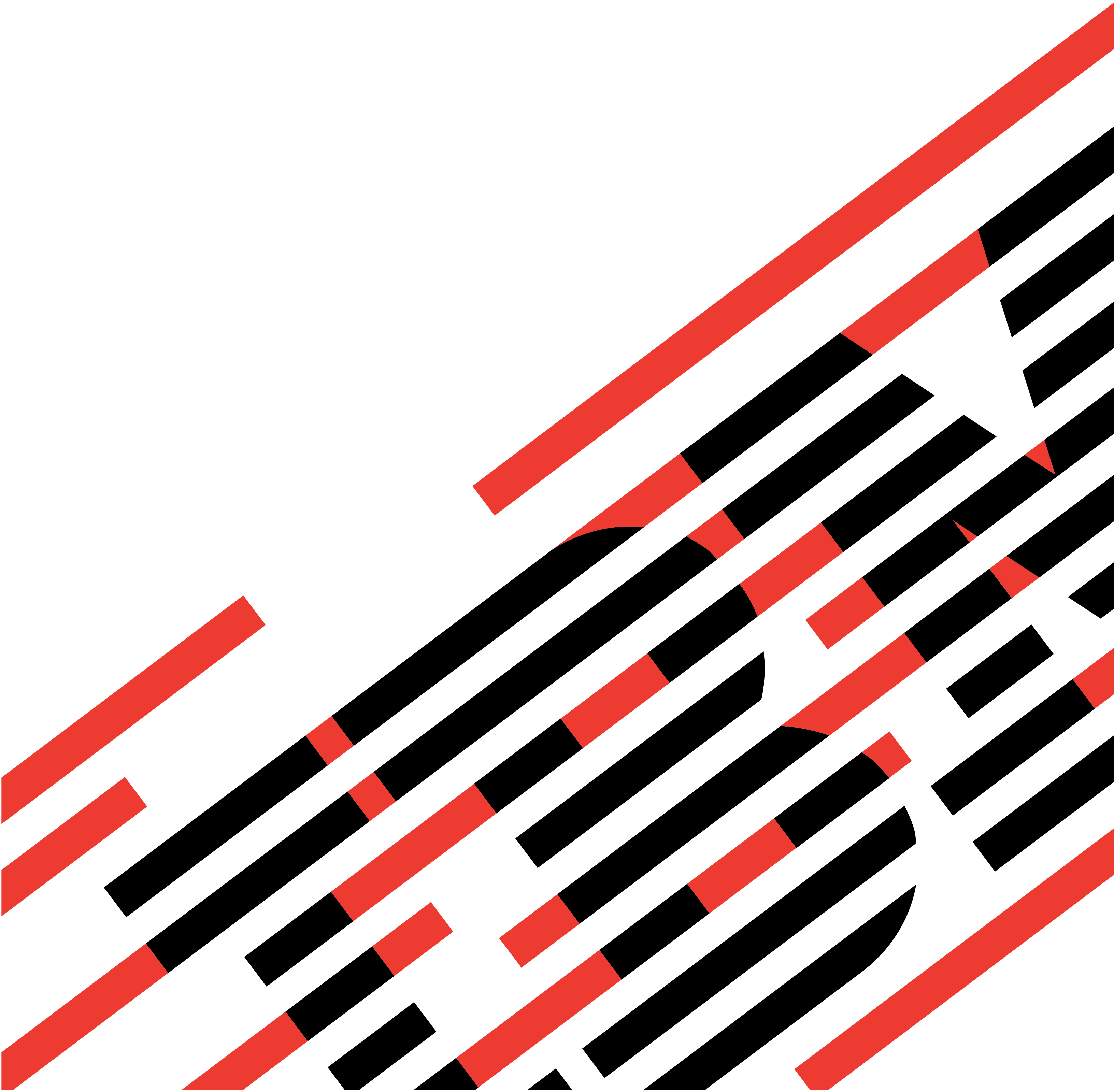
iSeries

IBM Connect for iSeries 2.0



iSeries

IBM Connect for iSeries 2.0

Contents

Chapter 1. IBM Connect for iSeries 2.0

IBM$^{(R)}$ Connect for iSeries$^{(TM)}$ is a software integration framework for business-to-business (B2B). A trained service provider can extend the framework to securely integrate your existing core business applications with the business applications of your trading partners. The framework is built on industry standards such as Java$^{(TM)}$, Extensible Markup Language (XML), and IBM MQSeries$^{(R)}$.

See these topics for more information about Connect for iSeries:

Print this topic
Provides a PDF file of this information that you can print.

“What’s new in iSeries Connect 2.0” on page 2
Highlights the product features that are new in version 2.0.

Chapter 2, “B2B and iSeries Connect concepts” on page 3
Introduces key concepts such as B2B models, iSeries Connect architecture, and related terminology.

See these topics for task-based information about setting up and running iSeries Connect:

Chapter 3, “Plan for iSeries Connect” on page 25
Provides details about preparing for B2B and iSeries Connect.

Chapter 4, “Install iSeries Connect” on page 29
Guides you through installing iSeries Connect.

Chapter 5, “Configure iSeries Connect” on page 39
Guides you through the process of configuring iSeries Connect.

Chapter 6, “Migrate iSeries Connect” on page 91
Describes how to migrate iSeries Connect and prerequisite middleware.

Chapter 7, “Customize iSeries Connect” on page 95
Describes iSeries Connect features that you can customize for your particular business situation.

Chapter 8, “Manage iSeries Connect” on page 103
Provides information about managing and administering iSeries Connect, including managing instances, troubleshooting, monitoring requests, tuning performance, backing up your data, and recovering from errors.

Use this topic to quickly find specific information about using Connect for iSeries:

Chapter 9, “Reference” on page 117
Contains API documentation, code examples, and other reference information that pertains to iSeries Connect.

This product includes software developed by the Apache Software Foundation (http://www.apache.org) .

What’s new in iSeries Connect 2.0

These functions are new in Connect for iSeries 2.0:

- Enhanced B2B supplier enablement function through:
  - Support for Application-to-Application (A2A) transactions
  - Support for the ability to handle any XML protocol
- Improved custom protocol development tools and e-catalog support
- New set of wizards for developing user-defined protocols
- Support for IBM’s latest middleware
- Support for WebSphere Application Server 4.0 and WebSphere Commerce Suite 5.4

Chapter 2. B2B and iSeries Connect concepts

Business-to-business (B2B) is the use of Web-based technologies to conduct business between two or more companies. Conducting business can mean buying or selling, or it can mean exchanging information. B2B transactions can take place directly between companies or through a third party who helps match buyers and sellers.

See these topics for conceptual information about B2B and iSeries Connect:

iSeries Connect B2B primer
Introduces basic B2B concepts and scenarios.

“Partner and provider model”
Provides information about the partner and provider model.

“iSeries Connect solution” on page 6
Describes the architecture of iSeries Connect and how it works.

Partner and provider model

The partner and provider model consists of two main scenarios for conducting business:

- The commerce scenario is a buy and sell specific type of scenario. In the commerce scenario, providers supply a catalog of products which the partner can browse and from which the partner can purchase items. Early B2B marketplace vendors (such as Ariba) developed this type of scenario.
- The non-commerce scenario is a more general scenario. In the non-commerce scenario, partners and providers exchange business data between business applications. For example, an independent insurance agent may request a life insurance quote from an insurance broker.

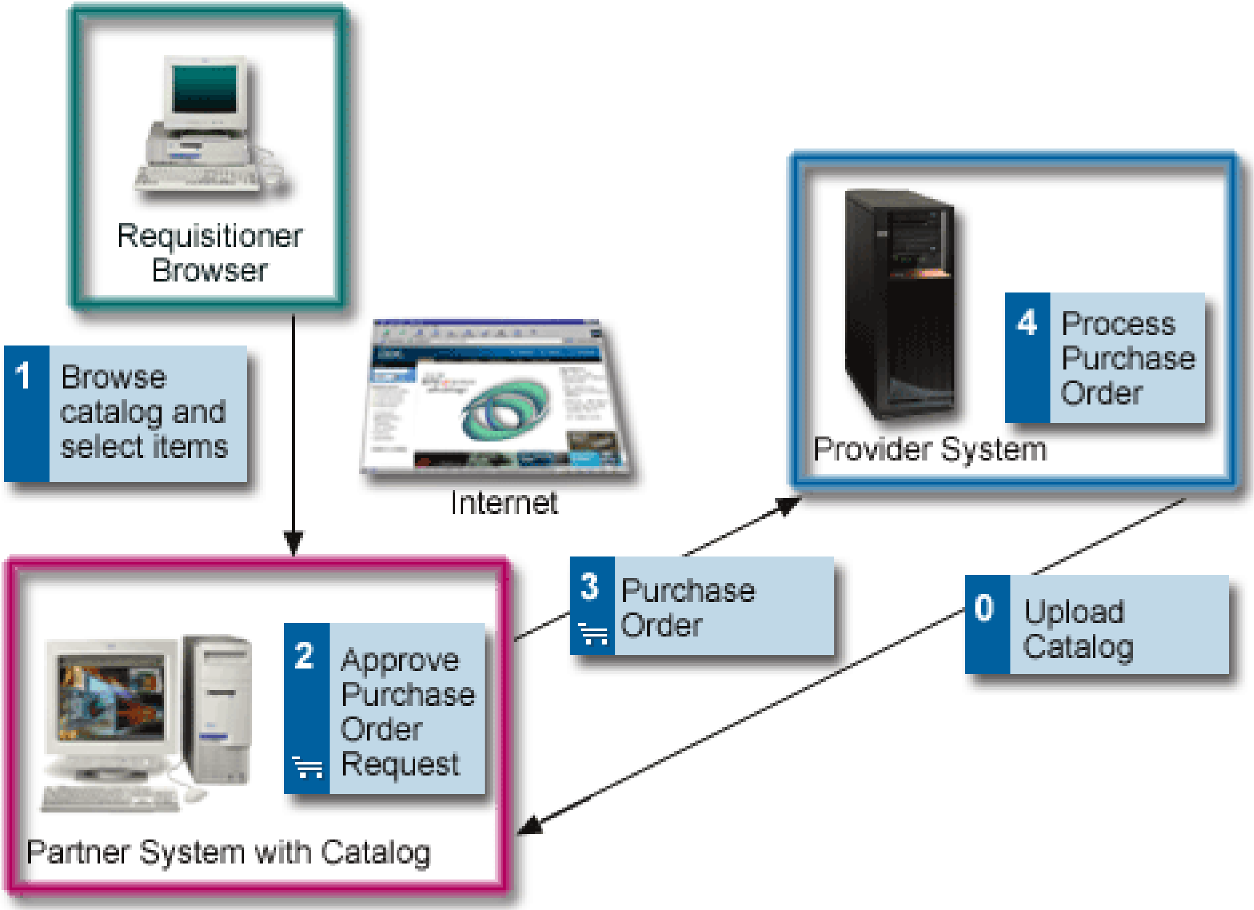
The commerce scenario supports different types of catalogs, which are named according to the catalog’s location, relative to the partner:

- **Local catalog**: Providers upload catalogs to the partner organization or marketplace server.
- **Remote catalog**: Providers host a catalog and the shopping experience on their Web site.

Local catalog: Hosted by the partner

In the partner and provider model, shown in Figure 1, a partner organization has purchased procurement software from a third party vendor, such as Ariba.

Figure 1: Partner and provider model for local catalog



This procurement software allows an individual in the partner organization (a requisitioner) to use a browser to make purchases. The requisitioner can choose from a list of approved catalogs that are hosted locally at the partner organization and shop for needed items. Each provider is responsible for uploading the catalog information to the partner site. The requisitioner selects the items and quantities needed. When finished, the order is submitted and captured by the procurement software.

The procurement software then notifies a designated approver that a new order request has been placed. This approver uses their browser to view the order and make any necessary changes in price or quantities. If the request looks satisfactory, the approver approves the order request.

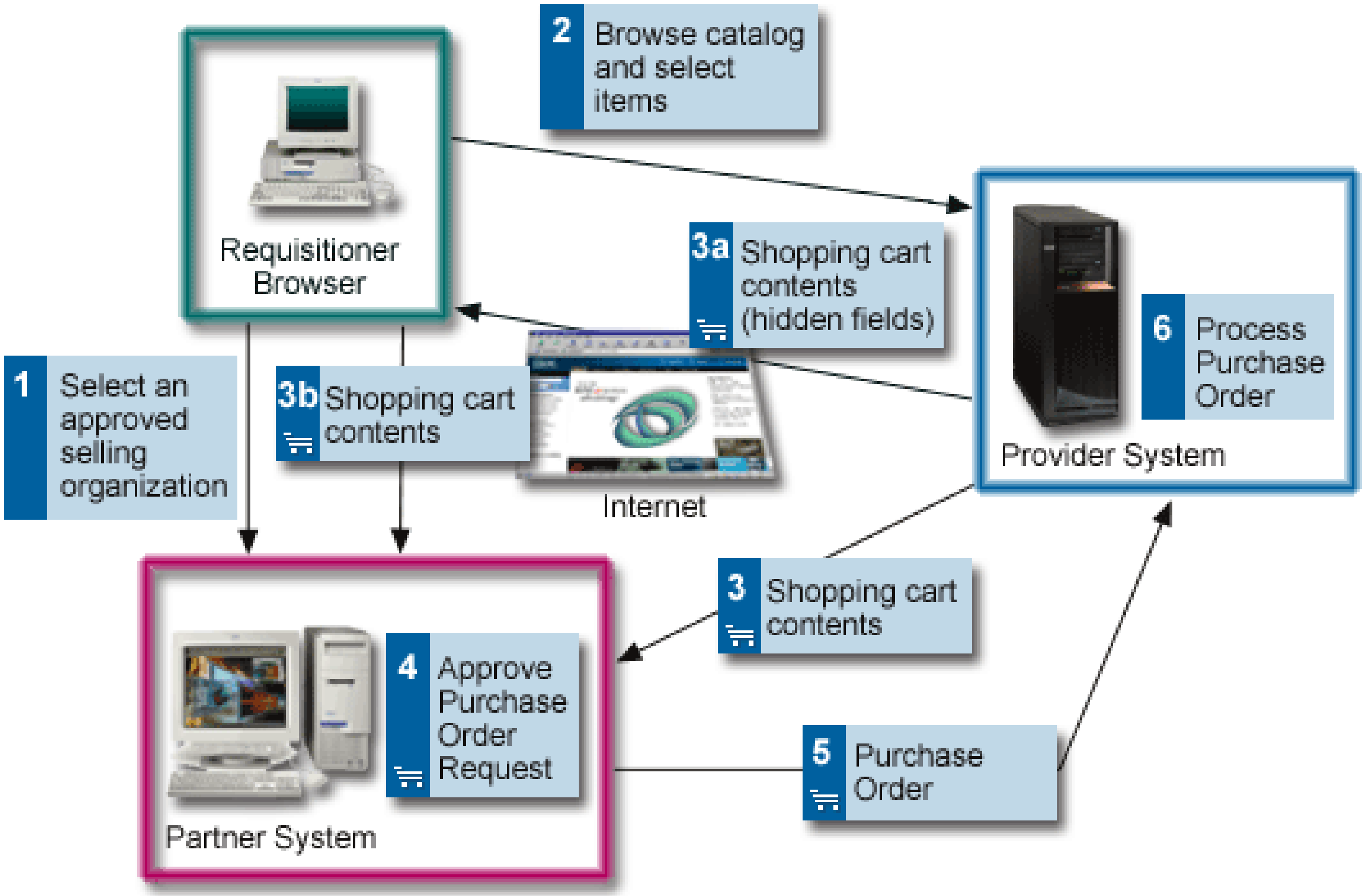
An approved order request results in a purchase order message being sent by the procurement software to the appropriate provider of the goods. The provider accepts the purchase order, processes it as necessary, and sends a purchase order response message to indicate that the order was accepted.

This last step in the process of accepting a purchase order request, processing it, and responding with a purchase order response is the job of iSeries Connect. iSeries Connect is responsible for accepting the various versions of this request message (for example, cXML for Ariba) and mapping the data from the request message to a format that is understandable by a flow manager application. It then maps the response from the flow manager application to a response format that is acceptable by the procurement software. iSeries Connect also helps build, upload, and manage the catalog information that the partner sees. This catalog information can be generated from the existing database tables of the provider.

Remote catalog: Hosted by the provider

Figure 2 shows a variation of the partner and provider model where the catalog and shopping experience is hosted at the provider site.

Figure 2: Partner and provider model for remote catalog



2 Browse catalog and select items
Requisitioner Browser
3a Shopping cart contents (hidden fields)
6 Process Purchase Order
Provider System
1 Select an approved selling organization
3b Shopping cart contents
Internet
3 Shopping cart contents
4 Approve Purchase Order Request
5 Purchase Order
Partner System

In this scenario, the requisitioner again uses his browser to choose an approved catalog to shop from. In this case, the procurement software indicates that the catalog is hosted remotely. The procurement software knows, or obtains directly from the provider site, the URL to use for shopping the catalog and returns this information to the browser. The requisitioner then shops the remote catalog and places items in his shopping cart. When he is done shopping, he confirms the contents of his shopping cart and checks out. A quote is sent to the procurement software.

When the requisitioner confirms his quote, the provider sends the shopping cart contents to the procurement system of the partner organization. The shopping cart contents contain all of the information about the items contained in the quote, and can be sent to the procurement system in one of two ways. It can be sent directly to the system as a separate message, or it can be sent as a redirect request to the browser of the requisitioner where it is redirected to the procurement system. The latter approach is usually selected, because it is the easiest way through various firewall systems.

The remaining steps are the same, as in the local catalog scenario, where a designated approver approves the quote and causes a purchase order to be sent to the provider system. A quote does not become an order until it is approved and sent to the provider. The provider processes the order by integrating it with the flow manager applications or by directing it to the commerce application (such as WebSphere Commerce Suite) for processing.

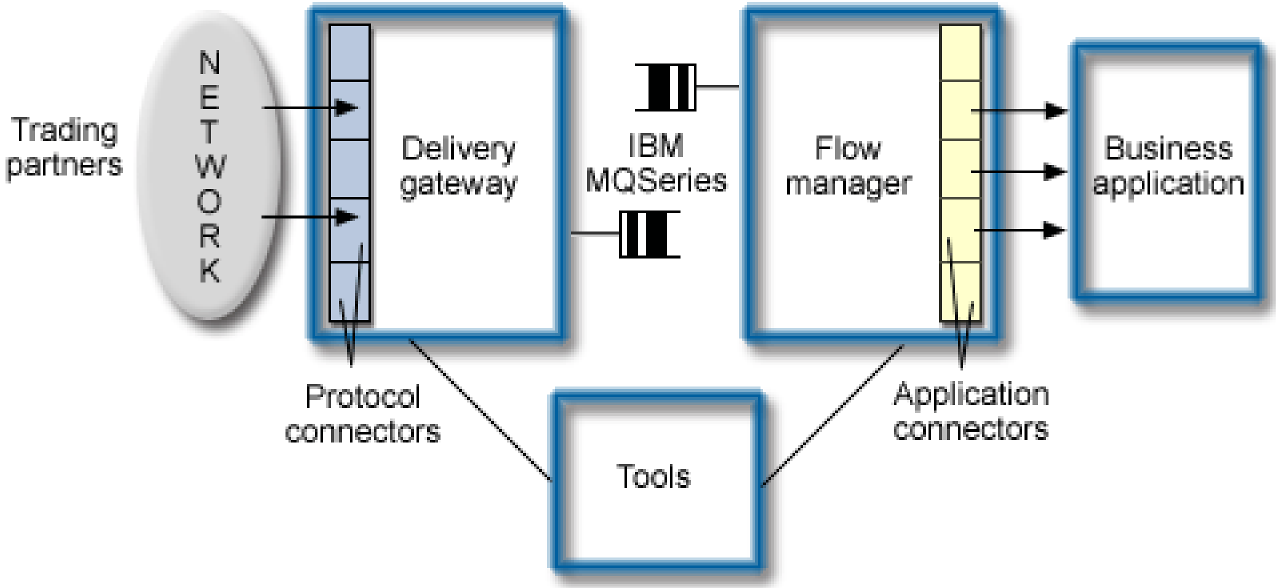
In the remote catalog scenario, iSeries Connect configures and extends the commerce application (such as WebSphere Commerce Suite) to handle the remote browsing requests and to generate the shopping cart contents. It also handles the resulting purchase order and integrates it with the flow manager application or routes it to the commerce application for completion.

iSeries Connect solution

iSeries Connect provides a highly configurable and pluggable architecture that is easy to use and extend. A series of graphical functions support this extendible framework. Use these graphical tools to install and configure iSeries Connect and to develop, deploy, and manage customized solutions.

Figure 3 shows a high-level overview of the components which comprise the iSeries Connect solution:

Figure 3: iSeries Connect solution



These are the iSeries Connect components:

- A **“The iSeries Connect delivery gateway” on page 7** handles interfacing with various trading partners over a variety of connectivity mechanisms and protocols. A collection of protocol connectors support the mechanisms that partner organizations and e-marketplaces use to submit requests, such as order placement, order status checking, and catalog maintenance.
- A **“The iSeries Connect flow manager” on page 16** deals primarily with processing requests by connecting them to existing enterprise resource planning (ERP), supply chain management (SCM), and other core business applications. The flow manager uses connectors that act as integrators between iSeries Connect and your business applications.
- The **“iSeries Connect tools” on page 17** are used to implement the iSeries Connect product. Primarily, you use the tools to install, configure, and manage the other iSeries Connect components and the solution in general.
- **IBM MQSeries** handles communication between the delivery gateway and the flow manager. A version of MQSeries is available on the iSeries Connect CD-ROM for use with the product. If you already have MQSeries, you can use your version if it meets the necessary prerequisites. For more information, see Chapter 4, “Install iSeries Connect” on page 29.
- **Business applications** are the part of the solution that process requests. iSeries Connect does not perform processing, so this responsibility falls to your business applications. iSeries Connect provides a variety of application connectors that the flow manager uses to convert XML-based request messages

into a format that is compatible with your application. For more information on supported application connector types, see “The iSeries Connect flow manager” on page 16.

iSeries Connect also supports the use of WebSphere Commerce Suite as an order processing application. For more information, see “WebSphere Commerce Suite extensions” on page 18.

The iSeries Connect delivery gateway

The delivery gateway is a flexible framework that prepares protocol requests for the flow manager to process and communicate responses back to the requester. The delivery gateway handles the interfaces with various partners over a variety of connectivity mechanisms and protocols.

The delivery gateway is implemented as a set of Java servlets and a set of MQSeries message queues that communicate with the flow manager. For details about servlets that communicate with the flow manager, see “Delivery gateway servlets”.

The delivery gateway servlets run in an instance of a prerequisite IBM WebSphere Application Server (and IBM HTTP Server)

The delivery gateway uses a set of connectors to process protocol requests. Depending on the needs of the protocol, there could be a single set of connectors or multiple sets of connectors for a single protocol. Collectively, the connectors perform the protocol-specific work of the delivery gateway.

For information about how the delivery gateway processes requests, see these topics:

- “Incoming request processing in the delivery gateway”
- “Outbound request processing in the delivery gateway” on page 12

Delivery gateway servlets

The delivery gateway uses these servlets to communicate with the flow manager. The servlets run in the IBM WebSphere Application Server servlet engine.

- **AdminServlet**

  Serves as the anchor point for common resources that the delivery gateway and the delivery gateway connectors use to process the B2B protocols. To see the status of the AdminServlet, enter this URL on your browser:

  ```
  http://server_name:port/BtoB/instance_name/Admin
  ```

  where *server_name* is the name of your server, *port* is the port number, and *instance_name* is the name of your iSeries Connect instance.

  If the AdminServlet was set up correctly, you receive a response page that shows the status of the services that the delivery gateway started.
- **HTTPServlet**

  Receives and responds to B2B protocol requests. Many instances of this servlet may be running, all with different names, depending on the needs of the B2B protocol. You can determine the URLs of the servlets that are servicing requests for your instance by displaying the properties of the configured instance.
- **AsynchMessageServlet**

  Serves as the internal entry point for the instance’s OMH requests sent from the OutboundRequest API.

Incoming request processing in the delivery gateway

The delivery gateway performs certain processing steps when it receives incoming requests. These steps may differ based on the requirements of the B2B protocol request. For more information on how the delivery gateway handles incoming requests for specific protocols, see “Processing a cXML request” on page 9.

Generally, this is how a protocol flow running in the delivery gateway processes incoming requests:

1. **Parses the incoming request and, optionally, validates the contents of the incoming request.** Transforms the request into an internal usable form. Some data in the B2B request may be copied to the sendable message header that is sent to the flow manager.
2. **Authenticates the request.** The delivery gateway authenticates the request using information from the request and information that is contained in the partner and provider repository. The protocol connectors use the information that is stored in the partner and provider repository to process the B2B protocol request. The information that is required and how it is used depends on the B2B protocol. The WebSphere Commerce Suite (WCS) connector also uses information in the repository and additional required information may be needed. For more information about WCS connectors, see “Run-time services for iSeries Connect” on page 20.
3. **Performs authorization checks.** The delivery gateway checks to see if the requester is authorized to this request. This step also involves using the information in the partner and provider repository.
4. **Logs the request to the delivery gateway audit file.** The delivery gateway logs requests to files with this naming format:

   ```
   /QIBM/UserData/Connect200/Commerce/instance_name/Logs/GW_Audit_timestamp.log
   ```

   where *instance_name* is the name of your instance, and *timestamp* is the data when the file is created. You can view these files in the iSeries Connect configuration tool. (View the Tracing properties for your instance. Then, under Delivery Gateway, click **View Audit File**.) You can also use the B2B Activiy Monitor to search the audit files. For more information, see “Monitor B2B transactions” on page 105.
5. **Sends the request to the flow manager.** If the expected response to the requester is an XML document, the delivery gateway connectors build a shell response document to send to the flow manager. Some B2B protocols may expect a simple return code so no shell response is needed.

   The shell document, along with the B2B protocol request and the header, are packaged and sent to the flow manager. The header summarizes information that may be present in the request and also contains information that may need to be communicated between the delivery gateway and the flow manager that might not be present in the B2B protocol request.
6. **Receives the response document from the flow manager.** The delivery gateway connectors check for errors from the flow manager or from the connected application.
7. **Optionally, validates the response document.** If the response to the request is an XML document, the delivery gateway connectors validate that the application called by the flow manager generated a valid response. Validation ensures that only syntactically correct responses are returned to the requester; it does not validate the data in the response.

   When you develop and test applications that connect to the iSeries Connect product, turn on validation. When you have thoroughly tested the application and put it into production, you can turn off validation. However, you may want to leave validation enabled if the resulting performance is acceptable.
8. **Logs and sends the response to the requester.**

NewQuote Request

The NewQuote request acts as an incoming request, so the delivery gateway framework and flow manager framework can send shopping cart contents back to the procurement application. This allows the catalog application to be more protocol-independent, allows use of the powerful capabilities of the flow manager, and allows all of the requests and responses to be logged together in the same delivery gateway log files. The catalog application calls the NewQuote request and passes these name value pairs: PostBackURL, SupplierNumber, SupplierNumberDomain, BuyerNumber, and BuyerNumberDomain. Additionally, the catalog application sends any other name value pairs that it needs to process the request. The delivery gateway connectors convert these name value pairs into an XML form and pass the request to the flow manager.

Example: Name value pairs

```
<NameValuePairs>
   <NameValuePair name="SupplierNumber">1234567890</NameValuePair>
   <NameValuePair name="SupplierNumberDomain">DUNS</NameValuePair>
   <NameValuePair name="BuyerNumber">0987654321</NameValuePair>
   <NameValuePair name="BuyerNumberDomain">DUNS</NameValuePair>
   <NameValuePair name="PostBackURL">http://server:port/ariba</NameValuePair>
</NameValuePairs>
```

Processing a cXML request: iSeries Connect contains a delivery gateway flow implementation that handles both cXML version 1.1 and version 1.2. cXML is the B2B protocol used by Ariba. For more information about the cXML protocol, see the Commerce XML Resources Web site.

The delivery gateway performs these functions when processing a cXML request:

1. **Parses the incoming request and optionally validates the contents of the incoming request.** Information from the cXML header is copied into the internal header.
2. **Authenticates the request.** The delivery gateway authenticates the request using information from the request and information that is contained in the partner and provider repository. The cXML delivery gateway connectors make use of the information contained in the partner and provider repository to process the B2B protocol request. You must enter a provider ID and domain for the provider that matches what is in the marketplace protocol request by using the provider registration function of the iSeries Connect configuration tool. You must enter a partner organization ID and domain for the partner organization that matches what is in the marketplace protocol request by using the partner organization registration function of the iSeries Connect configuration tool.

