Integration Guide for WebSphere Commerce and cXML

Version 6.0

Integration Guide for WebSphere Commerce and cXML

Version 6.0

Note
Before using this information and the product that it supports, read the information in "Notices" on page 53.

Edition Notice

This edition applies to IBM WebSphere Commerce, Version 6.0 and to all subsequent releases and modifications of the above listed products, until otherwise indicated in new editions. Make sure that you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office that serves your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. You can send your comments by using the online IBM WebSphere Commerce documentation feedback form, available at the following URL:

http://www.ibm.com/software/commerce/rcf.html

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

Before you begin v
Knowledge requirements vi
Conventions used in this guide vi
Paths variables vi

Chapter 1. Introduction 1
Terminology 1
Overview 2
Major capabilities 3
Business models enabled 3
Prerequisites 4
Software requirements 4
Hardware requirements 4
References 4

Chapter 2. cXML scenarios and messages for procurement system integration 5
Scenarios for procurement system integration . . . 5
PunchOut using Ariba SN 5
Profile transaction using Ariba SN 6
Using local catalogs 6
cXML messages for procurement system integration 6

Chapter 3. Configuring the reference application 9
Configuring a store 9
Configuring WebSphere Commerce for the supplier and buyer 11
Configuring supplier settings 11
Loading the access control policy 12
Getting buyer invitations and soliciting buyer information 12
Registering the buyer 12
Creating a business account and a contract between the supplier and the buyer 13
Configuring buyer settings 14
Enabling the WebSphere Commerce messaging subsystem for cXML 15

Chapter 4. Verification Procedure . . . 19
Verification procedure for PunchOut 19
Verification procedure for BatchOrder 20
Verification procedure for ProfileRequest . . . 20

Chapter 5. Exporting catalogs from WebSphere Commerce 21
Extracting and converting catalog data to cXML . . 21
Extracting and converting catalog data to CIF . . . 22

Chapter 6. Customizing procurement system integration. 25
Enabling procurement system integration for other procurement systems 25
Registering a new protocol with procurement system integration 25
Creating a configuration file 26
Customizing JSP files 27
Customizing commands for the PunchOut process 27
Customizing CIData 27
Customizing authentication 28
Customizing store catalog display in PunchOutSetupCmd 29
Customizing commands for the PurchaseOrder process 30
Customizing CIData 30
Customizing authentication 31
Customizing the catalog subsystem 31
Customizing the store catalog display 32
Customizing the shopping cart 32

Chapter 7. Migrating from WebSphere Commerce 5.6.1 to WebSphere Commerce 6.0 33

Appendix A. Sample XML messages . . 35
PunchOutSetupRequest message 35
BatchOrderRequest message 36
PunchOutSetupResponse message 37
BatchOrderResponse message 38
ProfileRequest message 38

Appendix B. Sample buyer information form 41

Appendix C. Sample catalog files for cXML 43
Sample cXML index file 43
Sample cXML supplier file 44
Sample cXML contract file 45

Appendix D. Sample catalog files for CIF 47
Sample CIF index file 47
Sample CIF contract file 47

Appendix E. Mapping Information . . . 49

Notices 53
Trademarks 54

Before you begin

The *Integration Guide WebSphere®® Commerce and cXML* provides information about the features and the major capabilities of the integration between WebSphere Commerce 6.0 Enterprise and Ariba Buyer through the Ariba Supplier Network (Ariba SN) using cXML. Customers using WebSphere Commerce as a seller system can connect to buyers through Ariba Buyer and the Ariba SN, using the capabilities described in this guide.

This guide is intended for WebSphere Commerce administrators, programmers, and other experts. The WebSphere Commerce procurement subsystem is a generic framework that enables WebSphere Commerce Version 6.0 Enterprise to handle B2B transactions using industry-standard protocols. It provides an extensible and customizable functionality in WebSphere Commerce, which allows you to extend the message, schema, or business logic.

This guide talks about the business models enabled with this integration, creating and configuring buyers and suppliers, configuring a business-to-business (B2B) store, customizing procurement system integration, and supporting local catalogs.

The guide is divided into the following sections:

Chapter 1. Introduction A brief overview of procurement system integration, cXML, CIF (Catalog Interface Format) as well as the definition of the terms used in this guide, prerequisites, and related references.

Chapter 2. cXML scenarios and messages for procurement system integration Describes the scenarios to integrate WebSphere Commerce with Ariba using cXML. The cXML messages that procurement system integration uses are listed.

Chapter 3. Configuring the reference application Describes how to configure the buyer and supplier settings. It provides information on configuring a store for procurement system integration.

Chapter 4. Verification procedure This chapter provides the procedural details for ensuring the end-to-end flow of a test message.

Chapter 5. Exporting catalogs from WebSphere Commerce Describes how to export catalog data in cXML and CIF catalog file formats from WebSphere Commerce.

Chapter 6. Migrating from WebSphere Commerce 5.6.1 to WebSphere Commerce 6.0 This chapter provides the procedural details on how to migrate from WebSphere Commerce 5.6.1 to WebSphere Commerce 6.0.

Appendix A. Sample XML messages Describes the sample XML messages used by this reference application.

Appendix B. Sample buyer information form Contains a sample buyer information form.

Appendix C. Sample catalog files for cXML Contains the sample catalog files for cXML format.

Appendix D. Sample catalog files for CIF Contains the sample catalog files for CIF format.

Appendix E. Mapping information Contains the mapping information for this reference application.

Knowledge requirements

In order to use or customize this reference application , knowledge of cXML, and PunchOut is mandatory. For more information, refer to http://www.cxml.org. You will also require knowledge in the following areas:

- WebSphere Commerce 6.0 Enterprise
- IBM® WebSphere Application Server
- Java™
- JavaServer Pages technology
- Enterprise Java Beans™ (enterprise beans)
- e-procurement systems
- WebSphere Commerce catalogs and data management utilities
- Struts
- XML and CIF

Conventions used in this guide

This guide uses the following conventions:

Boldface type	Indicates commands or graphical user interface (GUI) controls such as names of fields, buttons, or menu choices.
`monospaced type`	Indicates examples of text that you enter exactly as shown.
Italic type	Used to emphasize words. Italics also indicate names for which you must substitute the appropriate values for your system.

Paths variables

This guide uses the following variables to represent directory paths:

WC_installdir This is the installation directory for WebSphere Commerce. The default installation directory for WebSphere Commerce is

```
C:\ProgramFiles\WebSphere\CommerceServer60
```

WAS_installdir This is the installation directory for WebSphere Application Server. The default installation directory for WebSphere Application Server is

```
C:\ProgramFiles\WebSphere\AppServer
```

Chapter 1. Introduction

This chapter gives an overview of procurement integration. It defines the terms used in this guide, lists the software and hardware prerequisites, and provides references to other related sources of information.

Terminology

The following terms are used in this guide:

Catalogs
Catalogs convey product and service content to buying organizations. They describe the products and services that you offer, as well as the prices that you charge for them. Catalogs are the primary mode of communication from you to your customers.

Catalog file formats
Catalog file formats are files that hold a list of catalog items and their details in a particular format, such as cXML or CIF.

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.

CIF Catalog Interchange Format is a catalog format that uses a comma-delimited text file.

Contract
A contract is an agreement between a seller and one or more buyers. Through this contract, buyers can purchase goods and services from a 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.

cXML Commerce eXtensible Markup Language (cXML) is an open standard promoted by Ariba. It is designed to communicate the details of e-commerce transactions, including catalogs, supplier information, and purchase orders. A unique feature of cXML is its support for PunchOut catalogs, which consist of live catalog content hosted on a supplier's interactive Web sites. Because cXML is an XML-based language, you can use a variety of applications to generate and parse it. For more information on cXML, refer to http://www.cxml.org .

Data management utilities
Data management utilities provide customized content management solutions for a particular production environment. WebSphere Commerce includes data-management utilities such as the Extractor, XML Transformer,

and Text Transformer. These utilities help you to execute the tasks of extracting, transforming between different XML formats, and loading catalog data.

Export Exporting involves extracting data from WebSphere Commerce and transforming it into cXML or CIF using XSL rules.

Note: In this guide, the following are used interchangeably:

- WebSphere Commerce and WebSphere Commerce 6.0, Enterprise
- Ariba and Ariba Buyer
- Ariba is used as an example for cXML based procurement integration

Overview

Procurement system integration capabilities in WebSphere Commerce enables sellers using WebSphere Commerce to integrate external buy-side systems. This functionality allows buyers using a procurement system to interact with the supplier's catalog on WebSphere Commerce for conducting B2B e-commerce transactions. This results in increased sales and enhances the organization's B2B presence on the Web.

Procurement system integration provides connective capabilities to systems such as Ariba Buyer using cXML. Ariba supports cXML, which enables two different ways of shopping: Through local catalogs and Internet catalogs. Procurement system integration enables the export of catalog data from WebSphere Commerce in cXML and CIF catalog file formats. You can load catalog files onto the Ariba Buyer system to use the catalogs.

Local catalog orders

In the local catalog mode, suppliers have their catalogs replicated on the procurement system. Buyers browse the catalog and build their shopping carts without connecting to WebSphere Commerce. When the requisitioning buyer submits an order, the `OrderRequest` message is sent to WebSphere Commerce (supplier) and batch processing of the order is performed. WebSphere Commerce sends an (`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 conflict 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 in 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 the `PunchOutSetupRequest`message. 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 a `PunchOutOrderMessage` XML message 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.

Procurement integration provides the following benefits:

- Allows suppliers to maintain a single catalog in WebSphere Commerce and use it to enable their Web presence by participating in the procurement system's network.
- Reduces costs of order processing through the WebSphere Commerce connectivity to Supply Chain Management, Retail Business System, and Order Management backend systems. These systems automate the flow of orders from the procurement system's buyer to your business systems.
- Uses the updated B2B features of WebSphere Commerce to utilize and maintain buyer organizations, buyer-specific catalogs, price lists, and contract pricing.

Major capabilities

XML over HTTPS is used to support procurement system integration. WebSphere Commerce provides the following functionality, which is used by this reference application:

- 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.
- Support for contracts between buyers and sellers, where buyers can get specific views and prices of products based on the contract.
- Send the shopping cart to the buyer systems for approval.
- Process purchase orders sent from an external system.
- Buyers can edit their shopping carts from the procurement system. For this, the procurement system reconnects to WebSphere Commerce where you can make the required changes.
- Buyers inspect the availability of a product before placing an order.
- Buyers can obtain more information about a product from their external procurement systems.
- Register requisitioning buyers from the external buyer organizations during runtime.
- Authenticate and validate requests from buyers from external systems.
- Export catalogs from WebSphere Commerce tables to cXML and CIF formats.

Business models enabled

Procurement system integration enables the following business models:

- For buyers using e-procurement buy-side applications, suppliers can maintain a single catalog while allowing buyer applications remote access to the supplier catalog.
- Suppliers can process XML messages from buyer applications as part of the procurement process.

