this directory describes how to run the `config.bat` file. The following is a code fragment from `PISupplierConfig.xml`:

```
<member
   Member_id=”1”
   Type=”O”
   State=”1”
/>

<orgcode
   orgcode_id="1"
   orgentity_id=<StoreOwnerId>
   codetype="DUNS"
   code="sreed"
/>

<member
   Member_id="1"
   Type="O"
   State="1"
/>

<viewreg
   viewname="PunchOutAribaView"
   devicefmt_id="-10000"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand”
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=PunchOutSetupResponse.jsp"
/>

<viewreg
   viewname="PunchOutAribaErrorView"
   devicefmt_id="-10000"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=PunchOutAribaError.jsp"
/>
```

```
<viewreg
   viewname="PunchOutCatalogView"
   devicefmt_id="-1"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=PunchOutCatalogDisplay.jsp"
/>

<viewreg
   viewname="SubmitShoppingCartView"
   devicefmt_id="-1"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=SubmitShoppingCart.jsp"
/>

<viewreg
   viewname="SubmitShoppingCartErrorView"
   devicefmt_id="-1"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=error.jsp"
/>

<viewreg
   viewname="SendShoppingCartView"
   devicefmt_id="-1"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=ComposeShoppingCartAriba.jsp"
/>

<viewreg
   viewname="SendShoppingCartErrorView"
   devicefmt_id="-1"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=error.jsp"
/>

<viewreg
   viewname="PurchaseOrderResponseAribaView"
   devicefmt_id="-10000"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=PurchaseOrderResponseAriba.jsp"
/>

<viewreg
   viewname="PunchOutCatalogEditView"
   devicefmt_id="-1"
   storeent_id="0"
   interfacename="com.ibm.commerce.command.HttpForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=PunchOutCatalogDisplay.jsp"
/>
```

Getting buyer invitations and soliciting buyer information

To transact business with a buyer over the procurement system, a buyer must invite you to become a supplier. Once invited, you must obtain some basic information about the buyer and enter that information into WebSphere Commerce. Information specific to the procurement system is collected through new forms that are part of procurement integration.

See “Appendix D: Sample buyer information form” for a sample form You may implement and deliver this form in any manner you wish – as a printed form, a file sent as an e-mail attachment, a web-based form, and so on. Typically, you would send this form to a buyer or have them complete the form on the Internet.

Once the buyer completes the form or you obtain the buyer information in some other manner, you can proceed to register a new buyer organization.

Configuring buyer settings

A buyer organization or a buyer is any business organization that wants to procure goods and services. There may be multiple buyer organizations within a buyer’s enterprise. To create and configure the buyer, you must update the database tables with the buyer information. You can do this by creating an XML file.

In the `PIBuyerConfig.xml` file present in the `procurementintegration.zip` modify the supplier information to configure a new PIBuyer as a member of WebSphere Commerce. A `config.bat` file is provided to load the configuration data. This file resides in the same directory as the configuration XML files. The `Config.txt` file in this directory describes how to run the `config.bat` file. The following is a code fragment from `PIBuyerConfig.xml`:

```
<member
   Member_id=“5000”
   Type=”O”
   State=”1”
/>
```

Type ‘O’ specifies that the member is an organization.

```
<orgentity
   Orgentity_id=“5000”
   Orgentitytype=”O”
   Orgentityname=”Staples”
/>
```

`Orgentitytype` indicates that the type is an organization.

`Orgentityname,` is the name of the department or organization.

```
<orgcode
   orgcode_id=”2”
   orgentity_id=”5000”
   codetype=”AribaNetworkId”
   code=”nkrishna@in.ibm.com”
/>
```

`code` and `codetype` represent the DUNS number or any other mutually agreed-on number.

Add an address book entry to the address book and address tables.

```
<addrbook
      addrbook_id="4000"
      displayname="Staples address Book"
      member_id="5000"
/>

<address
      addrbook_id="4000"
      address_id="3000"
      member_id="5000"
      nickname="Staples_NickName1"
      selfaddress="1"
      address1="Airport Road"
      city="Bangalore"
      country="India"
      state="Karnataka"
      zipcode="560008"
      status="P"
/>

<mbrattrval
      mbrattrval_id="790"
      member_id="5000"
      attrtype_id="INTEGER"
      mbrattr_id="-21"
      integervalue="1"
/>

<!-- Now register the protocol which the buyer supports -->

<procsys
   procsysname=”Ariba”
/>

<procprotcl
   procprotcl_id=”1”
   procsysname=”Ariba”
   protocolname=”cXML”
   version=”1.0”
   authtype=”1”
   twostepmode=”Y”
   classifdomain=”UNSPSC”
   uomstandard=””
/>
```

```
<procmsgvw
   procprotcl_id=”1”
   orgentity_id=”5000”
   msgname=”PunchOutSetup”
   viewname=”PunchOutAribaView”
/>

<procmsgvw
   procprotcl_id=”1”
   orgentity_id=”5000”
   msgname=”SendShoppingCart”
   viewname=”SendShoppingCartView”
/>

<procmsgvw
   procprotcl_id=”1”
   orgentity_id=”5000”
   msgname=”PurchaseOrderResponse”
   viewname=”PurchaseOrderResponseAribaView”
/>

<procbuyprf
   procprotcl_id=”1”
   orgentity_id=”5000”
   reqidparm=”User”
   orgunitparm=<deptName>
/>
```

Creating member groups for buyers

A member group is a collection of buyers as defined by the supplier, who share a common interest. Member groups are similar to the exclusive clubs offered by large stores for their frequent or preferred buyers. Being part of a member group can entitle buyers to discounts or other bonuses when purchasing products. For example, if market research shows that certain buyers repeatedly purchase certain products, you can assign these buyers to the same member group. Alternatively, you can create a member group to reward frequent buyers for their business. You can assign different prices to products for different member groups. You can also customize the way products and categories are shown to member groups.

A procurement integration buyer can belong to one member group, the default group. Creating the procurement integration buyer group is similar to creating a member group for buyers. The `fontmbrgrp_id` must be present in the BUYSUPMAP table with the `fontbuyorgunit_id` and the `fontsuporg_id` to form a unique index.

A sample XML file, `PIBuyerConfig.xml` is provided to create member groups for buyers. The following is a code fragment from the file `PIBuyerConfig.xml`:

```
<member
   Member_id=“101”
   Type=“G”
   State=”1”
 />
```

```
<mbrgrp
   Mbrgrp_id="101"
   mbrgrpname="Store Requisitioners for BuyOrg"
   Owner_id="5000"

/>
```

This fragment creates a member with the type 'O', which represents the member group.

The `Owner_id` is the `Member_id` that owns the group. If the buyer group will apply site-wide, set this to zero. Otherwise, consider setting the ID to the `orgentity_id` of the organization within which this buyer group is used.

```
<buysupmap
    procprotcl_id="1"
    buyorgunit_id="5000"
    suporg_id=<StoreOwnerId>
    catalog_id="1"
    mbrgrp_id="101"
/>
```

The buyer and supplier must have access to execute the WebSphere Commerce or procurement integration commands. The following XML fragment assigns roles to the buyers and suppliers required to execute the commands.

```
<mbrole
    member_id="5000"
    role_id="-25"
    orgentity_id="-2001"
/>
<mbrrel
    descendant_id="5000"
    ancestor_id="-2001"
    sequence=2
/>
```

Register a buyer with the Ariba `NetworkUserId` as the `logonId` and `SharedSecret` as the password for the buyer organization from the adminconsole. Assign the 'Procurement Buyer Administrator' role to this buyer. This default buyer can now dynamically register other buyers from the procurement system. Assign this buyer to the member group created previously. For instructions on how to create a user refer to the WebSphere Commerce online documentation

Note: Restart the instance after configuring the buyer and supplier.

Creating and configuring your store

To provide an Internet catalog, you can create a new store in your WebSphere Commerce Installation or configure an existing store. Use the `PIRefApp.sar` provided with this reference application as a template.

Note: You must disable the cache in the WebSphere Commerce instance that is being used for this reference application.

1. **Publish a new procurement – enabled store**

a. Before you begin, make sure that the following services are running:

- IBM HTTP Server
- IBM WS AdminServer
- WebSphere Commerce Server instance

For more details on Starting and stopping services, refer to the *WebSphere Commerce Installation Guide*.

b. From procurementintegration.zip copy `PIRefApp.sar` to the `WCBE_Install_Path\samples\stores\ToolTech` directory.

c. Go to `WCBE_Install_Path\xml\tools\devtools`. Open the file `SARRegistry.xml`. Append the following code to the end before the last line, `</SAR-properties>`

```
<SampleSAR
   fileName="PIRefApp.sar"
   relativePath="ToolTech">
   <html locale="en_US"
   featureFile="ToolTech/Feature_refapp_en_US.html"
   sampleSite="ToolTech/preview/en_US/index.html"/>
</SampleSAR>
```

d. Copy the file `Feature_refapp_en_US.html` from the downloadable location to `WCBE_Install_Path\samplestores\ToolTech.`

e. Edit the file `store-all.dtd` in `WCBE_Install_Path\xml\sar`.