   When you register as an Ariba provider you register with your company DUNS number and a password. This password is your shared secret (in cXML terms) with the Ariba marketplace. When a request is submitted by a partner organization, it first goes to the Ariba Network. The Ariba Network validates the request by using the shared secret of the buying organization. The Ariba Network then removes the shared secret of the buying organization and inserts the shared secret of the provider into the request. Therefore, the information that is validated by the delivery gateway authentication connector is the provider DUNS number and shared secret. This method of authentication relieves the provider of having to maintain a shared secret (or password) for every buying organization. All the provider must do is be sure to put the correct DUNS number and shared secret in the repository when associating the provider with the marketplace.
3. **Performs authorization checks.** This checks to see if the requester is authorized to this request. This step also involves using the information in the partner and provider repository.
4. **Logs the request to the delivery gateway audit file.** The delivery gateway logs requests in files with this naming format:

   ```
   /QIBM/UserData/Connect200/Commerce/instance_name/Logs/GW_Audit_timestamp.log
   ```

   where *instance_name* is the name of your instance, and *timestamp* is the date on which the file is created.
5. **Sends the request to the flow manager.** The appropriate cXML response shell document is built. The shell response along with the cXML request and the internal header is packaged and sent to the flow manager. For more information, see “Response shells for the request” on page 10.
6. **Receives the response document from the flow manager.** The delivery gateway connectors check for errors from the flow manager or from the connected application.
7. **Optionally, validates the response document.** If validation is turned on, the cXML response is validated.
8. **Logs and sends the response to the requester.**

The cXML flow implementation handles these requests and request types:

Request	Request types	Description
ProfileRequest		The ProfileRequest is handled entirely by the delivery gateway. It determines the supported request and request types based on the configuration and authorization of the partner organization. It does not pass the request to the flow manager.
PunchOutSetupResponse	Create Edit Inspect	The PunchOutSetupResponse returns the URL of the provider catalog to the requester. For more information about the cXML protocol, see the Commerce XML Resources Web site. The delivery gateway builds this “Example: PunchOutSetupResponse shell document” on page 11.
PunchOutOrderMessage		The PunchoutOrderMessage is used by the provider to send a quote back to the partner application. This quote is based upon the contents of an online shopping cart which was filled in by the partner during the B2B shopping experience.
OrderRequest	New Update Delete	The OrderRequest request receives a purchase order from a buying organization. The response is a cXML response. The delivery gateway connectors build this “Example: OrderRequest shell response” on page 11.
NewQuote [1]		The response to the NewQuote is the PunchOutOrderMessage. The delivery gateway connectors build this “Example: PunchOutOrderMessage shell response” on page 11.

[1] Additionally, a request called NewQuote, which is not part of the cXML protocol, is implemented.

The flow manager connector that you choose may not be capable of handling all of the requests and request types of the delivery gateway flow implementation.

Response shells for the request: The delivery gateway connectors create the response shell in the request before sending the request to the flow manager if the response is to be an Extensible Markup Language (XML) document.

The shell document is a partially completed XML document that contains both XML elements and attributes that the delivery gateway can either determine or not determine. The elements and attributes that cannot be determined still need to be completed by steps in a business process flow. This relieves the application and flow manager from having to perform this work and can be helpful in isolating the protocol-specific information in the delivery gateway instead of in the application. You should try to keep the applications as protocol-independent as possible.

Here are some examples of requests that contain a response shell:

- "Example: PunchOutSetupResponse shell document"
- "Example: OrderRequest shell response"
- "Example: PunchOutOrderMessage shell response"

Example: PunchOutSetupResponse shell document: The delivery gateway builds this shell document for the PunchOutSetupResponse when it returns the URL of the provider catalog to the requester.

The element and attribute values are sample values from the examples provided in the cXML specification. Actual values are determined at run time.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.009/cXML.dtd">
<cXML payloadID="456778-199@cxml.workchairs.com"
      xml:lang="en-US" timestamp="1999-03-12T18:39:09-08:00">
    <Response>
        <Status code="200" text="OK"/>
        <PunchOutSetupResponse>
            <StartPage>
                <URL></URL>
            </StartPage>
        </PunchOutSetupResponse>
    </Response>
</cXML>
```

Example: OrderRequest shell response: The delivery gateway connectors build this shell cXML response when the OrderRequest request receives an order from a provider organization.

The element and attribute values are sample values from the examples provided in the cXML specification. Actual values are determined at run time.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.009/cXML.dtd">
<cXML payloadID="9949494@cxml.workchairs.com" xml:lang="en-US"
      timestamp="1999-03-12T18:39:09-08:00">
    <Response>
        <Status code="200" text="OK"/>
    </Response>
</cXML>
```

Example: PunchOutOrderMessage shell response: The delivery gateway connectors build this shell response when the NewQuote request acts as an incoming request so that the delivery gateway and flow manager can send the PunchOutOrderMessage.

The element and attribute values are sample values from the examples provided in the cXML specification. Actual values are determined at run time.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.009/cXML.dtd">
<cXML payloadID="456778-198@premier.workchairs.com"
      xml:lang="en-US" timestamp="1999-03-12T18:39:09-08:00">
    <Header>
        <From>
            <Credential domain="DUNS">
                <Identity>942888711</Identity>
            </Credential>
        </From>
        <To>
            <Credential domain="AribaNetworkUserId">
                <Identity>admin@acme.com</Identity>
            </Credential>
        </To>
        <Sender>
            <Credential domain="DUNS">
```

```
                <Identity>942888711</Identity>
            </Credential>
            <UserAgent>Workchairs cXML V1.1</UserAgent>
        </Sender>
    </Header>
    <Message>
        <PunchOutOrderMessage>
            <PunchOutOrderMessageHeader operationAllowed="edit">
                <Total>
                    <Money currency=""></Money>
                </Total>
            </PunchOutOrderMessageHeader>
        </PunchOutOrderMessage>
    </Message>
</cXML>
```

Outbound request processing in the delivery gateway

As opposed to inbound requests (which are sent by remote trading partners), outbound requests are originated by iSeries Connect—particularly, by the delivery gateway. An outbound request can be a confirmation that an order was successfully processed or notice that an order was shipped.

Outbound requests require the use of an outbound message handler. For more information, see “Create an outbound message handler” on page 99.

Here is how the delivery gateway processes an outbound request:

1. **An inbound request is received.** See “Incoming request processing in the delivery gateway” on page 7 for the how the delivery gateway handles an incoming request. Note that information from the inbound request is stored in the sendable message header. (You can later map this information to fields in the outbound request through a request token.)
2. **The outbound request process is initiated.** The process is initiated by either a step in the business process flow or by your business application. If your business application initiates the outbound request process, you need to provide a proxy application that builds the appropriate data and calls the outbound request API. See “Create an outbound message handler” on page 99 for detailed information about these options.
3. **The outbound request is sent to the delivery gateway.**
4. **The delivery gateway receives the request and runs the protocol flow for the outbound request.** The protocol connectors transform the request into an internal usable form. Some data in the B2B request may be copied to the sendable message header that is sent to the flow manager. For example, an application token sent through the API is copied into the sendable message header under the com_ibm_connect_header_appToken value. The application token can then be retrieved by a connector in your process flow.
5. **The delivery gateway authorizes the request.** The delivery gateway authorizes the request using information from the request and information that is contained in the partner and provider repository. The protocol connectors use the information that is stored in the partner and provider repository to process the B2B protocol request. The information that is required and how it is used depends on the B2B protocol.
6. **The delivery gateway logs the request.** The delivery gateway logs requests to files with this naming format:

   `/QIBM/UserData/Connect200/Commerce/`*`instance_name`*`/Logs/GW_Audit_`*`timestamp`*`.log`

   where *instance_name* is the name of your instance, and *timestamp* is the data when the file is created.
7. **The delivery gateway sends the request to the flow manager.** If the expected response to the requester is an XML document, the delivery gateway connectors build a shell response document to send to the flow manager. See “Response shells for outbound requests” on page 13 for examples of these shell documents.

Not all of the required data for the outbound request is accessible to the delivery gateway. In the example of a StatusUpdateRequest, the shell document contains a Status element. The flow manager must gather information from the back-end business application to complete this portion of the shell document.

To configure the flow manager to handle outbound requests, you create and deploy a process flow for the specific outbound request type. This includes defining a connector instance (and mapping fields to the business application) and processing flow. Thus, when the flow manager receives the outbound request (which contains the incomplete shell document), the flow manager can query the business application for the data that is required.

8. **The delivery gateway receives the completed response document from the flow manager.** The delivery gateway connectors check for errors from the flow manager or from the connected application.
9. **The delivery gateway logs and sends the response.** The delivery gateway sends the completed outbound request to one of two places, depending on the protocol that is used:
   - **To the remote trading partner.** This is known as **push** technology. The Ariba marketplace, which supports the cXML protocol, uses this method for handling outbound requests.
   - **To a message database on the local system.** A remote trading partner then periodically checks for messages and retrieves them. This is known as **pull** technology. Currently, none of the officially supported protocols use the pull method; however, if you define and install a customized protocol, you can use the pull messaging support in iSeries Connect.

Outbound request types for cXML

The outbound request types for the cXML protocol are tied to incoming OrderRequest. When the flow manager processes an OrderRequest, it sends a response to the delivery gateway. The response is sent either through response step in the process flow or at the end of processing if an explicit response step is not defined. The delivery gateway then forwards the response to the requester.

The cXML protocol defines these outbound requests that can occur after an OrderRequest response is sent:

- **StatusUpdateRequest**

  A StatusUpdateRequest informs the requester of the original OrderRequest when the status of the order changes. For example, the request can be sent when the order is queued for processing by the business application, when the back-end business application processes the order, or if order processing fails for some reason.
- **ConfirmationRequest**

  The ConfirmationRequest contains more detailed information than the StatusUpdateRequest, which can also be used to report that an order was successfully processed. Data included in this request type can be detailed item information and shipping addresses. ConfirmationRequest is only supported in cXML 1.2.
- **ShipNoticeRequest**

  The ShipNoticeRequest can be sent when an order is shipped. This request is initiated by the business application through an outbound message handler proxy application. For more information, see “Create an outbound message handler” on page 99. ShipNoticeRequest is only supported in cXML 1.2.

For more information about the cXML outbound request types, see the cXML specification at Commerce XML Resources

Response shells for outbound requests: The delivery gateway connectors create the response shell in the request before sending the request to the flow manager if the response is to be an Extensible Markup Language (XML) document.

The shell document is a partially completed XML document that contains both XML elements and attributes that the delivery gateway can either determine or not determine. The elements and attributes that cannot be determined still need to be completed by steps in a business process flow. This relieves the application and flow manager from having to perform this work and can be helpful in isolating the protocol-specific information in the delivery gateway instead of in the application. You should try to keep the applications as protocol-independent as possible.

Here are some examples of cXML requests that contain a response shell:

- “Example: StatusUpdateRequest shell document”
- “Example: ConfirmationRequest shell document”
- “Example: ShipNoticeRequest shell document” on page 15

For further information about the request types and shell documents, see the cXML specification at Commerce XML Resources

Example: StatusUpdateRequest shell document: The delivery gateway builds this shell document for the StatusUpdateRequest when it returns the status of order processing.

The element and attribute values are sample values from the examples provided in the cXML specification. Actual values are determined at run time.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "gateway/connectors/cXML12Ariba/CXML12004.DTD">
<cXML payloadID="9983986241291@cxml.workchairs.com"
    timestamp="2001-08-21T12:57:04+00:00" version="1.2">
    <Header>
        <From>
            <Credential domain="DUNS">
                <Identity>123456789</Identity>
            </Credential>
        </From>
        <To>
            <Credential domain="DUNS">
                <Identity>987654321</Identity>
            </Credential>
        </To>
        <Sender>
            <Credential domain="DUNS">
                <Identity>123456789</Identity>
                <SharedSecret>********</SharedSecret>
            </Credential>
            <UserAgent>Connect for iSeries</UserAgent>
        </Sender>
    </Header>
    <Request>
        <StatusUpdateRequest>
            <DocumentReference/>
            <Status/>
        </StatusUpdateRequest>
    </Request>
</cXML>
```

If the an OrderRequest is successfully submitted for processing, the flow manager returns updates the Status element with code 201/Accepted.

Example: ConfirmationRequest shell document: The delivery gateway builds this shell document for the ConfirmationRequest when it returns a detailed status update.

The element and attribute values are sample values from the examples provided in the cXML specification. Actual values are determined at run time.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/1.2.004/Fulfill.dtd">
<cXML payloadID="9983987253612@cxml.workchairs.com"
    timestamp="2001-08-21T12:58:45+00:00" version="1.2">
    <Header>
        <From>
            <Credential domain="DUNS">
                <Identity>123456789</Identity>
            </Credential>
        </From>
        <To>
            <Credential domain="DUNS">
                <Identity>987654321</Identity>
            </Credential>
        </To>
        <Sender>
            <Credential domain="DUNS">
                <Identity>123456789</Identity>
                <SharedSecret>********</SharedSecret>
            </Credential>
            <UserAgent>Connect for iSeries</UserAgent>
        </Sender>
    </Header>
    <Request>
        <ConfirmationRequest/>
    </Request>
</cXML>
```

When the flow manager completes the shell document, it updates the ConfirmationRequest element with detailed information about the order.

Example: ShipNoticeRequest shell document: The delivery gateway builds this shell document for the ShipNoticeRequest when it sends notification that an order has shipped.

The element and attribute values are sample values from the examples provided in the cXML specification. Actual values are determined at run time.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/1.2.004/Fulfill.dtd">
<cXML payloadID="9983987852833@cxml.workchairs.com"
    timestamp="2001-08-21T12:59:45+00:00" version="1.2">
    <Header>
        <From>
            <Credential domain="DUNS">
                <Identity>123456789</Identity>
            </Credential>
        </From>
        <To>
            <Credential domain="DUNS">
                <Identity>987654321</Identity>
            </Credential>
        </To>
        <Sender>
            <Credential domain="DUNS">
                <Identity>123456789</Identity>
                <SharedSecret>********</SharedSecret>
            </Credential>
            <UserAgent>Connect for iSeries</UserAgent>
        </Sender>
    </Header>
    <Request>
        <ShipNoticeRequest/>
    </Request>
</cXML>
```

The flow manager completes the ShipNoticeRequest element in the shell document with information about the shipped order.

The iSeries Connect flow manager

The flow manager is a flexible framework that accepts requests from the delivery gateway, invokes a multi-step business process flow to implement the request and generate a response, and passes the response back to the delivery gateway.

The flow manager is implemented as a Java application and a set of MQSeries message queues that communicate with the delivery gateway.

With the business process flow, the flow mananger uses application connectors to interface with business applications. These connector types are supported:

- The program call connector uses Program Call Markup Language (PCML) to run a program or service program.
- The JDBC connector accesses a database.
- The MQSeries queue connectors format messages and place them on an MQSeries queue.
- The data queue connector formats messages and places them on OS/400 data queue.
- The Java connector invokes methods on a defined Java class.

The flow manager performs these functions when processing a request:

1. **Receives an incoming request from the delivery gateway.** Incoming requests are sent by the delivery gateway. The delivery gateway places a message on the queue that contains common header information from the protocol-specific request.
2. **Finds the defined business process flow that corresponds to the request.** The flow manager determines the business process flow that was defined to handle this specific request based on protocol, protocol group, request, partner, and provider.
3. **Logs the request to the flow manager log file.** By default, the flow manager logs the request in files with this naming format:

   ```
   /QIBM/ProdData/Connect200/Commerce/instance_name/Logs/FM_Audit_timestamp.log
   ```

   where *instance_name* is the name of your instance, and *timestamp* is the date on which the file is created. You can view these files in the iSeries Connect configuration tool. (View the Tracing properties for your instance. Then, under Flow Manager, click **View Audit File**.) You can also use the Management Central B2B Transaction Manager to search the log files. For more information, see "Monitor B2B transactions" on page 105.
4. **Processes each step defined in the business process flow.** The business process flow can contain these step types: connector, decision, copy, or response. For more information about the step types, see "Create business process flows" on page 69. The flow manager performs each function that is defined by the step types.

   The **connector step** type defines these functions that the flow manager performs:

   a. **Maps the incoming request fields, header fields, or constants to the application defined inputs.** Based on the selected flow, the input mappings that are defined for that flow are performed. This allows protocol-specific information to be made available to applications in a non protocol-specific manner. This results in common applications that handle multiple protocols.

   b. **Calls the application.** Depending on the application connector type, the defined application is called through the defined connector.

      Some applications called from the flow manager require initialization processing that must occur before processing requests. For more information, see "Initialization and termination exit processing" on page 17.

   c. **Maps the application output fields to the response document.** Based on the selected flow, the output mappings that are defined for that flow are performed.

In a **response step**, the flow manager passes return information to the delivery gateway. This information includes the results of processing this request and any mapped response data. The delivery gateway takes this return information and sends it back to the originator in the appropriate format for the specific protocol that the originator used.

5. The flow manager stops processing the flow when all steps have been processed.
6. **Logs the request to the flow manager log file.** The flow manager logs the request in a files with this naming format:

   ```
   /QIBM/ProdData/Connect200/Commerce/instance_name/Logs/FM_Audit_timestamp.log
   ```

   where *instance_name* is the name of your instance, and *timestamp* is the date on which the file is created.
7. (Optional) **Sends the result to the delivery gateway.** If an explicit response step is not defined in the business process flow, the flow manager sends a response to the delivery gateway when it completes processing of the request.

Initialization and termination exit processing

Some applications that the flow manager calls require initialization processing that must occur prior to processing requests.

For example, if you are using a flow that uses the data queue connector when requests come in, a message is placed on the defined queue. You might have to start the application that takes requests from the defined queue and processes them. You also might have to provide termination or cleanup when requests no longer need to be processed. In this example, when the flow manager is shut down, you can shut down the application that was listening for requests on the defined queue.

The flow manager framework provides a facility (the B2BExit class) so you can define Java classes that are called when the flow manager framework is started and stopped. The Java classes need to implement the B2BExit class that is provided in the flowmanagerapi.jar file in the /QIBM/ProdData/Connect200/Classes directory.

The B2BExit interface is defined as follows:

```
package com.ibm.connect.flowmanager.interfaces;
public interface B2BExit  {
    void initialize() throws java.lang.Exception;
    void terminate() throws java.lang.Exception;
}
```

The initialize() method implementation runs when the flow manager starts up. If the call throws an exception, it is logged, and the initialization of the flow manager fails.

The terminate() method implementation runs when the flow manager shuts down. If the call throws an exception, it is logged, and shutdown continues.

You specify the user classes on the instance properties for your B2B instance.

iSeries Connect tools

Use the iSeries Connect tools to install, configure, customize, and manage the product.

iSeries Connect configuration tool

The Web-based configuration tool is a set of wizards, functions, and editors that you use to implement iSeries Connect. The configuration tool provides utilities to accomplish these general tasks:

- Configure and manage (start, stop, edit, and migrate) instances.
- Register partner and provider organizations with iSeries Connect.
- Create, manage, and publish product catalogs.

- Configure application connectors that convert requests into data that your business application expects.
- Configure business process flow processing.
- Deploy your business process flows.
- Configure and install user-defined protocols.

For more information about the configuration tool, see Chapter 5, “Configure iSeries Connect” on page 39.

B2B Activity Monitor

You can use the B2B Activity Monitor (part of iSeries Navigator - Management Central) to monitor requests that come into iSeries Connect. You can search your requests for specific criteria, such as date received and status.

The B2B Activity Monitor is not included with the iSeries Connect product. For more information about the B2B Activity Monitor and how to get it, see “Monitor B2B transactions” on page 105.

WebSphere Commerce Suite extensions

WebSphere Commerce Suite (WCS) extensions allow the existing WebSphere Commerce Suite 5.1 or 5.4 business-to-consumer (B2C) offering to participate in B2B transactions within the iSeries Connect framework. The WCS extensions provide the ability to use WCS for communicating with marketplaces to perform local catalog or remote catalog and purchase order processing using B2B protocols.

> **Note:** It is assumed that you or someone you are working with are familiar with the WCS product and its documentation. Installing and configuring a WCS instance is beyond the scope of the iSeries Connect information.

For more information about the implementation of the WCS extensions (including current limitations), “Assumptions for WebSphere Commerce Suite (WCS) extensions”.

The WCS extensions support these main functions:

- **“WebSphere Commerce Suite (WCS) extensions configuration services” on page 19**
  Provide the connection between the iSeries Connect configuration data and the existing WCS configuration support. This topic describes the tools you use to configure your WCS and instances.
- **“Run-time services for iSeries Connect” on page 20**
  Utilize the configuration information to support the various transactions that are required to support remote catalog shopping and purchase order processing.

For specific WCS information about security, migration and coexistence, and troubleshooting concerns, see “WebSphere Commerce Suite considerations for iSeries Connect” on page 23.

Assumptions for WebSphere Commerce Suite (WCS) extensions

Here are some of the assumptions regarding the iSeries Connect WCS extensions support:

- **System topology**
  The WCS instance can reside on a system remote to the flow manager.
- **Instances**
  An iSeries Connect instance is associated with a maximum of one WCS instance. This single iSeries Connect instance supports the configuration of multiple providers, marketplaces, and partner organizations. You can use the provider registration function and partner organization registration function of the iSeries Connect configuration tool to configure these items. If you have multiple WCS instances on a single iSeries that trading partners need to access, multiple instances are required.

  A WCS instance can be used for B2B and B2C shopping environments concurrently. For more information, see these topics:
  - “Enabling a WebSphere Commerce Suite 5.1 store for remote catalog support” on page 72

  - “Enabling a WebSphere Commerce Suite 5.4 store for remote catalog support” on page 80
- **Providers and partners**
  A key WebSphere Commerce Suite extension assumption, regardless of which B2B protocol is used to communicate with the trading partner, is that for each of the transactions that WCS extensions supports, there is a means of identifying the correct provider and partner organization. This provider and partner organization information is used to identify the corresponding WCS merchant and WCS shopper.

  A single provider maps to a single WCS merchant (or store). A single partner organization maps to a single WCS shopper entry. Use the iSeries Connect configuration tool to create these mappings:
  - The provider registration function defines mappings between providers and WCS merchants.
  - The partner organization registration function provides a mechanism for you to configure partner organizations and have those partner organizations automatically registered as WCS shoppers.
- **Catalog**
  Use the WCS administrative functions to configure your stores, products, and list prices before you use the catalog management function of the iSeries Connect configuration tools.
- **Maintenance**
  To perform its functions, the WCS extensions support queries the iSeries Connect instance registry for key environment information, such as the WCS instance name, WCS hostname (which is required for sending HTTP requests to WCS), and WCS database location. If you change these WCS configuration values, you must use the iSeries Connect configuration tool to update your iSeries Connect instance registry.
- **Communication**
  Generally, WCS extensions communicate with the WCS product by sending HTTP requests and receiving HTTP responses (similar to how a browser interacts with a WCS electronic store front).

  However, in cases where extensive amounts of data are accessed (such as when the WCS catalog is extended), the WCS extensions support uses JDBC to directly access the WCS instance database tables. The WCS extensions configuration services directly access databases only during configuration of an instance. The runtime services do not access the WCS database tables directly.

WebSphere Commerce Suite (WCS) extensions configuration services

The WCS extensions configuration services for iSeries Connect provide the connection between the iSeries Connect configuration data and the existing WCS configuration support. To set up this connection, you must configure both your existing WCS instance (or create a new one) and your iSeries Connect instance. Thus, you use a combination of WebSphere Commerce Suite and iSeries Connect configuration tools.

> **Note:** When you configure your WCS and instances, it is important that you follow the steps described in “Configure WebSphere Commerce Suite” on page 70.

WCS configuration tools

Use the WCS configuration tools to perform these tasks:

- Configure the WCS instance. This configuration step creates the WCS instance, associated database tables, and other resources. You can also incorporate a sample store into the WCS instance.
- Populate your WCS instance with merchant and product information using existing methods.

For more information about the WCS configuration tools, see these resources:

- Installation Guide: WebSphere Commerce Suite Professional Edition for e-server iSeries, Version 5.1
- Installation Guide: WebSphere Commerce Suite Professional Edition for e-server iSeries, Version 5.4

iSeries Connect configuration tool

The iSeries Connect configuration tool performs these tasks:

- Allows an iSeries Connect instance to be associated with a WCS instance.
- Maps providers to their corresponding WCS merchants with the provider registration function.
- Allows a partner organization to become automatically registered in the WCS environment (as a registered shopper) with the partner organization registration function. Also, use the partner organization registration function to administer address and password information for these shoppers. If you use tools within the WCS environment to modify this information, you must update the corresponding information in the iSeries Connect environment (with the iSeries Connect configuration tool) to make the changes known to the WCS extensions services.
- Extends the product catalog data that is contained within the WCS instance. This step is necessary for both local catalog and remote catalog processing. This extended product data includes manufacturer name, manufacturer part number, Universal Standard Products and Services Classification (UNSPSC) code, United Nations Units of Measure (UNUOM) specifications, and lead time for each product that can be sold through the B2B marketplace. You define this information for all products that are offered in the B2B shopping experience.

  If you provide a subset of your B2C products for your remote B2B catalog, you need only provide extended product data for the B2B products. You can define separate catalogs to support a different product list for your B2C and B2B shoppers.

For more information about the iSeries Connect configuration tool, see Chapter 5, “Configure iSeries Connect” on page 39.

Run-time services for iSeries Connect

The WebSphere Commerce Suite runtime services for iSeries Connect utilize the configuration information to support the various transactions that are required to support remote catalog shopping and purchase order processing.

All of the WCS extensions configuration information provided by the administrator is performed so that various B2B requests supported by the WCS extensions can be accomplished. These runtime requests involve the various messages (request and request type) exchanged between the B2B trading partners to carry out business transactions.

Note: The WCS extensions runtime support is designed to be B2B protocol independent. This means that the same business logic is run when processing the supported B2B transactions, regardless of the B2B protocol being used between the local system and the B2B trading partner. All of the protocol differences are handled by supplying different protocol flow models for the different protocols supported.

WCS extensions supports handling requests for both local and remote catalog processing. For local catalog processing, the request involved is called OrderRequest for the cXML 1.1 and 1.2 protocols. PurchaseOrder is used for discussing this request if you are describing an aspect of processing that is B2B protocol independent. For remote catalog processing using the cXML protocol, requests that can be used include: PunchOutSetupRequest, NewQuote, and OrderRequest. Remote catalog processing is sometimes referred to as PunchOut processing. The cXML PunchOutSetupRequest is used to initiate a remote catalog session between the B2B trading partner and the local system. RemoteCatalogInitiation is used for discussing this request if you are describing an aspect of processing that is B2B protocol independent.

Note: WCS extensions support new, update, and delete PurchaseOrder requests. For remote catalog processing, two additional requests are involved. For cXML the two additional requests are PunchoutSetupRequest and NewQuote. WCS extensions support the different request types associated with the cXML PunchoutSetupRequest (create, edit, and inspect). The NewQuote request does not have additional request types associated with it.

Request processing for WCS extensions can be further broken down into these steps:

- Mapping incoming messages (XML data) into parameters understood by the WCS extensions Java objects that support these various B2B requests.
- Using the parameters received in the previous step to drive WCS to perform the function requested. For example, a valid incoming PurchaseOrder request causes a new order to be added to the WCS ORDERS database table, and that order is associated with the partner organization that sent in the PurchaseOrder.

Here are details regarding the mapping of incoming messages:

There are several WCS extensions Java connectors that are shipped as part of the iSeries Connect product. There is one of these WCS extensions Java connectors for each B2B request supported by WCS extensions. These connectors are designed to provide protocol independent mapping of incoming and outgoing B2B protocol data. As with any other customer or IBM Business Partner supplied iSeries Connect connector implementation, there are application connector document and protocol flow model definitions associated with the WCS extensions Java connector. These WCS extensions application connector document and process flow model definitions are shipped with the iSeries Connect product. For information about how to update these files, see “Customize WebSphere Commerce Suite” on page 96.

The general idea is that data that is understood and can be used by the WCS extensions Java objects that support the B2B requests (PurchaseOrder, RemoteCatalogInitiation, and NewQuote) is defined as fields in the application connector documents. The application connector document provides a protocol independent description of the parameters that the WCS extensions Java objects need to fill in for processing a given B2B request. The process flow model document is used to describe the protocol dependent mapping of these fields from the incoming XML message (for example, a cXML OrderRequest) into the WCS extensions Java object that supports the particular B2B transaction. Once these WCS extensions Java objects are filled in, the WCS extensions Java object interprets the data and determines the appropriate action to take. For example, the provider identity is determined so the correct WCS merchant can be identified. The partner organization is determined so the correct WCS shopper can be identified. The provider part number is identified so the correct WCS product is identified. Once it has been determined that all of the data required to perform the B2B PurchaseOrder has been received, the WCS extensions Java object interfaces with the WCS instance to cause the order to get placed into the WCS database tables.