- The shopping cart can be sent in the format specific to the procurement system.
- Sellers can receive Internet catalog orders.
- Sellers can receive orders created through buyer systems using local catalogs.
- Buyers in the procurement system can view and place orders locally from the WebSphere Commerce catalog, using the catalog export feature.

Prerequisites

This section covers the software and hardware requirements for this reference application.

Software requirements

The following are the software requirements for this reference application:

- IBM WebSphere Commerce Version 6.0, Enterprise on Windows®®.
- IBM DB2 Universal Database™ Version 8.2.3 Enterprise Server Edition.

Hardware requirements

The hardware configuration required for this reference application is the same as that described in the WebSphere Commerce product documentation. There are no additional hardware requirements.

References

For more information on procurement system integration and related technologies, you can refer to the following Web sites:

- For complete information about IBM WebSphere Commerce Enterprise software, refer to http://www.ibm.com/software/webservers/commerce/wcbe/
- For more information on procurement system integration refer to the WebSphere Commerce information center
- For information on cXML, refer to http://www.cxml.org

Note: The preceding Web addresses can change at any time without notice. IBM is not responsible for the authenticity or correctness of information from non-IBM Web sites.

Chapter 2. cXML scenarios and messages for procurement system integration

This chapter describes the scenarios to integrate WebSphere Commerce with Ariba using cXML. The cXML messages that procurement system integration uses are listed.

Scenarios for procurement system integration

Procurement system integration uses one of the following scenarios:

- PunchOut using Ariba SN
- Using local catalogs

When using either of the scenarios described in this reference application WebSphere Commerce enables the following:

- Creating and managing catalogs with multiple categories and sub categories using the Catalog subsystem.
- Managing organizations using the Member subsystem. Buyer organizations and organizational units can be registered with WebSphere Commerce and the organizational hierarchies can be maintained.
- Capturing relationships between the suppliers and buyers in terms of accounts.
- Creating contracts between each buyer and the supplier organization. This allows you to provide a buyer specific view of the catalog and buyer specific prices for the products.
- Registering the requisitioners automatically with the buyer organization to which they belong, depending on the credentials presented.

PunchOut using Ariba SN

Any supplier that hosts an online catalog can sell products to multiple corporate buyers. A buyer can use Ariba Buyer as its procurement software to access the supplier's online catalog and place an order. This requires the supplier to enable PunchOut for the catalog from the Ariba SN. cXML PunchOut is a process that allows buyers to rapidly access content on a suppliers' interactive eCommerce Web site.

If the supplier uses WebSphere Commerce to host the interactive electronic catalog on the Web, then this reference application can be used as a guide to enable the cXML PunchOut. If a supplier hosts a catalog with dynamic content and custom views of the catalog are required for each buyer organization, then the PunchOut process that WebSphere Commerce supports can be utilized. The buyer can connect to the Ariba SN, which in turn will connect to WebSphere Commerce where the buyer can view the catalog and place orders.

Deploying this reference application on WebSphere Commerce enables the following:

- Publishing the WebSphere Commerce catalog entry point into the Ariba SN. This allows buyers to access the WebSphere Commerce catalog from Ariba Buyer through the Ariba SN using the PunchOutSetupRequest message.

- Enabling the WebSphere Commerce catalog to receive the PunchOutSetupRequest message in cXML format. This initiates the PunchOut process.
- Sending the shopping cart to the buyer organization for approval using the OrderRequest message.
- Receiving an approved order from the buyer organization for fulfillment using the PunchOutOrderMessage message.

Profile transaction using Ariba SN

The Profile transaction is used to retrieve cXML server capabilities, including the supported cXML version, transactions, and options on those transactions. The Profile transaction enables a buyer using Ariba SN to query the cXML capabilities of a supplier using WebSphere Commerce to host its catalog. To inquire about server capabilities, send a ProfileRequest document. The server returns a ProfileResponse document containing the server information.

When WebSphere Commerce receives a ProfileRequest from Ariba SN, WebSphere Commerce sends the information in the ProfileResponse document to Ariba. Since this reference application supports PunchOut and OrderResponse messages, the ProfileResponse document lists these as the supported messages and the URLs to which these messages must be routed to from Ariba SN.

A ProfileRequest is generally pulled by the network no more than once every 24 hours, and less if no request is sent to that supplier within a particular window.

Using local catalogs

In the PunchOut scenario, the buyer requires online access to the supplier's catalog. In some cases buyers may choose to view the supplier's catalog offline. This requires the supplier to export their catalogs in the CIF or cXML catalog file format and give it to the buyer. The buyer can load this catalog on their Ariba Buyer procurement system, view the catalog, and create orders locally. Once an order is created and approved, the order can be sent to the supplier through the Ariba SN for fulfillment.

The benefit in using this approach is that the buyer is not required to connect to the supplier's catalog each time to browse a catalog or add an item to the shopping cart. Though the buyer must ensure that the latest copy of the supplier's catalog is loaded into the procurement application at all times. If the supplier is using WebSphere Commerce to host the catalog, then the suppliers can use this reference application to export catalogs in the CIF or cXML catalog file format.

Deploying this reference application on WebSphere Commerce enables the following:

- Transforming the exported catalog data into the CIF or cXML format, to ensure that the Ariba Buyer application understands it.
- Receiving a local catalog order from Ariba SN using the PunchOutOrderMessage message. WebSphere Commerce then sends the OrderResponse message to Ariba SN.

cXML messages for procurement system integration

Procurement system integration uses cXML messages to integrate with Ariba. This includes three inbound request messages to WebSphere Commerce from Ariba, and four 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):

Table 1. cXML messages for procurement system integration

CXML request message	Command (ME)	**CXML response message**
`PunchOutSetupRequest`	`PunchOutSetup`	`PunchOutSetupResponse`
	`SendShoppingCart`	`PunchOutOrderMessage`
`OrderRequest`	`BatchOrderRequest`	`OrderResponse`
`ProfileRequest`	`ProfileResponse view command`	`ProfileResponse`

Here, `PunchOutOrderMessage` is the shopping cart sent to Ariba for approval. For sample messages see, Appendix A: Sample XML messages. For more information, refer cXML User's Guide in the following link. http://www.cxml.org

Chapter 3. Configuring the reference application

The instructions in this reference application assume that you have published the WebSphere Commerce AdvancedB2BDirect store. Configuring this reference application involves the following:

1. Configuring a store
2. Configuring WebSphere Commerce for the supplier and the buyer
3. Enabling the WebSphere Commerce messaging subsystem for cXML

Configuring a store

To provide an Internet catalog, you can configure an existing store. For more information on Internet catalogs, see "Scenarios for procurement system integration" on page 5 Scenarios for procurement system integration

The procedures in this section assume that you have already published an AdvancedB2BDirect store using Store Services. In the B2B shopping flow after you complete shopping you need to checkout. In the procurement system integration shopping flow after you complete shopping, you need to send your shopping cart to a procurement application. This requires you to modify certain JSP files. To enable an existing AdvancedB2BDirect store in WebSphere Commerce with procurement system integration, do the following:

1. Make a copy of the AdvancedB2BDirect directory for your existing WebSphere Commerce catalog from *WAS_installdir*\profiles\instance_name\installedApps\cell_name\WC_instance_name.ear\Stores.war. where the variables are defined as follows:

   WAS_installdir Default values for this variables are listed in "Paths variables" on page vi

   cell_name This is the cell name of the profile on which WebSphere Application Server is installed.

   WC_instance_name This is the name of the WebSphere Commerce instance.
2. Navigate to *WAS_installdir*\profiles\instance_name\installedApps\cell_name\WC_instance_name.ear\Stores.war\AdvancedB2BDirect\ShoppingArea\CurrentOrderSection
3. Open CurrentOrderDisplay.jsp in an editor.
4. During the normal shopping flow, when you click Submit in the CurrentOrderDisplay.jsp, the Billing and Shipping Address Selection displays. In a procurement enabled shopping flow you do not need to complete the order, but you must send the order to the procurement system for approval. For this, you must make certain changes to the CurrentOrderDisplay.jsp in your store. The following changes are required to call the SubmitShoppingCartCmd command during checkout:
   a. Open CurrentOrderDisplay.jsp from

      WAS_installdir\profiles\instance_name\installedAppa\cell_name\WC_instance_name.ear\Stores.war\\AdvancedB2BDirect\ShoppingArea\CurrentOrderSection.
   b. Add the function submitShopCart() as follows:

      function submitShopCart(form,url)

```
{
form.URL.value = "SubmitShoppingCart?orderId=<c:out
value="${thisOrderId}"/>&storeId=<c:out
value="${storeId}"/>&userId=<c:out
value="${userId}"/>&orderStatus=<c:out value="P" />";
var shopCart = "SubmitShoppingCart?orderId=<c:out
value="${thisOrderId}" />&storeId=<c:out
value="${storeId}"/>&userId=<c:out
value="${userId}"/>&orderStatus=<c:out value="P" />";
document.ShopCartForm.action=shopCart;
form.submit();
}
```

c. Search for the following line :

```
<a href="javascript:submitForm(document.ShopCartForm)"
   class="button" id="WC_CurrentOrderDisplay_Link_6">
   <fmt:message key="YourOrder_Submit"
   bundle="${storeText}"/>
</a>
```

Change submitForm to submitShopCart in the above line and make it similar to the line below:

```
<a href="javascript:submitShopCart(document.ShopCartForm)"
   class="button" id="WC_CurrentOrderDisplay_Link_6">
   <fmt:message key="YourOrder_Submit"
   bundle="${storeText}"/>
</a>
```

d. Search for the following below :

```
<form name="ShopCartForm" action="OrderItemUpdate" method="post"
 id="ShopCartForm">
```

Remove the action from the above line and make it similar to the line below:

```
<form name="ShopCartForm"  method="post" id="ShopCartForm">
```

Note: You may want to retain the normal B2B shopping flow in your store while providing support for procurement system integration. In this case, you can modify the shopping flow to detect if the user is a procurement buyer and redirect to a procurement enabled shopping flow. Otherwise the user will be directed to the normal shopping flow.

5. Copy the following JSP files from cXMLintegration.zip to the following locations :
   - Copy ProfileResponse.jsp, PunchOutAribaError.jsp and PunchOutSetupResponse.jsp to

     WAS_installdir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\Stores.war
   - Copy PurchaseOrderResponseAriba.jsp, PunchOutCatalogDisplay.jsp, SubmitShoppingCart.jsp, ComposeShoppingCartAriba.jsp to

     WAS_installdir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\Stores.war\AdvancedB2Bdirect

Table 2. JSP file names and descriptions

JSP file	Description
PunchoutSetupResponse.jsp	This JSP composes the cXML response message to the PunchOutSetupRequest.