You need to add the following to the <store-asset> element just before the symbol “`)*)>`”

```
|clasifcode | clsfcodeds |catclsfcod
```

Add the following lines at the end:

```
<!ELEMENT clasifcode EMPTY>


<!ATTLIST clasifcode
   clasifcode_id        CDATA       #REQUIRED
   domain               CDATA       #REQUIRED
   code                 CDATA       #REQUIRED
   parentcode           CDATA       #IMPLIED
>

<!ELEMENT clsfcodeds EMPTY>
<!ATTLIST clsfcodeds
   language_id          CDATA       #REQUIRED
   clasifcode_id        CDATA       #REQUIRED
   description          CDATA       #REQUIRED
>

<!ELEMENT catclsfcod EMPTY>
<!ATTLIST catclsfcod
   catentry_id          CDATA       #REQUIRED
   domain               CDATA       #REQUIRED
```

```
    code                    CDATA       #REQUIRED
>
```

f. Restart the WebSphere Commerce Server – *instance_name* by selecting **Start >> Programs >> IBM WebSphere >> Application Server V3.5 >> Administrator's Console**. Expand your hostname, and go to WebSphere Commerce Server – *instance_name*. Right click, and select **Stop**. After it stops successfully, start it again.

g. Start Store Services: from the Start menu, select **Programs >> IBM WebSphere Commerce >> Store Services**.

h. Click the **New** button on the right-hand pane. All the sample store archives are listed in a list box named "sample."

i. Provide the store archive and store directory name. Select the Owner organization for this store and select the **`PIRefApp.sar`** file from samples list. Click **OK**.

j. Select the store archive created in the previous step and click on **Publish**. This may take some time. Click **Refresh** to see the Publish Status change from 'Not Published' to 'Publishing' and then to 'Publishing Completed Successfully'.

Enabling an existing store with procurement integration

The procedures in this section assume that you have already published a store using Store Services. This process creates stores with a standard structure. Since stores created by means other than Store Creator can vary dramatically in organization, capability, and layout, there is no standard set of instructions that can explain how to enable these stores.

Note: You must disable the cache in the WebSphere Commerce instance.

To enable a store that already exists in WebSphere Commerce, do the following:

1. Make a backup of the web directory for your existing WebSphere Commerce catalog, *WebSphereAppServer_Installation_Path*`InstalledApps/WC_Enterprise_App_`*instance_name* `ear/wcstores.war/`. Where, `instance_name` is the name of the WebSphere Commerce instance in your installation.

2. Navigate to the *WebSphereAppServer_Installation_Path*`/InstalledApps/WC_Enterprise_App_`*instance_name*`.ear/wcstores.war/`. Where, *instance_name* is the name of the WebSphere Commerce instance in your installation and <*storedirectory*> `directory where` *store_directory* contains all the pages of your website. Open the `shoppingcart.jsp` in any editor.

3. Extract `PIRefApp.sar` into a temporary directory and open `shoppingcart.jsp`.

4. Make the necessary changes to the `shoppingcart.jsp` in your store to call the `SubmitShoppingCartCmd` during checkout (or) `addToOrder` by comparing it with the `PIRefApp's shoppingcart.jsp`.

5. Copy `PunchoutSetupResponse.jsp`, `PunchoutCatalogDisplay.jsp`, `PunchoutOutAribaError.jsp`, `SubmitShoppingCart.jsp`, `PurchaseOrderResponse.jsp`, and `ComposeShoppingCartAriba.jsp` into your *`WebSphereAppServer_Installation_Path`*`/InstalledApps/WC_Enterprise_App_<instance_name>.ear\wcstores.war/`directory.

6. From `PIRefApp.sar`, extract `data/refapp.dtd` and `data/refapp.xml` into any temporary directory. Edit `refapp.xml` to change the element from <refapp-asset> to <import> and the closing tag from </refapp-asset> to </import>. Edit `refapp.dtd` and change the element type from refapp to import. Change the corresponding `catentry_id` values to those of the existing store catalog's `catentry_ids`. Add more or delete some if required. Set the classpath to contain the following jar or zip files.

```
classpath=%classpath%;
WCBE_Install_Path\lib\loader\jgl2.0.0.jar;
WCBE_Install_Path\lib\loader\jlog.jar;
WCBE_Install_Path\lib\loader\wcmxmlp.jar;
WCBE_Install_Path\loader\wcmxslt.jar;
WCBE_Install_Path\lib\loader\Logger.zip;
WCBE_Install_Path\lib\loader\WCALogger.zip;
WCBE_Install_Path\lib\loader\SAFServ.zip;
WCBE_Install_Path\lib\loader\IdResGen.zip;
WCBE_Install_Path\lib\loader\MassLoader.zip;
WCBE_Install_Path\lib\loader\WCMCommon.zip;
WCBE_Install_Path\lib\loader\db2\dbconnect.zip;
WCBE_Install_Path\commerce\xml\loader;
```

   Run the following command from the command window:

```
java com.ibm.wca.MassLoader.MassLoad -dbname <dbname>
-dbuser <dbuser> -dbpwd <dbpwd> -infile refapp.xml
-method sqlimport
```

   This will insert the required classification code for the respective catentries.

7. Restart the WebSphere Commerce instance.

Note: When buyers access your catalog though a remote session, they see a slightly different navigation sequence than when viewing your store directly through WebSphere Commerce. This is normal. Message extensions replace certain standard operations such as the checkout process with applications that incorporate message extension order processing workflow in place of the standard WebSphere Commerce flow.

Business account for the supplier and buyer

1. Create a business account

   For instructions on how to create a business account refer to the WebSphere Commerce online documentation. When creating a business account note the following:

- Don't select the **Allow customers to purchase under the terms and conditions of store's default contract** checkbox in the Customer page.

- Don’t select the **Purchase order number may be specified at the time of the order** checkbox in the Purchase Order page.

- Specify the credit line account number in the Credit Line page.

2. Create a contract for the business account created previously.

   For instructions on how to create a contract for the business account refer to WebSphere Commerce online documentation. When creating a contract note the following:

- Select the **Allow the payment using the account’s credit line** checkbox.

- Don’t specify any contract shipping addresses. This is required only if shipping is necessary for various shipping addresses.

3. Modify the payment method information for the store using the following sql:

```
update cmdreg set
classname=’com.ibm.commerce.payment.commands.DoPaymentMP
FCmdImpl’ where storeent_id=<SupplierOrgStoreID> and
interfacename=’com.ibm.commerce.payment.DoPaymentCmd’
```

Your store is now enabled with procurement integration.

Chapter 5. Customizing procurement integration

Procurement integration provides an extensible and customizable framework to support B2B transactions on top of WebSphere Commerce so that one can extend the message, schema, or business logic. This component is built over WebSphere Commerce in order to enable the functionality to integrate external buy-side systems.

Enabling procurement integration for other procurement systems

To enable procurement integration for different procurement systems such as Ariba, mySAP, Commerce One, and so on, means supporting the system with the functionality to handle different protocols such as cXML, OCI, and XCBL.

Procurement integration provides out-of-the-box cXML support. It can also support any other protocol as long as it receives XML over HTTP. To support new protocols, you need to customize procurement integration. This customization involves the following:

- Registering the new protocol with procurement integration.
- Creating the configuration file (that is, mapping protocol-specific XML to procurement integration specific variables).
- Customizing the procurement integration specific Java Server Pages (JSP).
- Customizing the different subsystems like member subsystem, catalog subsystem, and order subsystem.

1. **Registering a new protocol with procurement integration**

   This includes creating an entry for the new protocol in the database and associating it with the suppliers and buyers by doing the following:

   a. Populate the procurement system information into the PROCSYS table for the new procurement system.

   b. Populate the protocol-specific information like protocol name, version, format, and so on into the PROCPROTCL table.

   c. Populate the view task name for the specific protocol. Each procurement integration command (`PunchOutSetup, BatchOrderRequest, SendShoppingCart`) that needs to compose a response message to be sent to the procurement system will look up the PROCMSGVW table to get the correct view task name.

   d. Populate the PROCBUYPRF table that contains the buyer organization's profile information such as the requisition id, department name, and so on.

   e. Associate the buying organization unit with the supplier. Populate the BUYSUPMAP table that will contain the `protocol id`, `catalog id`, and the member `group id` created for the buyer.

See, “Configuring the buyer and supplier” for sample XML files to populate these tables.

2. **Creating configuration files**

   To let procurement integration understand the incoming XML message, the message must be mapped to a WebSphere Commerce or procurement integration command and the elements of the message must be mapped to the parameters of that command. For instance, map the `logon request`, `order request`, and other messages to their respective commands using a system template or user template. Refer to “Appendix A: System_template.xml” for more information.

   To receive XML over HTTP enable the HTTP adapter in *`WCBE_Install_Path\instances\instance_name\xml\<instance_name>.xml`* by changing enabled="true" in the following section of the XML file.

Note: Where *`instance_name`* is the name of the instance in your WebSphere Commerce installation.

```
<HttpAdapters display="false">
  <HttpAdapter deviceFormatTypeId="-10000"
    enabled="true"
    deviceFormatType="XmlHttp"
    deviceFormatId="-10000"
    name="XML/HTTP"
    factoryClassname="com.ibm.commerce.programadapter.
      HttpProgramAdapterImpl">
    <ProgramAdapter>
      <SessionContext
        class="com.ibm.commerce.messaging.programadapter.security.
          CredentialsSpecifiedProgramAdapterSessionContextImpl">
                 <SessionContextConfig />
      </SessionContext>
        <Configuration supportedMessageMappers="WCS.INTEGRATION"
          supportedContentTypes="text/xml, text/xml;charset=UTF-8,
            text/xml-SOAP"
          supportedMethods="POST, M-POST"
          supportedCharacterEncoding="ISO8859-1, UTF-8" />
    </ProgramAdapter>
  </HttpAdapter>
</HttpAdapters>
```

You can configure the XML or HTTP adapter support using a separate template XML file for each procurement system. To do this, add the following section to the message-mapper group section for each message mapper in demo.xml.
The path is:
`<`*`drive`*`>:\`*`WCBE_Install_Path`*`\instances\`*`instance_name`*`\xml`.
Where *`instance_name`* is the name of the instance in your WebSphere Commerce Installation.

```
<MessageMapper
  messageMapperId="1"
  classname="com.ibm.commerce.messaging.programadapter.
    messagemapper.ecsax.ECSAXMessageMapper"
  enable="true"
  name="WCS.INTEGRATION">
  <configuration
/>
```

Add the following lines under the configuration element before the end tag, in the code given above.