Here is some additional information regarding the processing that occurs in the WCS extensions runtime code and within the WCS environment when processing an incoming PurchaseOrder.

The successful processing of a PurchaseOrder request results in an order entry being added into the WCS ORDERS table. PurchaseOrder processing represents an actual agreement for the partner organization to pay for the products that are listed in the PurchaseOrder. Once the order has been placed, the other processes regarding order fulfillment and supply chain management can proceed.

A PurchaseOrder results in a new entry being added to the WCS Orders table. The Orders table records the order level information, such as store, shopper number, billing address, and total price of the order. Each item in the PurchaseOrder is inserted into the WCS Shipto (Order Item) table. Order item information includes product number and quantity ordered. An Order Item table entry provides a means of storing the shipping address on a per item basis. The cXML protocol allows specification of a shipping address for an entire order or on a per item basis. The WCS extensions runtime services is designed to be protocol independent. If the shipping address is provided on a per item basis, then it uses that address. Otherwise, the shipping address affecting the entire order will be used. In either case, shipping information is stored on a per item basis. Upon successful addition of the order into WCS tables, the status of the entry in the WCS ORDERS table is set to complete, which indicates that the order was successfully placed. If all of the PurchaseOrder processing is successful, a good response is returned to the partner organization. If runtime services encounters an error in the processing of this PurchaseOrder, then the response provides an error code that reflects the type of error that was encountered.

As mentioned earlier, WCS extensions supports OrderRequest update and OrderRequest delete flows for cXML. For OrderRequest delete, the WCS extensions verifies that a WCS administrator user ID and password has been configured in the instance, and that the iSeries Connect support can log in to WCS using this user ID. It will also determine the WCS order number that corresponds to the orderID field received in the incoming OrderRequest. This information is determined using the QABECORDMP database table. After the WCS order number has been determined, a request is sent to WCS to cause that previous order to be cancelled. The processing of this command requires logging in with a user ID that has customer service representative authority in WCS. The WCS extensions support will process the cancel request if it is determined that the WCS Order has an order status of Complete. If the order status has already been moved to something other than Complete, such as Shipped, then the Cancel request will not be processed. If WCS extensions determines that the Cancel request can be honored, it will modify the status of the WCS order to indicate the request has been cancelled.

The OrderRequest Update flow is processed in a similar fashion to the OrderRequest delete. This flow also requires logging into WCS with a user ID having customer service representative authority. The update is actually handled by changing the status of the previous order to Cancelled, and then creating a new WCS Order entry based upon the information received in the OrderRequest update. If a problem is encountered in either modifying the previous order to cancelled status or in the processing of the new order, then the state of the WCS Orders database will be rolled back to the state it was in prior to receiving the OrderRequest update.

An additional point to make regarding PurchaseOrder processing has to deal with the way the runtime services handles duplicate purchase orders. A duplicate PurchaseOrder can occur when the partner side application or marketplace resends the exact same XML message that it had previously sent, because for some reason, it had not received a response from the provider (possibly due to a network outage). By default, the WCS extensions runtime services code detects these duplicate PurchaseOrder requests and handles the situation based upon its own knowledge of what has occurred regarding that PurchaseOrder. For example, if the order had been successfully placed within WCS previously, then upon receiving this duplicate PurchaseOrder, a response is returned to the marketplace or partner side application to indicate that the order was successfully placed (without actually placing the order a second time within WCS). There is a database table that is created as part of the instance, called QABECORDMP, which is used to implement this functionality. By default, the maximum number of distinct order mapping entries in this database table is 3000. See “Customize WebSphere Commerce Suite” on page 96 for information about how this function can be disabled or the number of table entries modified.

The processing involved in supporting a remote catalog shopping experience between a B2B trading partner and a WCS store is a little more complicated than purchase order processing. There are more requests sent and received between the parties involved.

The remote catalog shopping experience begins by having a requisitioner initiate the remote shopping experience by selecting a remote catalog to punchout to. The assumption here is that all of the configuration steps involved on the local (provider side system) and the partner systems have been performed. The partner sends in the RemoteCatalogInitiation request. This request validates that this partner organization is authorized to shop at this provider site. This authorization process actually ensures that this B2B buying organization is authorized to access this provider using the iSeries Connect support and also ensures that this partner organization has a WCS shopper table entry that has been configured and associated with this partner organization. The RemoteCatalogInitiation is also used by the partner to retrieve a URL, called the StartPageURL, which contains the actual website that it needs to access to begin the shopping experience. For cXML, this StartPageURL is returned to the partner application in the PunchOutSetupResponse cXML message.

For cXML, the partner application then launches a new browser session that directs the requisitioner directly to the WCS store to begin browsing the catalog, inserting items into a quote, and so on. At the end of the shopping experience, the requisitioner has the option of causing the quote to be generated or to end the shopping experience without generating a quote. In either case, this ending of the shopping experience causes a NewQuote message to get redirected to the iSeries Connect delivery gateway. This NewQuote request contains information that allows the WCS extensions support to correlate this NewQuote with the previous RemoteCatalogInitiation request, and it will also have information that identifies the quote number that had been generated within WCS. As part of this processing, the WCS extensions support interacts with the WCS instance to retrieve all of the detailed information regarding the quote, such as products selected, quantities, prices, and so on.

The WCS extensions runtime services uses this information to generate a message containing the quote. For cXML, the message generated is a PunchOutOrderMessage. It is important to note that from the WCS extensions perspective, it does not matter which message is being generated. All of the protocol specifics are handled by using the appropriate Protocol Flow Model within the Connect Flow Manager. The appropriate message containing the quote is then packaged by the delivery gateway that supports this iSeries Connect instance and is returned to the partner application. The partner can then decide to submit this quote for approval, and if the quote is approved by their organization, a PurchaseOrder can be submitted.

Note: The PurchaseOrder processing performed by the WCS extensions and the WCS support is the same as described above. The processing does not differ based on whether the PurchaseOrder was preceded by the remote catalog shopping experience or not. For cXML, after a quote is returned to the partner application, it is possible for the partner application to either inspect or edit the previous quote. The flows involved for these inspection or edit requests are the same as when the initial quote generation was performed. In these cases, the PunchOutSetupRequest is qualified with a different request type (either inspect or edit). The authorization and generation of the StartPageURL occurs. If the option specified was edit, then the requisitioner is allowed to continue shopping, add items, delete items, change quantities, and so on. If the option is inspect, then the requisitioner is only allowed to view information regarding the quote that was previously generated. For example, no modifications of the quote are allowed.

WebSphere Commerce Suite considerations for iSeries Connect

Other things to consider when using WebSphere Commerce Suite (WCS) and iSeries Connect are security, migration, coexistence, and troubleshooting.

Security

The iSeries Connect configuration tool functions, specifically provider registration, partner organization registration, and catalog management, run under a user profile on iSeries that has all object authority. When the WebSphere Commerce Suite (WCS) Extensions configuration services runs on their behalf, the instance database tables and the WCS instance database tables are directly accessible through JDBC. However, when WCS extensions runtime services is running, functions run under the user profile of the iSeries Connect instance. This profile has access to the instance database tables, but does not have direct access to the WCS database tables.

When runtime services is running, it only runs commands that are related to remote catalog shopping or order placement on behalf of a particular partner organization. The WCS shopper logon ID and password needs to be accessible to runtime services. WCS extensions runtime services determine the shopper logon ID that is associated with the partner organization. The WCS extensions support has access to the WCS shopper logon ID and password to perform functions within WCS on the behalf of the B2B trading partner.

The WCS administrator ID and password are optional parameters that can be used when an iSeries Connect instance is associated with a WCS instance. The WCS administrator userID and password must be provided if WCS extensions are providing support for OrderRequest Update and OrderRequest Delete process flows. The WCS administrator ID and password entered needs to have customer service representative authority in the associated WCS instance. It is recommended that you specifically create a user ID with this authority get created for use by iSeries Connect, using the Websphere Commerce Suite administrator tool.

The reason for creating a separate user ID to be used by Connect (as opposed to using the same administrator userID and password that you use in WCS for performing the administrator, accelerator, and store services functions) is to avoid cases where iSeries Connect is attempting to login to WCS at the same time someone is performing WCS administration using the service tools shipped with WCS. If both Connect and WCS were attempting to use the same administrator user ID concurrently, unpredictable results will occur.

Migration and coexistence

An administrator is required to create a new WCS instance and create a new store definition to participate in B2B transactions. This is required to force the ending of the shopping experience to work properly. For example, when a partner organization has finished selecting the items to be included in a quote, the iSeries Connect framework must gain control, so the quote can be generated and returned in the correct message format to the partner organization. The implementation of the store controls what happens at the end of the shopping experience. Existing WCS store definitions that are only set up for B2C shopping experiences are not set up to handle this.

iSeries Connect supports upgrading your iSeries Connect instance support from WCS 5.1 to WCS 5.4. For more information, see “Migrate iSeries Connect middleware” on page 92.

Troubleshooting

WCS extensions detect errors, exceptions, and other trace information, and use the flow manager logging support when the WCS extensions runtime services is running. When troubleshooting problems that occur between iSeries Connect and WCS, the log files that are associated with the WCS instance may also provide useful information and should be collected by service personnel. For WCS 5.1, these log files are located in the /QIBM/UserData/CommerceSuite5/instances/*instance_name*/logs directory. (*instance_name* is the name of your WCS instance.)

The WCS extensions trace and exception entries are logged into the same files that the flow manager uses. The WCS extensions runtime services does not create any audit log entries. When the WCS extensions configuration services function is running on behalf of the B2B catalog management function or provider registration function and partner organization registration function of the iSeries Connect configuration tool, error messages, exception data, and tracing information are logged into the same file into which the configuration tool is logging. These files are located in the QIBM/UserData/Connect200/Commerce/servlet/logs directory.

See the “WebSphere Commerce Suite extensions runtime return codes” on page 110 for additional troubleshooting information.

The QABECORDMP database table

WCS extensions use the QABECORDMP database table to provide mappings. The table detects duplicate purchase orders that are received from a B2B trading partner and handles them accordingly. This table is also used for keeping track of the previous orders that have been placed so that cXML OrderRequest Update and OrderRequest Delete requests can be processed.

The QABECORDMP database table is part of the instance library—not a new database table within the WCS database collection. The table requires no special processing.

Chapter 3. Plan for iSeries Connect

See these topics for information on planning for your iSeries Connect implementation:

“Migrate the iSeries Connect product” on page 91
If you have implemented iSeries Connect 1.1, see this topic for information about migrating to version 2.0.

“Plan for handling requests with your business applications”
See this topic for information about preparing your business for B2B and iSeries Connect.

“Plan for the marketplace” on page 26
This topic describes planning steps to take to meet requirements for the marketplace.

Plan for handling requests with your business applications

Many businesses find themselves being rushed by their partners to enter the world of business-to-business (B2B). These businesses have a tendency to start implementing their solution before making sure that their business processes are properly set up to handle this new way of doing business. This could be a costly mistake.

Before you install and configure iSeries Connect, you have several decisions to make about how iSeries Connect is implemented. A good understanding of B2B concepts, iSeries Connect, and your business processes and applications is important to making the best decisions for your situation.

See these topics for information about the key decisions in planning for your iSeries Connect implementation.

- “Choose a way to process requests”.
- “Choose an application server to support iSeries Connect”.
- “Choose an iSeries Connect application connector” on page 26.
- “Choose a protocol” on page 26.
- “Define a business process flow” on page 26.

Choose a way to process requests

B2B integration products, such as iSeries Connect, provide the connectivity that is needed to accept requests from external trading partners and pass them on to your business processes. It is ultimately the responsibility of your existing business applications, such as your order processing application, to process these requests. iSeries Connect does not process these requests directly.

More than likely, you have a variety of existing business applications that support order transactions. Analyze your business applications for possible integration points to iSeries Connect.

Note: iSeries Connect also supports WebSphere Commerce Suite (WCS) as a back-end application through the iSeries Connect “WebSphere Commerce Suite extensions” on page 18. If you have implemented a WCS instance to handle online shopping and order requests, you can leverage your existing WCS implementation for B2B transactions.

Choose an application server to support iSeries Connect

iSeries Connect requires a Web application server (IBM WebSphere Application Server Version 4.0, Advanced Single Server Edition for iSeries or IBM WebSphere Application Server Version 4.0, Advanced

Edition for iSeries) as prerequisite software. iSeries Connect automatically configures an application server instance with the necessary support, so technical experience with an application server is not required to implement iSeries Connect.

Choose an iSeries Connect application connector

iSeries Connect uses components called **application connectors** to access back-end applications. Choose an application connector that best fits your applications. See “Configure an application connector” on page 44 for a list of currently supported application connectors.

If your application cannot be accessed by the application connectors (such as, your application cannot be externally called), you will most likely have to create a proxy application that can be accessed by one of the connectors. The proxy application would then be responsible for accessing the business application.

Choose a protocol

You and your partners determine which protocol your iSeries Connect instance uses. It is important that you understand the protocol and how the data in the requests are formatted.

iSeries Connect supports the following protocols:

Protocol name	Description
cXML 1.1 or 1.2	For more information, see the cXML Web page .
User defined	You can use iSeries Connect tools to define your protocol. For more information, see “User defined protocols” on page 96.

Define a business process flow

A **process flow** defines the sequence of processing steps that are required to handle a request. These steps may include the starting of connectors and other steps. For more information about process flow step types, see “Create business process flows” on page 69.

You need to define a process flow to handle each type of request that your iSeries Connect instance is configured to handle. To plan for the process flows, analyze your current business logic. For example, if your order processing application performs a database call to check the inventory of a particular product, you may want to define a decision step in your iSeries Connect process flow that performs the same function.

Plan for the marketplace

See these topics for information about supporting your B2B marketplace.

Order a DUNS number

If you plan to use a B2B marketplace, make sure your company has a Data Universal Numbering System (DUNS) number. B2B marketplaces require that each company has a unique DUNS number. A DUNS number is a nine-digit number that Dun & Bradstreet issues to identify each corporate location of a business. You can order a DUNS number from the Dun & Bradstreet Web site.

Choose your catalog method

Determine how to make your available products and services known to your partner organizations. This helps you determine the best way to configure and use iSeries Connect. iSeries Connect can build and maintain electronic catalogs for you and distribute them to your partner organizations or provider system to host. This determination is often made on a per partner organization basis as each has its own preference. In fact, some partner organizations may prefer to work from printed catalog numbers or spreadsheets. Knowing how you are going to handle this ahead of time helps with your planning.

For more information about the iSeries Connect catalog services, see “Create a catalog” on page 42.

Chapter 4. Install iSeries Connect

This topic guides you through installing iSeries Connect, the prerequisite software that is shipped in the Connect for iSeries CD-ROM package, and product fixes.

> **Tip**: Before you install iSeries Connect, check the iSeries Connect 2.0 Release Notes for any special instructions.

To install iSeries Connect on your system, perform these steps:

1. Based on your operating system version, ensure that your iSeries system meets all iSeries Connect prerequisites:
   - “V5R1 prerequisites for iSeries Connect 2.0”
   - “V5R2 prerequisites for iSeries Connect 2.0” on page 30
2. “Ensure TCP/IP is ready” on page 31.
3. “Install IBM MQSeries” on page 31.
4. “Install fixes for the prerequisite products” on page 33.
5. “Install the iSeries Connect product” on page 34.
6. “Install fixes for iSeries Connect” on page 35.
7. “Perform post-installation steps” on page 35.
8. To verify the installation of iSeries Connect, “Run the PCML verification sample” on page 85.

If you want to remove iSeries Connect from your system either permanently or so you can re-install the product, see “Uninstall iSeries Connect” on page 37 for instructions.

V5R1 prerequisites for iSeries Connect 2.0

Check to see that your iSeries and workstation systems meet these hardware and software requirements.

iSeries hardware
These models and processor features are the minimum recommended requirements:

- iSeries 400 Model 270 with processor feature 2250
- iSeries 400 Model 820 with processor feature 2395
- 512 MB of memory

Connect for iSeries 2.0 also operates with all earlier RISC hardware models that meet comparable performance specifications and that run OS/400 Version 5 Release 1.

iSeries software
These licensed programs are necessary to install and run iSeries Connect:

- OS/400 Version 5 Release 1 (5722-SS1) with cumulative PTF package **C2134510** or later
- Qshell Interpreter (5722-SS1, Option 30)
- IBM Digital Certificate Manager (5722-SS1, Option 34)
- Cryptographic Access Provider for AS/400 (5722-AC3)
- IBM HTTP Server for AS/400 (5722-DG1)
- IBM Developer Kit for Java 1.3 (5722-JV1, Option 5)
- TCP/IP Connectivity Utilities (5722-TC1)
- One of these Web application servers:
  – IBM WebSphere Application Server Version 4.0 or later, Advanced Edition (5733-WA4)

  - IBM WebSphere Application Server Version 4.0 or later, Advanced Edition Single Server Edition (5733-WS4)
- (Optional) IBM WebSphere Commerce Suite 5.4 (5733-WC5) or 5.1 (5798-WC5)

  Notes:

  1. If WebSphere Commerce Suite (WCS) 5.1 (5798-WC5) is installed, then IBM WebSphere Application Server (WAS) Version 3.5.4, Advanced Edition (5733-WA3) must be installed in addition to WAS 4.0.
  2. WCS does not need to be locally installed. iSeries Connect supports WCS on a remote system. Also note that installing and configuring a WCS instance is beyond the scope of the iSeries Connect documentation. You or someone working with you needs to be familiar with the WCS product and its documentation.
- The latest PTF levels for all prerequisite software products. See Connect for iSeries: Program Temporary Fixes (PTFs) for information and instructions to ensure you have the correct PTF levels for each product.

Workstation hardware
These are the minimum recommended workstation hardware requirements for running the Web-based configuration tool:

- 300 MHz processor
- 64 MB of memory

Workstation software configuration tool:

- Microsoft Internet Explorer 5.x with JavaScript and cookies enabled.

V5R2 prerequisites for iSeries Connect 2.0

Check to see that your iSeries and workstation systems meet these hardware and software requirements.

iSeries hardware
These models and processor features are the minimum recommended requirements:

- iSeries 400 Model 270 with processor feature 2250
- iSeries 400 Model 820 with processor feature 2395
- 512 MB of memory

Connect for iSeries 2.0 also operates with all earlier RISC hardware models that meet comparable performance specifications and that run OS/400 Version 5 Release 2.

iSeries software
These licensed programs are necessary to install and run iSeries Connect:

- OS/400 Version 5 Release 2 (5722-SS1) with cumulative PTF package **C2211520** or later
- Qshell Interpreter (5722-SS1, Option 30)
- IBM Digital Certificate Manager (5722-SS1, Option 34)
- Cryptographic Access Provider for iSeries (5722-AC3)
- IBM HTTP Server for iSeries (5722-DG1)
- IBM Developer Kit for Java 1.3 (5722-JV1, Option 5)
- TCP/IP Connectivity Utilities (5722-TC1)
- One of these Web application servers:
  - IBM WebSphere Application Server Version 4.0 or later, Advanced Edition (5733-WA4)

  - IBM WebSphere Application Server Version 4.0 or later, Advanced Edition Single Server Edition (5733-WS4)
- (Optional) IBM WebSphere Commerce Suite 5.4 (5733-WC5)

  Note: WebSphere Commerce Suite (WCS) does not need to be locally installed. iSeries Connect supports WCS on a remote system. Also note that installing and configuring a WCS instance is beyond the scope of the iSeries Connect documentation. You or someone working with you needs to be familiar with the WCS product and its documentation.
- The latest PTF levels for all prerequisite software products. See Connect for iSeries: Program Temporary Fixes (PTFs) for information and instructions to ensure you have the correct PTF levels for each product.

Workstation hardware
These are the minimum recommended workstation hardware requirements for running the Web-based configuration tool:

- 300 MHz processor
- 64 MB of memory

Workstation software configuration tool:

- Microsoft Internet Explorer 5.x with JavaScript and cookies enabled.

Ensure TCP/IP is ready

The iSeries Connect installation relies on TCP/IP utilities. Follow these steps to ensure that the TCP/IP utilities are configured and started:

1. TCP/IP must be started before you install the iSeries Connect product. If it is not started, run the Start TCP/IP (STRTCP) command from the iSeries command line.
2. The TCP/IP loopback interface must be configured and active. To check the status of the loopback interface, follow these steps:
   a. On the iSeries command line, enter the Configure TCP/IP (CFGTCP) command.
   b. Select Option 1 (Work with TCP/IP Interfaces).
   c. In the Work with TCP/IP Interfaces display, look for an entry that has these values:
      - **Internet Address**: 127.0.0.1
      - **Subnet Mask**: 255.0.0.0
      - **Line Description**: *LOOPBACK
      - **Line Type**: *NONE

      If this entry does not exist, select Option 1 (Add) to add it with the values that are specified.
   d. In the Work with TCP/IP Interfaces display, press **F11** to display interface status. If `Active` does not appear as the value of **Interface Status** for the loopback interface, select Option 9 (Start) to start it.

Install IBM MQSeries

iSeries Connect requires these IBM MQSeries products:

- MQSeries 5.2 (5733-A38)
- MQSeries Classes for Java Product Extension Support Pac (5648-C60)
- MQSeries Application Messaging Interface (AMI) Product Extension Support Pac (5724-A23)

If these MQSeries products are already installed on your server, you may skip this step.

If your server does not have these product installed, MQSeries 5.2 and the required Support Pacs are provided on your iSeries Connect product CD-ROM.

To install the MQSeries products, follow these steps:

1. If you have version prior to MQSeries 5.1, you must delete it before you install MQSeries 5.2. To determine if you have an earlier version, run the Display Software Resource (DSPSFWRSC) command from the iSeries command line.

   If you have a previous version, use the Delete Licensed Program (DLTLICPGM) command to remove the earlier version. For example, to uninstall MQSeries Version 4.2, run this command:

   ```
   DLTLICPGM LICPGM(5769MQ2) RLS(V4R2M0)
   ```
2. If you have MQSeries 5.1, you can install MQSeries 5.2 over it. You do not need to delete version 5.1.
3. Insert the **Connect for iSeries 5733-CO2 V2.0** CD-ROM (which contains the MQSeries 5.2 products) in your iSeries CD-ROM drive. If you are installing MQSeries on a non-English system, see “MQSeries supported languages”.

   Note: These instructions assume that your iSeries CD-ROM drive is named OPT01. If it is not, replace `OPT01` in the commands with the name of your CD-ROM drive.
4. Install the base MQSeries product. If the primary language of your system is supported by MQSeries, enter these commands on an iSeries command line:

   ```
   RSTLICPGM LICPGM(5733A38) DEV(OPT01) RLS(V5R2M0) RSTOBJ(*PGM)
   RSTLICPGM LICPGM(5733A38) DEV(OPT01) RLS(V5R2M0) RSTOBJ(*LNG)
   ```

   The appropriate language objects are installed.

   If your system’s primary language is not supported by MQSeries, then use the language (LNG) parameter to override the system default and install the English version:

   ```
   RSTLICPGM LICPGM(5733A38) DEV(OPT01) RLS(V5R2M0) RSTOBJ(*PGM) LNG(2924)
   RSTLICPGM LICPGM(5733A38) DEV(OPT01) RLS(V5R2M0) RSTOBJ(*LNG) LNG(2924)
   ```
5. To install MQSeries Classes for Java Support Pac, run this command:

   ```
   RSTLICPGM LICPGM(5648C60) DEV(OPT01) RLS(*FIRST)
   ```

   The Support Pac is only available in the English language. When you install the Support Pac on a system with a primary language other than 2924, use the language (LNG) parameter to override the system default and install the English language version:

   ```
   RSTLICPGM LICPGM(5648C60) DEV(OPT01) RLS(*FIRST) LNG(2924)
   ```
6. To install the MQSeries AMI Support Pac, run this command:

   ```
   RSTLICPGM LICPGM(5724A23) DEV(OPT01) RLS(*FIRST)
   ```

   The Support Pac is only available in the English language. When you install the Support Pac on a system with a primary language other than 2924, use the language (LNG) parameter to override the system default and install the English language version:

   ```
   RSTLICPGM LICPGM(5724A23) DEV(OPT01) RLS(*FIRST) LNG(2924)
   ```

Note: To support iSeries Connect, you need to install fixes for MQSeries and perform some steps after the fixes have been applied. Installing the fixes and the post-installation steps are documented later in the “Install fixes for the prerequisite products” on page 33 topic.

MQSeries supported languages

These languages are supported by IBM MQSeries:

- Belgian English (2909)
- Belgian French (2966)
- Canadian French (2981)

- English Uppercase (2950)
- English Uppercase and Lowercase (2924)
- English Uppercase DBCS (2938)
- English Uppercase and Lowercase DBCS (2984)
- French (2928)
- French MNCS (2940)
- Italian (2932)
- Italian MNCS (2942)
- Spanish (2931)
- Japanese (2962)
- Korean (2986)
- Simplified Chinese (2989)

Install fixes for the prerequisite products

Before you install the iSeries Connect product, you must install certain critical fixes to the prerequsitie software products to ensure that they function correctly with iSeries Connect. These fixes are included in the iSeries Connect Group PTF.

> **Note:** The fixes in the iSeries Connect Group PTF do not contain every fix you need to apply for the prerequisite products. It is recommended that you install all current fixes that are available for these products. See Connect for iSeries Support Information for links to support pages for the prerequisite products.

To install fixes for the prerequisite products, follow these steps:

1. Install all product prerequisites and the fixes that are available for these products.
2. Install the fixes that are necessary to support iSeries Connect. Insert the **iSeries Connect Group PTF** CD-ROM in your iSeries CD-ROM drive. The CD-ROM you use depends on your operating system version:
   - For V5R1, the Group PTF number is **SF99165**.
   - For V5R2, the Group PTF number is **SF99166**.
3. From the iSeries command line, run the Program Temporary Fix (GO PTF) command.
4. In the Program Temporary Fix display, specify option 8 (Install program temporary fix package).
5. In the Install Options for Program Temporary Fixes display, specify these values and press **Enter**:
   - **Device**: OPT01 (or the name of your CD-ROM drive)
   - **Automatic IPL**:
     - Specify Y if you want an initial program load (IPL) performed automatically)
     - Specify N if you want to manually perform the IPL or wait for a scheduled IPL)
   - **Restart type**: *SYS
   - **PTF type**: 1 (All PTFs)
   - **Other options**: N

   Note: You must perform an IPL to apply the fixes. If you want to wait for a scheduled IPL, do not try to install the iSeries Connect product before the scheduled IPL completes.
6. To view installation messages, run the Work with Licensed Programs (GO LICPGM) command and select option 50 (Display log for messages).
7. To verify the status of the fixes, run the Display PTF Status (DSPPTF) command.

8. The MQSeries 5.2 fixes require that you update the user profiles that run MQSeries. To update the user profiles, follow these steps:
   a. On an iSeries command line, run these commands:
      ```
      CHGUSRPRF USRPRF(QMQM) STATUS(*ENABLED) PWDEXPITV(*NOMAX)
      CHGUSRPRF USRPRF(QMQMADM) STATUS(*ENABLED) PWDEXPITV(*NOMAX)
      ```
   b. To verify this fix, run the Work with MQM Queues (WRKMQM) command. If the Work with MQM Queues display appears, the fix was successful.
9. See the Connect for iSeries Support Information page for a current list of required fixes.

Install the iSeries Connect product

Before you install iSeries Connect, follow these steps to prepare your system for the installation:

Notes:

- If iSeries Connect Version 1.0 is installed, it is deleted when iSeries Connect Version 2.0 is installed. Version 1.0 instances and user data are preserved, but they cannot be directly migrated to version 2.0. Version 1.0 instances and user data must first be migrated using iSeries Connect 1.1.
- If iSeries Connect Version 1.1 is installed, it is not deleted when iSeries Connect Version 2.0 is installed. The two products can both exist on the same system, and they both can be functioning at the same time. Version 1.1 instances can be migrated to Version 2.0.