Table 2. JSP file names and descriptions (continued)

JSP file	Description
PunchoutCatalogDisplay.jsp	This JSP is called after executing the PunchOutCatalogCmd. It redirects the request to the TopCategoriesDisplay.jsp page in the Advanced B2B store model.
PunchoutOutAribaError.jsp	This is an error JSP used if an error occurs during PunchOutSetupRequest.
SubmitShoppingCart.jsp	This JSP is called after the SubmitShoppingCart command is executed. It displays a summary of the shopping cart. From this page you can send the details of the shopping cart back to the procurement system.
PurchaseOrderResponseAriba.jsp	This JSP composes the cXML OrderResponse message for the OrderRequest.
ComposeShoppingCartAriba.jsp	This JSP composes the cXML PunchOutOrderMessage response message. This message contains the data of the shopping cart that must be sent to the procurement system
ProfileResponse.jsp	This JSP composes the cXML response message to the ProfileRequest message. It is used to specify the supported transactions.

6. Restart the WebSphere Commerce Server instance.

Note: When buyers access your catalog through a PunchOut session, they see a slightly different navigation sequence than the standard WebSphere Commerce operations such as the checkout process etc. Procurement system integration replaces these standard processes with applications that incorporate a message extension order processing workflow in place of the standard WebSphere Commerce flow.

Configuring WebSphere Commerce for the supplier and buyer

This section explains how to configure the supplier and buyer settings and how to solicit buyer information. The cXMLintegration.zip file that you have downloaded contains the following files required to configure the supplier and buyer settings:

- PISupplierConfig.xml
- PIBuyerConfig.xml

Configuring supplier settings

Stores in WebSphere Commerce have suppliers who are identified as owners of the store. For procurement system integration using cXML, these suppliers require an identifier, which typically is the DUNS number. The following steps explain how to configure a member who owns a store in WebSphere Commerce with a DUNS number. The DUNS number can be replaced with any mutually agreed on value between the supplier and the buyer.

A supplier organization or a supplier is any business organization that wants to supply goods and services. To configure the supplier settings, do the following:

1. From a DB2® command prompt execute the following SQL:

```
db2 select MEMBER_ID from storeent where Identifier='AdvancedB2BDirect'
```

Note: If you are using an existing store, then substitute the store identifier with that of your store.

2. Edit the PISupplierConfig.xml file:
   a. Assign the value of the MEMBER_ID from step 1 to orgentity_id, as shown in the following code fragment:

```
<orgcode
orgcode_ID="1"
orgentity_id="StoreOwnerID"
codetype="DUNS"
code="DUNS number of the supplier organization"/>
```

   b. Assign the value of codetype and code according to the mutually agreed on code between the supplier and the buyers as shown in the preceding code fragment.

      Note: If the supplier organization does not have a DUNS number, then any mutually agreed on code between the supplier and the buyer can be used.

3. To load the supplier's data from a command window run the following:

```
massLoad -dbname databasename -dbuser dbusername -dbpwd dbpassword
–customizer filename infile PISupplierConfig.xml -method sqlimport
```

Loading the access control policy

This section describes how to load the access control policy for the ProfileResponse view command. The ProfileRequest message uses this view command. Do the following to load the policy:

1. Copy the `PRAccessControlPolicy.xml`file from the `cXMLintegration.zip` file into *`WC_installdir`*`\xml\policies\xml`
2. Open a command prompt and switch to *`WC_installdir`*`\bin.`
3. Run the following:

```
acpload databasename dbusername dbpassword PRAccessControlPolicy.xml
```

   This completes configuring the supplier settings for this reference application.

Getting buyer invitations and soliciting buyer information

To transact business with a buyer, a buyer must invite you to become a supplier. Once invited, you must obtain basic information about the buyer and enter that information into WebSphere Commerce. See "Appendix B. Sample buyer information form" on page 41 for the data that maybe required for procurement system integration.

You may implement and deliver this form in any manner you wish, as a printed form, as an e-mail attachment, or as a web-based form. Typically, you will mail 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.

Registering the buyer

Use the WebSphere Commerce Organization Administration Console to do the following:

1. Register the new buyer organization.
2. Register users with Administrator rights for this buyer organization. This is required for each buyer connecting directly from external cXML-enabled sites such as PunchOut sites to WebSphere Commerce.

3. If you have multiple buyers connecting from the Ariba SN, then register users with the following logonId and password:

   LogonId

   When the PunchOut or order request is coming from Ariba SN, the organization code is already defined in Ariba SN as sysadmin@ariba.com. As a result, register the user with the logonId as sysadmin@ariba.com under the root organization of the buyer. When the ProfileRequest is coming from Ariba SN, the organization code is already defined in Ariba SN as AN01001010101. As a result, register another user with the logonId as AN01001010101 under the root organization of the buyer.

   Password

   The password of both users must be the Shared Secret of the supplier organization. For secure access, Ariba SN and external cXML-enabled sites such as PunchOut sites authenticate the cXML documents they receive. This authentication ensures that the documents are legitimate because they are sent from recognized organizations. WebSphere Commerce supports shared secret as a mode of authentication. This reference application requires the shared secret to be the password of the user for the buyer organization. The user must have Administrator rights.
4. Assign the role of a procurement buyer administrator to the registered buyer organization.
5. Assign the role of a procurement buyer administrator to the registered users of the buyer organization.

For more information refer to the WebSphere Commerce information center.

Creating a business account and a contract between the supplier and the buyer

The instructions in this reference application require the credit line mode of payment. This assumes that payment is made offline. Defining the credit line mode of payment requires the creation of a contract between the buyer and the supplier.

1. Create a business account using the instructions provided in the WebSphere Commerce information center. When creating a business account note the following:
   - Do not select the **Allow customers to purchase under the terms and conditions of store's default contract** check box in the Customer page.
   - Do not select the **Purchase order number may be specified at the time of the order** check box 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 using the instructions provided in the WebSphere Commerce information center. When creating a contract note the following:
   - Do not specify any contract shipping addresses. If you specify a shipping address in the contract, then you must provide this same address in the `OrderRequest`message.

For more information refer to the WebSphere Commerce information center.

Configuring buyer settings

A buyer organization registered with WebSphere Commerce must use the cXML messaging protocol. This requires the following configuration:

1. Open the sample `PIBuyerConfig.xml`file.
2. Modify the buyer information in the `PIBuyerConfig.xml` file to configure a new `PIBuyer` as a member of WebSphere Commerce as shown in the following code fragment:

```
<orgcode
orgcode_id="2"
orgentity_id="orgentity Id of the buyer organization"
codetype="Organization code domain"
code="Organization code"/>
```

Where,

- `Organization code domain` is the organization code domain from the buyer information form. For example, the DUNS or AribaNetworkId.
- `Organization code` is the DUNS number or any other mutually agreed upon code of the buyer organization.

To find the orgentity_id from an DB2 command window run the following SQL:

```
db2 select orgentity_id from orgentity where orgentityname=Organization identifier
```

The `code` and `codetype` can represent the DUNS number or any other mutually agreed on code between the supplier the and buyer. For example, the codetype is the organization code domain from the buyer information form, which is AribaNetworkUserId in this case and the code is the DUNS number of the buyer organization.

```
<procsys
procsysname="Ariba"
/>

<procprotcl
procprotcl_id="1"
procsysname="Ariba"
protocolname="cXML"
version="1.2.014"
authtype="1"
twostepmode="Y"
classifdomain="UNSPSC"
uomstandard=""
/>
<procmsgvw
procprotcl_id="1"
orgentity_id="orgentity Id of the buyer organization"
msgname="PunchOut setup"
viwename="PunchOutAribaView"
/>

<procmsgvw
procprotcl_id="1"
orgentity_id="orgentity Id of the buyer organization"
msgname="SendShoppingCart"
viwename="SendShoppingCartView"
/>

<procmsgvw
procprotcl_id="1"
orgentity_id="orgentity Id of the buyer organization"
msgname="PurchaseorderResponse"
viwename="purchaseorderResponseAribaView"
/>
```

```
<procbuyprf
procprotcl_id="1"
orgentity_id="orgentity Id of the buyer organization"
reqidparm="User"
orgunitparm="deptName"
/>
```

Where, orgentity_id is the orgentity Id of the buyer organization.

```
<member member_id="101" type="G" state="1"/>
<mbrgrp mbrgrp_id="101" owner_id="orgentity Id of the buyer organization"
mbrgrpname="Member group name of the buyer organization"
/>
```

Where, owner_id is the orgentity Id of the buyer organization and mbrgrpname is the Member group name of the buyer organization. The values of member_id, mbrgrp_id, and procprotcl_id are provided as examples. You may need to change these values so that they are unique to your WebSphere Commerce database.

```
<buysupmap
procprotcl_id="1"
buyorgunit_id="orgentity Id of the buyer organization"
suporg_id="StoreOwnerID"
catalog_id="catalog Id of the published store"
mbrgrp_id="101"

/>
```

Where,

- `buyorgunit_id` is the orgentity Id of the buyer organization
- `suporg_id` is the StoreOwnerID
- `catalog_id` is the catalog Id of the published store

3. To load the buyer's data from a command window run the following:

```
massload -dbname databasename -dbuser dbusername -dbpwd dbpassword
–customizer filename -infile PIBuyerConfig.xml -method sqlimport
```

This completes configuring the buyer settings for this reference application.

Enabling the WebSphere Commerce messaging subsystem for cXML

This reference application uses the WebSphere Commerce messaging subsystem to communicate between WebSphere Commerce and buyer applications using cXML like Ariba. Do the following to enable the WebSphere Commerce messaging subsystem to support the cXML messaging protocol:

1. Download the cXML.dtd file for cXML 1.2.014 specification from http://www.cxml.org. Copy the cXML.dtd file to

   WAS_installdir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\xml\messaging
2. Add the configuration parameters to the XML message mapper:
   - Open the wc-server.xml file from

     WAS_installdir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\xml\config in an editor.

     Where `instance_name` is the name of your WebSphere Commerce instance.
   - Search for the `<messaging>` element in the configuration file. The `<messaging>` element contains an attribute called EcInboundMessageDtdFiles.
   - Register cxml.dtd with the messaging system. To do this add cxml.dtd to the list of supported DTD files to the EcInboundMessageDtdFiles attribute.

- Save the file.
- Open the ariba_sys_template.xml file from the cXMLintegration.zip file.
- Copy the content between the<TemplateDocument> and </TemplateDocument> and also the content between<TemplateTag> and </ TemplateTag> sections into the *WAS_install_dir*\profiles\instance_name\installedApps\ WC_instance_name_cell\WC_instance_name.ear\xml\messaging\ user_template.xml file.
- Save user_template.xml.