NT, 2000

```
EcSystemTemplateFile="ariba_sys_template.xml"
EcTemplatePath="C:\WebSphere\Commerce\xml\messaging"
EcInboundMessageDtdFiles="cXML.dtd"
cInboundMessageDtdPath="C:\WebSphere\Commerce\xml\mes
saging"

/>
```

AIX

```
EcSystemTemplateFile="ariba_sys_template.xml"
EcTemplatePath="/usr/WebSphere/CommerceServer/xml/mes
saging"
EcInboundMessageDtdFiles="cXML.dtd"
cInboundMessageDtdPath="/usr/WebSphere/CommerceServer
/xml/messaging"

/>
```

SOLARIS

```
EcSystemTemplateFile="ariba_sys_template.xml"
EcTemplatePath="/opt/WebSphere/CommerceServer/xml/mes
saging"
EcInboundMessageDtdFiles="cXML.dtd"
cInboundMessageDtdPath="/opt/WebSphere/CommerceServer/xml/m
essaging"

/>
```

Note: Some e-procurement protocols use HTML as the transport format instead of XML. Procurement integration can support such messages. Refer to “Appendix C: LogonRequest in HTML format” to configure an HTML message.

3. Customizing JSPs

JSPs generate the messages that need to be transferred to different procurement systems. To generate messages in any format specific to a protocol, customize the following JSPs:

- Logon Response JSP
- Shopping Cart JSP
- Purchase Order Request JSP
- Order Response JSP

For protocol specific messages, modify the following JSPs and then update the VIEWREG table accordingly:

- `PunchOutCatalogDisplay.jsp`
- `PunchOutSetupResponse.jsp`
- `OrderItemDisplay.jsp`
- `SubmitShoppingCart.jsp`

- `ComposeShoppingCartAriba.jsp`
- `PurchaseOrderResponseAriba.jsp`

4. Customizing the member subsystem

The member subsystem may be customized in the following ways:

- Adding extension tables to the existing member subsystem schema or modifying the existing tables.
- Adding entity beans corresponding to the new tables or modifying the existing entity beans to reflect changes in the table schema.
- Implementing new controller and task commands.

  For instructions, refer to the *WebSphere Commerce Programmers Guide Version 5.4.*
- Extending the controller and task commands.
- Receiving LogonRequest in HTML format.

  Refer to “Appendix C: LogonRequest in HTML format” to customize `PunchOutSetupCmd`.

a. Customizing commands

You can extend any of the procurement integration controller and task commands to provide custom behavior by over-riding certain methods. For instructions, refer to the *Programmers Guide for WebSphere Commerce Version 5.4.* Each command is based on an interface; and one way to customize a certain command would be to provide a custom implementation of the required interface. A default implementation class has been provided for some interfaces.

The following is a list of all the membership subsystem interfaces and their default implementation classes:

Controller commands

Interface	Default Implementation
PunchOutSetupCmd	PunchOutSetupCmdImpl
PunchOutCatalogDisplayCmd	PunchOutCatalogDisplayCmdImpl

Task commands

Interface	Default Implementation
AuthenticationHelperCmd	AuthenticationHelperCmdImpl
ProcurementDBAuthenticationCmd	ProcurementDBAuthenticationCmdImpl
LdapAuthenticationCmd	
RegisterRequisitionerCmd	RegisterRequisitionerCmdImpl
ThirdPartyB2BauthCmd	

b. Customizing CIData

The `PunchOutSetupCmd` uses the CIData object to store all the XML parameters that it receives from the `PunchOutSetupRequest` message. A default implementation of the CIData interface is provided in CIDataImpl. To customize CIData, the buyer can provide a custom implementation of the CIData class. This captures custom information from the XML parser.

The `PunchOutSetupCmd` receives the XML parameters in the form of a `TypedProperty` data structure that extends the HashTable class. It then passes the `TypedProperty` object to the CIData object in the `setRequestProperties()` method as follows:

Example:

```
public void setRequestProperties(TypedProperty
typedproperty)throws ECException
  {
      String s = "setRequestProperties";
      ECTrace.entry (ECTraceIdentifiers.COMPONENT_USER,
     getClass().getName(), s);
      requestProperties = typedproperty;
      ciData.setLogonData(typedproperty);
      ECTrace.exit(16L, getClass().getName(), s);
  }
```

The procurement integration data then processes the `TypedProperty`, that is `typedproperty` in the method `processHeader()`. The `processHeader()` method in turn populates the following objects:

i) SupplierCred – Stores the supplier credentials for authentication purposes.

ii) BuyerCred – Stores the buyer organization credentials for authentication purposes.

iii) MpCred – If the `PunchOutSetupRequest` comes from an online marketplace, then this object stores the marketplace credentials for authentication purposes.

iv) B2BAgent – Captures the information about the messaging protocol.

v) SessionInfo – Stores information relevant to register the requisitioning buyer.

Any of these may be modified to store information received through custom elements in the XML message. The following example shows how some of the supplier credentials are set.

Example:

```
private void processHeader(TypedProperty
typedproperty)
   {
      String s = "processHeader";
      supplierCred = new Credentials();
   supplierCred.setCode(typedproperty.getString
     ("supplierCode", null));

supplierCred.setCodeDomain(typedproperty.getString
     ("supplierCodeType", null));
   .
   .
   .
}
```

c. Customizing authentication

The authentication functionality provided with procurement integration can be modified in one of the two ways:

i) Customizing the authentication type

The default authentication mechanism provided is the authentication against credential information stored in the WebSphere Commerce database. This authentication is performed by the ProcurementDBAuthenticationCmd task command.

You have the option to customize authentication so that it will be performed against a Lightweight Directory Access Protocol (LDAP) directory by implementing the interface `LdapAuthenticationCmd`. Alternately, you can choose to use a third-party authentication mechanism by implementing the interface `ThirdPartyB2BAuthCmd`.

ii) Customizing the authentication level

In procurement integration, you are provided with four possible levels of authentication. For a description of these authentication levels please refer to the description of the `ProcurementDBAuthenticationCmd.`

You may also specify a fifth level of authentication to customize the authentication level. This must be specified in the database during buyer registration. In addition, the authentication command, either `ProcurementDBAuthenticationCmd`, `LdapAuthenticationCmd`, or `ThirdPartyAuthenticationCmd` must be modified to handle the new authentication level.

d. Customizing registration

The controller command `PunchOutSetupCmd` makes a call to the task command `RegisterRequisitionerCmd` to register the requisitioning buyer based on the session information stored in the `sessionInfo` object. To add, remove, or modify registration information complete the following steps:

i) Modify the tables in the database or any other persistent data storage being used to reflect this change. If the WebSphere Commerce database is being used, then modify the BUSPROF table that stores the information about the requisitioning buyer.

ii) Provide a custom implementation of the procurement integration data interface to capture custom information about the requisitioning buyer from the XML message. Store the information in the appropriate `sessionInfo` object variables, or provide added class variables in the `registerRequisitionerCmd` command with their appropriate getter and setter methods. These variables will then be set directly rather than being passed as members of the `sessionInfo` object.

iii) Modify the `performExecute()` method of the `registerRequisitionerCmd` to update the persistent data storage, the BUSPROF table for example, with the custom data.

e. Customizing store catalog display in `PunchOutSetupCmd`

Once a buyer organization is authenticated and the requisitioning buyer has been registered, do the following in the `performExecute()` method of the `PunchOutSetupCmd`:

i) All the parameters to be passed on to `PunchOutCatalogDisplayCmd` are stored in the `BuyerRequestInfo` object.

ii) The `BuyerRequestInfo` object is stored in the SupplierCookieTable.

iii) `PunchOutSetupCmd` will return the URL of `PunchOutCatalogDisplayCmd`, which will be invoked by the procurement system along with the supplier cookie value as its parameter.

Example:

The following is the code that performs this function in `PunchOutSetupCmd`:

```
public void performExecute()throws ECException
   {
      BuyerRequestInfo buyerrequestinfo = new
        BuyerRequestInfo();
      buyerrequestinfo.setUsersId(userId);
      ...
      String s1 =
        SupplierCookieTable.put(buyerrequestinfo);
      getInstance()

      responseProperties.put("commandName",
        PunchOutCatalogDisplayCmd?supplierCookie="
        + s1);
      ...
   }
```

iv) The `PunchOutCatalogDisplayCmd` will set the view name as `PunchOutCatalogVie`, which in turn will call `PunchOutCatalogDisplay.jsp.`

v) The `PunchOutCatalogDisplay.jsp` will then call the command invoked by the URL that will bring up the store’s home page.

f. Customizing the store catalog display

Doing the following can customize the store catalog display:

i) Modify the parameters being passed on to the store catalog display page. To do this add the parameter with its value to the `responseProperties` in the `performExecute()` method of `PunchOutCatalogDisplayCmd`. These parameters can then be captured in `PunchOutCatalogDisplay.jsp`.

ii) Modify the target command from `PunchOutCatalogDisplay.jsp`.

iii) Specify the new URL in place of the default URL provided in the appropriate logon mode.

iv) Modify the target .JSP invoked from the URL command. To do this change the appropriate value on the URLREG table in the WebSphere Commerce database.