1. If you have iSeries Connect 1.0 or 1.1 installed, follow these steps:
   a. In the iSeries Connect 1.0 or 1.1 configuration tool, stop all of your instances.
   b. End the QCONNECT subsystem. Run this command from an iSeries command line:
      ```
      ENDSBS QCONNECT
      ```
   c. Make sure that the QCONNECT subsystem has ended. Run the Work with Active Jobs (WRKACTJOB) command to verify. If QCONNECT is still running, use option 4 (End) to stop it.
2. End the MQSeries subsystem. Run this command:
   ```
   ENDSBS QMQM
   ```
3. If you have installed WebSphere Application Server, end the QEJBSBS subsystem. Run this command:
   ```
   ENDSBS QEJBSBS
   ```
4. If the HTTP Server administrative instance is not ended, end the HTTP Server administrative instance:
   ```
   ENDTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
   ```

To install the Connect for iSeries licensed program, follow these steps:

1. Insert the **Connect for iSeries 5733-CO2 V2.0** CD-ROM in your iSeries CD-ROM drive.
2. Install the base option of iSeries Connect. iSeries Connect is currently only available in the English language version. If the primary language of your system is English (2924), English Uppercase (2950), English DBCS Uppercase (2938), or English DBCS Uppercase and Lowercase, enter this command on the iSeries command line:
   ```
   RSTLICPGM LICPGM(5733CO2) DEV(OPT01) RLS(V2R0M0) OPTION(*BASE)
   ```

   If the primary language of your system is not English, use the language (LNG) parameter to override the default language. On the iSeries command line, enter this command:
   ```
   RSTLICPGM LICPGM(5733CO2) DEV(OPT01) RLS(V2R0M0) OPTION(*BASE) LNG(2924)
   ```
3. Install option 1 of iSeries Connect. If the primary language of your system is a version of English, enter this command:
   ```
   RSTLICPGM LICPGM(5733CO2) DEV(OPT01) RLS(V2R0M0) OPTION(1)
   ```

For non-English systems, enter this command:

```
RSTLICPGM LICPGM(5733CO2) DEV(OPT01) RLS(V2R0M0) OPTION(1) LNG(2924)
```

Notes:

a. You may receive the message AMQ7010 - The queue manager does not exist. You can ignore this message.

b. If you have not previously installed iSeries Connect 1.0 or 1.1 on your system, you may receive the message CPD3D91. You can ignore this message.

Before you configure iSeries Connect, you must install the necessary fixes for the product. See “Install fixes for iSeries Connect” for more information.

Install fixes for iSeries Connect

Before you configure iSeries Connect, you must install fixes for the product.

Note: It is strongly recommended that you install these fixes before you configure an instance.

To install fixes for iSeries Connect, follow these steps:

1. Insert the **iSeries Connect Group PTF** CD-ROM in your iSeries CD-ROM drive. The CD-ROM you use depends on your operating system version:
   - For V5R1, the Group PTF number is **SF99165**.
   - For V5R2, the Group PTF number is **SF99166**.
2. From the OS/400 command line, run the Program Temporary Fix (GO PTF) command.
3. In the Program Temporary Fix display, specify option 8 (Install program temporary fix package).
4. In the Install Options for Program Temporary Fixes display, specify these values and press **Enter**:
   - **Device**: OPT01 (or the name of your CD-ROM drive)
   - **Automatic IPL**: N
   - **PTF type**: 1 (All PTFs)
   - **Other options**: Y

   Note: You do not need to perform an initial program load (IPL) to apply these fixes.
5. In the Other Install Options display, specify these values and press **Enter**:
   - **Omit PTFs**: N
   - **Apply Type**: 3 (Apply only immediate PTFs)
6. To verify the correct group PTF level, follow the instructions on the iSeries Connect Support Information Web page.
7. To verify the status of the fixes, run the Display PTF Status (DSPPTF) command.

Perform post-installation steps

This topic describes steps you need to perform to complete your installation.

1. To add the MQSeries Support Pac libraries (QMQMJAVA and QMQMAMI) to the user library list, follow these steps:
   a. From the iSeries command line, run this command:
      ```
      WRKSYSVAL QUSRLIBL
      ```
   b. In the Work with System Values display, select option 2 (Change).
   c. In the Change System Value display, add QMQMJAVA and QMQMAMI in separate Library fields. You can add the libraries anywhere within the list. Press **Enter**.

2. If your system runs with a coded character set identifier (CCSID) value of 65535, see “Change the coded character set identifier (CCSID)”.
3. The installation wizard normally starts all required subsystems and jobs. However, to ensure that iSeries Connect is properly started, you can manually start them. If the subsystems and jobs are already started, the system ignores these commands. See “Start and stop iSeries Connect” on page 103 for a list of commands.
4. If you are using WebSphere Commerce Suite (WCS), check the WCS support page for fixes and follow any special instructions listed there.

   - For WCS Pro Edition, see WCS Pro Edition Support
   - For WCS Business Edition, see WCS Business Edition Support
5. Check your iSeries system QUTCOFFSET value. Run this command from the iSeries command line:

   ```
   WRKSYSVAL SYSVAL(QUTCOFFSET)
   ```

   The value depends on your time zone offset from Greenwich Mean Time (GMT). It is important to set this properly so that integrated file system files have proper time stamps.

   To change the value, select option 2 (Change) in the Work with System Values display.

Change the coded character set identifier (CCSID)

Because of database support limitations, it is necessary to run the IBM HTTP Server administrative instance and the iSeries Connect flow manager with a coded character set identifier (CCSID) other than 65535 so that data is tagged appropriately. 65535 is the default value that instructs the operating system not to perform data conversions, which causes problems when iSeries Connect tries to access data from a database table because it cannot properly convert the data.

You can either change the CCSID setting on the system level, or manually change the necessary CCSID values.

Change the system-wide CCSID value

Follow these steps to change the default CCSID value for your system:

1. On the iSeries command line, run this command:

   ```
   WRKSYSVAL SYSVAL(QCCSID)
   ```
2. In the Work with System Values display, select option 2 (Change) to change the CCSID to a value other than 65535. Specify an EBCDIC CCSID value. For English systems, specify a value of 37. For non-English systems, see “Coded character set identifier (CCSID) values for EBCDIC” on page 37 for a list appropriate values.

Manually change the necessary CCSID values

If you do not want to change the system-wide CCSID setting, you can instead specify the -fsccsid parameter when you start the HTTP Server administrative instance. Specify an EBCDIC CCSID value other than 65535 (such as 37 for English systems). For a list of non-English EBCDIC CCSID values, see “Coded character set identifier (CCSID) values for EBCDIC” on page 37.

For example, if you system runs in an English language environment, use this command:

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN '-fsccsid 37')
```

Additionally, the user profile under which your instance runs must not use CCSID 65535. The user profile name is the same as the instance name. For example, if your instance is named MYINST, change the CCSID value for the MYINST user profile.

To change the CCSID value for a user profile, use the Change User Profile (CHGUSRPRF) command with the Coded character set ID (CCSID) parameter. Specify an EBCDIC CCSID value other than 65535. For a list of non-English EBCDIC CCSID values, see “Coded character set identifier (CCSID) values for EBCDIC”.

For example, to change the MYINST user profile to use a CCSID of 37, run this command from the iSeries command line:

```
CHGUSRPRF USRPRF(MYINST) CCSID(37)
```

Coded character set identifier (CCSID) values for EBCDIC

This list contains EBCDIC coded character set identifier (CCSID) values by language:

- Simplified Chinese: 00935
- Traditional Chinese: 00937
- Swiss German: 00500
- German: 00273
- United States English: 00037
- Spanish: 00284
- French: 00297
- Belgian French: 00500
- Canadian French: 00500
- Swiss French: 00500
- Italian: 00280
- Swiss Italian: 00500
- Korean: 00933
- Belgian Dutch: 00500
- Dutch: 00037
- Brazilian Portuguese: 00037
- Portuguese: 00037
- Japanese Katakana: 05035

Uninstall iSeries Connect

Follow these steps to uninstall iSeries Connect:

1. Stop all of your instances. Use the iSeries Connect configuration tool to do this.

   Note: If you are uninstalling iSeries Connect and do not plan to re-install it, use the iSeries Connect configuration tool to delete all of your instances. This facilitates in removing all objects that were created as part of your instances.

2. End the MQSeries queue manager. When you do this, you remove all the queues under this queue manager.

   Note: If the queue manager was created under a user profile that is different than the one you are using to end it, you may need to give yourself authority to the queue manager. The queue manager name is QIBM.ICONNECT.*unique_name*.QUEUE.MANAGER, where *unique_name* is a unique name for the queue manager.

   To end the MQSeries queue manager, follow these steps:

   a. On the iSeries command line, run the Work with Queues (WRKMQM) command.
   b. In the Work with Queues display, locate the QIBM.ICONNECT.*unique_name*.QUEUE.MANAGER queue. Specify option 18 (Work with Queues) for that queue.

c. Clear the queues for each instance. In the Work with MQM Queues display, specify option 13 (Clear) for all queues names that start with `QIBM.ICONNECT.`*`instance_name`*, where *instance_name* is the name of your instance. Press **Enter**.

d. End the queue manager. In the Work with Queues display, specify option 15 (End) and press **F4** to prompt the option. In the End Message Queue Manager display, specify *IMMED in the **Option** field.

e. After the queue manager has ended, delete the queue manager by specifying option 4 (Delete).

f. End the MQSeries subsystem. Run this command from the iSeries command line:

```
ENDSBS QMQM
```

3. To delete the iSeries Connect product, run this command from the iSeries command line:

```
DLTLICPGM LICPGM(5733CO2)
```

Chapter 5. Configure iSeries Connect

Configure iSeries Connect with the Web-based configuration tool. Use the tool to set up iSeries Connect to handle requests.

> **Note:** This topic provides an overview for using the configuration tool. Concepts and fields of the tool are documented in the configuration tool online help text. To display the help text for a particular page in the configuration tool, click the help icon ? at the top of the page.

See these topics for information about configuring iSeries Connect with the configuration tool:

“Start the iSeries Connect configuration tool”
Describes prerequisites for running the configuration tool and how to start it.

“Roadmap for configuring iSeries Connect” on page 40
Guides you through creating and configuring an iSeries Connect instance.

“Configure WebSphere Commerce Suite” on page 70
If you are using WebSphere Commerce Suite (WCS) for remote catalog support, see this topic for steps to configure your WCS instance and your iSeries Connect instance.

“Run the PCML verification sample” on page 85
Shows you how to configure an iSeries Connect instance from start to finish. The PCML verification sample ships with the Connect for iSeries product. Running the sample is a good place to start for learning how to configure iSeries Connect.

Start the iSeries Connect configuration tool

Follow these steps to access the configuration tool:

1. Make sure you meet these prerequisites for running the configuration tool:
   - If you have not done so, install iSeries Connect and all the prerequisite software on your iSeries system. See Chapter 4, “Install iSeries Connect” on page 29 for more information.
   - Start the administrative server of the HTTP Server. Enter this command on the OS/400 command line:
     ```
     STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
     ```
   - Because of database support limitations, do not run the IBM HTTP Server administrative server instance with a CCSID value of 65535. For more information, including instructions for changing CCSID values, see “Change the coded character set identifier (CCSID)” on page 36.
   - You access the configuration tool from a browser on your workstation. Make sure your browser meets the minimum requirements:
     – Microsoft Internet Explorer 5.x with JavaScript and cookies enabled
   - The user profile you use to access the tool must have these authorities:
     – *ALLOBJ special authority
     – *IOSYSCFG special authority
     – *JOBCTL special authority
     – *SECADM special authority
     – *USER user class
2. Enter the following URL in your browser: `http://`*`system_name`*`:2001/Connect/Commerce`

   Enter your iSeries user profile name and password when prompted.

Roadmap for configuring iSeries Connect

To configure iSeries Connect with the configuration tool, follow these steps:

1. "Start the iSeries Connect configuration tool" on page 39.
   The configuration tool is a Web-based configuration utility. See this topic for prerequisites for running the tool and how to start it.
2. "Create an instance".
   An *instance* is an entity that has customized configuration data associated with it. You can create multiple instances of iSeries Connect.

   Note: After you create your instance, make sure it is selected in the Manage Connect Instances page before you continue with the next steps.
3. "Create a provider" on page 41.
   A *provider* is an organization that sells products or services to buying organizations. Generally, the provider is the organization that is receiving requests from the marketplace. Use the iSeries Connect configuration tool to register a provider and associate that provider with a particular marketplace.
4. "Register partner organizations" on page 42.
   Partners are the provider's customers. Register your partners with iSeries Connect so it can determine authorized partners and retrieve information about the partners when a request is processed.
5. "Create a catalog" on page 42.
   Create a catalog to provide marketplace partners with a list of services and goods to purchase. In the configuration tool, you can manually create a catalog or import one from sources such as Domino, WCS, and DB2. You then publish the catalog to your marketplace of choice.
6. "Configure an application connector" on page 44.
   An *application connector* connects your business application to iSeries Connect and enables your application to process B2B requests. You configure the application connector by creating an application connector document, which defines the inputs and outputs to your applications and any additional properties.
7. "Create business process flows" on page 69.
   Create a *process flow model* to define how a request is processed by iSeries Connect. For each request, you specify the application connector instance to use, define how request fields are mapped to your application, and define how your application output is mapped to response fields.
8. "Deploy your business process flows" on page 70.
   To complete the configuration, you must integrate (or *deploy*) the process flow models and application connector documents into your B2B instance's runtime environment.
9. "Start your instance" on page 70.
   Start your configured instance.
10. "Test your instance" on page 70.
    Run the Test Drive Connect utility to test your instance configuration.

Create an instance

The first step you perform when configuring iSeries Connect is to create an instance. An instance is a collection of software that works together to process traffic. It contains the customized configuration that supports your protocol and your business environment.

To create an instance, click the **Instances** tab in the iSeries Connect configuration tool. If you have not previously created any instances, the New Instance wizard automatically starts. If one or more instances are already defined, click **New Instance** to start the wizard. The wizard guides you through the steps you need to perform to configure a new instance.

Note: If you migrated from version 1.1 and have existing 1.1 instances, the first time you access the Instances tab in the configuration tool, you are prompted to either create a new instance or migrate your 1.1 instances.

After you create your instance, you can modify the instance properties (the properties you specified in the wizard and other properties, such as performance, classpath, and trace settings. To edit the instance properties, select your instance in the Manage Instances page and click **Properties**.

Before you start
Before you create an instance, you should have completed these tasks:

- Install iSeries Connect and validate the installation. See Chapter 4, “Install iSeries Connect” on page 29 for details.
- Determine your system topology. The delivery gateway component can reside on a system that is remote to the flow manager component. This configuration option allows you to protect the flow manager and your back-end business applications with a firewall. If you plan to integrate WebSphere Commerce Suite (WCS) with iSeries Connect, your WCS instance can reside on a system that is remote from the flow manager component. (For more information about integrating WCS, see “Configure WebSphere Commerce Suite” on page 70.)

  If you plan to use more than one system, you should have a valid user profile and password on each system. In a split-instance configuration, the user profile on the gateway system must have the same authorities as the user profile that you use to access the iSeries Connect configuration tool on the flow manager system.
- Determine to which marketplace you are connecting (if applicable), which protocols are required, and which requests that you want to support. For more information, see “Plan for handling requests with your business applications” on page 25 and “Plan for the marketplace” on page 26.

Create a provider

A provider is an organization that sells products or services to buying organizations. Registering a provider in your instance serves these purposes:

- Provides basic information about the provider that can be made available to the partner organizations through catalog management services. This includes information such as provider name, address information, and contact information.
- Associates the provider with one or more protocols and provides information about how the protocol identifies itself to the provider (access information), what functions (request and request types) are allowed, and what the preferred order methods (phone, fax, e-mail, Web address) are as specified in the catalog.
- Associates a provider with a WebSphere Commerce Suite (WCS) merchant if WCS is configured for the instance. This allows requests coming in for the provider, which the provider ID and domain identify, to be associated with a back-end WCS merchant.

To create a provider, use the iSeries Connect configuration tool. Click the **Providers** tab to start the provider management functions. This allows you to create providers and associate them with protocols that are known to your instance.

If your instance supports the cXML protocol, see “Tips for registering provider and partner organizations for the cXML protocol”.

You can register a provider exit program to integrate the iSeries Connect provider services with a different application. For more information, see “Exit programs” on page 42.

Tips for registering provider and partner organizations for the cXML protocol

If you are registering a provider or partner organization for association with a marketplace that uses the cXML protocol, here is specific information that you should understand:

- Both the provider and the partner organization are identified in the cXML protocol by an ID value (which is a DUNS number) and a domain (which is the DUNS domain). The DUNS number is a required input when you register a provider. When you make a provider to marketplace association, the ID and domain are automatically populated for you. When you register a partner organization, DUNS is optional, but highly recommended. If provided, the ID and domain are automatically populated with the DUNS number when you make a marketplace association for partner organization, otherwise you must manually provide the DUNs number and a domain value of DUNS.
- With cXML, when you select **Logon Information** when associating a marketplace for a partner organization, the authentication of the marketplace request to access the provider system occurs on the provider system. Therefore, you must provide the necessary authentication information. You must select that logon authentication is required by the marketplace protocol on the Available marketplaces dialog, and provide a logon ID, domain, and shared secret or password. For cXML, the logon ID and domain match the provider ID and domain. You just need to provide the shared secret or password that was determined when registering with marketplaces. Remember that the password is case sensitive.
- With cXML, you do not need to check **Logon Information** when associating a marketplace for a partner organization. Partner organizations are authenticated at the marketplace and not authenticated on the provider system. However, you need to identify each partner organization by ID and domain coming from that marketplace, because that information is necessary for verifying the authorization of a partner organization to provider services through a marketplace request.

Exit programs

In the Providers and Partners tabs of the configuration tool, you can register an exit program (click **Registered Exit Program**), which allows iSeries Connect to interact with back-end applications (such as WebSphere Commerce Suite). For example, you could register an exit program that changes WCS Shopper data when you update iSeries Connect partner data.

The exit program is a Java class that uses iSeries Connect application programming interfaces (APIs). For more information on these APIs, see the iSeries Connect Customization Guide .

Register partner organizations

Click **Partners** to start the partner organization management function. A partner organization uses a procurement system to buy products or services from providers.

Registering a partner organization serves these purposes:

- Provides basic information about the partner organization that can be made available to the providers through catalog management services. This includes information such as partner organization name, address information, and contact information.
- Associates the partner organization with a provider and protocol, and indicates access information about how the partner organization is identified in the protocol.
- Associates the partner organization by its partner organization ID and domain with a WebSphere Commerce Suite (WCS) shopper if WCS is associated with the instance. By making this association, you create a WCS shopper or customer with information that reflects the information that is provided when you register a partner organization.

If your instance supports the cXML protocol, see “Tips for registering provider and partner organizations for the cXML protocol” on page 41.

You can register a partner exit program to integrate the iSeries Connect partner organization services with a different application. For more information, see “Exit programs”.

Create a catalog

Use the catalog management function in the iSeries Connect configuration tool to to create or manage catalogs in the current instance. You then integrate your catalog with the B2B marketplace.

In the configuration tool, click the **Catalog** tab to access the catalog management functions.

Creating a catalog

The iSeries Catalog services provides these methods for creating a catalog:

- Import a catalog from one or more local or remote database tables.
- Import a catalog from a local Lotus Domino database. For more information, see “Set up Lotus Domino for importing catalogs”.
- Import a catalog from a WebSphere Commerce Suite (WCS).
- Import a catalog from a Catalog Interchange Format (CIF) version 2.1 or 3.0 file.
- Create a catalog manually.

Publishing a catalog

The iSeries Connect catalog services support these methods for publishing a catalog:

- **Local catalog**
  A local catalog resides on a server that is local to the partner (and remote from the provider). The catalog is published on the B2B marketplace Web site.
- **Remote catalog**
  A remote catalog resides on a server that is remote from the partner (and local to the provider). The catalog is hosted on the provider’s Web site. The URL for the provider’s Web site is published to the B2B marketplace. When partners want to view products from a remote catalog provider, they ″punch out″ from the marketplace to the provider’s site.
- **Detailed remote catalog**
  The detailed remote catalog is similar to the remote catalog in that a partner punches out to the provider Web site. However, in detailed remote catalog implementations, partners punch out of the marketplace to individual product listings on the provider’s Web site, not to the main entry point of the provider’s store.

Set up Lotus Domino for importing catalogs

To use Domino as a source for the catalog, follow these steps:

1. Install Domino 5.0.7 or later on the same iSeries machine on which iSeries Connect is running.
2. Enable Domino HTTP and DIIOP servers through iSeries Operations Navigator or by running this command on the iSeries command line:

   ```
   CHGDOMSVR SERVER(domino_server_name ) WEB(*HTTP *IIOP)
   ```

   where *domino_server_name* is the name of your Domino server.
3. The Domino HTTP server must be configured to be listening on the default port, port 80. The Domino server name must resolve (from the iSeries) to this HTTP server listening on port 80. For the name to resolve, it must be added either to the iSeries host table or added to the DNS.

   If port 80 is already in use by another application, another logical interface (IP address) must be defined for Domino HTTP server. Both the Domino server and the other application must be configured to bind specific in order for both to share port 80.

   Use the Domino Administrator client to set the binding information for Domino. Under Internet Protocols, locate HTTP. Enter the **Host name**, and set **Bind to host name** to `Enabled`.
4. Open the Domino Administrator client. Under **Internet Protocols**, locate **HTTP**. Set **Allow HTTP clients to browse databases** to `Yes`.
5. Use the Domino Adminstrator client to set the proper authorities. Under the Security tab, add your Domino user profile (that you are using to access the iSeries Connect configuration tool) to the **Access server** and **Run unrestricted Java/Javascript/COM** settings.

Configure an application connector

iSeries Connect uses components called application connectors to access back-end applications. Currently, iSeries Connect provides these application connectors:

- “Program call connector”
  Use the program call connector to call iSeries programs and service programs written in languages such as C, RPG, and COBOL.
- “Java connector” on page 45
  Use the Java connector to make method calls to Java programs.
- “JDBC connector” on page 46
  Use the JDBC connector to access local or remote databases with JDBC.
- “MQSeries queue connectors” on page 52
  Use the MQSeries queue connector to work with MQSeries-enabled applications. The MQSeries queue connector supports these interfaces: MQSeries Queueing Interface (MQI) and MQSeries Application Messaging Interface (AMI).
- “OS/400 data queue connector” on page 59
  Use the OS/400 data queue connector to access queue-enabled applications.

In the iSeries Connect configuration tool, click the **Tools** tab. Click **Application Connectors** to configure your application connector instance.

Before you start

Before you configure your application connector instance, you should have completed these tasks:

- Select an application connector type that works best with your back-end application.
- If necessary, modify your back-end application to facilitate your chosen application connector. This may include writing an intermediary (or proxy) application.
- If it is required by your chosen application connector, create an input file and optionally, an output file. See information about your application connector for more information.

Program call connector

The program call connector starts iSeries programs or service programs with Program Call Markup Language (PCML). For more information about PCML, see Program Call Markup Language (PCML) in the iSeries Information Center.

To create an instance of the program call connector (that calls your specific application), first create a PCML document to describe the interface to your application. You can use any method (such as using a text editor) to create a PCML document.

> **Note:** iSeries Connect does not create PCML documents for you. iSeries Connect assumes that you have already created the PCML documents that you need.

Use the iSeries Connect configuration tool (click **Application Connectors** under the Tools tab) to create and update instances of the program call connector.

This table describes the properties that are used by the program call connector to use an iSeries program in response to a B2B request.

Property name	Description	Default	Set by
Host system (system)	iSeries server hosting the program to be used for this connector instance.	`localhost`	Application connector editor in the configuration tool

Property name	Description	Default	Set by
PCML document (pcmldocument)	File that contains the PCML definition of the program or service program and its interface. The PCML document is used to gather the information about input and output fields for the application connector. If the PCML document does not reside in the default directory (/QIBM/UserData/Connect200/Commerce/*instance_name*/Connector) the non-default must have a corresponding entry in the flow manager classpath. (You can update the classpath from your instance's properties page.)		Application connector editor in the configuration tool
Description	A description of the connector.		Application connector editor in the configuration tool
User ID (userid)	User ID used when the iSeries Connect program runs.		Deployment function of the configuration tool
Password (password)	Password associated with user ID.		Deployment function of the configuration tool

For information about tracing a PCML connector, see "Diagnose and solve iSeries Connect problems using trace" on page 108.

Java connector

The implementation of the Java connector differs from the other iSeries Connect connector types. Other connector types drive the mappings for applications and then use the flow manager application with a fully-resolved parameter list. This design is successful for those connector types because they provide a consistent, generic way to describe the application interface through an XML-based grammar.

Instead, the Java connector allows you to write the application logic in Java (as opposed existing RPG or C applications), which provides much more flexibility. The other connector types require you to describe structure with Program Call Markup Language (PCML), Extensible Markup Language (XML), or XML document type definition (DTD). The Java connector can directly access all of the mapping information. For the Java connector type, this access technique is necessary because Java parameters can be objects. Objects are more complex and there is no consistent, generic way to build all of the diverse parameter types for all user-defined objects.

The Java connector application is a Java class that implements the JavaConnectorInterface The flow manager Java connector code calls the run() method of the JavaConnectorInterface. The Java connector application provides the implementation of the run() method. In the run() method, you can access the input fields passed in the ConnectorParm object. It can then call any other Java classes that are necessary to perform the processing for this request. The ConnectorParm object can then be used to pass back any output fields to be mapped to the response message. The run() method has to be written with very specific knowledge of the interface to the targeted flow manager application.

The Java connector utilizes an interface class to describe the interaction between the Java connector and a Java method. The interface class definition is as follows:

```
package com.ibm.connect.flowmanager.interfaces;
 import com.ibm.connect.flowmanager.connector.ConnectorResult; import
 com.ibm.connect.flowmanager.interfaces.ConnectorParm;
```