3. To receive XML messages over HTTP, enable the HTTP adapter in *WC_installdir*\instances\instance_name\xml\instance_name.xml. For this, set enabled=true for the HttpAdapter with deviceFormatType=XmlHttp.
   - Search for the string deviceformattypeId=10000.
   - By default, the adapter is disabled, which means that it is set to enabled=false. Set enabled=true. For more information refer to the WebSphere Commerce information center.
4. Open the struts-config.xml in an editor from

   WAS_install_dir\profiles\instance_name\installedApps\ WC_instance_name_cell\WC_instance_name.ear\Stores.war\WEB-INF and add the following entries

   a. Add the following actions between the <action-mappings> and </action-mappings> tag.

```
<action path="/PunchOutAribaView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/PunchOutSetupErrorView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/PunchOutCatalogView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/SubmitShoppingCartView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/SubmitShoppingCartErrorView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<path="/SendShoppingCartView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/SendShoppingCartErrorView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/PurchaseOrderResponseAribaView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/PurchaseOrderErrorView" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
<action path="/ProfileResponse" type="com.ibm.commerce.struts.BaseAction">
<set-property property="https" value="0:1"/>
</action>
```

   b. b) Add the following forward entries between the <global-forwards> and </global-forwards> tag.

```
<forward className="com.ibm.commerce.struts.ECActionForward">
name="PunchOutAribaView/0/-10000" path="/PunchOutSetupResponse.jsp">
<set-property propertyresourceClassName
value="com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
```

```
 name="PunchOutSetupErrorView" path="/PunchOutAribaError.jsp">
<set-property property="resourceClassName"
value="com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
name="PunchOutCatalogView" path="/PunchOutCatalogDisplay.jsp">
<set-property property="resourceClassName"
value="com.ibm.commerce.command.HttpForwardViewCommandImpl/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
name="SubmitShoppingCartView" path="/SubmitShoppingCart.jsp">
<set-property property="resourceClassName
value="com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
name="SubmitShoppingCartErrorView" path="/GenericError.jsp">
<set-property property="resourceClassName
value="com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
name="SendShoppingCartView" path="/ComposeShoppingCartAriba.jsp">
<set-property property="resourceClassName
value="com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
name="SendShoppingCartErrorView" path="/GenericError.jsp">
<set-property property="resourceClassName
value="com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
name="PurchaseOrderResponseAribaView/0/-10000" path="/PurchaseOrderResponseAriba.jsp">
<set-property property="resourceClassNamevalue="com.ibm.commerce.command.HttpForwardViewCom
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward"
name="PurchaseOrderErrorView/0/-10000" path="/GenericApplicationError.jsp">
<set-property property="resourceClassName
value="com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
</forward>
<forward className="com.ibm.commerce.struts.ECActionForward" name="ProfileResponse/0/-
10000" path="/ProfileResponse.jsp">
<set-property property="resourceClassName
value="com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
</forward>
```

5. Restart the WebSphere Commerce Server instance for the changes to take effect.

Chapter 4. Verification Procedure

This chapter provides the procedural details for ensuring the end-to-end flow for a test message. Before you verify the message flow ensure the following:

1. Ensure that the WebSphere Commerce server instance is started and the configuration mentioned in Chapter 3, “Configuring the reference application,” on page 9 is completed successfully
2. Verify the following elements in the cXML header:

Table 3.

cXML header element	Verification
<Header> <From> <Credential domain= "AribaNetworkId"> <Identity>sysadmin@ariba.com </Identity> </Credential> </From>	• The value of the credential domain must match the codetype mentioned in step 2 of “Configuring buyer settings” on page 14 • The value of the identity must match the code mentioned in step 2 of “Configuring buyer settings” on page 14
<Credential domain="DUNS"> <Identity>9143470144 </Identity> </Credential> </To>	• The value of the credential domain must match the codetype mentioned in step 3 of “Configuring supplier settings” on page 11 • The value of the identity must match the code mentioned in step 3 of “Configuring supplier settings” on page 11
<Credential domain= "AribaNetworkUserId"> <Identity>user@ibm.com </Identity> <SharedSecret> *Secret* </SharedSecret> </Credential> </Sender> </Header>	• The value of the credential domain must match the codetype mentioned in step 2 of “Configuring buyer settings” on page 14 • The value of the identity must be the logonId mentioned in step 2 of “Registering the buyer” on page 12 • The value of SharedSecret is the password of the user registered in step 2 of the “Registering the buyer” on page 12

Verification procedure for PunchOut

- Set the PunchOut and Order URL in the settings for cXML in the procurement system to https://*hostname*/webapp/wcs/stores/servlet/

  where *hostname* is the fully qualified hostname of the machine running WebSphere Commerce
- For all PunchOut requests configure the Ariba system to send an additional extrinsic called "User", which must be the requisitions' logonId in the procurement system.

When a PunchOut message is sent from the Ariba Buyer System through the Ariba SN, the WebSphere Commerce Catalog page must display and the user must be able to shop. Depending on the operation requested for in the input message, users

can create, edit, or inspect the shopping cart. This verifies the PunchOut message flow for this reference application. Similarly, you can test the PunchOut order message.

Verification procedure for BatchOrder

Set the BatchOrder request in the cXML settings in the procurement system to https://*hostname*/webapp/wcs/stores/servlet/

where *hostname* is the fully qualified hostname of the machine running WebSphere Commerce

For all the BatchOrder requests, configure the Ariba system to send an additional extrinsic called "User" which must be the requisitioner logonId in the procurement system.

- Ariba sends the BatchOrder request message to WebSphere Commerce with the buyer or supplier credentials. It also sends the details of the shopping cart as well as the BILL TO and payment information
- The message is mapped to the BatchOrderRequest command. This command detects if the shopping cart is already in the ORDERS using the unique PAYLOADID. It creates a fresh order in batch processing by calling a series of order related commands
- At the end of the processing, the OrderResponse message that gives the status code and status message is sent back to Ariba.

Verification procedure for ProfileRequest

Set the Profile request in the cXML settings in the procurement system to https://*hostname*/webapp/wcs/stores/servlet/

where *hostname* is the fully qualified hostname of the machine running WebSphere Commerce

- WebSphere Commerce receives a ProfileRequest from Ariba with the buyer and supplier credentials
- WebSphere Commerce sends the information like supported cXML version, transactions,status code etc. in the ProfileResponse document to Ariba

Chapter 5. Exporting catalogs from WebSphere Commerce

This chapter explains how to export catalog data from WebSphere Commerce in cXML and CIF catalog file formats. You can use the exported catalog data as local catalogs. For more information on local catalogs, see "Using local catalogs" on page 6. The WebSphere Commerce catalog data is mapped to the corresponding cXML and CIF elements. For more information, see "Appendix E. Mapping Information" on page 49. Exporting catalogs from WebSphere Commerce involves the following:

1. Extracting the files contained in the cXMLintegration.zip file.
2. Using the rules to transform data from massloadable XML to cXML.
3. Using the rules to transform data from massloadable XML to CIF.

Extracting and converting catalog data to cXML

To extract the catalog data from WebSphere Commerce and transform the extracted massloadable XML to cXML, do the following:

1. From cXMLintegration.zip extract the contents of \catalog\CatalogExport.zip into *WC_installdir*
2. Make the following changes in `WC_installdir` to match your installation and configuration settings:
   a. Open the `WC_installdir` \CatalogExport\cxml\WcsTocXML.bat to batch file in an editor.
   b. Change the following literals to match your installation, and save the changes.

      ```
      DB_NAME=db2
      WCS_DBNAME=mallplan
      UID=user
      PWD=password
      TRANSFORMED_INDEXOUT_FILE_NAME=IndexOut.xml
      TRANSFORMED_CONTRACTOUT_FILE_NAME=ContractOut.xml
      TRANSFORMED_SUPPLIEROUT_FILE_NAME=SupplierOut.xml
      ```

      Where:

      - `DB_NAME` is the database type for which catalog export is required, for example, DB2.
      - `WCS_DBNAME` is the name of the database in use, for example, Mall.
      - `UID` is the user ID for the database in use.
      - `PWD` is the password for the database in use.
      - `TRANSFORMED_INDEXOUT_FILE_NAME` is the output file where details about the index are stored after transformation.
      - `TRANSFORMED_CONTRACTOUT_FILE_NAME` is the output file where details of the contract are stored after transformation.
      - `TRANSFORMED_SUPPLIEROUT_FILE_NAME` is the output file where details of the supplier are stored after transformation.
3. Open the `WC_installdir` \CatalogExport\cxml\ExportFilter.xml in an editor. Change the values for the following to that of the catalog you are exporting:
   a. Buyer organization name, for example orgentityname='*Organization name*'.
   b. Unique identifier for the store, for example identifier=' *AdvancedB2BDirect*'.
   c. Supplier ID, for example *member_id*='100' and *orgentity_id*='100'

d. Code from the ORGCODE table, for example code='organizationcode '.
e. Buyorgunit_id from the BUYSUPMAP table, for example buyorgunit_id=7000000000000000002.
f. Save the ExportFilter.xml file.

4. Run the WcsTocXML.bat batch file from a DB2 command window. After the execution completes, the output comprises index, contract, and supplier information. This information is contained in the following transformed cXML files:
   - TRANSFORMED_INDEXOUT_FILE_NAME
   - TRANSFORMED_CONTRACTOUT_FILE_NAME
   - TRANSFORMED_SUPPLIEROUT_FILE_NAME
5. Verify the format of the output files with the sample cXML files provided in "Appendix C. Sample catalog files for cXML" on page 43

Extracting and converting catalog data to CIF

To extract the catalog data from WebSphere Commerce and transform the extracted massloadable XML to CIF, do the following:

1. If you did not complete step 1 in "Extracting and converting catalog data to cXML" on page 21then from cXMLintegration.zip extract the contents of \catalog\CatalogExport.zip into *WC_installdir*
2. Make the following changes in WcsTocif.bat to match your installation and configuration settings:
   a. Open the *WC_installdir*\CatalogExport\cif\WcsTocif.bat batch file in an editor.
   b. Change the following literals as per your installation, and save the changes.

```
DB_NAME=db2
WCS_DBNAME=mallplan
UID=user
PWD=password
TRANSFORMED_INDEXOUT_FILE_NAME=IndexOut.cif
TRANSFORMED_CONTRACTOUT_FILE_NAME=ContractOut.cif
```

   Where:
   - DB_NAME is the database type for which catalog export is required, for example, DB2.
   - WCS_DBNAME is the name of the database in use, for example, Mall.
   - UID is the user ID for the database in use.
   - PWD is the password for the database in use.
   - TRANSFORMED_INDEXOUT_FILE_NAME is the output file where details about the index are stored after transformation.
   - TRANSFORMED_CONTRACTOUT_FILE_NAME is the output file where details of the contract are stored after transformation.
3. Open the WC_installdir installdir\CatalogExport\cif\ExportFilter.xml in an editor. Change the following values to that of the catalog you are exporting:
   a. Buyer organization name, for example orgentityname=*Organization Name*'.
   b. Unique identifier for the store, example identifier=' *AdvancedB2BDirect*'.
   c. Supplier ID, for example *member_id*='100' and *orgentity_id*='100'
   d. Code from the ORGCODE table, for example code='organizationcode '.
   e. Buyorgunit_id from the BUYSUPMAP table, for example buyorgunit_id=7000000000000000002.

f. Save the ExportFilter.xml file.