5. Customizing the order subsystem

The order subsystem can be customized in the following ways:

- Extend the controller and task commands.
- Add extension tables to the existing order subsystem schema or modifying the existing tables.
- Add entity beans corresponding to the new tables or modify the existing entity beans to reflect changes in the table schema.
- Implement new controller and task commands.

For instructions, refer to the *WebSphere Commerce Programmers Guide Version 5.4.*

a. Customizing commands

You can extend any of the controller and task commands to provide custom behavior by over-riding certain methods. For instructions, refer to the *WebSphere Commerce Programmers Guide Version 5.4.* Each command is based on an interface, and one way to customize a certain command would be to provide a custom implementation of the required interface. A default implementation class has been provided for each interface.

The following is a list of all the order subsystem interfaces and their default implementation classes:

Controller commands

Interface	Default implementation
BatchOrderRequestCmd	BatchOrderRequestCmdImpl
ProcurementOrderPrepareCmd	ProcurementorderPrepareCmdImpl

Task commands

Interface	Default implementation
AuthenticationHelperCmd	AuthenticationHelperCmdImpl
ProcurementDBAuthenticationCmd	ProcurementDBAuthenticationCmdImpl
ShipBillToAddressCmd	ShipBillToAddressCmdImpl
LdapAuthenticationCmd	
RegisterRequisitionerCmd	RegisterRequisitionerCmdImpl
ThirdPartyB2BauthCmd	

b. Customizing CIData

The `BatchOrderRequestCmd` uses the CIData object to store all the XML parameters that it receives from the `OrderRequest` message. A default implementation of the CIData interface is provided in CIDataImpl. To customize CIData, provide a custom implementation of the CIData class to capture custom information from the XML parser.

The `BatchOrderRequestCmd` receives the XML parameters in the form of a `TypedProperty` data structure that extends the HashTable class. It then passes the `TypedProperty` object to the CIData object, which then processes it using the method `processHeader()`. The `processHeader()` method populates the following objects:

i) SupplierCred – Stores the supplier credentials for authentication.

ii) BuyerCred – Stores the buyer organization credentials for authentication.

iii) MpCred – If PunchOutSetupRequest has come from an online marketplace then this object stores the marketplace credentials for authentication purposes.

iv) B2BAgent – Captures the information about the messaging protocol.

v) SessionInfo – Stores information relevant to register the requisitioning buyer.

vi) Any of the above may be modified to store information received via custom elements in the XML message.

c. Customizing authentication

The authentication functionality provided with procurement integration can be modified in one of the two ways:

i) Customizing the authentication type

The default authentication mechanism provided is authentication against credential information stored in the WebSphere Commerce database. This authentication is performed by DBAuthenticationCmd task command.

You have the option to customize authentication so that it is performed against an LDAP directory by implementing the interface `LdapAuthenticationCmd`.

Add to above Alternately, you can choose to use a third-party authentication mechanism by implementing the interface ThirdPartyB2BAuthCmd.

ii) Customizing authentication level

Procurement integration provides you with four possible levels of authentication. For a description of these authentication levels, please refer to the description of the `DBAuthenticationCmd`.

You may also specify a fifth level of authentication to customize the authentication level. You must specify this in the database during buyer registration. In addition, the authentication command (`DBAuthenticationCmd`, `LdapAuthenticationCmd` or `ThirdPartyAuthenticationCmd`) will have to be modified to handle the new authentication level.

d. Customizing registration

The controller command `BatchOrderRequestCmd` makes a call to the task command `RegisterRequisitionerCmd` to register the requisitioning buyer based on the session information stored in the `sessionInfo` object. To add, remove, or modify registration information, do the following:

i) Modify the tables in the database or any other persistent data storage being used to reflect this change. If the WebSphere Commerce database is being used, then modify the B2BRequisitioner table that stores the information about the requisitioning buyer.

ii) Provide a custom implementation of the CIData interface to capture custom information about the requisitioning buyer from the XML message and store the information in the appropriate `sessionInfo` object variables, or provide added class variables in the `registerRequisitionerCmd` command with their appropriate getter and setter methods. These properties will then be set directly, as opposed to being passed as members of the `sessionInfo` object.

iii) Modify the `performExecute()` method of the `registerRequisitionerCmd` to update the persistent data storage, the BUSPROF table for example, with the custom data.

6. Customizing the catalog subsystem

The catalog subsystem may be customized in the following ways:

- Add extension tables to the existing order subsystem schema or modify the existing tables.

- Add entity beans corresponding to the new tables or modifying the existing entity beans to reflect changes in the table schema.

- Implement new commands.

  For instructions, refer to the *WebSphere Commerce Programmers Guide Version 5.4.*

- Extend the commands

  - You can extend any of the procurement integration controller and task commands to provide custom behavior by over-riding certain methods. For instructions, refer to the *WebSphere Commerce Programmers Guide Version 5.4.*

  - Each command is based on an interface, and one way to customize a certain command is to provide a custom implementation of the required interface. A default implementation class has been provided for each interface.

a. Customizing the commands

Most of the business logic of the subsystem resides in the commands. The following table shows the list of the commands in the catalog subsystem, their interface names and the default implementation class names.

Interface name	**Default implementation class**
SendShopppingCartCmd	SendShoppingCartCmdImpl
SubmitShoppingCartCmd	SubmitShoppingCartCmdImpl

To customize the commands or develop new business logic, override the default implementation of the command interfaces.

b. Customizing the store catalog display

The catalog is displayed using a set of JSPs, some of which are common to all the stores and the rest are specific to a particular store. The common JSPs can be found in the directory `WCS_installation_Directory\WCBE_Install_Path\WCS\stores\web,` and store-specific JSPs can be found in the directory `WCS_installation_Directory>\WCBE_Install_Path\stores\web\store_name.`

c. Customizing the shopping cart

i) Customizing the quote

The CIQuote interface provides a generic interface to be implemented by any quote class. CIQuoteImpl provides the default implementation for the quote.

The quote interface consists of methods to authenticate a buyer, populate the data into the quote object, and get the shopping cart items from the quote object. The default implementation uses the PROCPROTCL and USERREG table to authenticate the buyer, and it retrieves the `logon_id` and `password` associated with the buyer from that table. To customize the quote, override the respective methods in CIQuoteImpl or give a new implementation to the CIQuote interface.

ii) Customizing the message format

Once the line items are populated into the quote object, the `ComposeShoppingCartAriba.jsp` is invoked to generate the order request message in the required format. The default implementation of this view command generates the order request in the procurement system specific XML format. To generate messages in other formats replace this JSP file with another JSP file and then register it in the VIEWREG table.

iii) Customizing database extensions

The database extensions consist of the tables CATCLSFCOD, QTYUNITMAP, CLASIFCODE, and CLSFCODEDS as well as the corresponding enterprise beans. The CATCLSFCOD table is an extension to the WebSphere Commerce table CatEntry and relates a catalog entry or product to its classification code. The CLASIFCODE and CLSFCODEDS tables list the codes for different classification schemes like SPSC and UNSPSC as well as their descriptions.

The QTYUNITMAP table contains the unit-of-measure information about the products. This table links to the existing WebSphere Commerce base tables QTYUNIT and CATENTSHIP. The QTYUNIT table lists all the UN or CEFACT standard codes. CATENTSHIP gives the assignment of these codes to catalog entries. The QTYUNITMAP table provides the one-to-one mapping of codes between UN/CEFACT standard and any NON-UN standards, for example, ISO.

Chapter 6. Use cases

The following use cases detail the flow of events when a buyer uses a procurement system with WebSphere Commerce.

Use case 1: The requisitioning buyer selects a WebSphere Commerce supplier to shop from Ariba

1. Ariba sends the `PunchOutSetupRequest` message to WebSphere Commerce with operation= “create.” This includes buyer and supplier credentials and the requisitioning buyer’s session ID in Ariba.

2. The message is mapped to the `PunchOutSetup` command.

3. The `PunchOutSetup` command invokes the `AuthenticationHelper` task command to authenticate the buyer and supplier credentials. The `AuthenticationHelper` task command, depending on the configured authentication mode (DB, LDAP or THIRD PARTY), calls the appropriate task command to perform the authentication.

4. If the authentication is successful, then the `PunchOutSetup` command invokes the `RegisterRequisitioner` task command. This command checks if the requisitioning buyer is already registered in the system. If so, it updates the relevant requisitioning buyer information such as POSTBACK URL, SESSION_ID on Ariba. If the requisitioning buyer does not exist, it registers the requisitioning buyer as a new buyer and adds the buyer to the WebSphere Commerce member group specified in the BUYSUPMAP table.

5. The supplier cookie is generated. This is the encrypted form of the information sent through `PunchOutSetupRequest` message. It is sent to Ariba, and Ariba sends it back to WebSphere Commerce for future HTTP requests.

6. The catalog that the buyer will view using the BUYSUPMAP table is determined, and the URL that must be sent back to Ariba is constructed. This is the `B2BCatalogDispaly` command URL, and the supplier cookie is the parameter.

7. The constructed URL is sent in the `PunchOutSetupResponse` XML message. When Ariba receives this message it launches a new browser window with the `B2BCatalogDisplay` command. This command retrieves the logon mode from the supplier cookie and determines it as “CREATE” so that the CatalogDisplay page is shown to the requisitioning buyer. The requisitioning buyer can now perform the usual browsing and shopping using only HTTP.