```
/**
* JavaConnectorInterface is the interface that must be implemented by connector classes written in Java.
*
* The JavaConnectorInterface interface allows users to write Java code that can be executed in the Java Connector.
*
*/ public interface JavaConnectorInterface {
/**
* execute the user Java code..
* @param parameters The ConnectorParm object that can be used to extract any mapped parms.
* @returns A JavaConnectorResult object that allows a non zero return code to indicate an error and an error string
* to describe the error.
*/ public JavaConnectorResult run (ConnectorParm parameters);}
```

When you compile the Java connector application, you need to include /QIBM/ProdData/Connect200/Classes/flowmanagerapi.jar in your classpath.

As the flow manager processes requests, it needs to access your Java connector application class and any other classes that your class references. To make your classes available to the flow manager, update the Java classpath that the flow manager uses to include your classes. Use the iSeries Connect configuration tool to update the flow manager Java virtual machine classpath (under the flow manager tab of the Instance Properties page).

For more specific information, see the iSeries Connect Javadoc index.

The Java connector has properties to define the user-written Java connector application class, and an optional properties file that is made available to that class.

This table describes the properties that are used by the Java connector to use a method on a Java class in response to a B2B request.

Property name	Description	Default	Set by
Class name (classname)	Name of the class that the Java connector uses.		Application connector editor in the configuration tool
Property file name	File that defines the set of properties used by the Java class when called by the Java connector.	blank	Application connector editor in the configuration tool

For more detailed information about developing a Java connector application, see the Connect for iSeries Developer Resources Web site.

JDBC connector

The JDBC connector allows connector instances to access a local or remote database with JDBC.

When you create a JDBC connector in the application connector editor of the iSeries Connect configuration tool (under the Tools tab), you specify information that is used to connect to the database (such as JDBC driver, JDBC URL, user ID, and password).

To access a local database, it is recommended that you use the iSeries IBM Developer Kit for Java JDBC driver (com.ibm.db2.jdbc.app.DB2Driver). To access a remote iSeries database, the IBM Toolbox for Java JDBC driver is recommended (com.ibm.as400.access.AS400JDBCDriver).

You provide an SQL statement that is executed when the process flow is invoked for a request. Input fields are mapped as the parameter markers in the SQL statement. Any output parameter markers or result set data may also be mapped.

Input and output fields

Input fields represent parameter markers and are designated with ?. Output fields include any data that is generated by the SQL statement, such as fields returned in a result set or output fields from a stored procedure call.

In the application connector editor, you also specify input and output fields for the connector with a SQL statement or an “Application connector document field set (ACDFieldSet)” on page 68.

If you are accessing an iSeries DB2 database with either the iSeries IBM Developer Kit for Java JDBC driver or the IBM Toolbox for Java JDBC driver, you can use an SQL statement to define input and output fields. When the connector instance is created, the database is accessed to gather information about input and output fields for the SQL statement.

> **Note:** The application connector editor generates parameter names in the format of ″Parm1″, ″Parm2″, and so on. It is recommended that you rename these fields (using the editor) to more meaningful names. The meaningful names make it easier to map fields later.

You can also use an ACDFieldSet document to define input and output fields. If you are accessing a remote non-iSeries database, use an ACDFieldSet document to define input and output fields.

Input fields for the JDBC connector are grouped under a structure, typically called InputParms. The fields within the InputParms structure correspond to the parameter markers or the input parameters (IN or INOUT) for a stored procedure. If the mulit-statement execution property is set to Yes or the InputParms structure was defined as repeating in the ACDFieldSet, the SQL statement provided is executed for each set of repeating input provided. That is, the input parameters are set for the first element of each of the fields in the InputParms structure and the SQL statement is run. Then, the next set of field values are set and it is again run. And so on.

Fields under the InputParms structure can be mapped from multiple places and structures, but only one of the associated structures can be repeating, and this is to what the InputParms structure should be mapped. For example, some of the input parameters may be mapped from the request header and others from fields within a repeating structure in the request. In this case, the InputParms should be mapped to the repeating structure.

The JDBC connector has three possible output fields:

- **Row**
  A Row structure is returned for a query. The items in the result set are set into the fields defined under the Row structure. Generally, the Row structure is repeating, since a query can return multiple records. If the multi-row result set property is set to No for an input type of SQL or the Row structure is defined as non-repeating in an ACDFieldSet, the JDBC connector signals an exception if multiple rows are returned from JDBC. If the InputParms structure is repeating, that Row is an accumulation of the result set from each execution of the query.
- **RowCount**
  A RowCount in an integer that is the number of rows affected by an INSERT, UPDATE, or DELETE statement or the number of rows returned from a SELECT statement. If the InputParms structure is repeating, RowCount is an accumulation of the row count values for each execution of the SQL statement.
- **OutputParms**
  The OutputParms structure contains field definitions for the output parameters markers (OUT or INOUT) from a stored procedure call. If the multi-statement execution property is set to Yes for an input type of SQL or the OutputParms structure is set to repeating in the ACDFieldSet, the OutputParms structure will contain results from each invocation of the stored procedure call. When an ACDFieldSet is specified, the repeating attribute of the InputParms and OutputParms structures must match.

ACDFieldSet and the JDBC connector

If you are creating your own ACDFieldSet document, you should follow these guidelines. (For an input type of SQL, this information is automatically generated.)

The location values in the Field definition are numeric values for the JDBC connector. The names of the pre-defined fields may vary, but the location values for the pre-defined fields must be as follows:

- `-1` for RowCount
- `-2` for Row
- `-3` for InputParms
- `-4` for OutputParms

The location values for the InputParms and OutputParms structure are the location of the parameter marker (`?`) in the SQL statement. That is, the first parameter marker would have a location of `1`, the second would have a location of `2` and so on. The location value of the fields within the Row structure should be the location of the field in the result set. For example, if you have a query that returns two columns, the location of the first column is `1` and the location of the second column is `2`.

If InputParms, Row, or OutputParms structures are repeating, a mapping must be provided for them when creating the process flow. If they are non-repeating, a mapping is optional.

The field definitions within the InputParms, OutputParms, and Row structures must also contain the SubType attribute. The SubType should be the defined as the SQL type for the parameter marker or result set field. For example, the subtype for a field in the Row structure may be `VarChar` if the underlying database column is defined as a VARCHAR. Also, in a manually created ACDFieldSet for a query, the RowCount should be listed as the CountField for the Row structure.

Supported data subtypes

When you specify SQL as the Input parameter for the JDBC connector, the connector generates a data type based on the SQL data type. Also, when you create an ACDFieldSet document to define input fields, the connector data type is what you should specify for the Type attribute on the Field definition.

The valid SubType attributes are listed in the table below. They are case-sensitive. Match the data type to the type that the JDBC connector expects.

Here is a list of SQL data types and the corresponding type that is generated by the JDBC connector:

SQL type	Connector type
Char VarChar LongVarChar	String
CLOB	String
Graphic VarGraphic LongVarGraphic	String
DBCLOB	String
Numeric Decimal	BigDecimal
SmallInt	short
Integer	int
BigInt	long

SQL type	Connector type
Real	float
Float	double
Double	double
Binary VarBinary LongBinary	byte[]
BLOB	byte[]
Date [1] (See 49)	long
Time [1] (See 49)	long
Timestamp [1] (See 49)	long
Bit	Boolean
TinyInt	byte

[1] Date, time, and timestamp fields are treated as long values in iSeries Connect. The long value is the number of milleseconds since January 1, 1970 00:00:00.000 GMT. There are operators available to convert the long values to and from String values in JDBC escape format. For Date, this format is ″yyyy-mm-dd″ format. For Time, the format is ″hh:mm:ss″ format. For Timestamp, the format is ″yyyy-mm-dd hh:mm:ss.fffffffff″ format (where fffffffff represents nanoseconds).

Limitations

At this time, use of the JDBC connector is restricted by these limitations:

- You can specify only one SQL statement for a JDBC connector instance.
- The JDBC connector supports only these SQL statement types: SELECT, INSERT, UPDATE, DELETE, and CALL.
- The JDBC connector does not support result sets from stored procedures. If you need this function, create another connector type (such as a Java connector) that uses JDBC directly to run the stored procedure and manipulate the result sets.
- Binary fields can only be mapped to other binary fields, and are therefore only available between intermediate steps in the process flow model. This is an overall limitation and does not only apply to the JDBC connector.
- Individual input and output fields cannot be repeatable. The JDBC connector does support the set of InputParms as repeating and will iterate through the array, running the SQL statement that is provided on each individual ″row″ in the array.
- Do not run the JDBC connector (and database operations in general) under a job that uses a CCSID of 65535. This specifies that data conversions are not to be performed, which can cause critical problems in the iSeries Connect environment. If you have not changed the system-wide CCSID value to a value other than 65535 (as described in “Change the coded character set identifier (CCSID)” on page 36, change the CCSID value of the job under which database operations are being performed.

JDBC connector properties

For information about properties that the JDBC connector uses, see “JDBC connector properties” on page 50.

Sample ACDFieldSet document for the JDBC connector

This ACDField set document applies to an SQL statement that queries a customer ID from the DUNS number in the B2B request:

```
SELECT CUSTOMER_ID FROM collection.tablename WHERE BUYER_ID = ?
```

```
<?xml version="1.0"?>
<acdfieldset version="2.0" >
  <input >
    <field display="yes" location="-3" name="InputParms" repeating="no" type="struct" >
      <children >
        <field display="yes" location="1" name="BUYER_ID" repeating="no" type="string" >
          <length >9</length>
          <subtype >char</subtype>
        </field>
      </children>
    </field>
  </input>
  <output >
    <field display="yes" location="-1" name="RowCount" repeating="no" type="int" >
        <length >4</length>
    </field>
    <field display="yes" location="-2" name="Row" repeating="yes" type="struct" >
      <countfield >RowCount</countfield>
      <children >
        <field display="yes" location="1" name="CUSTOMER_ID" repeating="no" type="string" >
          <length >10</length>
          <subtype >char</subtype>
        </field>
      </children>
    </field>
  </output>
</acdfieldset>
```

JDBC connector properties: When you create a JDBC connector instance in the application connector editor of the iSeries Connect configuration tool (under the Tools tab), here are properties you specify:

Property name	Description	Required or optional	Default	Set by
JDBC driver	Specifies the JDBC driver to use. For more information see the documentation for the JDBC driver.	Required	com.ibm.db2.jdbc.app.DB2Driver (IBM Developer Kit for Java JDBC driver)	Application connector editor in the configuration tool
JDBC URL	Specifies the URL of the JDBC driver. For more information see the documentation for the JDBC driver.	Required	jdbc:db2:localhost	Application connector editor in the configuration tool
Input type	Specifies either SQL or ACDFieldSet, the type of input that is used to derive input and output fields for the application connector. The SQL input type is only allowed when either the iSeries IBM Developer Kit for Java JDBC driver or the IBM Toolbox for Java JDBC driver is specified. When an input type of SQL is specified, the input and output field definitions are gathered from the SQL statement and the database. For ACDFieldSet, you manually specify all of the field information in an ACDFieldSet document.	Required	SQL	Application connector editor in the configuration tool

Property name	Description	Required or optional	Default	Set by
Transaction isolation level	Specifies the commitment control level for the database transaction.	Required	None	Application connector editor in the configuration tool
SQL statement	Specifies an SQL statement to use to execute for this JDBC connector instance. The SQL statement is also used to derive input and output field for the application connector.	Required		Application connector editor in the configuration tool
Multi statement execution	Specifies to support running supplied SQL statement multiple times and to support a repeating set of input and output parameters.	Required	No	Application connector editor in the configuration tool
Multi row result set	By default, the database can return multiple rows from an SQL SELECT statement. If no is specified and multiple rows are returned from the database, the JDBC connector treats this as an error. This value should only be set to no when it is guarunteed that only one row should match the selection criterea. You cannot specify this field if the Multi statement execution property is set to Yes.	Required	Yes	Application connector editor in the configuration tool
ACD field set	Path name of the ACDFieldSet document to be used for the message if Message Type is ACD field set.	Required when input type is ACDFieldSet. (SQL input type automatically generates the equivalent field information.)		Application connector editor in the configuration tool
Admin Logon	Specifies a user ID for accessing the configuration tool. The user ID is used to gather field information when input type is SQL.	Required when input type is SQL.		Application connector editor in the configuration tool
Admin Password	Password for the Logon ID	Required when input type is SQL.		Application connector editor in the configuration tool
Admin Port	Specifies the port number on which the iSeries configuration tool runs.	Required when input type is SQL.	2001	Application connector editor in the configuration tool

Property name	Description	Required or optional	Default	Set by
Database user	Specifies the user ID that is used to connect to the database to gather the information about input and output fields when the input type is SQL. The user ID must have the authority to run the SQL statement provided. You are prompted again to provide the database user ID and password when you deploy your process flow. The values gathered at deployment are saved to be used at runtime to connect to the database with JDBC.	Required		Application connector editor gathers it when input type is SQL, but it is stored during deployment.
Database password	Password for the userID	Required		Application connector editor gathers it when input type is SQL, but it is stored during deployment.

MQSeries queue connectors

There are two forms of connectors for accessing an application through MQSeries: the MQSeries queue connector and the MQSeries Application Messaging Interface (AMI) connector.

Each of these queuing methods can use one of these message definition techniques, either Program Call Markup Language (PCML), Extensible Markup Language (XML), Comma Separated Variables (CSV), or Application Connector Document Field Set (ACDFieldSet). The MQSeries queue connectors transform B2B requests into messages of these formats and place them on the queue. Therefore, the properties define a send message transformation, send queue, receive queue, and receive message transformation. Use the application connector editor in the iSeries Connect configuration tool (under the Tools tab) to set the properties for the MQSeries queue connector.

An IBM Business Partner or customer that wants to create an instance of the MQSeries queue connector to pass a message to their application has to define queue identification information and has a choice of specifying the message format as an XML document, a CSV stream, or a structured buffer described by PCML or an ACDFieldSet document.

When you use the application connector editor in the iSeries Connect configuration tool (under the Tools tab), you specify the name and location of your document.

Selecting a queue connector type

When selecting between MQSeries queues or MQSeries AMI queues, both connectors send and receive messages to MQSeries queues. However, they each offer different methods of configuring the queueing interface. MQSeries queue directly configures queue usage information. MQSeries AMI indirectly configures queue usage through a policy.

MQSeries queue usage requires that the connector is configured with MQSeries information that defines the queue manager, queue names, and message options directly in the connector properties. If the MQSeries configuration changes, the connector must be redeployed with matching configuration changes.

MQSeries AMI allows a user-defined AMI repository to be used, which can insulate the queue connector from MQSeries configuration information. The AMI policy allows the connector to be decoupled from a specific MQSeries configuration. The information in the policy can be changed without requiring any connector changes. The AMI policy also offers some additional control over how the connector uses the MQSeries. An MQSeries configuration tool for Windows NT is available for creating and changing AMI repositories. The tool is available as an MQSeries product extension MA0G at

http://www.software.ibm.com/ts/mqseries/txppacs/ma0g.html.

You can use MQSeries queue connectors in these modes:

- **Send queue only**
  Sends a message to a queue that is identified in the connector properties. No response message is received. This mode is only appropriate for flows that do not require any response data, such as a cXML PunchOut message.
- **Receive queue only**
  Allows send and receive to be performed in separate steps of a process flow. For this option, mapping of message and correlation IDs to intermediate fields may be useful.
- **Send and receive queues**
  Generates a message to send to the queue that is identified in the connector properties and waits for a response message on another queue that is identified in the connector properties. The flow manager is multithreaded, so it is possible for multiple threads to wait on a common receive queue. This requires the responses to be received with a queueing method that can select a specific response. Response messages to MQSeries queues should use message-id or correlation-id matching through appropriate queue connector properties. The simplest method for keys is to use the GENERATE value for these properties.

Queue connector properties

For information about properties that the queue connectors use:

- “MQSeries queue connector properties”
  To use MQSeries, you must grant the instance profile MQI authority to the queue manager and queues.
- “MQSeries AMI connector properties” on page 56
  To use MQSeries AMI, you must grant the instance profile MQI authority to the queue manager and queues.

MQSeries queue connector properties: This table describes the properties that are used by the MQSeries queue connector to call an application through an MQSeries queue.

Property name	Description	Required or optional	Default	Set by
MQSeries Manager	Name of the MQSeries message queue manager.	Required		Application connector editor in the configuration tool.
MQSeries host name	Name of the system on which the MQSeries manager runs.	Required	`localhost`	Application connector editor in the configuration tool.
MQSeries server port	Port number with which to access the MQSeries manager.	Optional	If not defined, a default of 1414 is used.	Application connector editor in the configuration tool.

Property name	**Description**	**Required or optional**	**Default**	**Set by**
MQSeries server channel	Specifies a channel for communications between two queue managers.	Optional	For MQSeries 5.1, if this attribute is not defined, bindings mode is used. Otherwise, the default is SYSTEM.DEF.SVRCONN.	Application connector editor in the configuration tool.
Message Type	Specifies the type of message to send or receive. Valid values are PCML, XML, CSV, and ACDFieldSet [1]	Required		Application connector editor in the configuration tool.
Tracing	Specifies whether to turn on MQSeries client for Java trace facility. Valid values are ENABLED and DISABLED.	Required	DISABLED	Application connector editor in the configuration tool.
Trace File	Specifies the trace file name and directory path, if tracing is enabled. For example, /home/trace/mqb2b.trc.	Optional	/QIBM/UserData/mqm/trace/B2BMQ.trc	Application connector editor in the configuration tool.
Send Queue	Name of the MQSeries queue to send the request message.	Required		Application connector editor in the configuration tool.
Send Message Persistence	Specifies the MQSeries persistence of the message. Valid values are YES and NO.	Required	Persistence will be as the Send Queue is defined	Application connector editor in the configuration tool.
Send Message Priority	Specifies the MQSeries message priority. Valid values are 0 through 9.	Required	Priority will be as the SendQueue is defined	Application connector editor in the configuration tool.
Send Message Id	Specifies the MQSeries message identifier to use when sending the message. Valid values are NONE, GENERATED, and MAPPED [2]	Required	NONE	Application connector editor in the configuration tool.
Send Message Correlation Id	Specifies the MQSeries correlation identifier to use when sending the message. Valid values are NONE, GENERATED, and MAPPED [2]	Required	NONE	Application connector editor in the configuration tool.

Property name	Description	Required or optional	Default	Set by
Receive Queue	Name of the MQSeries queue to receive the response message.	Required for receiving a response message. Empty property means send only.		Application connector editor in the configuration tool.
Receive queue name prefix	Prefix name of the MQSeries queue to receive the response message.	Optional		Application connector editor in the configuration tool.
Receive Message Wait Time	Specifies the length of time, in milliseconds, to wait on the receive queue for a message. Valid values are *nnnnn* and UNLIMITED	Optional	300000 milliseconds (for example, 5 minutes)	Application connector editor in the configuration tool.
Receive message ID match	Specifies how a receiving application should set the message ID when sending a reply message.	Required	`None`	Application connector editor in the configuration tool.
Receive correlation ID match	Specifies how a receiving application should set the correlation ID when sending a reply message.	Required	`MessageIdSent`	Application connector editor in the configuration tool.
PCML Document	Name of the PCML document to be used for the message if message type is PCML or CSV.	Required for PCML message type		Application connector editor in the configuration tool.
PCML system	Name of the system on which the PCML document is located.	Required for PCML message type	`localhost`	Application connector editor in the configuration tool.
Input field template	Specifies the fully-qualified path name of an XML, DTD, or XSD document that is used to derive input fields for the application connector.	Required for XML message type		Application connector editor in the configuration tool.

Property name	Description	Required or optional	Default	Set by
Output field template	Specifies the fully-qualified path name of an XML, DTD, or XSD document that is used to derive input fields for the application connector.	Required for the XML message type		Application connector editor in the configuration tool.
ACD field set	Name of the ACDFieldSet document to be used for the message if message type is ACDFieldSet or CSV.	Required for ACDFieldSet message type.		Application connector editor in the configuration tool.
Default CCSID	Specifies the Coded Character Set ID (CCSID) to be used for conversion of property values and inputs when the CCSID is not otherwise specified.	Optional	If this attribute is omitted, the default CCSID of the host environment is used.	Application connector editor in the configuration tool.
iSeries System	System used for PCML documents or CSV text conversions.	Required	localhost	Application connector editor in the configuration tool.
iSeries Userid	Userid used for the iSeries system.	Required		Deployment function of the iSeries Connect configuration tool
iSeries Password	Password used for the iSeries system.	Required		Deployment function of the iSeries Connect configuration tool

[1] PCML is Program Call Markup Language. XML is Extensible Markup Language. CSV is Comma Separated Variables. ACDFieldSet is Application Connector Document Field Set. The PCML method of mapping IDs should be considered deprecated. Instead, they can be mapped to fixed fields.

[2] GENERATED has MQSeries generate a new value for each message. MAPPED extracts the value from the PCML document with the appropriate structure or data name. MAPPED is supported for message type XML only with fixed fields. For more information, see “Define messages for queue connectors” on page 62.

MQSeries AMI connector properties: This table describes the properties that are used by the MQSeries application messaging interface (AMI) connector to call an application through the MQSeries AMI interface.

Property name	Description	Required or optional	Default	Set by
Message Type	Specifies the type of message to send or receive. Valid values are PCML, XML, CSV, and ACDFieldSet [1]	Required		Application connector editor in the configuration tool.
Directory Path	Specifies the directory for the local host and repository files.	Optional	/QIBM/UserData/mqm/amt	Application connector editor in the configuration tool.
Repository File Name	Specifies the name for the AMI repository file.	Optional	amt.xml	Application connector editor in the configuration tool.
AMI Tracing	Specifies whether to turn on the AMI client for Java trace facility. Valid values are ENABLED and DISABLED	Optional	DISABLED	Application connector editor in the configuration tool.
Trace Level	Specifies the AMI trace level to use if AMI Tracing is enabled. Valid values are 0 through 9.	Optional	9	Application connector editor in the configuration tool.
Trace Location	Specifies the trace location directory, if tracing is enabled. The filenames are generated by AMI and will be AMTnnnnn.TRC.	Optional	/QIBM/UserData/mqm/amt/trace	Application connector editor in the configuration tool.
Policy Name	Specifies the name of the AMI AMPolicy to use for processing the messages.	Required		Application connector editor in the configuration tool.
Sender Name	Specifies the AMI AMSender name. This represents an MQSeries queue on a local queue manager.	Not required for receive only.		Application connector editor in the configuration tool.
Receive	Specifies if a AMIReceiver is used.	Optional	Yes	Application connector editor in the configuration tool.

Property name	Description	Required or optional	Default	Set by
Receiver Name	Specifies the AMI AMReceiver name. This represents an MQSeries queue on a local queue manager.	Required for receiving a response message. Empty property means send only.		Application connector editor in the configuration tool.
Correlation Id	Specifies the MQSeries correlation identifier to use when sending the message. Valid values are NONE, GENERATED, and MAPPED [2]	Optional	GENERATED	Application connector editor in the configuration tool.
Receive correlation ID match	Specifies how correlation ID matching should be done when receiving a reply message. Possible values are None, MessageIdSent, CorrelationIdSent, or Mapped.	Required	`MessageIdSent`	Application connector editor in the configuration tool.
PCML Document	Name of the PCML document to be used for the message if message type is PCML.	Required for PCML message type		Application connector editor in the configuration tool.
PCML system	Name of the system on which the PCML document is located.	Required for PCML message type	`localhost`	Application connector editor in the configuration tool.
Input field template	Specifies the fully-qualified path name of an XML, DTD, or XSD document that is used to derive input fields for the application connector.	Required for XML message type		Application connector editor in the configuration tool.
Output field template	Specifies the fully-qualified path name of an XML, DTD, or XSD document that is used to derive input fields for the application connector.	Required for the XML message type		Application connector editor in the configuration tool.

Property name	Description	Required or optional	Default	Set by
ACD field set	Name of the ACDFieldSet document to be used for the message if message type is ACDFieldSet or CSV.	Required for ACDFieldSet message type.		Application connector editor in the configuration tool.
Default CCSID	Specifies the Coded Character Set ID (CCSID) to be used for conversion of property values and inputs when the CCSID is not otherwise specified.	Optional	If this attribute is omitted, the default CCSID of the host environment is used.	Application connector editor in the configuration tool.
iSeries System	System used for PCML documents or CSV text conversions.	Required	`localhost`	Application connector editor in the configuration tool.
iSeries Userid	Userid used for the iSeries system.	Required		Deployment function of the iSeries Connect configuration tool
iSeries Password	Password used for the iSeries system.	Required		Deployment function of the iSeries Connect configuration tool

[1] PCML is Program Call Markup Language. XML is Extensible Markup Language. CSV is Comma Separated Variables.

[2] GENERATED has MQSeries generate a new value for each message. The PCML method of mapping IDs should be considered deprecated. Instead, they can be mapped to fixed fields. MAPPED is supported for message type XML only with fixed fields. For more information, see “Define messages for queue connectors” on page 62.

OS/400 data queue connector

The OS/400 data queue connector supports OS/400 data queues. This queuing method can use one of these message definition techniques, either Program Call Markup Language (PCML), Extensible Markup Language (XML), Comma Separated Variables (CSV), or Application Connector Field Set (ACDFieldSet). Therefore, the properties define a sending message transformation, sending queue, receiving queue, and received message transformation. The application connector editor in the iSeries Connect configuration tool (under the Tools tab) sets these properties.

An IBM Business Partner or customer that wants to create an instance of the OS/400 data queue connector to pass a message to their application has to define queue identification information and has a choice of specifying the message format as an XML document, a structured buffer with PCML, a CSV stream, or an ACDFieldSet document. To put a structured message on the queue, specify the input and output with PCML or ACDFieldSet by using the application Connector editor. To put XML on the queue, use XML to define the input and outputs. To put CSV on the queue, use PCML to specify the inputs and outputs.

You can use an OS/400 data queue connector in these modes:

- **Send queue only**
  Sends a message to a queue that is identified in the connector properties. No response message is received. This mode is only appropriate for flows that do not require any response data, such as a cXML PunchOut message.
- **Receive queue only**
  Allows send and receive to be performed in separate steps of a process flow. For this option, mapping of message and correlation IDs to intermediate fields may be useful.
- **Send and receive queues**
  Generates a message to send to the queue that is identified in the connector properties and waits for a response message on another queue that is identified in the connector properties. The flow manager is multithreaded, so it is possible for multiple threads to wait on a common receive queue. This requires the responses to be received with a queueing method that can select a specific response. Response messages to iSeries data queues should be keyed. The simplest method for keys is to use the GENERATE value for these properties.

Queue connector properties

For information about properties that the OS/400 data queue connector uses, see “OS/400 data queue connector properties”. To use OS/400 data queues, you must grant the instance user profile *USE authority to the library and queues.

OS/400 data queue connector properties: This table describes the properties that are used by the data queue connector to call an application through an iSeries data queue.

Property name	Description	Required or optional	Default	Set by
Message Type	Specifies the type of message to send or receive. Valid values are PCML, XML, CSV, and ACDFieldSet [1]	Required		Application connector editor in the configuration tool.
Keyed Send Queue	Specifies whether or not the send queue name is a keyed data queue. Valid values are YES and NO	Required for sending a message		Application connector editor in the configuration tool.
Send Queue Key	Specifies the method or the value of the key, if keyed send queue is set to YES. Valid values are GENERATED, MAPPED, or *xxxxxxx* [2] where *xxxxxxx* is your user-defined key value.	Required if Keyed Send Queue is set to YES		Application connector editor in the configuration tool.
Send Queue Name	Specifies the name of the iSeries data queue (*DTAQ) object to send the request message.	Required for sending a message		Application connector editor in the configuration tool.

Property name	Description	Required or optional	Default	Set by
Send Queue Library	Specifies the iSeries library name which contains the *DTAQ object.	Required for sending a message		Application connector editor in the configuration tool.
Keyed Receive Queue	Specifies whether or not the receive queue name is a keyed data queue. Valid values are YES and NO	Required for receiving a response message. Empty property means Receive only.	Yes	Application connector editor in the configuration tool.
Receive Queue Key	Specifies the method or the value of the key, if keyed receive queue is set to YES. Valid values are GENERATED, MAPPED, and *xxxxxxx* [2] where *xxxxxxx* is your user-defined key value. Read off the queue uses searchType of EQ for equal key compare.	Required if Keyed Receive Queue is set to YES		Application connector editor in the configuration tool.
Receive Queue Name	Specifies the name of the iSeries data queue (*DTAQ) object to receive the response message from.	Required for receiving a message		Application connector editor in the configuration tool.
Receive Queue Library	Specifies the iSeries library name that contains the *DTAQ object.	Required for receiving a message		Application connector editor in the configuration tool.
Receive Queue Wait Time	Specifies the length of time, in seconds, to wait on the receive queue for a message. Valid values are *nnnnn* and UNLIMITED	Optional	300 (5 minutes)	Application connector editor in the configuration tool.
Receive queue operator	Specifies the operator to use for comparing keyed messages to the receive key.	Required for keyed receive queue.	EQ (equals)	Application connector editor in the configuration tool.
Key padding character	Specifies the character with which to pad keys that are less than the specified key length of the queue.	Optional, however key values must be the appropriate length if this is not specified.		Application connector editor in the configuration tool.
PCML Document	Name of the PCML document to be used for the message if Message Type is PCML.	Required for PCML and CSV message types		Application connector editor in the configuration tool.

Property name	Description	Required or optional	Default	Set by
ACD field set	Path name of the ACDFieldSet document to be used for the message if Message Type is ACD field set.	Required for ACDFieldSet message type		Application connector editor in the configuration tool.
Send-Receive Queue System	Specifies the name of the iSeries system that contains the *DTAQ object.	Required	localhost	Application connector editor in the configuration tool.
Input field template	Specifies the fully-qualified path name of an XML, DTD, or XSD document that is used to derive input fields for the application connector.	Required for XML message type		Application connector editor in the configuration tool.
Output field template	Specifies the fully-qualified path name of an XML, DTD, or XSD document that is used to derive input fields for the application connector.	Required for the XML message type		Application connector editor in the configuration tool.
Default CCSID	Specifies the Coded Character Set ID (CCSID) to be used for conversion of property values and inputs when the CCSID is not otherwise specified.	Optional	If this attribute is omitted, the default CCSID of the host environment is used.	Application connector editor in the configuration tool.
Send-Receive Queue Userid	User ID that the program uses to connect to the *DTAQ object.	Required if Send-Receive Queue System is not local host or the name of the iSeries system		Deployment function of the iSeries Connect configuration tool.
Send-Receive Queue Password	Password associated with the Send-Receive Queue Userid.	Required if Send-Receive Queue System is not local host or the name of the iSeries system		Deployment function of the iSeries Connect configuration tool.

[1] PCML is Program Call Markup Language. XML is Extensible Markup Language. CSV is Comma Separated Variables.

[2] GENERATED generates a value by using the unique transaction value that the flow manager provides. The PCML method of mapping IDs should be considered deprecated. Instead, they can be mapped to fixed fields. MAPPED is supported for message type XML only with fixed fields. For more information, see "Define messages for queue connectors".

Define messages for queue connectors: There are a few things that you need to be aware of when defining messages for each of the queue connectors. Follow these guidelines for defining Extensible

Markup Language (XML) messages (See 63), Program Call Markup Language (PCML) messages (See 63), Comma Separated Variable (CSV) messages (See 64), and “Application connector document field set (ACDFieldSet)” on page 68. See the code examples that follow for additional information.

Each application connector defines a fixed set of input and output fields in addition to those that are derived from a PCML or XML document. These fixed fields are used to map additional information used to control the runtime behavior of a given connector instance. The additional information varies according to what connector type is used. For example, a MQSeries connector can use input fields to define the MQSeries message identifier or correlation identifier. iSeries data queue connectors use input fields to construct a message key. This technique allows each call of the connector to generate different values for these fields. In contrast, the properties for a connector instance provide constant values for each call of a connector.

Defining XML messages

You must define input and output fields for XML messages by using a sample XML document with the application connector editor in the iSeries Connect configuration tool (under the Tools tab). The pcmldocument property is not used when specifying XML message types. Incoming messages with DOCTYPE information specified are validated. Incoming messages without DOCTYPE information are not validated.

For document type information that is in the document used for input mapping, the following occurs:

- <!DOCTYPE docname publicId systemId /> element is generated in outgoing messages to the SendQueue. The ″docname″, ″publicId″, and ″systemId″ values are copied from the input mapping document.
- Incoming messages are validated, thus requiring the document DTD to be available as indicated in the DOCTYPE information. A simple DTD name, such as systemID `cXML.dtd`, requires the DTD file to be located in the flow manager classpath. The default flow manager classpath includes the /QIBM/UserData/Connect200/Commerce/*instance_name*/Connector directory. You can place DTD documents in this directory. iSeries Connect deployment does not automatically copy DTD files to this location.

For document type information that is not in the document used for input mapping, the following occurs:

- No <!DOCTYPE /> information is generated on outgoing messages.
- Incoming messages are not validated with a validating parser for document validity. Parsing the XML message is still performed in non-validating mode, which requires the XML document to be well formed.
- There is no requirement for a DTD to be available.

When using the application connector editor to create the input and output mappings these guidelines apply:

- Repeating elements in the source must be mapped to repeating elements in the target. Any repeated element which occurs in a source message creates repeating elements in the target.
- All elements in a target should be mapped to ensure that the correct XML is generated.
- The sample XML document must contain elements for each element that is to be mapped.
- Element attributes may be mapped to or from elements, such as a source element attribute can be mapped to a target element. For example, you can map <sourcetag srcattr=″xxx″ /> to <targettag>xxx</targettag>.

XML messages support MAPPED values for key, message ID, and correlation ID fields only through fixed fields.

Defining PCML messages

You must create the PCML document as described here to ensure correct operation of the queue connectors using PCML described messages:

- You must use the same PCML document for input and output fields.
- The PCML document must contain a <program> element with the same name as the document. For example, Message.pcml must contain a <program name=″Message″> element.
- The path attribute in the <program> element is not used and can have any value.
- The structure and data elements for the program must match the entries specified by and elements defined in the PCML document.

  The SendMessage structure is required to define the message to be sent and must have usage=″input″ attribute (for example, <struct name=″SendMessage″ usage=″input″ ...>). SendMessage must not have a countfield.

  The ReceiveMessage structure is required to define the message to be received and must be usage=″output″ (for example, <struct name=″ReceiveMessage″ usage=″output″...>). ReceiveMessage must not have a countfield.

  Other data elements listed in the table are deprecated and are overridden if the corresponding fixed fields are used.
- The structure and data elements that describe input and output can be in any order. The recommended order is to put the SendMessage structure first and ReceiveMessage structure second.
- Names of structure and data elements within any defined name structure are not restricted. Elements with count fields are allowed.
- The contents of the input fields are converted to a stream of bytes that are used as described for the element name. For example, a SendMessage structure is converted to a byte stream used as the message placed on a queue.
- The contents of the output fields are populated from the data generated by the queueing method and are then mapped to flow manager response fields. For example, a ReceiveMessage structure is populated from the byte stream of a message received from a queue.

Defining CSV messages

CSV messages implement messages as a series of characters that represent a field with commas between each field. The message characters are placed on the queue as converted to the CCSID specified in the properties, or the default CCSID of the host environment.

Input and output fields for CSV messages use a PCML document to describe how input and output fields are mapped to a CSV message. The document is only used for mapping. It is not used when running a connector. As a result, you can use a different PCML document for input and output field definition. The pcmldocument property is used to determine the set of input and output fields to be represented in CSV format, but it is not directly used at runtime. PCML documents must follow the rules described in **defining PCML messages** regarding the PCML document structure and names.

Here are additional restrictions on PCML documents that are used for CSV definition:

- All fields are inserted in the CSV message as strings in the CCSID specified by the CCSID property. If no CCSID is specified, the default CCSID of the host environment is used.
- Commas are inserted between CSV fields, where a field is a data element (for example, <data name=item...>). Structure elements do not create any input or output.
- Input data elements with types of int, double, and float are converted to the respective type and then back to a string to ensure the values are the correct type. Other types are mapped directly from input to output without conversion and put in quoted form..
- Repeating fields are allowed. Repeating fields have count fields associated with them that provide the application with the number of repeating fields at runtime. (Thus, defining a Countfield is required when you have repeating fields.) At runtime, the count for a repeating field is always placed immediately before the repeating data. For example, a <data name=″mychar″ type=″char″ countfield=″mycount″

usage=″input″/> when mapped to an input field with three instances ″a″,″b″, and ″c″ produce a CSV string 3,a,b,c. The location of the Countfield in the PCML document is ignored.

PCML structure or data names

The queue connectors use a PCML document to describe the message and optional parameters to the queue process. The document must contain a <program> element with a name attribute that matches the name of the PCML document and contains highest level structure or data elements that are described in the following table. Elements with other names are ignored.

Structure or data name	Usage	Connector
SendMessage	Input	MQSeries queue, MQSeries application messaging interface (AMI), and data queue
ReceiveMessage	Output	MQSeries queue, MQSeries AMI, and data queue

Defining ACDFieldSet messages

See “Application connector document field set (ACDFieldSet)” on page 68

Example: PCML document

This PCML Document (checkinvio.pcml) contains SendMessage and ReceiveMessage definitions. You can use this document for input and output fields with PCML or CSV messages and any queue connector type. Any field that is not mapped must have an init attribute for PCML messages.