4. Run the `WcsTocif.bat` batch file from a DB2 command window. After the execution completes, the output comprises index and contract information. This information is contained in the following transformed CIF files:
   - TRANSFORMED_INDEXOUT_FILE_NAME
   - TRANSFORMED_CONTRACTOUT_FILE_NAME.
5. Verify the format of the output files with the sample CIF files provided in "Appendix D. Sample catalog files for CIF" on page 47.

Chapter 6. Customizing procurement system integration

Procurement system integration provides an extensible framework that can be customized to support B2B transactions in WebSphere Commerce so that you can extend the message, schema, or business logic. Procurement system integration is a component of WebSphere Commerce that enables integration with external buy-side systems.

Enabling procurement system integration for other procurement systems

Procurement system integration can support other protocols as long as it receives requests over HTTP. To support new protocols you must customize procurement system integration. This involves the following:

- Registering the new protocol with the procurement system integration system.
- Creating a configuration file, which involves mapping protocol-specific XML to procurement system integration specific variables.

  Note: You may require a configuration file only if the protocol you are using needs one. For example, if you are using an XML message, then you will need a corresponding message mapping template. Depending on the protocol that you are using, the request is directly sent as URL parameters
- Customizing the procurement system integration specific JSP files.
- Customizing the different subsystems such as the member subsystem.

Registering a new protocol with procurement system integration

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

1. Populate the procurement system information into the PROCSYS table for the new procurement system. For more information refer to the WebSphere Commerce Developer information center.
2. Populate the protocol specific information like protocol name, version and so on, into the PROCPROTCL table. For more information refer to the WebSphere Commerce Developer information center.
3. Populate the protocol specific view task name. Each procurement system integration command (PunchOutSetup, BatchOrderRequest, and SendShoppingCart) that composes a response message looks up the PROCMSGVW table for the correct view task name. You can specify a unique view for each protocol of the buyer organization. For more information refer to the WebSphere Commerce Developer information center.
4. Populate the PROCBUYPRF table that contains the buyer organization's profile information such as the requisition ID, department name, and other information. For more information refer to procurement system integration in the WebSphere Commerce Developer information center.
5. Associate the buying organization unit with the supplier. Populate the BUYSUPMAP table that contains the protocol ID, catalog ID and the member group ID created for the buyer. For more information refer to procurement system integration in the WebSphere Commerce Developer information center.

See, "Configuring WebSphere Commerce for the supplier and buyer" on page 11 for sample XML files to populate these tables.

Creating a configuration file

For procurement system integration to understand the incoming XML message, the message must be mapped to a WebSphere Commerce or procurement system integration command and the elements of that message must be mapped to the parameters of the command. Do the following to create a configuration file:

1. Create a *my_user_template.xml* with the necessary mapping and elements required to map to the parameters of the commands in WebSphere Commerce. For more details, refer to *ariba_sys_template.xml* from cXMLIntegration.zip
2. Overwrite the contents of the *user_template.xml* file from *WAS_install_dir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\xml\messaging\user_template.xml* with the contents of *my_user_template.xml* file.
3. Save the file.
4. Copy the <TemplateDocument> and <TemplateTag> section into the *WAS_install_dir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\xml\messaging\user_template.xml* file.
5. Save *user_template.xml*.
6. To receive XML messages over HTTPS, enable the HTTP adapter in the instance_name.xml and wc-server.xml located at *WC_installdir\instances\instance_name\xml* and *WAS_install_dir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\xml\config* respectively. For this, set enabled=true for the HttpAdapter with deviceFormatType=XmlHttp.
   a. Search for the string deviceformattypeId=10000.
   b. By default, the adapter is disabled, which means that it is set to enabled=false. Set enabled=true. For more information refer to the WebSphere Commerce information center.

You can configure the XML or HTTP adapter support using a separate XML template file for each procurement system. To do this, add the following section to the message-mapper group section for each message mapper in the *wc-server.xml* and *instance_name.xml* located at *WAS_install_dir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\xml/config/wc-server.xml* and *WC_installdir\instances\WC_instance_name\xml\.instance_name.xml* respectively. where *WC_instance_name* is the name of your WebSphere Commerce instance.

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

In the preceding code, add the following lines under the configuration element and before the end tag:

```
EcSystemTemplateFile="my_user_template.xml"
 EcTemplatePath="WAS_install_dir\profiles\instance_name\installedApps
 \WC_instance_name_cell\WC_instance_name.ear\xml\messaging"
EcInboundMessageDtdFiles="protocol.dtd"
 EcInboundMessageDtdPath="WAS_install_dir\profiles\instance_name\installedApps
 \WC_instance_name_cell\WC_instance_name.ear\xml\messaging"
/>
```

Customizing JSP files

JSP files are used to compose the messages that need to be sent to different procurement systems in different message formats. The messages sent to the procurement system depend on the view command that is used. For example the PunchOutSetup command determines the view to be used by looking at the PROCMSGVW table. If no entries are found, then the default PunchOutSetupOKView view is used.

For protocol specific messages that assist in the PunchOut process modify the existing JSP files or replace them with new JSP templates and then update the the struts-config.xml accordingly.The struts-config.xml is located at *WAS_install_dir\profiles\instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\Stores.war\WEB-INF* For a list of the existing JSP files see step 5 in the "Configuring a store" on page 9 section.

Customizing commands for the PunchOut process

You can extend any of the procurement system integration controller and task commands to provide custom behavior by overriding certain methods. For more information, refer to the WebSphere Commerce Developer information center. Each command is based on an interface and one way to customize a command is to provide a custom implementation of the required interface. A default implementation class is provided for some interfaces. The following is a list of the interfaces and their default implementation classes used in the PunchOut flow:

Table 4. Controller commands

Interface	Default Implementation
PunchOutSetupCmd	PunchOutSetupCmdImpl
PunchOutCatalogDisplayCmd	PunchOutCatalogDisplayCmdImpl

Table 5. Task commands

Interface	Default implementation
AuthenticationHelperCmd	AuthenticationHelperCmdImpl
DBAuthenticationCmd	DBAuthenticationCmdImpl
LdapAuthenticationCmd	
RegisterRequisitionerCmd	RegisterRequisitionerCmdImpl
ThirdPartyB2BauthCmd	

Customizing CIData

The PunchOutSetupCmd command uses the CIData object to store the XML parameters received 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 command 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 CIData class then processes the TypedProperty object in the processHeader() method. The processHeader() method in turn populates the following objects:

SupplierCred
Stores the supplier credentials for authentication purposes.

BuyerCred
Stores the buyer organization credentials for authentication purposes.

MpCred
If the PunchOutSetupRequest request comes from an online marketplace, then this object stores the marketplace credentials for authentication purposes.

SessionInfo
Stores information relevant to register the requisitioning buyer.