8. The requisitioning buyer checks out the shopping cart in the browser. This invokes the WebSphere Commerce `OrderPrepare` command. The order (shopping cart) is changed to ‘P’ (pending) status.

9. The display result (view task) JSP page is customized for WebSphere Commerce to call the `SubmitShoppingCart` command. This command is invoked when the requisitioning buyer presses the SUBMIT button on the JSP display page.

10. The `SubmitShoppingCart` command changes the status of the order from ‘P’ to ‘I’. This differentiates it from the non-procurement WebSphere Commerce shopping flow.

11. The `SubmitShoppingCart` command invokes the `SendShoppingCart` command. This command composes the `PurchaseOrderMessage` XML message and sends it to the POSTBACK URL in Ariba for approval. This ends the session.

Use case 2: The Approver on Ariba edits the order

1. If the Approver on Ariba is not satisfied with the shopping cart sent for approval, the approver or any other requisitioning buyer tries to edit (modify) the shopping cart (order).

2. Ariba sends the `PunchOutSetupRequest` message to WebSphere Commerce with operation=”edit.” It includes the buyer and supplier credentials, the requisitioning buyer’s session ID in Ariba, and the ShoppingCartID (ORDERS_ID column value in the ORDERS table).

3. The message is mapped to the `PunchOutSetup` command.

4. The `PunchOutSetup` command invokes the `AuthenticationHelper` task command to authenticate the buyer or supplier credentials.

5. If the authentication is successful, then the `PunchOutSetup` command invokes the `RegisterRequisitioner` task command. This command checks if the requisitioning buyer is already registered in the system. If so, it updates information such as POSTBACK URL and SESSION_ID on Ariba. If the requisitioning buyer does not exist, it registers the requisitioning buyer as a new buyer and adds the buyer to the WebSphere Commerce member group specified in the BUYSUPMAP table.

6. The order status changes from 'I' to 'P'.

7. The supplier cookie is generated.

8. The `B2BCatalogDisplay` command URL that must be sent back to Ariba is constructed, with the supplier cookie as its parameter.

9. The constructed URL is sent in the `PunchOutSetupResponse` XML message.

10. When Ariba receives this message, it launches a new browser window with the `B2BCatalogDisplay` command. This command retrieves the logonmode from the supplier cookie, based on which the Edit Shopping Cart page displays.

Use case 3: The shopping cart is approved on Ariba

1. Ariba sends the `OrderRequest` message to WebSphere Commerce with the buyer or supplier credentials. It also sends details about the shopping cart as well as the BILL TO and payment information.

2. The message is mapped to the `BatchOrderRequest` command. This command detects if the shopping cart is already in the ORDERS table using the unique PAYLOADID in the message, and it changes the existing shopping cart (order) status to ‘R’. It also creates a fresh order in batch processing by calling a series of order-related commands.

3. At the end of the processing, the `OrderResponse` message that gives the status code and status message is sent back to Ariba.

Use case 4: The requisitioning buyer builds the shopping cart using the local catalog and completes the order

1. Ariba sends the `OrderRequest` message to WebSphere Commerce with the buyer or supplier credentials. It also sends details of the shopping cart as well as the BILL TO and payment information.

2. The message is mapped to the `BatchOrderRequest` command. This command detects if the shopping cart is already in the ORDERS table using the unique PAYLOADID. It creates a fresh order in batch processing by calling a series of order-related commands.

3. At the end of processing, the `OrderResponse` message that gives the status code and status message is sent back to Ariba.

Appendix A: System_template.xml

The sample below shows an example of `system_template.xml`. This maps the `PunchOutSetupRequest` to the `PunchOutSetup` command and the `OrderRequest` to the `BatchOrderRequest` command.

```
<ECTemplate>
 <TemplateDocument>
    <DocumentType version='1.0'>cXML</DocumentType>
    <StartElement>cXML</StartElement>
    <TemplateTagName>CXMLMap</TemplateTagName>
    <CommandMapping>
      <Command CommandName='PunchOutSetup' Condition='PunchOutReq'>
        <Constant Field='protocolName'>cXML</Constant>
        <Constant Field='protocolVersion'>1.0</Constant>
      </Command>
      <Command CommandName='BatchOrderRequest' Condition='OrderReq'>
        <Constant Field='protocolName'>cXML</Constant>
        <Constant Field='protocolVersion'>1.0</Constant>
      </Command>
    </CommandMapping>
  </TemplateDocument>
  <TemplateTag name='CXMLMap'>
   <Tag XPath='Request/PunchOutSetupRequest'
        Field='PunchOutReq' FieldInfo='Command'/>
   <Tag XPath='Request/OrderRequest' Field='OrderReq'
        FieldInfo='Command'/>
   <Tag XPath='@version' Field='agentMessageVersion' />
   <!-- Unique Message Id which is used  to check Duplicate Orders -->
   <Tag XPath='@payloadID' Field='messageId' />
   <!–
        Map Buyer /MarketPlace Credentials like Organization code,
        code type. to command variables buyerCode, buyerCodeType etc.,
   -->
   <Tag XPath='Header/From/Credential@domain'  Field='buyerCodeType'/>
   <Tag XPath='Header/From/Credential/Identity'  Field='buyerCode'/>
   <Tag XPath='Header/From/Credential@type' XPathType='ATTRIBUTE'/>
   <Tag XPath='Header/From/Credential@type[@type="marketplace"]'
        Field='buyerType' />
   <Tag XPath='Header/From/Credential@domain[@type="marketplace"]'
        Field='marketPlaceType' />
   <Tag XPath='Header/From/Credential[@type="marketplace"]/Identity'
        Field='marketPlaceCode' />
   <!–
        Map Supplier Credentials like Organization code, code type.
        to command variables supplierCode, supplierCodeType etc.,
   -->
   <Tag XPath='Header/To/Credential@domain' Field='supplierCodeType' />
   <Tag XPath='Header/To/Credential/Identity' Field='supplierCode' />
   <Tag XPath='Header/Sender/Credential@domain'
        Field='supplierUserIDType' />
   <Tag XPath='Header/Sender/Credential/Identity' Field='logonId'
        FieldInfo='CONTROL' />
```

```
<Tag XPath='Header/Sender/Credential/SharedSecret'
     Field='logonPassword' FieldInfo='CONTROL' />
<Tag XPath='Header/Sender/UserAgent' Field='agentName' />
<!--
     Map PunchOutSetupRequest message operation mode to logonMode.
     This operation’s can be ‘create’,’edit’, and ‘inspect’
-->
<Tag XPath='Request/PunchOutSetupRequest@operation'
     Field='logonMode'/>
<!--
     Map  AribaBuyer’s  BuyerCookie to sessionId. This is included
     in the Shopping cart message which is sent to procurement
     system.
-->
<Tag XPath='Request/PunchOutSetupRequest/BuyerCookie'
     Field='sessionId'/>
<!--
     User Id and Department name is sent using Extrinsics. This
     information is configured during Buyer Organization registration.
-->
<Tag XPath='Request/PunchOutSetupRequest/Extrinsic'
     XPathType='USERDATA'/>
<!--
     orderApprovalURL is the Url in the procurement system to which
     shopping cart is sent
-->
<Tag XPath='Request/PunchOutSetupRequest/BrowserFormPost/URL'
     Field='orderApprovalURL'/>
<!--
     quoteNumber is the WCS order id which was created when the
     shopping cart was created.
-->
<Tag XPath='Request/PunchOutSetupRequest/ItemOut/ItemID/
     SupplierPartAuxiliaryID' Field='quoteNumber'/>
<!--
     Map BuyerOrderId which is created at procurement system side.
-->
<Tag XPath='Request/OrderRequest/OrderRequestHeader@type'
     Field='orderMode'/>
<Tag XPath='Request/OrderRequest/OrderRequestHeader@orderID'
     Field='buyerOrderID' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader@orderDate'
     Field='buyerOrderDate' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/Total/Money'
     Field='totalAmount' />
<!--
     Map the Requisitioner ship to address to appropriate fields in
     the command.
-->
<!-- ShipTo Address at the Order Level -->
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo'
     XPathType='VECTOR' Field='order_shipTo_vector' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Name' Field='name' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/PostalAddress/DeliverTo' Field='deliverTo' />
 <Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/Address/
```

```
     PostalAddress/Street' XPathType='REPEAT' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/PostalAddress/Street[1]' Field='streetAddress1' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/Address/
     PostalAddress/Street[2]'Field='streetAddress2' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/PostalAddress/Street[3]' Field='streetAddress3' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/PostalAddress/City' Field='city' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/PostalAddress/State' Field='state' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/PostalAddress/PostalCode' Field='postalCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/PostalAddress/Country' Field='country' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Phone@name' Field='telephoneType' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Phone/TelephoneNumber/CountryCode'
     Field='phoneCountryCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Phone/TelephoneNumber/AreaOrCityCode'
     Field='phoneAreaCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Phone/TelephoneNumber/Number'  Field='phoneNumber' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Fax@name'Field='faxType' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Fax/TelephoneNumber/CountryCode'
     Field='faxCountryCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Fax/TelephoneNumber/AreaOrCityCode'
     Field='faxAreaCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Fax/TelephoneNumber/Number'  Field='faxNumber' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Email' Field='email' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/ShipTo/
     Address/Email@name' Field='emailType' />
<!-- BillTo Address at the Order Level -->
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo'
     XPathType='VECTOR'Field='order_billTo_vector' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Name' Field='name' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/PostalAddress/DeliverTo'Field='deliverTo' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/PostalAddress/Street'XPathType='REPEAT' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/PostalAddress/Street[1]' Field='streetAddress1' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/PostalAddress/Street[2]' Field='streetAddress2' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/PostalAddress/Street[3]' Field='streetAddress3' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/PostalAddress/City' Field='city' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
```

```
     Address/PostalAddress/State' Field='state' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/PostalAddress/PostalCode' Field='postalCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/PostalAddress/Country' Field='country' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Phone@name' Field='telephoneType' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Phone/TelephoneNumber/CountryCode'
     Field='phoneCountryCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Phone/TelephoneNumber/AreaOrCityCode'
     Field='phoneAreaCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Phone/TelephoneNumber/Number' Field='phoneNumber' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Fax@name' Field='faxType'       />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Fax/TelephoneNumber/CountryCode'
     Field='faxCountryCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Fax/TelephoneNumber/AreaOrCityCode'
     Field='faxAreaCode' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Fax/TelephoneNumber/Number' Field='faxNumber' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Email' Field='email' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/BillTo/
     Address/Email@name' Field='emailType' />
<!-- Shipping Info -->
<Tag XPath='Request/OrderRequest/OrderRequestHeader/Shipping'
     XPathType='VECTOR'Field='shippingInfo' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/Shipping/Money'
     Field='shippingCharge' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/
     Shipping/Money@currency'  Field='currency'      />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/
     Shipping/Description'  Field='discription' />
<Tag XPath='Request/OrderRequest/OrderRequestHeader/Comments'
     Field='comment' />
<!--  Extrinisics at the Order level -->
<Tag XPath='Request/OrderRequest/OrderRequestHeader/Extrinsic'
     XPathType='USERDATA' />
<!--  Order Items -->
<Tag XPath='Request/OrderRequest/ItemOut' XPathType='VECTOR'
     Field='order_items_vector' />
<Tag XPath='Request/OrderRequest/ItemOut@quantity' Field='quantity' />
<Tag XPath='Request/OrderRequest/ItemOut@requestedDeliveryDate'
     Field='requestDeliveryDate' />
<Tag XPath='Request/OrderRequest/ItemOut/ItemID/SupplierPartID'
     Field='itemID' />
<Tag XPath='Request/OrderRequest/ItemOut/ItemID/
     SupplierPartAuxiliaryID' Field='quoteNumber'/>
<Tag XPath='Request/OrderRequest/ItemOut/ItemDetail/
     UnitPrice/Money@currenty' Field='currency' />
<Tag XPath='Request/OrderRequest/ItemOut/ItemDetail/
     UnitPrice/Money' Field='itemPrice' />
```

```
    <Tag XPath='Request/OrderRequest/Itemout/ItemDetail/Description'
         Field='itemDescription' />
    <Tag XPath='Request/OrderRequest/ItemOut/ItemDetail/UnitOfMeasure'
         Field='unitOfMeasure' />
    <Tag XPath='Request/OrderRequest/ItemOut/ItemDetail/
        Classification@domain' Field='classificationDomain' />
    <Tag XPath='Request/OrderRequest/ItemOut/ItemDetail/Classification'
         Field='classificationCode' />
    <!-- Shipto at the Order Item level -->
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo' XPathType='VECTOR'
         Field='order_Item_shipTo' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/Name'
         Field='name' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/DeliverTo'Field='deliverTo' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/Street' XPathType='REPEAT' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/Street[1]' Field='streetAddress1' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/Street[2]' Field='streetAddress2' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/Street[3]' Field='streetAddress3' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/City' Field='city' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/State' Field='state' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/PostalCode' Field='postalCode' />
    <Tag XPath='Request/OrderRequest/ItemOut/ShipTo/Address/
         PostalAddress/Country' Field='country'
     />
    <!--  Extrinisics at the Order Item level -->
    <Tag XPath='Request/OrderRequest/ItemOut/Extrinsic'
         XPathType='USERDATA'  />
    </TemplateTag>
</ECTemplate>
```