```
<pcml version="1.0">

 <!-- PCML source for Inventory Check, document name must match program name -->
 <program name="checkinvio" path="not_used">

   <!-- INPUT struct defining the message to receive, name must be SendMessage-->
    <struct name="SendMessage" usage="input" >
      <data name="msglabel" type="char"length="8" usage="input" init="REQUEST " />
      <data name="incount" type="int" length="4" usage="input" />
      <data name="hdrdesc" type="char" length="64" usage="input" init="n/a" />
      <struct name="item"  usage="input" count="checkinvio.SendMessage.incount" >
        <data name="identifier" type="int" length="4" usage="input" />
        <data name="quantity" type="int" length="4" usage="input" />
        <data name="price" type="float" length="4" usage="input" />
        <data name="description" type="char" length="64" usage="input"/>
      </struct>
   </struct>

   <!-- OUTPUT struct defining message to receive, name must be ReceiveMessage-->
   <struct name="ReceiveMessage" usage="output" >
      <data name="msglabel" type="char"length="8" usage="input" init="RESPONSE" />
      <data name="outcount" type="int" length="4" usage="output" />
      <struct name="itemout" usage="output"
             count="checkinvio.ReceiveMessage.outcount" outputsize="1024" >
        <data name="identifier" type="int" length="4" usage="output" />
        <data name="quantity" type="int" length="4" usage="output" />
        <data name="price" type="float" length="4" usage="output" />
        <data name="status" type="char" length="64" usage="output" />
      </struct>
      <data name="ostatus" type="int" length="4" usage="output" />
    </struct>

  </program>

</pcml>
```

Example: MQSeries queue connector

This connector allows you map input values to send and receive values to match through PCML structure or data elements. In addition, the values that you use to send or receive can be mapped into your output document. This is an example of all independent possibilities. They can be specified in any combination and any order. Only the highest level structure or data names must match what is specified in the example.

```
<pcml version="1.0">

  <!-- PCML source for Inventory Check, document name must match program name -->
  <program name="checkinvio" path="not used">

   <!-- INPUT struct defining the message to receive, name must be SendMessage-->
   <struct name="SendMessage" usage="input" >
     <data name="incount" type="int" length="4" usage="input" />
     <data name="hdrdesc" type="char" length="64" usage="input" init="n/a" />
     <struct name="item"  usage="input" count="checkinvio.SendMessage.incount" >
       <data name="identifier" type="int" length="4" usage="input" />
      <data name="quantity" type="int" length="4" usage="input" />
       <data name="price" type="float" length="4" usage="input" />
       <data name="description" type="char" length="64" usage="input" init="n/a" />
     </struct>
     <data name="istatus" type="int" length="4" usage="input" init="0" />
   </struct>

   <!-- OUTPUT struct defining message to receive, name must be ReceiveMessage-->
   <struct name="ReceiveMessage" usage="output" >
     <data name="outcount" type="int" length="4" usage="output" />
     <data name="hdroutdsc" type="char" length="64" usage="output" />
     <struct name="itemout" usage="output" count="checkinvio.ReceiveMessage.outcount" outputsize="1024" >
       <data name="code" type="int" length="4" usage="output" />
       <data name="quant" type="int" length="4" usage="output" />
       <data name="outprice" type="float" length="4" usage="output" />
       <data name="status" type="char" length="64" usage="output" />
     </struct>
     <data name="msgoend" type="char" length="12" usage="output" />
     <data name="ostatus" type="int" length="4" usage="output" />
   </struct>
  </program>
</pcml>
```

Example: MQSeries AMI

QueueConnectorMQAMI allows mappings for the MQSeries AMI CorrelationId and MessageId. You can map input values to send and receive values to match. In addition, the values that you use to send or receive can be mapped into your output document. This an example of all independent possibilities. They can be specified in any combination and any order. Only the highest level structure or data names must match what is specified in the example.

```
<pcml version="1.0">

  <!-- PCML source for Inventory Check, document name must match program name -->
  <program name="checkinvio" path="not used">

   <!-- INPUT struct defining the message to receive, name must be SendMessage-->
   <struct name="SendMessage" usage="input" >
     <data name="incount" type="int" length="4" usage="input" />
     <data name="hdrdesc" type="char" length="64" usage="input" init="n/a" />
     <struct name="item"  usage="input" count="checkinvio.SendMessage.incount" >
       <data name="identifier" type="int" length="4" usage="input" />
      <data name="quantity" type="int" length="4" usage="input" />
       <data name="price" type="float" length="4" usage="input" />
       <data name="description" type="char" length="64" usage="input" init="n/a" />
     </struct>
     <data name="istatus" type="int" length="4" usage="input" init="0" />
   </struct>

   <!-- OUTPUT struct defining message to receive, name must be ReceiveMessage-->
   <struct name="ReceiveMessage" usage="output" >
     <data name="outcount" type="int" length="4" usage="output" />
     <data name="hdroutdsc" type="char" length="64" usage="output" />
     <struct name="itemout" usage="output" count="checkinvio.ReceiveMessage.outcount" outputsize="1024" >
       <data name="code" type="int" length="4" usage="output" />
       <data name="quant" type="int" length="4" usage="output" />
       <data name="outprice" type="float" length="4" usage="output" />
       <data name="status" type="char" length="64" usage="output" />
     </struct>
     <data name="msgoend" type="char" length="12" usage="output" />
     <data name="ostatus" type="int" length="4" usage="output" />
   </struct>
  </program>
</pcml>
```

Example: Data queue

QueueConnectorDQ allows mappings for the DataQueue key. You can map input values to send and receive values to match. In addition, the value that you use to send can be mapped into your output document. Here is an example of all independent possibilities. They can be specified in any combination, or any order. Only the highest level structure or data names must match what is specified in the example.

```
<pcml version="1.0">

  <!-- PCML source for Inventory Check, document name must match program name -->
  <program name="checkinvio" path="not used">

   <!-- INPUT struct defining the message to receive, name must be SendMessage-->
   <struct name="SendMessage" usage="input" >
     <data name="incount" type="int" length="4" usage="input" />
     <data name="hdrdesc" type="char" length="64" usage="input" init="n/a" />
     <struct name="item"  usage="input" count="checkinvio.SendMessage.incount" >
       <data name="identifier" type="int" length="4" usage="input" />
      <data name="quantity" type="int" length="4" usage="input" />
       <data name="price" type="float" length="4" usage="input" />
       <data name="description" type="char" length="64" usage="input" init="n/a" />
     </struct>
     <data name="istatus" type="int" length="4" usage="input" init="0" />
   </struct>

   <!-- OUTPUT struct defining message to receive, name must be ReceiveMessage-->
   <struct name="ReceiveMessage" usage="output" >
     <data name="outcount" type="int" length="4" usage="output" />
     <data name="hdroutdsc" type="char" length="64" usage="output" />
     <struct name="itemout" usage="output" count="checkinvio.ReceiveMessage.outcount" outputsize="1024" >
       <data name="code" type="int" length="4" usage="output" />
       <data name="quant" type="int" length="4" usage="output" />
       <data name="outprice" type="float" length="4" usage="output" />
       <data name="status" type="char" length="64" usage="output" />
     </struct>
     <data name="msgoend" type="char" length="12" usage="output" />
     <data name="ostatus" type="int" length="4" usage="output" />
   </struct>

   <!-- INPUT struct for mapping iSeries data queue key to use for receiving msg.
        Only used when Keyed Receive Queue property is YES and the Receive Queue Key property is MAPPED -->
   <struct name="DQKeyToReceive" usage="input" >
      <data name="dqrcvkey" type="char" length="10" usage="input" init=" " />
   </struct>
  </program>
</pcml>
```

Application connector document field set (ACDFieldSet): ACDFieldSet is an XML grammar that is designed for use with iSeries Connect. ACDFieldSet provides a mechanism to fully specify application connector fields and their attributes, such as attributes that define the type, length, precision, name, and repetition of each field.

The ACDFieldSet grammar provides certain advantages over the PCML grammar: ACDFieldSet is not restricted to the semantics and type scheme supported by the iSeries platform, and you can define type information for each field.

ACDFieldSet documents are used for message construction, buffer layout, and field descriptions.

Syntax

Here is the syntax for the ACDFieldSet format. When you create an ACDFieldSet document, use the file extension ACDFieldSet.

```
<field name="name" location="input or output message field name" display="yes | no"
[type = "int | float | byte | struct | double | bigdecimal | long | short | bytearray | string | boolean"]
[repeating ="yes | no"]
>
[<length> "number" </length>]
[<precision> "number" </precision]
[<countfield> "reference to a previous int,short,or long field"</countfield>]
</field>
```

These are the attributes on the Field element:

name Specifies the name of the field element.

location
Specifies the path to the field.

display
Indicates if this field should be displayed in the deployment editor.

type Indicates the data type of the field being used.

repeating
Indicates if the input field is a repeating field.

length Specifies the length of the data element.

precision
Specifies the precision (in bytes) for some numeric data types. It is used to define the explicitly understood number of decimal positions. Output fields only reflect these number of digits. The connector's processing code must use the same number of decimal digits.

countfield
Specifies a field where a count (the actual number of repeated fields) is stored. The count is an int, short, or long data type field. If there is a repeating field already within a repeating structure, then the countfield must reference a field within that structure.

Example: ACDFieldSet document

Here is a sample ACDFieldSet document, named sample.ACDFieldSet:

```
<?xml version="1.0"?>
<acdfieldset version="2.0" >
  <input >
```

```
    <field display="yes" location="/Order"
      name="Order" repeating="no" type="struct" >
      <children >
        <field display="no" location="/Order/item_count"
            name="item_count" repeating="no" type="int" />
        <field display="yes" location="/Order/Item"
            name="Item" repeating="yes" type="struct" >
          <countfield >/Order/item_count</countfield>
          <children >
            <field display="yes" location="/Order/Item/itemNumber"
                  name="itemNumber" repeating="no" type="string" >
              <length >7</length>
            </field>
            <field display="yes" location="/Order/Item/price"
                  name="price" repeating="no" type="float" />
            <field default="1" display="yes" location="/Order/Item/quantity"
                  name="quantity" repeating="no" type="int" />
            <field display="no" location="/Order/Item/line_count"
                  name="line_count" repeating="no" type="int" />
            <field display="yes" location="/Order/Item/Description"
                  name="Description" repeating="yes" type="string" >
              <length >80</length>
              <countfield >/Order/Item/line_count</countfield>
            </field>
          </children>
        </field>
        <field display="yes" location="/Order/Buyer"
            name="Buyer" repeating="no" type="string" >
            <length >9</length>
        </field>
      </children>
    </field>
  </input>
  <output />
</acdfieldset>
```

Create business process flows

A **process flow** defines the sequence of processing steps that are required to handle a B2B request. These steps may include invocation of connectors and the other steps outlined below. These are the step types that are supported in iSeries Connect process flows:

- **Decision**
  The decision step can evaluate one or more conditional expressions. The conditional expression are listed with corresponding step indicators. The conditional expressions are evaluated in order and the first expression that evaluates to `True` has its corresponding step taken. If no conditional expressions are evaluated to `True`, then the default step is taken. The result of the condition expression must be a boolean type.
- **Copy**
  You can use the copy step to copy data or perform data operations on fields to generate a new value. The resultant data type of the source expression must be the same data type as the target reference. The copy step does not support copying repeating elements or structures.
- **Response**
  You can use the response step to initiate a synchronous response to the delivery gateway. You can execute only one response step when processing a flow. If you do not define a response step, a response is sent at the end of the flow processing.
- **Connector**
  The connector step invokes an application connector instance that you have configured. You map fields (defined by your protocol type) to fields in your back-end application.

To configure your business process flows, use the iSeries Connect configuration tool. Click the **Tools** tab and then click **Process Flows**.

Before you start

Before you configure your business process flows, you should have completed these tasks:

- Determine the request types that are supported by your protocol.
- For each request type, define the steps included in the process flow.
- Configure an application connector for any back-end applications.

Deploy your business process flows

A business process flow is made up of process flow models and associated application connector documents. You need to deploy your business process flow to create the runtime configuration data format that iSeries Connect uses.

To deploy your process flows, click the **Deployment** tab in the iSeries Connect configuration tool. Click **Add Flow** to start the deployment wizard.

After you complete the deployment wizard, you must update the flow manager component with your process flow data. If the flow manager is running, click **Update flow manager** (under the Deployment tab) to refresh it. Otherwise, the changes you have made do not take effect until the next time you start the flow manager.

Start your instance

To start your instance, follow these steps:

1. In the iSeries Connect configuration tool, click the **Instances** tab. Select your instance.
2. Click **Start**.
3. On the Start B2B Instance page, select all listed components.
4. Click **Start**. It may take a few minutes for your instance to start. Wait until all components show a status of Started and you receive a message that the instance was successfully started.

Test your instance

iSeries Connect provides a utility, called Test Drive Connect, that simulates requests from a B2B marketplace. You can use this utility to ensure that your instance is correctly configured to process B2B requests.

To run the Test Drive Connect wizard, do the following: S and click **Test Drive Connect**.

1. Click the **Instances** tab and select your instance on the Manage Connect Instances page.
2. Click the **Tools** tab and then click **Test Drive Connect**.

For an example of using the Test Drive Connect utility, see “Run the PCML verification sample” on page 85.

Configure WebSphere Commerce Suite

To configure the “WebSphere Commerce Suite extensions” on page 18 instance and the instance to allow WCS stores to participate as providers in a B2B environment, follow these steps:

1. Create a new WCS instance on iSeries. See these resources for information about creating WCS instances:
   - Installation Guide: WebSphere Commerce Suite Professional Edition for e-server iSeries, Version 5.1
   - Installation Guide: WebSphere Commerce for e-server iSeries, Version 5.4
2. Before you start your new WCS instance follow these steps:

a. If you are using a remote WCS instance, an entry must be added to the relational database directory on the iSeries server where iSeries Connect is installed. Use the WRKRDBDIRE CL command to add the entry.

b. Your WCS instance has an associated HTTP Server. This HTTP Server must support Secure Sockets Layer (SSL) protocol. See these iSeries Information Center resources for more information about changing your HTTP Server configuration:

   - For V5R1, Configure a secure server on HTTP Server
   - For V5R2, Configure a secure server on HTTP Server

c. To support the iSeries Connect WCS extensions, a digital certificate must be associated with the HTTP Server instance. Use digital certificate manager (DCM) to associate the digital certificate.

   For WCS instances that reside on the same system as iSeries Connect, the digital certificate must have a common host name. For example, if the host name of the WCS instance is myhost.mydomain.mycompany.com, the digital certificate that is assigned to the HTTP Server instance must have a common name of myhost.mydomain.mycompany.com. Associate the digital certificate after the instance has been created, but before you start the instance. See these Information Center resources for more information:

   - For V5R1, Digital Certificate Management
   - For V5R2, Digital Certificate Management

d. For WCS instances that reside on a system remote to iSeries Connect, a copy of the certificate for the Certificate Authority that signed the server certificate for the WCS instance must be imported into the *SYSTEM key store on the system that runs iSeries Connect. If iSeries Connect is split into separate systems for running the delivery gateway and flow manager functions, the Certificate Authority certificate must be copied to the flow manager system. See “Copy digital certificates for remote WebSphere Commerce Suite (WCS) instances” on page 72.

e. If you are using a remote WCS instance, copy the JAR file that contains the WCS extensions support for WCS to the remote server:

   If you are using WCS 5.1, do the following:

   1) Transfer /QIBM/ProdData/Connect200/Classes/wcsiconnect.jar (in binary mode) to the remote system’s /QIBM/UserData/CommerceSuite5/Instances/*instance_name*/lib directory.

      Note: You can put the jar file in a different directory as long as you update the class path to refer to that location.

   2) On the remote system, update the classpath of the WebSphere Application Server instance that is associated with the WCS instance. Append this path to the front of the classpath: `/QIBM/UserData/CommerceSuite5/Instances/wcsiconnect.jar`. For more information about updating the classpath, see the ″Update class paths″ topic in Chapter 8 of the IBM WebSphere Commerce Suite Programmer’s Guide, Version 5.1

   If you are using WCS 5.4, do the following:

   1) Transfer /QIBM/ProdData/Connect200/Classes/wcsiconnect54.jar (in binary mode) to the remote system’s /QIBM/UserData/WebASAdv4/*WASAdminServer*/InstalledApps/ WC_Enterprise_App_*wcsinstancename*.ear/wcstores.war/WEB-INF/lib/ directory.

3. Use existing methods for populating your WCS instance with merchant and product information. If you plan to support remote catalog requests between B2B trading partners and this WCS instance, you need to use a B2B enabled store. For information about using a model B2B-enabled WCS store (which is supplied as part of iSeries Connect), see these topics:

   - For WCS 5.1, see “Enabling a WebSphere Commerce Suite 5.1 store for remote catalog support” on page 72.
   - For WCS 5.4, see “Enabling a WebSphere Commerce Suite 5.4 store for remote catalog support” on page 80.

4. "Create an instance" on page 40.
5. Start your WCS instance.
6. Configure provider information using the provider registration function. During this step, you can make associations between the provider and a WCS merchant.
7. Configure partner organizations using the partner organization registration function. When you associate a partner organization to a provider and marketplace association, you can also indicate that the partner organization is allowed to participate as a WCS shopper. During this configuration step, the WCS extensions configuration services automatically adds a unique shopper table entry into the WCS shopper table, and the associations between partner organizations and WCS shoppers are made.
8. (Optional) Make shopper or customer group assignments by using the WCS configuration tools if the administrator decides to allow special pricing for some of the partner organizations.
9. Augment the products that are associated with the B2B providers that have an association with WCS merchants by using the catalog services tool. The catalog services tool uses methods that the WCS extensions catalog services provides to perform this function.
10. Create the local catalog or remote catalog image that is required for downloading to the partner organization site. If you want to use WCS Partnumber or Manufacturer's Partnumber values, see "Customize WebSphere Commerce Suite" on page 96.
11. "Deploy process flows for WebSphere Commerce Suite (WCS)" on page 84.
12. Start the instance.
13. Follow the instructions required to download the local or remote catalog image onto your trading partner system or network.

Now, iSeries Connect and WCS can receive B2B requests from partner organizations.

> **Note:** If a partner organization with WCS shopper associations is deleted, the shoppers are not removed from WCS.

For additional information about WCS customization, see "Customize WebSphere Commerce Suite" on page 96.

Copy digital certificates for remote WebSphere Commerce Suite (WCS) instances

For authentication purposes between the iSeries Connect system and the remote WCS system, it is necessary to obtain a copy of the Certificate Authority (CA) certificate that signed the server certificate that is being used by the HTTP server supporting the WCS instance. Use IBM Digital Certificate Manager to copy the certificate. For more information about Digital Certificate Manager, see these iSeries Information Center resources:

- For V5R1, Digital Certificate Management
- For V5R2, Digital Certificate Management

> **Note:** If you created the iSeries Connect instance before the *SYSTEM certificate store was created on that same system, you must grant additional authority to the instance user profile (named *instance_name*). The instance user profile must have Read and Execute (*RX) authority to the entire path of the /QIBM/UserData/icss/cert/server/default.kdb *SYSTEM certificate store file.

Enabling a WebSphere Commerce Suite 5.1 store for remote catalog support

Remote catalog support requires a WebSphere Commerce Suite (WCS) store enabled for use with Connect for iSeries. The easiest way to create a WCS store enabled for remote catalog support is to look at the InFashionWithConnect sample provided with Connect for iSeries in the store archive named InFashionWithConnect_en_US_es_ES.sar in the /QIBM/ProdData/Connect200/Commerce/Samples/WCSStore directory. InFashionWithConnect is based on the InFashion sample provided with WCS, with changes made to allow it to concurrently participate in both B2C and B2B environments.

If you have an existing WCS store that you want to enable for use as a remote catalog with Connect for iSeries, you can learn how the InFashion sample was modified to produce the InFashionWithConnect sample by reading “Details: InFashionWithConnect sample store implementation” on page 74. You may also want to look at the actual implementation of the InFashionWithConnect sample, by looking at the contents of the store archive.

If you do not have an existing WCS store, but want to create a new WCS store that is enabled for use as a remote catalog, you could use the InFashionWithConnect sample as the starting point for your own customization or you could just use it as a source of ideas for how you want to implement your store. If you are creating a store only for remote catalog support with no need to support B2C users, then your implementation would not need to use the conditional logic that is in InFashionWithConnect to behave differently depending on whether the user is B2B or B2C.

If you want to publish a new store using the shipped InFashionWithConnect sample, follow these steps:

1. To make InFashionWithConnect available as a sample in WCS Store Services, follow the instructions in Chapter 4 ″Create a sample store archive″ of the IBM WebSphere Commerce Suite Store Developer: Building a Store Archive, Version 5.1 Follow all steps except for the first (the store archive is shipped, so you don’t need to build it), using InFashionWithConnect as the store name.
2. Use StoreServices to create a new store archive specifying the sample named InFashionWithConnect_en_US_es_ES.sar. For detailed instructions, see the ″Create a store archive using Store Services″ topic in Chapter 5 of the IBM WebSphere Commerce Suite Store Developer: Creating a Store Using the Store Services, Version 5.1 information.
3. Before publishing the store, change the store name to whatever name you want, and make whatever other modifications you want. For detailed instructions, see Chapters 6 through 12 of the IBM WebSphere Commerce Suite Store Developer: Creating a Store Using the Store Services, Version 5.1 information.
4. Use StoreServices to publish the new store. For detailed instructions, see the ″Publish a store archive from Store Services″ topic in Chapter 13 of the IBM WebSphere Commerce Suite Store Developer: Creating a Store Using the Store Services, Version 5.1 information.
5. Use the WCS Configuration Manager to disable caching for the WCS instance you are using, if it is currently enabled.

   By default, WCS does caching of the sample store web pages. The default caching does not look at the new b2bShopperType and b2bLogonMode parameters being used by the InFashionWithConnect sample. With caching enabled, that will cause all users to see the same pages, rather than seeing pages tailored based on those parameters. The simplest way to avoid that problem is to just disable caching for your WCS instance.

   For more information about WCS caching, see the ″Cache administration″ topic in Chapter 8 of the IBM WebSphere Commerce Suite Fundamentals, Version 5.1 information.

   For better performance, you may want to batch compile your JSPs. For information about how to do that, see the ″Compile the JavaServer Pages files″ topic in Chapter 7 of the Installation Guide: WebSphere Commerce Suite Professional Edition for e-server iSeries, Version 5.1

Details: InFashionWithConnect sample store implementation

The InFashionWithConnect sample store was created based on the WCS InFashion sample store, with modifications to support concurrent use by both B2C and B2B customers. As much as possible, the existing interface and implementation of the WCS InFashion sample were preserved as is or modified in a way that is meant to be consistent with the approach taken in the InFashion sample. InFashionWithConnect is just one sample, and you may have different implementation approaches that work better for you, so do not feel that your store needs to work the same as this sample.

Changes for B2C

The following are changes relative to the original WCS InFashion sample which are relevant to a B2C user of InFashionWithConnect:

- The PaymentManager checking was disabled in order to make it easier to run the sample. That means that the sample allows orders to be placed without validating payment information. For any production store you would create, you would want to make sure to implement payment validation that meets your requirements. See details below.
- The version of InFashion available at the time the InFashionWithConnect sample was created mistakenly had both products and items being marked as ″Buyable″. That mistake was corrected by changing the catalog.xml file to specify Buyable=″0″ for all ProductBeans.

Changes for B2B

The following are changes relative to the original WCS InFashion sample which are relevant to a B2B user of InFashionWithConnect:

- The Java Server Pages (JSPs) that support the browser pages a B2B user will see have been modified to support additional parameters passed with the url string. When invoking the WCS store for a punchout request, the Connect for iSeries support passes the first two of these parameters set to the appropriate values for the request. The JSP implementations then use these parameters to produce the appropriate output, and they propagate these parameters to other pages reachable from themselves in order to get the right behavior on those pages too.
  - **b2bShopperType**

    This parameter indicates whether the shopper type is B2B or B2C. The Connect for iSeries support always specifies this parameter with a value of `b2b`. The JSPs are implemented to interpret any other value or the complete absence of this parameter to mean the shopper type is B2C.
  - **b2bLogonMode**

    For B2B shoppers, this parameter is used to indicate whether the request is to create a new quote (b2bLogonMode=create), to edit an existing quote (b2bLogonMode=edit), or to inspect an existing quote (b2bLogonMode=inspect). The pages behave about the same for create and edit modes, but for inspect mode there are significant changes to prevent the user from making any modifications to the existing quote. For example, editable fields are made read-only, and buttons related to making changes are not shown.
  - **b2bOrderId**

    For B2B shoppers editing or inspecting an existing order, this parameter is used to propagate the order identifier from the punchout request through all the JSP pages so that it can be returned on the call to the B2BNewQuote command.
- A new properties file is provided, named infashiontext_connect_en_US.properties. This file contains the translatable text shown to the B2B shopper. Like the InFashion sample, the InFashionWithConnect sample is enabled for multiple languages, and the InFashionWithConnect sample store archive is configured for English and Spanish, and a B2C user can view the pages in Spanish, but the new infashiontext_connect_en_US.properties file has not been translated into Spanish so a B2B user always sees English. All it would take to enable the InFashionWithConnect sample store to be displayable in Spanish would be for you to create an infashiontext_connect_es_ES.properties file that contained the properties translated into Spanish.

- An implementation of the ItemDisplay JSP was provided to support punchout request for a specific item. The implementation is based on the ProductDisplay JSP, with appropriate modifications.
- Various functions are not shown on the pages for B2B users. For example, the ″My account″ options in the header and footer are not shown, and the ″Register now″ option in the sidebar is not shown.
- Appropriate changes were made to the terminology. For example, references to the term *order* were changed to *quote.*
- The text shown on the Help page was changed to be more appropriate for the B2B user.
- Function not considered appropriate or necessary for a B2B user in this sample was suppressed. This includes the billing address, ship-to address, shipping method, shipping charges, taxes, and payment information.
- A button has been added to the header and footer to support cancelling the B2B operation. The implementation uses the “WebSphere Commerce Suite B2BNewQuote command” on page 77 to ensure that the transaction is closed appropriately. For a create request, it simply cancels the operation so that no new quote is created. For an inspect request, it cancels the operation and there is no change made to the existing quote. For an edit request, it deletes the existing quote.

  If an edit is performed and no changes are to be made, the shopper must complete the edit transaction with the choice of items and quantities shown matching the existing quote and without using the cancel option.
- When a create or edit quote operation is completed, the JSP uses the “WebSphere Commerce Suite B2BNewQuote command” on page 77 to ensure that the transaction is performed appropriately. Instead of then showing the same confirmation page that a B2C shopper would see, the implementation redirects the browser to the appropriate partner page.

Disabling the Payment Manager

The Payment Manager checking was disabled in order to make it easier to run the sample. That means that the sample allows orders to be placed without validating payment information. For any production store you would create, you would want to make sure to implement payment validation that meets your requirements.

The payment manager was disabled by doing the following:

1. In the paymentinfo.xml file in the data/es_ES directory, on the line specifying <PaymentManager enable=″yes″/>, changed ″yes″ to ″no″.
2. In the command.xml file in the data directory, in the section specifying the command for the interface named ″com.ibm.commerce.payment.commands.DoCancelCmd″, changed the classname setting from ″com.ibm.commerce.payment.commands.DoCancelPMCmdImpl″ to ″com.ibm.commerce.payment.commands.DoCancelCmdImpl″.
3. In the command.xml file in the data directory, in the section specifying the command for the interface named ″com.ibm.commerce.payment.commands.DoPaymentCmd″, changed the classname setting from ″com.ibm.commerce.payment.commands.DoPaymentMPFCmdImpl″ to ″com.ibm.connect.wcsext.environment.Qbecb2bDoPaymentCmdImpl″.
4. Implemented the Qbecb2bDoPaymentCmdImpl Java class specified in the command.xml file above. This class implements the DoPaymentCmd interface and extends the TaskCommandImpl class. Since the sample intentionally is not performing any validation of payment information, the implementation of the performExecute method in this class simply returns without doing anything. For general information, see the ″Task command customization″ topic in Chapter 6 of the IBM WebSphere Commerce Suite Programmer’s Guide, Version 5.1 information.

Modified JSPs

These are the specific JSPs from the InFashion sample that were modified in producing the InFashionWithConnect sample. The files include comments describing the changes made. In general, if you scan for the string ″B2B″, you will find the changes.

- confirmation.jsp
- emptyshoppingcart.jsp
- footer.jsp
- getresource.jsp
- header.jsp
- help.jsp
- InventoryError.jsp
- ItemDisplay.jsp
- newarrivals.jsp
- OrderDisplayPending.jsp
- ProductDisplay.jsp
- shoppingcart.jsp
- sidebar.jsp
- StoreCatalogDisplay.jsp
- subcategory.jsp
- topcategory.jsp

New WebSphere Commerce Suite (WCS) commands to support iSeries Connect

These commands provide communication between WCS and iSeries Connect:

Command Name	Interface	Implementation	Description
WCSB2BCatalog	Qbecb2bcatCmd	Qbecb2bcatCmdImpl	Provides WCS catalog information using WCS 5.1 data access beans.
WCSB2BCatalog	Qbecb2bcatsqlCmd	Qbecb2bcatsqlCmdImpl	Provides WCS catalog information using direct SQL into the WCS 5.1 database.
B2BLogon	Qbecb2blogCmd	Qbecb2blogCmdImpl	Used by the internal iSeries Connect processes to log onto the WCS instance to perform transactions.
B2BNewQuote	Qbecb2bnqCmd	Qbecb2bnqCmdImpl	See “WebSphere Commerce Suite B2BNewQuote command” on page 77 for more information.
B2BOrder	Qbecb2borderCmd	Qbecb2borderCmdImpl	Tools to process a purchase order request in WCS.
WCSPunchOut	Qbecb2bpoCmd	Qbecb2bpoCmdImpl	Authenticates a shopper to WCS.
B2BShopperAdmin	Qbecb2bsaCmd	Qbecb2bsaCmdImpl	Administrative tools that maintain B2B shoppers in WCS.
WCSB2BShop	Qbecb2bshopCmd	Qbecb2bshopCmdImpl	See “WebSphere Commerce Suite WCSB2BShop command” on page 79 for more information.

Command Name	Interface	Implementation	Description
B2BVerify	Qbecb2bvfyCmd	Qbecb2bvfyCmdImpl	Verification tools that provide access to WCS information.

WebSphere Commerce Suite B2BNewQuote command: B2BNewQuote is a command used by the BtoB store and iSeries Connect to pass control back to iSeries Connect to allow the quote transaction to be processed. A WebSphere Commerce Suite (WCS) order is transformed into the quote. The order must exist and be in pending (″P″) status prior to starting this command. Typically, it can be called after the completion of an OrderDisplay command.

Parameters

These parameters are passed into B2BNewQuote during the punchout process:

Parameter name	Description	Required	Optional
order_rn	WCS order reference number that is used as the quote	X	
merchant_rn	WCS merchant reference number	X	
nq_mode	Activity code (create, cancel) to determine how to complete the transaction	X	

The nq_mode parameter (used in conjunction with the shopper logon_mode parameter) determines the process that is used to complete the transaction.

Logon mode	Nq mode	Process
create	create	Update status = ’q’, pass order_rn=new order reference number
create	cancel	No update to status, pass order_rn=-1
edit	create	Update status = ’q’, pass order_rn=new order reference number
edit	cancel	Update status = ’q’, pass order_rn=-1
inspect	create	Update status = ’q’, pass order_rn=new order reference number
inspect	cancel	Update status = ’q’, pass order_rn=-1

For example:

```
<A HREF="/webapp/wcs/stores/servlet/B2BNewQuote?merchant_rn=$(merfnbr)&order_rn=
       $(order_rn)&nq_mode=cancel" TARGET="homes">
   <FONT size="$(linkTextSize) FACE="$(textFont)>
      <center>Cancel Transaction</center>
   </FONT>
</A>
```

The logon_mode and nq_mode parameters determine the parameter values that are passed.

Logon mode	Nq mode	URL
create	create	http://*server_name*/webapp/wcs/stores/servlet/ B2BNewQuote?order_rn=XXXX&merchant_rn=XXXXX&nq_mode=create
create	cancel	http://*server_name*/webapp/wcs/stores/servlet/ B2BNewQuote?order_rn=XXXX&merchant_rn=XXXXX&nq_mode=cancel
edit	create	http://*server_name*/webapp/wcs/stores/servlet/ B2BNewQuote?order_rn=XXXX&merchant_rn=XXXXX&nq_mode=create
edit	cancel	http://*server_name*/webapp/wcs/stores/servlet/ B2BNewQuote?order_rn=XXXX&merchant_rn=XXXXX&nq_mode=cancel
inspect	create	http://*server_name*/webapp/wcs/stores/servlet/ B2BNewQuote?order_rn=XXXX&merchant_rn=XXXXX&nq_mode=create
inspect	cancel	http://*server_name*/webapp/wcs/stores/servlet/ B2BNewQuote?order_rn=XXXX&merchant_rn=XXXXX&nq_mode=cancel

Processing

When the command has successfully completed processing the transaction, the OrderOkView task is called. OrderOkView redirects to ConfirmationView which has confirmation.jsp as the assigned view.

These parameters are passed to confirmation.jsp:

Parameter name	Description	Required	Optional
SupplierNumber	Internal iSeries Connect	X	
SupplierDomainNumber	Internal iSeries Connect	X	
Buyer Number	Internal iSeries Connect	X	
BuyerDomainNumber	Internal iSeries Connect	X	
SupplierCookie	Unique shopper identifier	X	
QuoteNumber	WCS order number	X	
PostBackURL	Internal iSeries Connect	X	
NewQuoteURL	Internal iSeries Connect	X	
nqMode	create or cancel	X	
logonMode	create, edit, or inspect	X	
shopper_type	BtoB shopper identifier `b2b`	X	

The confirmation.jsp file contains JavaScript code that redirects the browser to the iSeries Connect delivery gateway and passes the required parameters for processing.

For example:

```
<script language="JavaScript">
   location.href="<%=newQuoteURL%>
      ?SupplierNumber=<%=supplierNumber%>
      &SupplierNumberDomain=<%=supplierNumberDomain%>
      &BuyerNumber=<%=buyerNumber%>
      &BuyerNumberDomain=<%=buyerNumberDomain%>
      &SupplierCookie=<%=supplierCookie%>
      &QuoteNumber=<%=quoteNumber%>
      &PostBackURL=<%=postBackURL%>";
</script>
```

WebSphere Commerce Suite WCSB2BShop command: WCSB2BShop is a command used by the BtoB store and iSeries Connect in the remote catalog punchout process to allow the user to logon to the WCS store. Store entry is controlled by the logon mode of either create, edit, or inspect. If the mode is create and no catalog entry id is specified, the user is directed to the store page that allows them to create a new quote. If the mode is create and a catalog entry id is specified, the user is directed to that item page and is allowed to create a new quote. If it is either edit or inspect, the existence of the quote is confirmed, the status of the quote is updated, and the user is directed to either a store page that allows edit functions or a page for inspection of the existing quote.

Parameters

These parameters are passed into WCSB2Bhop during the punchout process:

Parameter name	Description	Required	Optional
SupplierCookie	Unique shopper identifier	X	
NewQuoteURL	Internal iSeries Connect	X	
PostBackURL	Internal iSeries Connect	X	
SupplierCode	Internal iSeries Connect	X	
SupplierDomain	Internal iSeries Connect	X	
BuyOrgCode	Internal iSeries Connect	X	
BuyOrgDomain	Internal iSeries Connect	X	
B2B_Logon_Mode	Transaction type (create, edit, or inspect)	X	
shiptoRN	WCS ShipTo reference number for an item in a quote	edit or inspect	create
B2B_Merchant_RN	Used to determine the redirect URL	X	
catEntryId	Used for detailed remote catalog punchout	detailed	non-detailed

The B2B_Logon_Mode and catEntryId parameters determine the processing and redirection to the store home page.

Logon mode	catEntryId	Process
create	Specified	Redirect to item page.
create	Not specified	Redirect to store home page.
edit	Not applicable	Update quote status = 'P,' redirect to display the quote.
inspect	Not applicable	Update quote status = 'P,' redirect to display the quote.

The WCSB2BShop command determines what page to load (WCS command to process) based on the logon mode. The redirect parameters are determined by the B2B_Logon_Mode parameter.

Logon mode	catEntryId	Redirect URL
create	Specified	http://*server_name*/webapp/wcs/stores/servlet/ProductDisplay ?storeId=XX&langId=XX&catalogId=XX&productId=XXXX &b2bShopperType=b2b&b2bLogonMode=create
create	Not specified	http://*server_name*/webapp/wcs/stores/servlet/StoreCatalogDisplay ?storeId=XXXX&langId=XX&catalogId=XX &b2bShopperType=b2b&b2bLogonMode=create
edit	Not applicable	http://*server_name*/webapp/wcs/stores/servlet/OrderItemDisplay ?storeId=XX&langId=XX&catalogId=XX&orderId=XXXX &b2bShopperType=b2b&b2bLogonMode=edit

Logon mode	catEntryId	Redirect URL

inspect	Not applicable	`http://server_name/webapp/wcs/stores/servlet/OrderItemDisplay`
		`?storeId=XX&langId=XX&catalogId=XX&orderId=XXXX`
		`&b2bShopperType=b2b&b2bLogonMode=inspect`

Enabling a WebSphere Commerce Suite 5.4 store for remote catalog support

Remote catalog support requires a WebSphere Commerce Suite (WCS) store enabled for use with Connect for iSeries. The easiest way to create a WCS store enabled for remote catalog support is to look at the InFashionWithConnect sample provided with Connect for iSeries in the store archive named InFashionWithConnect54_en_US_es_ES.sar in the /QIBM/ProdData/Connect200/Commerce/Samples/WCSStore directory. InFashionWithConnect is based on the InFashion sample provided with WCS, with changes made to allow it to concurrently participate in both B2C and B2B environments.

If you have an existing WCS store that you want to enable for use as a remote catalog with Connect for iSeries, you can learn how the InFashion sample was modified to produce the InFashionWithConnect sample by reading “Details: InFashionWithConnect54 sample store implementation” on page 81. You may also want to look at the actual implementation of the InFashionWithConnect sample, by looking at the contents of the store archive.

If you do not have an existing WCS store, but want to create a new WCS store that is enabled for use as a remote catalog, you could use the InFashionWithConnect sample as the starting point for your own customization or you could just use it as a source of ideas for how you want to implement your store. If you are creating a store only for remote catalog support with no need to support B2C users, then your implementation would not need to use the conditional logic that is in InFashionWithConnect to behave differently depending on whether the user is B2B or B2C.

If you want to publish a new store using the shipped InFashionWithConnect sample, follow these steps:

1. To make InFashionWithConnect available as a sample in WCS Store Services, follow the instructions in ″Creating a store using a sample store archive″ topic in Chapter 6 of the IBM WebSphere Commerce Fundamentals, Version 5.4 .
   a. Use StoreServices to create a new store archive specifying the sample named InFashionWithConnect54_en_US_es_ES.sar.
   b. Before publishing the store, change the store name to whatever name you want, and make whatever other modifications you want.
   c. Use StoreServices to publish the new store.
2. Use the WCS Configuration Manager to disable caching for the WCS instance you are using, if it is currently enabled.

   By default, WCS does caching of the sample store Web pages. The default caching does not look at the new b2bShopperType and b2bLogonMode parameters being used by the InFashionWithConnect sample. With caching enabled, that will cause all users to see the same pages, rather than seeing pages tailored based on those parameters. The simplest way to avoid that problem is to just disable caching for your WCS instance.

   For more information about WCS caching, see the ″Dynamic page caching″ topic in Chapter 10 of the IBM WebSphere Commerce Fundamentals, Version 5.4 information.

   For better performance, you may want to batch compile your JSPs. For information about how to do that, see the ″Compile the JavaServer Pages files″ topic in Chapter 8 of the Installation Guide: WebSphere Commerce for e-server iSeries, Version 5.4

Details: InFashionWithConnect54 sample store implementation

The InFashionWithConnect sample store was created based on the WCS InFashion sample store, with modifications to support concurrent use by both B2C and B2B customers. As much as possible, the existing interface and implementation of the WCS InFashion sample were preserved as is or modified in a way that is meant to be consistent with the approach taken in the InFashion sample. InFashionWithConnect is just one sample, and you may have different implementation approaches that work better for you, so do not feel that your store needs to work the same as this sample.

Changes for B2C

The following are changes relative to the original WCS InFashion sample which are relevant to a B2C user of InFashionWithConnect:

- The PaymentManager checking was disabled in order to make it easier to run the sample. That means that the sample allows orders to be placed without validating payment information. For any production store you would create, you would want to make sure to implement payment validation that meets your requirements. See details below.
- The version of InFashion available at the time the InFashionWithConnect sample was created mistakenly had both products and items being marked as ″Buyable″. That mistake was corrected by changing the catalog.xml file to specify Buyable=″0″ for all ProductBeans.

Changes for B2B

The following are changes relative to the original WCS InFashion sample which are relevant to a B2B user of InFashionWithConnect:

- The Java Server Pages (JSPs) that support the browser pages a B2B user will see have been modified to support additional parameters passed with the url string. When invoking the WCS store for a punchout request, the Connect for iSeries support passes the first two of these parameters set to the appropriate values for the request. The JSP implementations then use these parameters to produce the appropriate output, and they propagate these parameters to other pages reachable from themselves in order to get the right behavior on those pages too.
  - **b2bShopperType**

    This parameter indicates whether the shopper type is B2B or B2C. The Connect for iSeries support always specifies this parameter with a value of `b2b`. The JSPs are implemented to interpret any other value or the complete absence of this parameter to mean the shopper type is B2C.
  - **b2bLogonMode**

    For B2B shoppers, this parameter is used to indicate whether the request is to create a new quote (b2bLogonMode=create), to edit an existing quote (b2bLogonMode=edit), or to inspect an existing quote (b2bLogonMode=inspect). The pages behave about the same for create and edit modes, but for inspect mode there are significant changes to prevent the user from making any modifications to the existing quote. For example, editable fields are made read-only, and buttons related to making changes are not shown.
  - **b2bOrderId**

    For B2B shoppers editing or inspecting an existing order, this parameter is used to propagate the order identifier from the punchout request through all the JSP pages so that it can be returned on the call to the B2BNewQuote command.
- A new properties file is provided, named infashiontext_connect_en_US.properties. This file contains the translatable text shown to the B2B shopper. Like the InFashion sample, the InFashionWithConnect sample is enabled for multiple languages, and the InFashionWithConnect sample store archive is configured for English and Spanish, and a B2C user can view the pages in Spanish, but the new infashiontext_connect_en_US.properties file has not been translated into Spanish so a B2B user always sees English. All it would take to enable the InFashionWithConnect sample store to be displayable in Spanish would be for you to create an infashiontext_connect_es_ES.properties file that contained the properties translated into Spanish.

- An implementation of the ItemDisplay JSP was provided to support punchout request for a specific item. The implementation is based on the ProductDisplay JSP, with appropriate modifications.
- Various functions are not shown on the pages for B2B users. For example, the ″My account″ options in the header and footer are not shown, and the ″Register now″ option in the sidebar is not shown.
- Appropriate changes were made to the terminology. For example, references to the term *order* were changed to *quote.*
- The text shown on the Help page was changed to be more appropriate for the B2B user.
- Function not considered appropriate or necessary for a B2B user in this sample was suppressed. This includes the billing address, ship-to address, shipping method, shipping charges, taxes, and payment information.
- A button has been added to the header and footer to support cancelling the B2B operation. The implementation uses the "WebSphere Commerce Suite B2BNewQuote command" on page 77 to ensure that the transaction is closed appropriately. For a create request, it simply cancels the operation so that no new quote is created. For an inspect request, it cancels the operation and there is no change made to the existing quote. For an edit request, it deletes the existing quote.

  If an edit is performed and no changes are to be made, the shopper must complete the edit transaction with the choice of items and quantities shown matching the existing quote and without using the cancel option.
- When a create or edit quote operation is completed, the JSP uses the "WebSphere Commerce Suite B2BNewQuote command" on page 77 to ensure that the transaction is performed appropriately. Instead of then showing the same confirmation page that a B2C shopper would see, the implementation redirects the browser to the appropriate partner page.

Disabling the Payment Manager

The Payment Manager checking was disabled in order to make it easier to run the sample. That means that the sample allows orders to be placed without validating payment information. For any production store you would create, you would want to make sure to implement payment validation that meets your requirements.

The payment manager was disabled by doing the following:

1. In the paymentinfo.xml file in the data/es_ES directory, on the line specifying <PaymentManager enable=″yes″/>, changed ″yes″ to ″no″.
2. In the command.xml file in the data directory, in the section specifying the command for the interface named ″com.ibm.commerce.payment.commands.DoCancelCmd″, changed the classname setting from ″com.ibm.commerce.payment.commands.DoCancelPMCmdImpl″ to ″com.ibm.commerce.payment.commands.DoCancelCmdImpl″.
3. In the command.xml file in the data directory, in the section specifying the command for the interface named ″com.ibm.commerce.payment.commands.DoPaymentCmd″, changed the classname setting from ″com.ibm.commerce.payment.commands.DoPaymentMPFCmdImpl″ to ″com.ibm.connect.wcsext.environment.Qbecb2bDoPaymentCmdImpl″.
4. Implemented the Qbecb2bDoPaymentCmdImpl Java class specified in the command.xml file above. This class implements the DoPaymentCmd interface and extends the TaskCommandImpl class. Since the sample intentionally is not performing any validation of payment information, the implementation of the performExecute method in this class simply returns without doing anything. For general information, see the ″Customizing existing task commands″ topic in Chapter 6 of the IBM WebSphere Commerce Programmer's Guide, Version 5.4 information.

Modified JSPs

These are the specific JSPs from the InFashion sample that were modified in producing the InFashionWithConnect sample. The files include comments describing the changes made. In general, if you scan for the string ″B2B″, you will find the changes.

- confirmation.jsp
- emptyshoppingcart.jsp
- help.jsp
- InventoryError.jsp
- ItemDisplay.jsp
- newarrivals.jsp
- OrderDisplayPending.jsp
- ProductDisplay.jsp
- shoppingcart.jsp
- StoreCatalogDisplay.jsp
- subcategory.jsp
- topcategory.jsp
- include/footer.jsp
- include/getresource.jsp
- include/header.jsp
- include/sidebar.jsp

New WebSphere Commerce Suite (WCS) commands to support iSeries Connect

These commands provide communication between WCS and iSeries Connect:

Command Name	Interface	Implementation	Description
WCSB2BCatalog	Qbecb2bcatCmd	Qbecb2bcatCmdImpl	Provides WCS catalog information using WCS data access beans.
WCSB2BCatalog	Qbecb2bcatsqlCmd	Qbecb2bcatsqlCmdImpl	Provides WCS catalog information using direct SQL into the WCS database.
B2BLogon	Qbecb2blogCmd	Qbecb2blogCmdImpl	Used by the internal iSeries Connect processes to log onto the WCS instance to perform transactions.
B2BNewQuote	Qbecb2bnqCmd	Qbecb2bnqCmdImpl	See “WebSphere Commerce Suite B2BNewQuote command” on page 77 for more information.
B2BOrder	Qbecb2borderCmd	Qbecb2borderCmdImpl	Tools to process a purchase order request in WCS.
WCSPunchOut	Qbecb2bpoCmd	Qbecb2bpoCmdImpl	Authenticates a shopper to WCS.
B2BShopperAdmin	Qbecb2bsaCmd	Qbecb2bsaCmdImpl	Administrative tools that maintain B2B shoppers in WCS.
WCSB2BShop	Qbecb2bshopCmd	Qbecb2bshopCmdImpl	See “WebSphere Commerce Suite WCSB2BShop command” on page 79 for more information.

Command Name	Interface	Implementation	Description
B2BVerify	Qbecb2bvfyCmd	Qbecb2bvfyCmdImpl	Verification tools that provide access to WCS information.

Deploy process flows for WebSphere Commerce Suite (WCS)

Deploy the process flows that support WebSphere Commerce Suite (WCS). The protocol flows you select for accessing the WCS extensions vary depending upon which protocol is being used. If more than one protocol is used to access the WCS extensions support, then each set of protocol flow mappings need to be selected.

These process flows do not need to be changed, however some customization is possible. See "Customize WebSphere Commerce Suite" on page 96 for more information.

Deploy flows based on the protocol your instance supports:

cXML 1.1

For deploying the support necessary to implement WCS local catalog support using the cXML 1.1 protocol, select the process flow for OrderRequest and the request type of new. For the cXML 1.1 protocol, the name of the process flow that is shipped with the system for supporting WCS local catalog is qwcscxmlorderrequestnew200.

If the iSeries Connect instance is associated with an WCS instance, then the process flows for OrderRequest and request types of update and delete are also supported. For the cXML 1.1 protocol, the names of these process flows are qwcscxmlorderrequestupdate200 and qwcscxmlorderrequestdelete200. Support of these update and delete flows are optional.

For deploying the support necessary to implement WCS remote catalog support using the cXML 1.1 protocol, select the process flow for NewQuote and PunchOutSetupRequest with all of the request types (create, edit, and inspect). Additionally, select the process flows for OrderRequest. Select all OrderRequest types (new, deleted, and update). The names of the process flows that are shipped with the system for supporting WCS remote catalog using the cXML 1.1 protocol are:

- qwcscxmlnewquote200
- qwcscxmlorderrequestnew200
- qwcscxmlorderrequestdelete200
- qwcscxmlorderrequestupdate200
- qwcscxmlpunchoutcreate200
- qwcscxmlpunchoutedit200
- qwcscxmlpunchoutinspect200

cXML 1.2

For deploying the support necessary to implement WCS local catalog support using the cXML 1.2 protocol, select the process flow for OrderRequest and the request type of new. For the cXML 1.2 protocol, the name of the process flow that is shipped with the system for supporting WCS local catalog is qwcscxml12orderrequestnew200.

If the iSeries Connect instance is associated with a WCS instance, then the process flows for OrderRequest and request types of update and delete are also supported. For the cXML 1.2 protocol, the names of these process flows are qwcscxml12orderrequestupdate200 and qwcscxml12orderrequestdelete200. Support of these update and delete flows are optional.

For deploying the support necessary to implement WCS remote catalog support using the cXML 1.2 protocol, select the process flow for NewQuote and PunchOutSetupRequest with all of the request types (create, edit, and inspect). Additionally, select the process flows for OrderRequest. Select all OrderRequest types (new, deleted, and update). The names of the process flows that are shipped with the system for supporting WCS remote catalog using the cXML 1.2 protocol are:

- qwcscxml12newquote200
- qwcscxml12orderrequestnew200
- qwcscxml12orderrequestdelete200
- qwcscxml12orderrequestupdate200
- qwcscxml12punchoutcreate200
- qwcscxml12punchoutedit200
- qwcscxml12punchoutinspect200

Run the PCML verification sample

The objective of this verification sample is to verify the successful installation and configuration of the Connect for iSeries sample instance. For detailed information about how the sample works, see “Flow of the PCML verification sample”.

> **Note:** The iSeries Connect catalog and WebSphere Commerce Suite (WCS) extensions are not demonstrated in this sample.

To configure and test the PCML verification sample, follow these steps:

1. If you have not done so, install Connect for iSeries, all prerequisite products, and necessary fixes. See Chapter 4, “Install iSeries Connect” on page 29 for more information.
2. “Start the iSeries Connect configuration tool” on page 39.
3. “PCML sample: Create an instance” on page 86.
4. “PCML sample: Register a new provider” on page 87.
5. “PCML sample: Register a new partner” on page 87.
6. “PCML sample: Deploy the process flow” on page 88.
7. “PCML sample: Start the instance” on page 88.
8. “PCML sample: Test the sample configuration” on page 89.

Flow of the PCML verification sample

Here is the processing flow of the PCML verification sample:

1. The Test Drive Connect tool is started using the **Tools** tab in the iSeries Connect configuration tool. You specify which type of request to simulate. The available test requests provided by this sample program are ProfileRequest and OrderRequest.

   > **Note:** Test Drive Connect also supports the PunchOutSetup request, which is outside of the scope of this sample.
2. Based on the type of request, the Test Drive Connect tool sends a simulated request (in the form of an XML document) to the iSeries Connect front-end delivery gateway. In this sample, the tool simulates two of the cXML documents that the Ariba Network can send to the Connect for iSeries instance:
   - **ProfileRequest**
     Sends a cXML message to the Connect for iSeries delivery gateway which handles the request (without involving the back-end flow manager). Successful completion of this request verifies that the delivery gateway is correctly installed and configured.
   - **OrderRequest**
     Simulates an Ariba order request that flows through the delivery gateway to the flow manager. Successful completion of this request verifies that both the delivery gateway and the flow manager

(and the intermediate MQSeries infrastructure) are correctly installed and configured. In addition, the OrderRequest simulation requires that the sample Application Connector Document and Process Flow Model are deployed, so you gain experience with the iSeries Connect deployment utilities.

3. Before you run the test, you deploy a process flow instance using the sample application connector document, process flow model (PFM), and Program Call Markup Language (PCML) files.
4. The PCML connector invokes the sample back-end business application, OrderReq (a program written in the C programming language).
5. In this scenario, the sample OrderReq program simulates an order entry application. It accepts input parameters and sends its output to a spool file. (In a real-world scenario, this back-end application would handle the actual processing of the order request.) The sample program does not return any parameters.
6. The flow manager sends a response to the delivery gateway that signals that the request was received.
7. The results (response) of the simulated request is displayed in the Results page of the Test Drive Connect tool.

PCML sample: Create an instance

To create an instance, follow these steps in the iSeries Connect configuration tool:

1. Click the **Instances** tab.
2. If you have no instances configured, click **Next** to start the New Instance wizard.

   If you have previously configured instances, click **New Instance** to start the wizard.

   Note: If you are migrating from version 1.1 and you have not migrated your version 1.1 instances, you see a page that asks if you want to create a new instance or migrate an existing (1.1) instance. Select **Create a new instance**.
3. On the Create a New iSeries Connect Instance page, click **Next**.
4. On the Instance Servers page, select **Run the delivery gateway on this server**. Click **Next**.
5. On the Protocol Information page:

Field	Value
Instance name	`sampInst`
Instance description	PCML sample instance
Protocols	`cxml:Ariba:1.1`
Protocol group name	Default

   Click **Next**.
6. On the Protocol Request Information page, select **OrderRequest:new**. Click **Next**.
7. On the IBM HTTP Web Server Information page, select **Create a new IBM HTTP server**. Click **Next**.
8. On the next IBM HTTP Web Server Information page, enter these fields:

Field	Value
HTTP server name	`SAMPINST`
HTTP port	(Type the number of an available port, such as 7080. Use the NETSTAT *CNN command to verify that the port number that is not already used by another application on your system.)
HTTPS port	(Leave this field blank.)

   Click **Next**.

9. On the IBM WebSphere Commerce Suite Server page, select **No**. Click **Next**.
10. On the Configuration Summary page, click **Finish**. Connect for iSeries now creates your instance.

    Note: This may take a few minutes. Wait until the Manage Connect Instances page appears.
11. On the Manage Connect Instances page, select **sampInst** if it is not already selected.

    Note: Do not start your instance until after you deploy your process flows.

PCML sample: Register a new provider

To register a new provider, follow these steps in the iSeries Connect configuration tool:

1. In the Manage Connect Instances page (click the **Instances** tab to see the page), select **sampInst**.
2. Click the **Providers** tab. On the Registered Providers page, click **Next**.
3. On the Provider Information page, enter these values:
   - **Provider**: `Sample Provider Co.`
   - **DUNS**: `123222888`

   Click **Next**
4. On the Provider Address page, click **Next**.
5. On the Contact Information page, enter these values:
   - **First Name**: `First_Name`
   - **Last Name**: `Last_Name`

   Click **Next**.
6. On the Associate Protocol with New Provider page, select **Yes, associate a protocol now**. Click **Next**.
7. On the Protocol for New Provider page, select **Default** protocol group. Click **Next**.
8. On the Protocol Access Information page, enter these values:

Field	**Value**
Provider Id	`123222888`
Provider Id Domain	`DUNS`
Logon Id	`123222888`
Domain	`DUNS`
Password/Shared Secret	`secret`

   Click **Next**.
9. On the Message Delivery Method Setup page, click **Next**.
10. On the Requests Available to this Provider page, select all of the requests that are listed. Click **Next**.
11. On the Order Request Method Preferences page, select `first` for the Web method. Click **Next.**
12. On the Summary for New Provider page, click **Finish**. Connect for iSeries now registers your new provider. The Registered Providers page appears after the process is completed.
13. On the Registered Providers page, select **Sample Provider Co.** if it is not already selected.

PCML sample: Register a new partner

To register a new partner, follow these steps in the iSeries Connect configuration tool:

1. Click the **Partners** tab. On the Registered partners page, click **Next**.

2. On the Partner Information page, type `Sample Partner Co.` in the **Partner** field. Leave the rest of the fields blank. Click **Next**.
3. On the Partner Address page, click **Next**.
4. On the Contact Information page, enter these values:
   - **First Name**: `First_Name`
   - **Last Name**: `Last_Name`

   Click **Next**.
5. On the Partner Billing page, click **Next**.
6. On the Partner Shipping page, click **Next**.
7. On the Partner Cost Center page, click **Next**.
8. On the Associate Protocol with New Partner page, select **Yes, associate a marketplace now**. Click **Next**.
9. On the Protocol for New Partner page, select **Sample Provider Co.** and the corresponding **cxml:Ariba:1.1** protocol. Click **Next**.
10. On the Protocol Access Information page, enter these values:
    - **Partner Id**: `321222888`
    - **Partner Id Domain**: `DUNS`

    Click **Next**.
11. On the Summary for New Partner page, click **Finish**. Connect for iSeries now registers your new partner. The Registered Partners page appears when the process is completed.

PCML sample: Deploy the process flow

To deploy the process flow for the OrderRequest request, follow these steps:

1. Copy the sample process flow files (orderreq.pcml, orderreq.AppConnector, and orderreq.ProcessFlow) from the /QIBM/ProdData/Connect200/Commerce/Samples/Application directory to the /QIBM/UserData/Connect200/Commerce/sampInst/Connector directory. It is recommended that you use a network drive mapped to your server to copy and paste the files.
2. In the iSeries Connect configuration tool, click the **Instances** tab and select **sampInst** in the Manage Connect Instances page.
3. Click the **Deployment** tab.
4. On the Deployment Flows page, click **Add Flow**.
5. On the Add Deployment Flow - Protocol page, select **cxml:Ariba:1.1**. Click **Next**.
6. On the Add Deployment Flow - Process Flows page, select **OrderRequest:new**. From the **Process Flow** pull-down menu for OrderRequest:new, select **orderreq.ProcessFlow**. Click **Next**.
7. On the Add Deployment Flow - Providers page, select **Selected Providers** and **Sample Provider Co**. Click **Next**.
8. On the Add Deployment Flow - Partners page, select **Selected Partners** and **Sample Partner Co**. Click **Next**.
9. On the Add Deployment Flow - Summary page, select **Deploy**. Click **Finish**.
10. On the Connector Prompts page, enter your iSeries user profile name and password. Click **OK**.

PCML sample: Start the instance

Follow these steps in the iSeries Connect configuration tool to start your instance:

1. Click the **Instances** tab.
2. On the Manage Connect Instances page, select **sampInst**. Click **Start**.
3. On the Start Instance page, select all components. Click **Start**.

Note: Starting your instance may take some time. Do not continue until all components show a status of ″Started.″

4. Click **Close**.

PCML sample: Test the sample configuration

Use the Drive Connect sample application to simulate a ProfileRequest and an OrderRequest.

> **Note:** The Drive Connect application can also simulate a PunchOutSetupRequest. This sample is configured only to test the ProfileRequest and OrderRequest.

To test the ProfileRequest request, follow these steps in the iSeries Connect configuration tool:

1. On the Manage Connect Instances page (click the **Instances** tab to view this page), select **sampInst**.
2. On the iSeries Connect Tools page (click the **Tools** tab to view this page), click **Test Drive Connect**.
3. On the Test Drive Connect page, click **Next**.
4. On the Request Type page, select **ProfileRequest**. Click **Next**.
5. On the Request Options page, select **No, I will specify the partner and provider information**. Click **Next**.
6. On the Select Provider page, select or enter these values:
   - **Provider**: Sample Provider Co.
   - **Provider Password**: `secret`

   Click **Next**.
7. On the Select Partner page, select **Sample Partner Co**. Click **Next**.
8. On the Request to Test page, click **Next**.
9. On the Send Request page, click **Send Request**. The Drive Connect application sends the request through your instance.
10. The Test Results page appears with the results of the request. See Sample ProfileRequest response below for an example of a successful response.

Sample ProfileRequest response

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.009/cXML.dtd">
<cXML
    payloadID="1030545632932.sampInst.1945764792@MYSYSTEM.IBM.COM" timestamp="2002-08-28T14:40:32+00:00">
    <Response>
        <Status code="200" text="OK"/>
        <ProfileResponse effectiveDate="2002-08-28T14:40:33+00:00">
            <Transaction requestName="ProfileRequest">
                <URL>http://MYSYSTEM.IBM.COM:7080/BtoB/sampInst/Ariba11</URL>
            </Transaction>
            <Transaction requestName="OrderRequest">
                <URL>http://MYSYSTEM.IBM.COM:7080/BtoB/sampInst/Ariba11</URL>
                <Option name="allowedOperation">new</Option>
                <Option name="attachments">No</Option>
            </Transaction>
        </ProfileResponse>
    </Response>
</cXML>
```