Any of the preceding objects 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));
 .
 .
 .
}
```

Customizing authentication

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

1. Customizing the authentication type
   The default authentication mechanism provided authenticates against the credential information stored in the WebSphere Commerce database. The DBAuthenticationCmd task command performs authentication.
   You have the option to customize authentication so that it will be performed against a Lightweight Directory Access Protocol (LDAP) directory by implementing the LdapAuthenticationCmd interface. Alternatively, you can choose to use a third-party authentication mechanism by implementing the ThirdPartyB2BAuthCmd interface. For more information refer to the WebSphere Commerce Developer information center.
2. Customizing the authentication level
   Procurement system integration provides four possible levels of authentication.

For a description of these authentication levels refer to the description of the DBAuthenticationCmd command.
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, modify the authentication command, either DBAuthenticationCmd, LdapAuthenticationCmd, or ThirdPartyAuthenticationCmd to handle the new authentication level. For more information refer to the WebSphere Commerce Developer information center.

Customizing store catalog display in PunchOutSetupCmd

After a buyer organization is authenticated and the requisitioning buyer is registered, the performExecute() method of the PunchOutSetupCmd command does the following:

1. All the parameters to be passed on to PunchOutCatalogDisplayCmd are stored in the BuyerRequestInfo object.
2. The BuyerRequestInfo object is stored in the SupplierCookieTable.
3. The PunchOutSetupCmd command returns the URL of the PunchOutCatalogDisplayCmd command, which the procurement system invokes along with the supplier cookie value as its parameter.
   Example:
   The following is the code that performs this function in PunchOutSetupCmd command:

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

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

4. The PunchOutCatalogDisplayCmd command sets the view name to PunchOutCatalogView, which in turn will call PunchOutCatalogDisplay.jsp.
5. The PunchOutCatalogDisplay.jsp will then call the command invoked by the URL that will bring up the store's home page.

The store catalog display can customized in the following ways:

1. To add new parameters that must be passed to the store catalog display page, you must customize the PunchOutCatalogDisplayCmd command. In the new implementation of the command add the parameters, that is the name-value pairs to the response properties in the performExecute() method of the command. These parameters can then be captured in PunchOutCatalogDisplay.jsp file
2. For a custom view of your catalog's home page, modify the PunchOutCatalogDisplay.jsp to specify a different view.
3. Based on the logon mode the PunchOutCatalogDisplay page is redirected to a different URL. You can modify PunchOutCatalogDisplay.jsp to specify a new URL for each logon mode.

Customizing commands for the PurchaseOrder process

You can extend any of the controller and task commands to provide custom behavior by overriding certain methods. For more information, refer to the WebSphere Commerce Developer information center. Each command is based on an interface and one way to customize a command is to provide a custom implementation of the required interface. A default implementation class is provided for each interface. The following is a list of the order subsystem interfaces and their default implementation classes for procurement system integration:

You can extend any of the procurement system integration controller and task commands to provide custom behavior by overriding certain methods. For more information, refer to the WebSphere Commerce Developer information center. Each command is based on an interface and one way to customize a command is to provide a custom implementation of the required interface. A default implementation class is provided for some interfaces. The following is a list of the interfaces and their default implementation classes used in the PunchOut flow:

Table 6. Controller commands

Interface	Default Implementation
BatchOrderRequestCmd	BatchOrderRequestCmdImpl
ProcurementOrderPrepareCmd	ProcurementorderPrepareCmdImpl

Table 7. Task commands

Interface	Default implementation
AuthenticationHelperCmd	AuthenticationHelperCmdImpl
DBAuthenticationCmd	DBAuthenticationCmdImpl
CreateShippingBillingAddressCmd	CreateShippingBillingAddressCmdImpl
LdapAuthenticationCmd	
RegisterRequisitionerCmd	RegisterRequisitionerCmdImpl
ThirdPartyB2BauthCmd	

Customizing CIData

The BatchOrderRequestCmd command uses the CIData object to store 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, which captures custom information from the XML parser.

The BatchOrderRequestCmd command 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 processHeader() method. The processHeader() method populates the following objects:

The CIData class then processes the TypedProperty object in the processHeader() method. The processHeader() method in turn populates the following objects:

SupplierCred
Stores the supplier credentials for authentication purposes.

BuyerCred
Stores the buyer organization credentials for authentication purposes.

MpCred
If the PunchOutSetupRequest request comes from an online marketplace, then this object stores the marketplace credentials for authentication purposes..

SessionInfo
Stores information relevant to register the requisitioning buyer.

You can modify any of the preceding objects to store information received through custom elements in the XML message.

Customizing authentication

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

1. Customizing the authentication type
   The default authentication mechanism provided authenticates against the credential information stored in the WebSphere Commerce database. The DBAuthenticationCmd task command performs authentication.
   You have the option to customize authentication so that it will be performed against an LDAP directory by implementing the LdapAuthenticationCmd interface. Alternatively, you can choose to use a third-party authentication mechanism by implementing the ThirdPartyB2BAuthCmd interface. For more information refer to the WebSphere Commerce Developer information center.
2. Customizing the authentication level
   Procurement system integration provides four possible levels of authentication. For a description of these authentication levels refer to the description of the DBAuthenticationCmd command.
   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, modify the authentication command, either DBAuthenticationCmd, LdapAuthenticationCmd, or ThirdPartyAuthenticationCmd to handle the new authentication level. For more information refer to the WebSphere Commerce Developer information center.

Customizing the catalog subsystem

The following table shows the list of the commands in the catalog subsystem related to procurement system integration, their interface names, and the default implementation class names:

Table 8.

Interface name	Default implementation class
SendShoppingCartCmd	SendShoppingCartCmdImpl
SubmitShoppingCartCmd	SubmitShoppingCartCmdImpl

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

Customizing the store catalog display

The catalog is displayed using a set of JSP files some of which are common to all the stores and the others are specific to a particular store. The common JSP files can be found in *WAS_installdir\profiles\instance_name\installedApps\cell_name\ WC_instance_name.ear\Stores.war* .

Customizing the shopping cart

Customizing the shopping cart involves the following:

1. Customizing the shopping cart implementation. The CIQuote interface provides a generic interface that any shopping cart can implement. CIQuoteImpl provides the default implementation for the shopping cart.
   The CIQuote interface consists of methods to populate the data into the shopping cart and get the line items from the shopping cart. To customize the quote override the respective methods in CIQuoteImpl or give a new implementation to the CIQuote interface.
2. 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 the SendShoppingCartView 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 struts-config.xml .The struts-config.xml is located at *WAS_install_dir\profiles\ instance_name\installedApps\WC_instance_name_cell\WC_instance_name.ear\ Stores.war\WEB-INF*.

Chapter 7. Migrating from WebSphere Commerce 5.6.1 to WebSphere Commerce 6.0

Migrate existing WebSphere Commerce 5.6.1 instance (which has cXML Reference Application configured) to WebSphere Commerce 6.0 level. Refer to the Migration Guide, Version 6.0 for more information. This guide is available at http://www.ibm.com/software/webservers/commerce/library/ . After you have finished the steps documented in WebSphere Commerce 6.0 migration guide, complete the following cXML specific post migration steps:

1. Add the cXML.dtd to the instance.xml and the wc-server.xml files.

   To add the cXML.dtd , do the following :
   Add cXML.dtd to the list of supported DTD files to the EcInboundMessagingDtdFiles attribute found in instance.xml and the wc-server.xml. Search for the Messaging node and add cXML.dtd to the EcInboundMessageDtdFiles as shown in the sample below:

   ```
   <Messaging EcInboundMessageDtdFiles="NCCommon.mod, NCCustomer_10.mod,.........,
   cXML.dtd" />
   ```

2. In the struts-config-migrate.xml, make sure that the HTTPS property for SubmitShoppingCartView and SendShoppingCartView is set to 0:1. The struts-config-migrate.xml can be found at *WAS_installdir*\profiles\ instance_name\installedApps\WC_instance_name_cell\ WC_instance_name.ear\Stores.war\WEB-INF. Search for the SendShoppingCartView and make the value of HTTPS as 0:1 as shown in the sample below:

   ```
   <action path="/SendShoppingCartView" type="com.ibm.commerce.struts.BaseAction">
   <set-property property="https" value="0:1"/>
   </action>
   <action path="/SubmitShoppingCartView" type="com.ibm.commerce.struts.BaseAction">
   <set-property property="https" value="0:1"/>
   </action>
   ```

3. In the struts-config-migrate.xml, remove the following lines from all the Ariba forward views if at all they exist.

   ```
   <set-property property="properties" value="storeDir= AdvancedB2BDirect"/>
   <set-property property="interfaceName" value=
   "com.ibm.commerce.command.HttpForwardViewCommand"/>
   <set-property property="implClassName" value=
   "com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
   <set-property property="direct" value="true"/>
   ```

   For instance, if entry for PunchOutAribaView exists as the following:

   ```
   <forward className="com.ibm.commerce.struts.ECActionForward"
             name="PunchOutAribaView/0/-10000" path="/PunchOutSetupResponse.jsp">
   <set-property property="resourceClassName" value=
           "com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
   <set-property property="properties" value="storeDir=AdvancedB2BDirect"/>
   <set-property property="interfaceName" value=
          "com.ibm.commerce.command.HttpForwardViewCommand"/>
   <set-property property="implClassName" value=
          "com.ibm.commerce.command.HttpForwardViewCommandImpl"/>
   <set-property property="direct" value="true"/>
   </forward>
   ```

   Remove the above lines in bold with the property as properties interfaceName, implClassName and direct. Make sure that the above instructions are followed for all the Ariba related views as ProfileResponse, PunchOutAribaView, PunchOutCatalogEditView, PunchOutCatalogView, PunchOutSetupErrorView,

PurchaseOrderResponseAribaView, SubmitShoppingCartErrorView, SubmitShoppingCartView, SendShoppingCartErrorView and SendShoppingCartView.

If the above lines do not exist in the struts-config-migrate.xml for an entry then please ignore the above instructions.

4. Make sure that the parameter sent to OrderListDataBean for orderStatus should be changed to 'W' if the orderStatus is 'P' in the SubmitShoppingCart.jsp. Search for

```
<wcbase:useBean and in the line<c:set target="${orderListBean}"property=
"orderStatus" value="W"/>
```

in the <wc:useBean> tag and if the value is 'P', change it to 'W'. See the sample below :

```
<wcbase:useBean id="orderListBean" classname="com.ibm.commerce.order.beans.OrderListDataBean">
<c:set target="${orderListBean}" property="storeId" value="${CommandContext.storeId}"/>
<c:set target="${orderListBean}" property="orderStatus" value="W"/>
<c:set target="${orderListBean}" property="userId" value="${userId}"/>
</wcbase:useBean>
```

The SubmitShoppingCart.jsp is located at *WAS_installdir*\profiles\ instance_name\installedApps\WC_instance_name_cell\ WC_instance_name.ear\Stores.war\AdvancedB2BDirect

Appendix A. Sample XML messages

This section contains the following sample XML messages:

- PunchOutSetupRequest message
- BatchOrderRequest message
- PunchOutSetupResponse message
- BatchOrderResponse message
- ProfileRequest message

PunchOutSetupRequest message

This message initiates the interaction from Ariba to WebSphere Commerce. The WebSphere Commerce messaging subsystem maps this message to the `PunchOutSetup` command.

Sample message:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "cXML.dtd">
<cXML version="1.2.014" xml:lang="en-US" payloadID="933694607118.1869318421@jlee"
  timestamp="2002-08-15T08:36:47-07:00">
 <Header>
  <From>
   <Credential domain="DUNS1">
    <Identity>65652314</Identity>
   </Credential>
  </From>
  <To>
   <Credential domain="DUNS">
    <Identity>83528721</Identity>
   </Credential>
  </To>
  <Sender>
   <Credential domain="DUNS1">
    <Identity>sysadmin@ariba.com</Identity>
    <SharedSecret>Shared Secret</SharedSecret>
   </Credential>
   <UserAgent>Ariba Buyer 7.0.4</UserAgent>
  </Sender>
 </Header>
 <Request>
  <PunchOutSetupRequest operation="create">
   <BuyerCookie>1CX3L4843PPZO</BuyerCookie>
   <Extrinsic name="User">dcode</Extrinsic>
   <BrowserFormPost>
    <URL>https://aribaorms:2600/punchout?client=NAwl4JsLIiuo</URL>
   </BrowserFormPost>
   <SupplierSetup>
    <URL>http://www.myweb.com/punchout.asp</URL>
   </SupplierSetup>
  </PunchOutSetupRequest>
 </Request>
</cXML>
```

BatchOrderRequest message

This message is sent from Ariba to WebSphere Commerce to create an approved order. The WebSphere Commerce messaging subsystem maps this message to the `BatchOrderRequest` command.

Sample message:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "cXML.dtd">
<cXML xml:lang="en-US" payloadID="933694607118.18693184210@jlee"
  timestamp="2002-08-15T08:36:47-07:00">
 <Header>
  <From>
   <Credential domain="DUNS1">
    <Identity>65652314</Identity>
   </Credential>
  </From>
  <To>
   <Credential domain="DUNS">
    <Identity>83528721</Identity>
   </Credential>
  </To>
  <Sender>
   <Credential domain="DUNS1">
    <Identity>sysadmin@ariba.com</Identity>
    <SharedSecret>abracadabra</SharedSecret>
   </Credential>
   <UserAgent>Ariba Buyer 7.0.4</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 addressID="1000487">
     <Name xml: lang="en">Los&#032;Gatos</Name>
     <PostalAddress name="default">
      <DeliverTo>Vincent&#032;Lo</DeliverTo>
      <DeliverTo>Lo&#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@<YourSmtpDomainName></Email>
     <Phone name="work">
      <TelephoneNumber>
       <Country isoCountryCode="US">1</Country>
       <AreaOrCityCode>408</AreaOrCityCode>
       <Number>3582100</Number>
      </TelephoneNumber>
     </Phone>
     <Fax name="work">
      <TelephoneNumber>
       <Country isoCountryCode="US">1</Country>
       <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>94080</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>
       <Extrinsic name="User">dcode</Extrinsic>
 </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="UNSPSC">43171800</Classification>
   <ManufacturerPartID>6565E2U</ManufacturerPartID>
   <ManufacturerName>IBM<ManufacturerName>
  <URL>http://budhoo.hawthorne.ibm.com/webapp/wcs/stores/servlet</URL>
   <Extrinsic name="PR&#032;No.">PR6150</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>
```

PunchOutSetupResponse message

This message is sent from WebSphere Commerce to Ariba in response to the PunchOutSetupResponse message. The WebSphere Commerce messaging subsystem maps this message to the `PunchOutSetup` command.

Sample message:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.2.014/cXML.dtd">
  <cXML xml:lang="en-US" payloadID="933694607739.1869318421@jlee"
   timestamp="2002-08-15T08:46:00-07:00"
  <Response>
  <Status code="200"> text="OK"></Status>
  <PunchOutSetupResponse>
   <StartPage>
    <URL>
        http://xyz.hawthorne.ibm.com/webapp/wcs/stores/servlet/PunchOutCatalogDisplay?
             supplierCookie=9bAxTQtK3TzcxamHpLhKYen5lG5MGILJOyFfNHuvDWo%3D
        </URL>
  </StartPage>
 </PunchOutSetupResponse>
</Response>
</cXML>
```

BatchOrderResponse message

This message is sent from WebSphere Commerce to Ariba in response to the PunchOutOrderRequest message.