Note: If the protocol has different versions and procurement integration is to support that protocol, you must add the TemplateDocument for that version as given below:

```
<TemplateDocument>
    <DocumentType version='1.1.008'>cXML</DocumentType>
    <StartElement>cXML</StartElement>
    <TemplateTagName>CXMLMap</TemplateTagName>
    <CommandMapping>
       <Command CommandName='PunchOutSetup'
             Condition='PunchOutReq'>
             <Constant Field='protocolName'>cXML</Constant>
             <Constant Field='protocolVersion'>1.0</Constant>
      </Command>
      <Command CommandName='BatchOrderRequest' Condition='OrderReq'>
       <Constant Field='protocolName'>cXML</Constant>
       <Constant Field='protocolVersion'>1.0</Constant>
      </Command>
    </CommandMapping>
  </TemplateDocument>
```

Appendix B: Sample XML messages

1. `PunchOutSetupRequest`

This message initiates the interaction from Ariba to WebSphere Commerce. It refers to use cases 1 and 2.

The WebSphere Commerce messaging subsystem maps this message to the `PunchOutSetup` command.

The following are other commands triggered by this message:

- `AuthenticatonHelperCmd`
- `DBAuthenticationCmd`
- `LdapAuthenticationCmd`
- `ThirdPartyB2BAuthCmd`
- `RegisterRequisitionerCmd`

Sample message:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "cXML.dtd">
<cXML version="1.0" payloadID="992114492875.591694834@sandbox21"
      timestamp="2001-06-09T12:21:32-07:00">
 <Header>
  <From>
   <Credential domain="AribaNetworkId">
    <Identity>nkrishna@in.ibm.com</Identity>
   </Credential>
  </From>
  <To>
   <Credential domain="DUNS">
    <Identity>9143470144</Identity>
   </Credential>
  </To>
  <Sender>
   <Credential domain="AribaNetworkUserId">
    <Identity>nkrishna@in.ibm.com</Identity>
    <SharedSecret>catalog</SharedSecret>
   </Credential>
   <UserAgent>Ariba ORMS 6.1</UserAgent>
  </Sender>
 </Header>
 <Request>
  <PunchOutSetupRequest operation="create">
   <BuyerCookie>V8UE19ALU9IL</BuyerCookie>
   <Extrinsic name="CostCenter">610</Extrinsic>
   <Extrinsic name="User">vlo</Extrinsic>
   <BrowserFormPost>
    <URL>http://sandbox21:3377/punchout</URL>
   </BrowserFormPost>
   <SupplierSetup>
```

```
    <URL>http://budhoo.hawthorne.ibm.com/webapp/cib2b/PunchOut</URL>
   </SupplierSetup>
   <ShipTo>
    <Address addressID="1000487">
     <Name xml:lang="en">Los Gatos</Name>
     <PostalAddress>
      <DeliverTo>Vincent Lo</DeliverTo>
      <Street>15 Camino del Cerro</Street>
      <City>Los Gatos</City>
      <State>CA</State>
      <PostalCode>95032</PostalCode>
      <Country isoCountryCode="US">United States</Country>
     </PostalAddress>
    </Address>
   </ShipTo>
  </PunchOutSetupRequest>
 </Request>
</cXML>
```

2. `OrderRequest`

This message is sent from Ariba to WebSphere Commerce to create an approved order. It refers to use cases 3 and 4.

The WebSphere Commerce messaging subsystem maps this message to the `CIPurchaseOrder` command.

The following are other commands that are indirectly invoked by this message:

- `B2BOrderPrepareCmd`
- `CreatePurchaseOrderCmd`
- `CreateBatchOrderItemCmd`
- `OrderCompleteCmd`
- `ShipBillToAddressCmd`

Sample message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM
     "http://xml.cxml.org/schemas/cXML/1.1.006/cXML.dtd">
<cXML version="1.1"
      payloadID="991013064000.1@sandbox21.nkrishna@in.ibm.com"
      timestamp="2001-05-27T18:24:24-07:00">
 <Header>
  <From>
   <Credential domain="AribaNetworkId">
    <Identity>nkrishna@in.ibm.com</Identity>
   </Credential>
  </From>
  <To>
   <Credential domain="DUNS">
    <Identity>9143470144</Identity>
   </Credential>
  </To>
  <Sender>
   <Credential domain="AribaNetworkUserId">
    <Identity>nkrishna@in.ibm.com</Identity>
    <SharedSecret>catalog</SharedSecret>
   </Credential>
   <UserAgent>Ariba Network V1.1 </UserAgent>
  </Sender>
 </Header>
 <Request>
  <OrderRequest>
   <OrderRequestHeader orderID="DO1452"
       orderDate="2001-05-27T18:24:24-07:00" type="new" >
    <Total>
     <Money currency="USD" >1,222</Money>
    </Total>
    <ShipTo>
     <Address isoCountryCode="US" addressID="1000487">
      <Name xml:lang="en">Los&#032;Gatos</Name>
      <PostalAddress name="default">
       <DeliverTo>Vincent&#032;Lo</DeliverTo>
       <DeliverTo>Los&#032;Gatos</DeliverTo>
       <Street>15 Camino del Cerro </Street>
       <City>Los&#032;Gatos</City>
       <State>CA</State>
       <PostalCode>95032</PostalCode>
       <Country isoCountryCode="US">United&#032;States</Country>
      </PostalAddress>
      <Email name="default">tvlo@&lt;YourSmtpDomainName&gt;</Email>
      <Phone name="work">
       <TelephoneNumber>
        <CountryCode isoCountryCode="US">1</CountryCode>
        <AreaOrCityCode>408</AreaOrCityCode>
        <Number>3582000</Number>
       </TelephoneNumber>
      </Phone>
      <Fax name="work">
       <TelephoneNumber>
        <CountryCode isoCountryCode="US">1</CountryCode>
        <AreaOrCityCode>408</AreaOrCityCode>
        <Number>3582100</Number>
       </TelephoneNumber>
      </Fax>
     </Address>
    </ShipTo>
    <BillTo>
     <Address isoCountryCode="US" addressID="15">
      <Name xml:lang="en">Ariba&#032;Headquarters</Name>
      <PostalAddress name="default">
       <Street>1314 Chesapeake Terrace</Street>
       <City>Sunnyvale</City>
       <State>CA</State>
       <PostalCode>94089</PostalCode>
       <Country isoCountryCode="US">United&#032;States</Country>
      </PostalAddress>
     </Address>
    </BillTo>
    <Shipping>
     <Money currency="USD">6</Money>
     <Description
        xml:lang="en">International&#032;mail</Description>
    </Shipping>
   </OrderRequestHeader>
    <ItemOut quantity="1"  lineNumber="1">
     <ItemID>
      <SupplierPartID>6565E2U</SupplierPartID>
      <SupplierPartAuxiliaryID>15051</SupplierPartAuxiliaryID>
     </ItemID>
     <ItemDetail>
      <UnitPrice>
       <Money currency="USD">1,222</Money>
      </UnitPrice>
      <Description xml:lang="en">PC&#032;300PL&#032;
       (with&#032;Pentium&#032;III&#032;processors)
      </Description>
      <UnitOfMeasure>EA</UnitOfMeasure>
      <Classification domain="Not Available">
         Not Available
      </Classification>
      <ManufacturerPartID>6565E2U</ManufacturerPartID>
      <ManufacturerName>IBM</ManufacturerName>
     <URL>http://budhoo.hawthorne.ibm.com/webapp/cib2b/PunchOut</URL>
      <Extrinsic name="PR&#032;No.">PR6150</Extrinsic>
      <Extrinsic name="Requester">Vincent&#032;Lo</Extrinsic>
     </ItemDetail>
     <Distribution>
      <Accounting name="DistributionCharge">
       <Segment type="Cost&#032;Center"
         id="Engineering&#032;Management"
         description="Department&#032;Name"/>
       <Segment type="Account" id="Office&#032;Supplies"
           description="Account&#032;Name"/>
      </Accounting>
      <Charge>
       <Money currency="USD">40</Money>
      </Charge>
     </Distribution>
    </ItemOut>
   </OrderRequest>
  </Request>
 </cXML>
```

3. `PunchOutSetupResponse`

This message initiates the interaction from WebSphere Commerce to Ariba. It refers to use case 3.

The WebSphere Commerce messaging subsystem maps this message to the `PunchOutSetup` command.

The following are other commands triggered by this message:

- `AuthenticatonHelperCmd`
- `DBAuthenticationCmd`
- `LdapAuthenticationCmd`
- `ThirdPartyB2BAuthCmd`
- `RegisterRequisitionerCmd`

Sample message

```
<?xml version="1.0" encoding="UTF-8"?>
            <!DOCTYPE cXML SYSTEM
                "http://xml.cXML.org/schemas/cXML/1.1.008/cXML.dtd" >
            <cXML version="1.1.008"
                  payloadID = "992444231515.1@budhoo.hawthorne.ibm.com"
                  timestamp = "2001-06-13T10:57:11-5.00"
                  xml:lang = "en-US" >
             <Response>
              <Status code="200" text="OK"></Status>
              <PunchOutSetupResponse>
               <StartPage>
                <URL>
                  https://budhoo.hawthorne.ibm.com/webapp/wcs/stores/
                  servlet/PunchOutCatalogDisplay?
                  supplierCookie=
                  gUZK0k6PWNDOAYKLrUKKwR/ZTUjPwxI14/6fTVP5IDQ=
                </URL>
               </StartPage>
              </PunchOutSetupResponse>
             </Response>
            </cXML>
```

4. `PunchOutOrderMessage`

This message initiates the interaction from WebSphere Commerce to Ariba. It refers to use cases 1 and 2.

The WebSphere Commerce messaging subsystem maps this message to the `SendShoppingCartCmd` command.

Sample message:

```
<?xml version="1.0" encoding="UTF-8"?>
            <!DOCTYPE cXML SYSTEM
                "http://xml.cXML.org/schemas/cXML/1.1.008/cXML.dtd">
            <cXML timestamp = "2001-06-13T11:07:27-5.00"
                  payloadID = "992444847906.1@sreed.in.ibm.com"
                  version =   "1.1.008" >
              <Header>
                <From>
                  <Credential domain="DUNS">
                    <Identity>9143470144</Identity>
                  </Credential>
                </From>
                <To>
                  <Credential domain="AribaNetworkId">
                    <Identity>nkrishna@in.ibm.com</Identity>
                  </Credential>
                </To>
                <Sender>
                  <Credential domain="AribaNetworkUserId">
                    <Identity>nkrishna@in.ibm.com</Identity>
                  </Credential>
                  <UserAgent>IBM WCS 5.1</UserAgent>
                </Sender>
              </Header>
```

```
  <Message>
    <PunchOutOrderMessage>
     <BuyerCookie>5FP3YIRS6U1T</BuyerCookie>
     <PunchOutOrderMessageHeader operationAllowed="edit">
      <Total>
       <Money currency="USD">35.00000</Money>
      </Total>
      <Shipping trackingDomain="-">
       <Money currency="USD">6.00000</Money>
       <Description xml:lang="en-US">
          International mail
       </Description>
      </Shipping>
     </PunchOutOrderMessageHeader>
     <ItemIn quantity="1">
      <ItemID>
       <SupplierPartID>1161210</SupplierPartID>
       <SupplierPartAuxiliaryID> 15551
       </SupplierPartAuxiliaryID>
      </ItemID>
      <ItemDetail>
       <UnitPrice>
        <Money currency="USD">35.00000</Money>
       </UnitPrice>
       <Description xml:lang="en-US">
         ThinkPad i Series 1200
       </Description>
       <UnitOfMeasure>EA</UnitOfMeasure>
       <Classification domain="null"></Classification>
       <ManufacturerPartID>1161210</ManufacturerPartID>
       <ManufacturerName>IBM</ManufacturerName>
       <Extrinsic name ="User">kitty</Extrinsic>
      </ItemDetail>
     </ItemIn>
    </PunchOutOrderMessage>
  </Message>
</cXML>
```

5. **`OrderResponse`**

   This message initiates the interaction from WebSphere Commerce to Ariba. It refers to use cases 3 and 4.

   The WebSphere Commerce messaging subsystem maps this message to the `BatchOrderRequestCmd` command.

Sample message:

```
<?xml version="1.0" encoding="UTF-8"?>
            <!DOCTYPE cXML SYSTEM
                "http://xml.cXML.org/schemas/cXML/1.1.008/cXML.dtd">
            <cXML timestamp = "2001-06-13T11:07:27-5.00"
                  payloadID = "992444847906.1@sreed.in.ibm.com"
                  version = "1.1.008" >
              <Response>
               <Status code="200" text="OK"> </Status>
              </Response>
            </cXML>
```

Appendix C: LogonRequest in HTML format

Some procurement systems send a `LogonRequest` message in HTML format as a set of name-value pairs either in the query string or as a part of post data. Procurement integration expects XML to be the message transport format; as a result, additional configuration is necessary to handle HTTP requests.

For example, Open Catalog Interface (OCI) is a protocol that supports a logon request to the supplier’s site in HTML format. OCI is used by the mySap e-Procurement system and the Commerce One BuySite e-Procurement system. Consider an OCI request coming in from the procurement system to WebSphere Commerce with the following parameters:

HOOK_URL	The URL to which the shopping cart is to be submitted	OCI Mandatory parameter. The URL used to return to the SAP Business-to-Business Procurement application from the catalog application. It is automatically filled at runtime by the procurement application.
SUPPLIER_CODE		Additional parameter for the supplier credential. The supplier requires this.
SUPPLIER_CODE_DOMAIN		Additional parameter for supplier credential. The supplier requires this.
BUYER_CODE		Additional parameter for buyer credential. The supplier requires this.
BUYER_CODE_DOMAIN		Additional parameter for buyer credential. The supplier requires this.
USERNAME		Additional parameter, the buyer’s group user id is registered with the supplier. The supplier requires this.
PASSWORD		Additional parameter, the buyer’s group password is registered with the supplier. The supplier requires this.
REQID		Additional parameter, unique requisitioner id used for dynamic registration of the buyers. The supplier requires this.
COST_CENTER		Additional parameter, indicating the buyers cost center. This is optional.
~OkCode	ADDI	OCI mandatory parameter. Contains the transaction code indicating that the function Add Items to SAP shopping basket is to be performed. Must be set to ADDI for SAP Business-to-Business Procurement.
~CALLER	CTLG	OCI mandatory parameter. Indicates that the data was sent by an external catalog. The content must be set to CTLG.
~TARGET	_top	OCI mandatory parameter. Specifies the frame to which a catalog is to return in a frame-based environment. If this field is not set, the catalog application must provide a default target of _top.
OCI_VERSION	2	OCI mandatory parameter. SAP B2B Procurement system version.

Note: For more details about the OCI parameters, refer to the *Open Catalog Interface, release 2.0B specification* document.

The additional functions supported by OCI include the following:

1. **Product details**

   The following are the name and value pairs that will be an additional transfer to the catalog:

   a. FUNCTION="DETAIL"
   b. PRODUCTID="database key of product in the catalog"

2. **Product validation**

   The following are the name and value pairs that will be an additional transfer to the catalog:

   a. FUNCTION="VALIDATE”
   b. PRODUCTID="database key of product in the catalog"

3. **Product sourcing**

   The following are the name and value pairs that will be an additional transfer to the catalog:

   a. FUNCTION="VALIDATE”
   b. SEARCHSTRING="string to directly start the catalog search" and
   c. VENDOR="vendor number in the buyer's system"

Note: The functions "DETAIL" and "VALIDATE" will only work if the parameter NEW_ITEM-EXT_PRODID[n] was filled from the catalog in a previous call.