Sample message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM
  "http://xml.cXML.org/schemas/cXML/1.2.014/cXML.dtd">
<cXML timestamp="2001-06-13T11:07:27-5:00"
   payloadID=992444847906.1@sreed.in.ibm.com
   version="1.2.014">
<Response>
 <Status code="200" text="OK"></Status>
</Response>
</cXML>
```

ProfileRequest message

This message is sent from Ariba to WebSphere Commerce to retrieve the server capabilities, including the supported cXML version, transactions and options on these transactions.

Sample message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM cXML.dtd">
  <cXML payloadID="9949494@ariba.acme.com" xml:lang="en-US"
         timestamp="2000-02-04T18:39:09-08:00"
<Header>
    <From>
       <Credential domain="AribaNetworkUserId">
        <Identity>admin@acme.com</Identity>
       </Credential>
    </From>
    <To>
       <Credential domain="AribaNetworkUserId">
        <Identity>admin@ariba.com</Identity>
       </Credential>
    </To>
    <Sender>
       <Credential domain="AribaNetworkUserId">
          <Identity>admin@acme.com</Identity>
          <SharedSecret>coyote</SharedSecret>
       </Credential>
       <UserAgent>ORMS V6.1</UserAgent>
    </Sender>
```

```
</Header>
<Request deploymentMode="test">
<ProfileRequest/>
</Request>
</cXML>
```

Appendix B. Sample buyer information form

The following is a sample buyer information form:

Table 9. Sample buyer information form

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 Extrinsic Name: User/Requisitioner Extrinsic Name:
Phone Number: Fax Number: Email Address: Address: Street: City: State: Zip/Postal Code: Country: Contact Information: Title: First Name: Middle Name: Last Name: Primary Phone Number: Alternative Phone Number: Fax Number: Email Address: Alternative Email Address:

Appendix C. Sample catalog files for cXML

This appendix lists the sample catalog files for cXML format

Sample cXML index file

The following is the sample index.xml file:

```
<?xml version='1.0' encoding="UTF-8" ?>
<!DOCTYPE Index SYSTEM "http://xml.cXML.org/schemas/cXML/1.2.014/cXML.dtd">
<Index>
 <SupplierID domain="InternalSupplierID">29</SupplierID>
 <Comments xml:lang="en-US">Sample cXML/Index</Comments>
 <IndexItem>
  <IndexitemAdd>
   <ItemID>
    <SupplierPartID>pn12345</SupplierPartID>
   </ItemID>
   <ItemDetail>
    <UnitPrice>
     <Money currency="USD">60</Money>
    </UnitPriceUnitOfMeasure>EA
    <Description xml:lang="en">Men's black shoes</Description>
    <UnitOfMeasureManufacturerPartID>MBS3.12EA</UnitOfMeasure>
    <Classification domain="SPSC">5136030000</Classification>
    <ManufacturerPartID>MBS3.12</ManufacturerPartID>
    <ManufacturerName>Florsheim</ManufacturerName>
    <URL>http://www.florsheim.com</URL>
    <Extrinsic name="ManufacturerURL>http://www.shoemaker.com</Extrinsic>
   </ItemDetail>
   <IndexItemDetail>
    <LeadTime>10</LeadTime>
    <ExpirationDate>2000-06-01</ExpirationDate>
    <EffectiveDate>1999-01-01</EffectiveDate>
    <TerritoryAvailable>USA</TerritoryAvailable>
   </IndexItemDetail>  </IndexItemAdd>
 </IndexItem>
<IndexItem>
 <IndexItemDelete>
  <ItemID>
   <SupplierPartID>pn12356</SupplierPartID>
  </ItemID>
 </IndexItemDelete>
 <IndexItemDelete>
  <ItemID>
   <SupplierPartID>pn12357</SupplierPartID>
  </ItemID>
 </IndexItemDelete>
</IndexItem>
<IndexItem>
 <IndexItemPunchout>
  <ItemID>
   <SupplierPartID>pn12399</SupplierPartID>
  </ItemID>
  <PunchoutDetail>
   <Description xml:lang="en-US">Ruby slippers</Description>
   <URL>http://mywebsite.com/Dorothy/shoes/red.htm</URL>
   <Classification domain="SPSC">5136030000</Classification>
   <ManufacturerName>Wizard Shoes</ManufacturerName>
   <ManufacturerPartID>WSRS1</ManufacturerPartID>
   <ExpirationDate>2010-01-01</ExpirationDate>
   <EffectiveDate>1999-11-24</EffectiveDate>
   <TerritoryAvailable>US</TerritoryAvailable>
```

```
   <TerritoryAvailable>UK</TerritoryAvailable>
  </PunchoutDetail>
 </IndexItemPunchout>
</IndexItem>
</Index>
```

Sample cXML supplier file

The following is the sample supplier.xml file:

```
<?xml version='1.0' encoding="UTF-8" ?>
<!DOCTYPE Supplier SYSTEM "http://xml.cXML.org/schemas/cXML/1.2.014/cXML.dtd">
 <Supplier corporateURL="http://www.myweb.com">
   storeFrontURL="http://buy.myweb.com">
<Name xml:lang="en-US">myweb</Name>
<Comments xml:lang="en-US">This is a cool company</Comments>
 <SupplierID domain="DUNS">942888711</SupplierID>
  <SupplierLocation>
   <Address>
     <Name xml: lang="en">main Office</Name>
     <PostalAddress>
      <DeliverTo>Bob A.Worker</DeliverTo>
      <Street>123 Front Street</Street>
      <City>TooSunny</City>
      <State>CA</State>
      <PostalCode>95000</PostalCode>
      <Country isoCountryCode="US">United States</Country>
     </PostalAddress>
     <Email>bobw@myweb.com</Email>
     <Phone name="Office">
      <TelephoneNumber>
       <CountryCode isoCountryCode="US">011</CountryCode>
       <AreaOrCityCode>800</AreaOrCityCode>
       <Number>555-1212</Number>
      </TelephoneNumber>
     </Phone>
     <Fax name="Order">
      <TelephoneNumber>
       <CountryCode isoCountryCode="US">011</CountryCode>
       <AreaOrCityCode>408</AreaOrCityCode>
       <Number>555-1234</Number>
      </TelephoneNumber>
     </Fax>
     <URL>http://www.myweb.com/Support.htm</URL>
    </Address>
    <OrderMethods>
     <OrderMethod>
      <OrderTarget>
       <URL>http://www.myweb.com/cxmloders</URL>
      </OrderTarget>
     </OrderMethod>
     <OrderMethod>
      <OrderTarget>
       <Email>plain.orders@myweb.com</Email>
      </OrderTarget>
     </OrderMethod>
     <Contact>
      <Name xml: lang="en">Mr. Smart E. Pants</Name>
      <PostalAddress>
      <Street>123 Front Street</Street>
      <City>TooSunny</City>
      <State>CA</State>
      <PostalCode>95000</PostalCode>
      <Country isoCountryCode="US">United States</Country>
     </PostalAddress>
     <Email>sepants@myweb.com</Email>
     <Phone name="Office">
```

```
      <TelephoneNumber>
       <CountryCode isoCountryCode="US">011</CountryCode>
       <AreaOrCityCode>800</AreaOrCityCode>
       <Number>555-1212</Number>
      </TelephoneNumber>
     </Phone>
    </Contact>
   </OrderMethods>
  </SupplierLocation>
 </Supplier>
```

Sample cXML contract file

The following is the sample MasterAgreementRequest.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.2.014/cXML.dtd">
<cXML payloadID="3223232@ariba.acme.com"
  timestamp="1999-02-12T18:39:09-08:00" xml lang="en-US">
 <Header>
  <From>
   <Credential domain="AribaNetworkUserId">
    <Identity>admin@acme.com</Identity>
   </Credential>
   <Credential domain="AribaNetworkUserId" type="marketplace">
    <Identity>bigadmin@marketplace.org</Identity>
   </Credential>
  </From>
  <To>
   <Credential domain="DUNS">
    <Identity>942888711</Identity>
   </Credential>
  </To>
  <Sender>
   <Credential domain="AribaNetworkUserId">
    <Identity>admin@acme.com</Identity>
    <SharedSecret>abracadabra</SharedSecret>
   </Credential>
   <UserAgent>Ariba.com Network V1.0</UserAgent>
  </Sender>
 </Header>
 <Request deploymentMode="test">
  <MasterAgreementRequest>
   <MasterAgreementRequestHeader agreementID="MA123"
     agreementDate="2001-12-01"
     type="value"
     effectiveDate="2002-01-01"
              expirationDate="2002-12-31"
              operation="new">
   <MaxAmount>
    <Money currency="USD">10000</Money>
   </MaxAmount>
   <MaxReleaseAmount>
    <Money currency="USD">10000</Money>
   </MaxReleaseAmount>
   <Contact role="BuyerLocation">
    <Name xml:lang="en">Buyer Location Address></Name>
   <PostalAddressName="Default">
     <DeliverTo>Joe Smith</DeliverTo>
     <DeliverTo>Mailstop M-543</DeliverTo>
     <Street>123 Anystreet</Street>
     <City>Sunnyvale</City>
     <State>CA</State>
     <PostalCode>90489</PostalCode>
     <Country isoCountryCode="US">United States</Country>
   </PostalAddress>
   </Contact>
```

```
    <Comments xml:lang="en-US">Anything well formed in XML can go here.</Comments>
    </MasterAgreementRequestHeader>
    <AgreementItemOut maxQuantity="100">
    <MaxAmount>
     <Money currency="USD">10000</Money>
    </MaxAmount>
    <MaxReleaseAmount>
     <Money currency="USD">10000</Money>
    </MaxReleaseAmount>
    <ItemOut quantity="1">
     <ItemID>
      <SupplierPartID>1233244</SupplierPartID>
      <SupplierPartAuxiliaryID>15051</SupplierPartAuxiliaryID>
     </ItemID>
     <ItemDetail>
      <UnitPrice>
       <Money currency="USD">1.34</Money>
      </UnitPrice>
      <Description xml: lang="en">hello</Description>
      <UnitOfMeasure>EA</UnitOfMeasure>
      <Classification domain="SPSC">12345</Classification>
      <ManufacturerPartID>234</ManufacturerPartID>
      <ManufacturerName>foobar<ManufacturerName>
      <URL>www.foo.com</URL>
     </ItemDetail>
     <Shipping trackingDomain="FedEx" trackingId="1234567890">
      <Money currency="USD">2.5</Money>
      <Description xml:lang="en-US">FedEx 2-day</Description>
     </Shipping>
     <Comments xml:lang="en-US">Anything well formed in XML can go here.</Comments>
    </ItemOut>
   </AgreementItemOut>
  </MasterAgreementRequest>
 </Request>
 </cXML>
```

Appendix D. Sample catalog files for CIF

This appendix lists the sample catalog files for CIF format

Sample CIF index file

The following is the sample index.cif file:

```
CIF_I_V3.0
DATA
942888711,100,,"Blue Ballpoint Pen",1213376,1.95,EA,,,,,
942888711,101,,"No. 2 Pencil",1213377,1.50,DZN,,,,,
942888711,102,,"Rubber Eraser",1213472,0.25,PK,,,,,
942888711,103,,"Stapler, Standard",1237461,2.95,BX,,,,,
ENDOFDATA
```

Sample CIF contract file

The following is the sample contract.cif file:

```
CIF_I_V3.0
SEMANTIC: CONTRACT
FIELDNAMES: Supplier ID, Supplier Part ID, Unit Price, Price Segment
DATA
AN0012345,ACF-100,2.85,{Plant12=2.75;Plant14=2.70;}
AN0012345,ACF-200,3.45,{Plant12=3.35;Plant14=3.40;}
ENDOFDATA
```

Appendix E. Mapping Information

The following are the attributes or elements in the cXML Supplier and Index files, and their mapping to corresponding placeholders in the WebSphere Commerce Schema. In some cases these values are assumed to be constants, for example, DUNS.

Table 10. Mapping information

Element or attribute name as given in the cXML	**Value for the element or attribute**
Supplier⇒**Name**	DISPLAYNAME in STOREENTDS
Supplier⇒**Comments**	DESCRIPTION in STOREENTDS
Supplier⇒**corporateURL**	<not mandatory>
Supplier⇒**storeFrontURL**	<not mandatory>
Supplier⇒**SupplierID**	MEMBER_ID in MEMBER table
Supplier⇒SupplierLocation⇒Contact ⇒**Name**	FIRSTNAME in ADDRESS
Supplier⇒SupplierLocation⇒ Contact⇒PostalAddress⇒**Street**	ADDRESS1+ADDRESS2+ADDRESS3 in ADDRESS
Supplier⇒SupplierLocation⇒ Contact⇒**PostalAddress City**	CITY in ADDRESS
Supplier⇒SupplierLocation⇒ Contact⇒**PostalAddress State**	STATE in ADDRESS
Supplier⇒SupplierLocation⇒ Contact⇒**PostalAddress PostalCode**	ZIPCODE in ADDRESS
Supplier⇒SupplierLocation⇒ Contact⇒**PostalAddress Country**	COUNTRY in ADDRESS
Supplier⇒SupplierLocation⇒ Contact⇒**Email**	EMAIL1+EMAIL2 in ADDRESS
Supplier⇒SupplierLocation⇒ Contact⇒Phone⇒**name**	PHONE1TYPE or PHONE2TYPE in ADDRESS (whichever is published)
Supplier⇒SupplierLocation⇒ Contact⇒Phone⇒Telephone Number⇒**CountryCode isoCountryCode**	US is constant (or pick from a config file)
Supplier⇒SupplierLocation⇒Contact ⇒Phone⇒TelephoneNumber ⇒**CountryCode**	PHONE1 or PHONE2 in ADDRESS (whichever is published)—need to use substring to pick the first three digits in the value of this column which will be the country code
Supplier⇒SupplierLocation⇒ Contact⇒Phone⇒TelephoneNumber ⇒**AreaOrCityCode**	PHONE1 or PHONE2 in ADDRESS (whichever is published) - need to use substring to pick the second three digits in the value of this column which will be the city code
Supplier⇒SupplierLocation⇒ Contact⇒Phone⇒TelephoneNumber ⇒**Number**	PHONE1 or PHONE2 in ADDRESS (whichever is published) - need to use substring to pick the digits after the first six digits in the value of this column which will be the number

Table 10. Mapping information (continued)

Element or attribute name as given in the cXML	**Value for the element or attribute**
Supplier⇒SupplierLocation⇒Address ⇒**Name**	IDENTIFIER in STOREENT
Supplier⇒SupplierLocation⇒Address ⇒PostalAddress⇒**DeliverTo**	FIRSTNAME in STADDRESS
Supplier⇒SupplierLocation⇒Address ⇒PostalAddress⇒**Street**	ADDRESS1+ADDRESS2+ADDRESS3 in STADDRESS
Supplier⇒SupplierLocation⇒Address ⇒PostalAddress⇒**City**	CITY in STADDRESS
Supplier⇒SupplierLocation⇒Address ⇒PostalAddress⇒**State**	STATE in STADDRESS
Supplier⇒SupplierLocation⇒Address ⇒PostalAddress⇒**PostalCode**	ZIPCODE in STADDRESS
Supplier⇒SupplierLocation⇒Address ⇒PostalAddress⇒**Country**	COUNTRY in STADDRESS
Supplier⇒SupplierLocation⇒Address ⇒**Email**	EMAIL1+EMAIL2 in STADDRESS
Supplier⇒SupplierLocation⇒Address ⇒Phone⇒**name**	<not mandatory>
Supplier⇒SupplierLocation⇒Address ⇒Phone⇒TelephoneNumber ⇒CountryCode⇒**isoCountryCode**	US is constant (or pick from a config file)
Supplier⇒SupplierLocation⇒Address ⇒Phone⇒TelephoneNumber ⇒CountryCode⇒**CountryCode**	PHONE1 or PHONE2 in STADDRESS (whichever is published)—need to use substring to pick the first three digits in the value of this column which will be the country code
Supplier⇒SupplierLocation⇒Address ⇒Phone⇒TelephoneNumber ⇒CountryCode⇒**AreaOrCityCode**	PHONE1 or PHONE2 in STADDRESS (whichever is published)—need to use substring to pick the second three digits in the value of this column which will be the city code
Supplier⇒SupplierLocation⇒Address ⇒Phone⇒TelephoneNumber ⇒CountryCode⇒**Number**	PHONE1 or PHONE2 in STADDRESS (whichever is published)—need to use substring to pick the digits after the first six digits in the value of this column which will be the number
Supplier⇒SupplierLocation⇒Address ⇒Fax⇒**name**	<not mandatory>
Supplier⇒SupplierLocation⇒Address ⇒**URL**	<not mandatory>
Supplier⇒SupplierLocation⇒ OrderMethods⇒OrderMethod⇒ OrderTagret⇒OtherOrderTagret⇒**name**	CBB is constant (we submit orders only through catalog based buying)
Index⇒SupplierID⇒**domain**	InternalSupplierID is a constant
Index⇒**SupplierID**	MEMBER_ID in CATENTRY
Index⇒Comments⇒**xml:lang**	en_US is a constant
Index⇒SearchGroup⇒**Name**	<not mandatory>
Index⇒SearchGroup⇒SearchAttribute ⇒**name**	<not mandatory>

Table 10. Mapping information (continued)

Element or attribute name as given in the cXML	**Value for the element or attribute**
Index⇒SearchGroup⇒SearchAttribute ⇒**type**	<not mandatory>
Index⇒IndexItem⇒ IndexItemAddItemID⇒**SupplierPartID**	CATENTRY_ID in CATENTRY
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒UnitPrice⇒ Money⇒**Currency**	CURRENCY in OFFERPRICE
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒UnitPrice⇒**Money**	PRICE in OFFERPRICE
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒**Description**	LONGDESCRIPTION in CATENTDESC
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒**UnitOfMeasure**	QUANTITYMEASURE as in CATENTSHIP
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒**Classification⇒domain**	DOMAIN in CATCLSFCODE
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒**Classification**	CODE in CATCLSFCOD
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒**ManufacturerPartID**	MFPARTNUMBER in CATENTRY
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒**ManufacturerName**	MFNAME in CATENTRY
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒**URL**	URL in CATENTRY
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒Extrinsic⇒**name**	<not mandatory>
Index⇒IndexItem⇒IndexItemAdd ⇒ItemDetail⇒**Extrinsic**	<not mandatory>
Index⇒IndexItem⇒IndexItemAdd ⇒IndexItemDetail⇒**LeadTime**	This cannot be mapped as per 5.1.. So the element will be mentioned with empty string as value
Index⇒IndexItem⇒IndexItemAdd ⇒IndexItemDetail⇒**ExpirationDate**	ENDDATE in OFFER
Index⇒IndexItem⇒IndexItemAdd ⇒IndexItemDetail⇒**EffectiveDate**	STARTDATE in OFFER
Index⇒IndexItem⇒IndexItemAdd ⇒IndexItemDetail⇒**TerritoryAvailabe**	<not mandatory>
Index⇒IndexItem⇒IndexItemPunchout ⇒PunchoutDetail⇒**Description**	LONGDESCRIPTION in CATENTDESC
Index⇒IndexItem⇒IndexItemPunchout ⇒PunchoutDetail⇒**URL**	URL in CATENTRY
Index⇒IndexItem⇒IndexItemPunchout ⇒PunchoutDetail⇒Classification ⇒**domain**	DOMAIN in CATCLSFCODE
Index⇒IndexItem⇒IndexItemPunchout ⇒PunchoutDetail⇒ItemDetail ⇒**Classification**	CODE in CATCLSFCODE
Index⇒IndexItem⇒IndexItemPunchout ⇒PunchoutDetail⇒**ManufacturerName**	MFNAME in CATENTRY

Table 10. Mapping information (continued)

Element or attribute name as given in the cXML	**Value for the element or attribute**
Index⇒IndexItem⇒IndexItemPunchout ⇒PunchoutDetail⇒**ManufacturerPartID**	MFPARTNUMBER in CATENTRY
Index⇒IndexItem⇒IndexItemPunchout ⇒PunchoutDetail⇒**TerritoryAvailable**	<not mandatory>

Notices

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 106-0032, 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 that has been exchanged, should contact:

IBM Canada Ltd.
Office of the Lab Director
8200 Warden Avenue,
Markham, Ontario L6G 1C7
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

IBM, and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or foreign countries.

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

AS/400®
AIX®
DB2
DB2 Universal Database
*e*server
IBM
iSeries™
MQSeries®
S/390®
VisualAge®
WebSphere
zSeries®

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

Solaris is a trademark of Sun Microsystems, Inc.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Oracle and Oracle Names are registered trademarks of Oracle Corporation. Oracle Objects, Oracle Workflow, Oracle8, Oracle8i, and Net8, are trademarks of Oracle Corporation.

D&B and D-U-N-S are trademarks or registered trademarks of D&B.

Credit card images trademarks and trade names provided in this product should be used only by merchants authorized by the credit card mark's owner to accept payment via that credit card.

Printed in USA