Modifications in `PunchOutSetupCmd`

The `PunchOutSetupCmd` instantiates the CIData class and receives the B2B logon requests. You must modify the implementation of the CIData class to parse and return the HTML parameters as represented in OCI. Once CIData is implemented to do this, modify the `PunchOutSetupCmd` to instantiate the new implementation of CIData. To achieve this override the `PunchOutSetupCmd` and register it with the CMDREG table within WebSphere Commerce. For more information on how to override a command, refer to the *WebSphere Commerce Programmer’s Guide Version 5.4*.

The following is a sample implementation of the setRequestProperties method of the `PunchOutSetupCmd` that is invoked to set the parameters to the command. Here, this method is modified to instantiate the new CIDataImpl.

```
public void
setRequestProperties(com.ibm.commerce.datatype.TypedProperty p)
throws com.ibm.commerce.exception.ECException
   {
      final String strMethodName = "setRequestProperties";
      ECTrace.entry(ECTraceIdentifiers.COMPONENT_USER,
       GetClass().getName(), strMethodName);
      this.requestProperties = p;
   // now set the properties in the new ciData object
   CiData = new NewCIData();
   ciData.setLogonData(p);
   ECTrace.exit(ECTraceIdentifiers.COMPONENT_USER,
          GetClass().getName(), strMethodName);
```

Modifications in the CIData

The `PunchOutSetupCmd` instantiates the CIData implementation class to parse and return the input parameters from the request. In this case, the input parameters are as per the OCI specifications. This section describes a different implementation for the CIData interface.

The CIData class encapsulates the login data and the purchase order data. The default implementation of CIData is CIDataImpl, which enables the implementation for cXML protocol. Refer to the WebSphere Commerce documentation for details of the methods in the CIData interface.

For an HTML request, a new implementation of CIData that implements the methods of the interface by parsing the HTML query string and then returning the respective data is necessary. The following is a sample implementation of CIData to parse the OCI-specific data.

```
package com.ibm.commerce.ci.oci.objects;

import com.ibm.commerce.me.datatype.SessionInfo;
import com.ibm.commerce.me.datatype.Credentials;
import com.ibm.commerce.me.datatype.Agent;
import com.ibm.commerce.me.datatype.PurchaseOrderHeader;

public class newCIData implements CIData
{
    // the parameters coming in as the logon request
    com.ibm.commerce.datatype.TypedProperty p;


    /**
     * get the bill to address
     *
     * @return com.ibm.commerce.me.datatype.Address
     */
    Address getBillTo()
    {
        // no billTo is available with OCI Protocol
        return null;
    }


    /**
     * get the department name
     *
     * @return java.lang.String
     */
    String getDepartment()
    {
        // in OCI the cost center indicates the
        // department name. The key is the HTTP
        // request parameter (COST_CENTER)
        return p.getString("COST_CENTER", null);
    }

    /**
     * get the logon data
```

```
 *
 * @return com.ibm.commerce.datatype.TypedProperty
 */
com.ibm.commerce.datatype.TypedProperty getLogonData
{
    return p;
}

/**
 * get the purchase order data
 *
 * @return com.ibm.commerce.datatype.TypedProperty
 */
com.ibm.commerce.datatype.TypedProperty getPOData()
{
    // no POData is available with OCI logon request
    return null;
}

/**
 * get the line items in the purchase order request
 *
 * @return java.util.Vector
 */
java.util.Vector getPOItems()
{
    // no line items is available with OCI logon request
    return null;
}

/**
 * get the name of the requisitioner
 *
 * @return java.lang.String
 */
String getRequisitioner()
{
    return p.getString("REQID", null);
}

/**
 * set the logon data
 *
 * @param props com.ibm.commerce.datatype.TypedProperty
 */
void setLogonData(com.ibm.commerce.datatype.TypedProperty props)
{
    this.p = props;
}

/**
 * set the purchase order data
 *
 * @param props com.ibm.commerce.datatype.TypedProperty
 */
void setPOData(com.ibm.commerce.datatype.TypedProperty props)
{
    // no need to implement...
}

/**
 * get the buyer's software agent
 *
 * @return com.ibm.commerce.me.datatype.Agent
 */
Agent getAgent()
{
    // not available
    return null;
}

/**
 * get the buyer credentials
 *
 * @return com.ibm.commerce.me.datatype.Credentials
 */
Credentials getBuyerCredentials()
{
    if (buyerCred == null)
    {
        // set the buyer credentials
        buyerCred = new Credentials();
        buyerCred.setCode(p.getString("BUYER_CODE", null));
        buyerCred.setCodeDomain(p.getString("BUYER_CODE_DOMAIN", null));
        buyerCred.setUserId(p.getString("USERNAME", null));
        buyerCred.setUserIdType(null);
        buyerCred.setPassword(p.getString("PASSWORD", null));
    }
    return buyerCred;
}

/**
 * get the marketPlace credentials
 *
 * @return com.ibm.commerce.me.datatype.Credentials
 */
Credentials getMarketPlaceCredentials()
{
    // not available
    return null;
}

/**
 * get the purchase order request header
 *
 * @return com.ibm.commerce.me.datatype.PurchaseOrderHeader
 */
PurchaseOrderHeader getPOHeader()
{
    // not available
    return null;
}

/**
```

```
     * get the session information
     *
     * @return com.ibm.commerce.me.datatype.SessionInfo
     */
    SessionInfo getSessionInfo()
    {
        if (this.sessionInfo == null)
        {
            sessionInfo = new SessionInfo();
            sessionInfo.setReqId(p.getString("REQID", null));
            sessionInfo.setReqName(null);
            sessionInfo.setSessionId(null);
            sessionInfo.setSessionType(null);
            sessionInfo.setDeptName(p.getString("COST_CENTER", null););
            sessionInfo.setPostBackURL(p.getString("HOOK_URL", null));
            sessionInfo.setOrderStatusUrl(null);

            // OCI can support functions like EDIT and DETAIL
            String s = p.getString("FUNCTION", null)
            if (s != null & s.equalsIgnoreCase("DETAIL"))
            {
                //detail function. see the details of a product.
                sessionInfo.setLogonMode(B2BMemberConstants.DISPLAY_MODE);
                Long l = p.getLong("PRODUCTID", null);
                if (l != null)
                {
                    sessionInfo.setItemId(l.longValue());
                }
            }
            else if (s != null & s.equalsIgnoreCase("VALIDATE"))
            {
                //Validate function. validate a product.
                sessionInfo.setLogonMode(B2BMemberConstants.INSPECT_MODE);
                Long l = p.getLong("PRODUCTID", null);
                sessionInfo.setItemId(l.longValue());

                String searchString = p.getString("SEARCHSTRING", null);
                if (searchString != null)
                {
                    // validate mode with search, not supported.
                    String vendor = p.getString("VENDOR", null);
                }
            }
            else
            {
                sessionInfo.setLogonMode(B2BMemberConstants.CREATE_MODE);
            }
        }
        return sessionInfo;
    }

    /**
     * get the supplier credentials
     *
     * @return com.ibm.commerce.me.datatype.Credentials
     */
    Credentials getSupplierCredentials()
    {
        if (supplierCred == null)
        {
            supplierCred = new Credentials();
            supplierCred.setCode(p.getString("SUPPLIER_CODE", null));
            supplierCred.setCodeDomain(p.getString("SUPPLIER_CODE_DOMAIN", null));
            supplierCred.setUserId(null);
            supplierCred.setUserIdType(null);
            supplierCred.setPassword(null);
        }
        return supplierCred;
    }
}
```

Appendix D: Sample buyer information form

The following is the sample buyer information form provided in the `BuyerInfo.txt` file:

```
Please fill out the form below and email it to merchant_admin@your_company.com
or fax it to (914) 555 0000.

Please fill out the following information about your organization:

Buyer Organization Name: _________________________________________________

Organization Code (test): ________________________________________________

Organization Code (production: _________________________________________

Organization Code Domain: ________________________________________________

Department Extrinsics Name: ______________________________________________

User/Requisitioner Extrinsic Name: _______________________________________


Phone Number: __________________ Fax Number: _____________________

Email Address: ___________________________________________

Fax Number: ______________________________________________

Address:
     Street: _____________________________________________
             _____________________________________________
             _____________________________________________
       City: _____________________________________________
      State: _____________________________________________
ZIP/Postal Code: _________________________________________
    Country: _____________________________________________________

Contact Information:
              Title: _____________________________________________
         First Name: _____________________________________________
             Middle: _____________________________________________
          Last Name: _____________________________________________
          Primary Phone Number: ______________ Alt Phone Number: ______________
             Fax Number: _________________________________
      Email Address: _____________________________________
  Alt Email Address: _____________________________________
```

Notices and trademarks

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Ltd.
Department 071
1150 Eglinton Avenue East
North York, Ontario, M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This document may contain information about other companies' products, including references to such companies' Internet sites. IBM has no responsibility for the accuracy, completeness, or use of such information.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States or other countries or both:

IBM **DB2 Extenders**
AIX **DB2 Universal Database**
DB2 **MQSeries**

Microsoft, Windows and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Solaris, Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc.

Lotus and Domino are trademarks or registered trademarks of Lotus Development Corporation.

SAP is a registered trademark of SAP AG.