To test the OrderRequest request, follow these steps in the iSeries Connect configuration tool:

1. After you have completed running the ProfileRequest test, click **New Request** on the Test Results page.
2. On the Request Type page, select **OrderRequest:new**. Click **Next**.

3. On the Request Options page, select **No, I will specify the partner and provider information**. Click **Next**.
4. On the Select Provider page, select or enter these values:
   - **Provider**: Sample Provider Co.
   - **Provider Password**: `secret`

   Click **Next**.
5. On the Select Partner page, select **Sample Partner Co**. Click **Next**.
6. On the Items to Order page, you can specify a product to be included in the OrderRequest. Select **C1** and enter whatever data you want in the remaining fields. For example, Description - Blender, Price - 20.00, Qty - 1. Click **Next**.
7. On the Request to Test page, click **Next**.
8. On the Send Request page, click **Send Request**. The Drive Connect application sends the request through your instance.
9. The Test Results page appears with the results of the request. Here is an example of a successful response:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.009/cXML.dtd">
   <cXML
       payloadID="1030546137977.sampInst.1254236671@MYSYSTEM.IBM.COM" timestamp="2002-08-28T14:48:57+00:00">
       <Response>
           <Status code="200" text="OK"/>
       </Response>
   </cXML>
   ```
10. Display the spool file to which the back-end application wrote its output:
    a. On the iSeries command line, run this command:

    ```
    WRKSPLF user_profile_name
    ```

    where *user_profile_name* is the name of the user profile that you specified when you deployed the process flow.

    b. In the Work with All Spooled Files display, specify option 5 (Display) for the last spool file in the list. Here are the contents of an example spool file:

    ```
    ******In ORDERREQ  ***********
    Number of arguments, argc = 3
    Name of the program called, argv[0] = QCON200SAM/ORDERREQ
    The following are input parameters from PCML Connector
    Number of items to process is 1
    Input element [0]
       identifier = C1
       quantity = 1
       price = 20.000000
       description = Blender
    ```
11. On the Test Results page, click **Close**.

Chapter 6. Migrate iSeries Connect

See these topics for information about migrating iSeries Connect and its prerequisite software:

“Migrate the iSeries Connect product”
Describes migration from iSeries Connect 1.1 to 2.0.

“Migrate iSeries Connect middleware” on page 92
Provides information about migrating middleware (prerequisite products) between releases of iSeries Connect.

“Considerations for migrating instances for WebSphere Commerce Suite (WCS) support” on page 93
Lists considerations for migrating instances that use the WebSphere Commerce Suite (WCS) extensions.

Migrate the iSeries Connect product

You do not have to uninstall iSeries Connect 1.1 before you install version 2.0. Both iSeries Connect 1.1 and 2.0 can exist on the same system, and they can both be functioning at the same time.

iSeries Connect Version 2.0 deletes Version 1.0 (if installed) as part of the installation process. Version 1.0 instances and user data are preserved, but they cannot be directly migrated to version 2.0. Version 1.0 instances and user data must first be migrated using iSeries Connect 1.1.

Protocols

Only iSeries Connect instances using the cXML 1.1 or 1.2 protocols are migrated.

WebSphere Commerce Suite

The iSeries Connect 2.0 Migration wizard will prompt you to migrate your WebSphere Commerce Suite 5.1 instances to WebSphere Commerce Suite 5.4 or WebSphere Commerce Suite Business Edition 5.4 instances if one of these products is installed.

> **Note:** iSeries Connect instances that use WebSphere Commerce Suite (WCS) 4.1 do not appear as instances that are available to migrate in the configuration tool. These instances must first be migrated from WebSphere Commerce Suite 4.1 to WebSphere Commerce Suite 5.1 using iSeries Connect 1.1.

To migrate an instance using WebSphere Commerce Suite 4.1 to 5.1 in iSeries Connect 1.1, do the following:

1. Click the **Instances** tab.
2. Click **Migrate Configurations**.
3. Specify **Migrate instances to use new versions of MQ Series, WAS, or WCS**.
4. Click **Next**.
5. Specify **WebSphere Commerce Suite**.
6. Specify the WebSphere Commerce Suite 5.1 instance.
7. Click Next.
8. Click Finish.

See “Considerations for migrating instances for WebSphere Commerce Suite (WCS) support” on page 93 for additional WCS migration considerations.

Domino

iSeries Connect 1.1 instances that use Domino as the Web application server are not migrated.

HTTP Server

A Connect 1.1 HTTP server will be migrated if Connect 1.1 has been uninstalled. If Connect 1.1 is still installed, then a new HTTP server is created for the migrated instance.

Migrating 1.1 instances

Use the iSeries Connect configuration tool to migrate your iSeries Connect 1.1 instances to version 2.0. Start the migration process by doing the following:

1. Click the **Instances** tab. If you have not created any 2.0 instances and you have existing 1.1 instances, you are prompted to migrate your 1.1 instances or create a new instance. Follow the prompts to complete the migration.

If you already have created 2.0 instances or migrated a 1.1 instance, you will not be prompted to migrate your instances. If you are not prompted to migrate your instances, you can still migrate your instances by doing the following:

1. Click the **Instances** tab.
2. Click **Migrate Configurations** and follow the prompts to complete the migration.

If you migrate a 1.1 instance to version 2.0 and subsequently delete the 2.0 instance, your 1.1 instance can be migrated again unless you explicitly deleted it. A 1.1 instance can be deleted in the following ways:

- If iSeries Connect 1.1 is installed, you can use the iSeries Connect configuration tool.
- If iSeries Connect 1.1 is no longer installed, and you choose to migrate the 1.1 instance, the migration wizard will ask if you want to delete the 1.1 instance upon successful migration.
- If iSeries Connect 1.1 is installed when you migrate, and then subsequently uninstalled, you can choose to migrate the 1.1 instance. The migration wizard will not migrate the instance since it already has been migrated, but will ask if you want to delete the 1.1 instance.

Migrate iSeries Connect middleware

iSeries Connect runs in conjunction with several middleware products, such as IBM WebSphere Application Server, IBM MQSeries, and IBM WebSphere Commerce Suite. iSeries Connect requires certain levels of these products; however, new versions of these products may be released between versions of iSeries Connect.

To support these middleware changes, iSeries Connect provides a utility to upgrade middleware support.

> **Note:** You must upgrade middleware support at the instance level. If you have multiple instances, you must migrate each instance separately. If, for some reason, migrating the first instance fails, you have better recovery options because your other instances have not changed. Also, migration problem are easier to isolate when they affect only one instance.

Here is an overview of the process for migrating middleware in iSeries Connect:

1. iSeries Connect announces support for the new product. Check the product home page at http://www.ibm.com/servers/eserver/iseries/btob/connect/ for middleware support announcements.

   > **Note:** Do not upgrade the middleware products when a new version is released. iSeries Connect must be able to support the new version before you migrate the product, or iSeries Connect will

not work properly. See the middleware support announcement for details on whether you need to perform the middleware migration function or not. Some middleware upgrades may not require special migration steps for iSeries Connect.

2. Support for a new middleware version may require changes to the iSeries Connect product. Check the middleware support announcement for fixes you need to apply to iSeries Connect and any special instructions you must follow.
3. Use the iSeries Connect configuration tool to stop all your instances before you upgrade the middleware product.
4. Install the new version of the middleware product according to instructions included in the fix package or on the iSeries Connect home page.
5. Install support for the new middleware version with the iSeries Connect configuration tool. Click the **Instances** tab, and click **Migrate Configurations**.

   The Migrate Instance Configuration wizard starts. Specify the information that is appropriate to your environment, including which middleware product and instance you want to migrate.
6. If the migration is successful, migrate your other instances. If it is not successful, see “Troubleshoot iSeries Connect” on page 106 for information about troubleshooting—including how to report problems to IBM Support.

Considerations for migrating instances for WebSphere Commerce Suite (WCS) support

WebSphere Commerce Suite (WCS) 4.1 is not supported in iSeries Connect 2.0. Migration of WCS 4.1 to WCS 5.1 must be performed in iSeries Connect 1.1.

If you have iSeries Connect 1.1 with WCS 4.1 and install iSeries Connect 2.0 and WCS 5.4, you are not allowed to perform middleware migration from WCS 4.1 to WCS 5.4.

If you have iSeries Connect 1.1 with WCS 5.1 and install iSeries Connect 2.0 and WCS 5.4, then you must do the following:

1. Migrate the iSeries Connect 1.1 WCS 5.1 instances to iSeries Connect 2.0 with WCS 5.1.
2. Perform middleware migration on iSeries Connect to WCS 5.4.

When following the Migrate Instance Configuration wizard, select the iSeries Connect instance that is currently linked to the WebSphere Commerce Suite 5.1 instance. Then, select which WebSphere Commerce Suite 5.4 instance with which you want the iSeries Connect instance to be associated. Make sure that both instances are currently active.

Chapter 7. Customize iSeries Connect

See these topics for information about customizing iSeries Connect:

“Customize the iSeries Connect configuration tool”
Describes how to customize pages in the configuration tool.

“User defined protocols” on page 96
Documents customizing iSeries Connect protocol support.

“Customize WebSphere Commerce Suite” on page 96
Documents WebSphere Commerce Suite (WCS) extensions support that you can customize.

“Initialization and termination exit processing” on page 17
Provides information about creating and registering Java classes that are called when the flow manager starts or stops. You can use these classes to start and stop applications with which the flow manager interacts.

“Exit programs” on page 42
Describes provider and partner exit programs, which can be used to integrate iSeries Connect provider and partner services with another application.

“Create an outbound message handler” on page 99
Describes using an outbound message handler to support asynchronous messages in iSeries Connect.

iSeries Connect API Javadoc
Documents the iSeries Connect application programming interfaces (APIs).

iSeries Connect Customization Guide
Describes the iSeries Connect application programming interfaces (APIs) and contains information about creating user defined protocols.

Customize the iSeries Connect configuration tool

You can customize the appearance of the iSeries Connect configuration tool and specify other preferences.

Application preferences

In the Home tab of the configuration tool, click **Preferences**. You can specify preferences such as:

- The first page that appears when you load the tool
- The default instance
- Trace levels
- Cache expiration and validation mapping for the entity resolver

  The entity resolver locates XML validation documents. Protocol designers can use the entity resolver and XML validation documents to validate that XML documents are syntactically and semantically correct. The iSeries Connect Customization Guide contains additional information about creating your own protocol and extending iSeries Connect to support user defined protocols.

Customize tabs

Some of the tabbed sections of the configuration tool contain **Customize** functions:

- **Providers** and **Partners**: Use the **Customize** function to control which properties are gathered for providers and partners. You can add or remove pages and fields.
- **Catalog**: Use the **Customize** function to specify units of measure and currencies to be used in your catalogs. You can also customize the way products are displayed in the Edit Products wizard.

User defined protocols

iSeries Connect currently supports the cXML 1.1, and cXML 1.2 protocols. If your business scenario requries a different protocol, it is possible to extend iSeries Connect to support your own user defined protocol.

Creating protocols

To use the iSeries Connect configuration tool to create your protocol, click **Create and Edit Protocols** under the **Tools** tab. If you do not use the iSeries Connect configuration tool to create your protocol, place your protocol files in the /QIBM/UserData/Connect200/Protocols/*protocol_name* directory.

See the Connect for iSeries: Developer Resources for a sample of how to create a custom protocol. Instructions and sample files are included.

The iSeries Connect Customization Guide contains additional information about creating your own protocol. It is recommended that you work with a trained IBM Business Partner to develop a customized protocol.

Deploying protocols

To deploy a protocol to iSeries Connect, do the following:

1. Click **Create and Edit Protocols** under the **Tools** tab.
2. Select the protocol that you want to add and click **Publish**.
3. Select the protocol that you want to add and click **Add Support**.
4. Select the instance where you want to deploy the protocol under the **Instances** tab and click **Properties**.
5. Click **Protocols**.
6. Select the protocol group where you would like to add the protocol (for example click **Default**) and then click **Add Protocol**.
7. Select the protocol that you want to add and click **Next**.
8. Select the protocol requests for the protocol that you want to add. You may need to click on the triangle to view the protocol requests.
9. Click **Finish**.

If you change your deployed protocol, do the following to update your protocol:

1. Click **Manage Protocols** under the **Tools** tab.
2. Select the protocol and click **Update**.

> **Note:** If you had an update to or a deletion of an RMF, then you need to remove support for the protocol, add support for the protocol, and change the instance properties to add the protocol.

Customize WebSphere Commerce Suite

Customizing runtime services

Application connector documents and process flow models that are shipped with iSeries Connect to support integration with WebSphere Commerce Suite (WCS) do not require modification. However, there are several fields within the application connector documents for processing NewQuote and PurchaseOrder that can be customized to affect some of the runtime characteristics of the WCS extensions support. You can make these modifications by using the Application Connector and Process Flow editors in the iSeries Connect configuration tool (under the Tools tab).

Changing locale returned during NewQuote processing

For the cXML protocol, there is a field in the application connector document for NewQuote called productDescriptionLanguage. When the WCS extensions Java connector is constructing a cXML PunchOutOrderMessage to return a quote at the end of a remote catalog shopping experience, there is an XML element called Description which carries a language specification to indicate what language the description of the product is in. By default, the value en-US is sent when using the WCS application connector documents that are shipped with the system. A partner application can use this information to determine if it needs to perform any sort of textual translation on the product description as information is presented through their application. The value that the iSeries Connect product sends for WCS is governed by a configurable default field, which can be modified in the appropriate application connector document.

To change the language value sent in the PunchOutOrderMessage, follow these steps:

1. Copy the qwcsnewquote200.AppConnector file from the /QIBM/ProdData/Connect200/classes/connectors directory to /QIBM/UserData/Connect200/Commerce/*instance_name*/connector directory (where *instance_name* is the name of your instance.
2. In the iSeries Connect configuration tool, select your instance and click the **Tools** tab.
3. Click **Application Connectors**.
4. Select **Create a new application connector document**.
5. Change the default setting for FieldId ProductDescriptionLanguage to a value other than en-US.
6. Click **Save**.
7. Click **Add Flow** to deploy your changes.
8. Click **Update flow manager**.

Changing fields that affect PurchaseOrder processing

The WCS extensions uses a database table called QABECORDMP to maintain order information. One field (maxOrderMapEntries) that can be modified is used to control the size of the QABECORDMP database table. This field also controls whether the duplicate transaction detection processing of the WCS extensions support is used or note.

The default value of maxOrderMapEntries is 3000, which means the QABECORDMP table will grow to a maximum size of 3000. (Note that this table is also used to support OrderRequest update and OrderRequest delete processing, so a value of at least 3000 entries is suggested.) When the number of orders received during the life of the instance exceeds this number, entries are removed from the table using a First In First Out (FIFO) scheme. To turn off the duplicate transaction processing, you can set the default value for this field to 0. (Note that this action also turns off the delete and update functions.) If the default setting for this maxOrderMapEntries is modified, you must specify an integer value.

The respondAfterValidation field controls whether or not WCS extensions fully processes a purchase order prior to sending a response to the B2B trading partner. By specifying a value of `0` (the default value) in this field, WCS extensions will validate the incoming purchase order request, and also wait for a response from Websphere Commerce Suite indicating the success or failure of processing the received order. A response will not be sent to the B2B trading partner until this response is received from WCS.

If a value of 1 is specified in the respondAfterValidation field, then the WCS extensions support will cause a response to get sent to the B2B trading partner after initial validation of the purchase order has occurred, but before having received a response from Websphere Commerce Suite. This option could be used in cases where a large number of items are being processed in a single purchase order, and the B2B trading partner is experiencing timeouts due to the amount of time required to process the large orders. This option should only be used in rare cases as it is possible that some failure could occur after successfully validating the incoming purchase order, but before WCS is able to complete purchase order processing.

The other fields in the application connector document, which can be modified, pertaining to Purchase Order processing affect the billing and shipping address entries which are recorded in the WCS SHADDR database table as part of order processing. These fields can be thought of as eye-catchers for identifying these billing and shipping addresses as B2B transactions. In the fields orderShipToAddress, billToAddress, and itemShipping, there are subfields called firstName and lastName. By default, firstName is set to B2B and lastName is set to SHIPPING or BILLING. When orders are received from B2B trading partners, the firstName setting is recorded in the SAFNAME column and the lastName setting is set in the SALNAME column of the SHADDR table. These fields can be modified in the application connector document to cause different values to get recorded in the SHADDR table. The default value specified must be a valid value that WCS understands. These fields are limited to 30 characters in length.

To change fields that affect PurchaseOrder processing, follow these steps:

1. Copy the qwcsorderrequest200.AppConnector document from the /QIBM/ProdData/Connect200/Classes/Connectors directory to the /QIBM/UserData/Connect200/Commerce/*instance_name*/Connector directory (where *instance_name* is the name of your iSeries Connect instance).
2. In the iSeries Connect configuration tool, select your instance.
3. Click the **Tools** tab.
4. Click **Application Connectors**.
5. Select **Create a new application connector document**.
6. Change the default values for the fields that you want to change. This depends on whether you are attempting to change the duplicate transaction processing, address information, or both.
7. Click **Save**.
8. Click **Add Flow** to deploy your changes.
9. Click **Update flow manager**.

External URL

If the WCS server is behind a firewall, the firewall may expose a different URL for the WCS instance. To direct the partner’s (buyer’s) server to the correct URL, you can specify an external URL in the **External URL** field under the WebSphere Commerce Suite tab of the Instance Properties page.

For example, company XYZ has a WCS instance on a server named wcssys and this sserver is behind a firewall. The internal address for the server is wcssys.xyz.com, and the internal URL is https://wcssys.xyz.com/. Since the firewall causes the wcssys server to be seen from the outside as https://www.xyz.com/mywcs/, the External URL should be set to: https://www.xyz.com/mywcs/.

Allow WCS Partnumber or Manufacturer’s Partnumber

If you want the WCS Partnumber (SKU) or Manufacturer's Partnumber values to be recognized by iSeries Connect in the cXML SupplierPartId field on the NewQuote and OredrRequest flows, you must do the following:

1. Create the iSeries Connect catalog using the option to create the catalog from an existing database. A view table can be created in WCS that uses both the CATENTRY ProductReference Number (CATENTRY_ID) and the PARTNUMBER or MFPARTNUMBER. The iSeries Connect catalog can then be created based on this view table. When setting the iSeries Connect catalog mapping fields, the WCS PARTNUMBER or MFPARTNUMBER field in the view table should be linked to the iSeries Connect catalog's product number.
2. Under the WebSphere Commerce Suite tab of the Provider Properties page, specify the following values:

   Mapping Table
   The view table created in the WCS collection.

   Collection
   The name of the WCS instance. This is the same name as the WCD database collection.

   Internal Product Reference Field
   The field for the WCS product reference number from the WCS view table (CATENTRY_ID).

   External Part Number Field
   PARTNUMBER or MFPARTNUMBER.

Create an outbound message handler

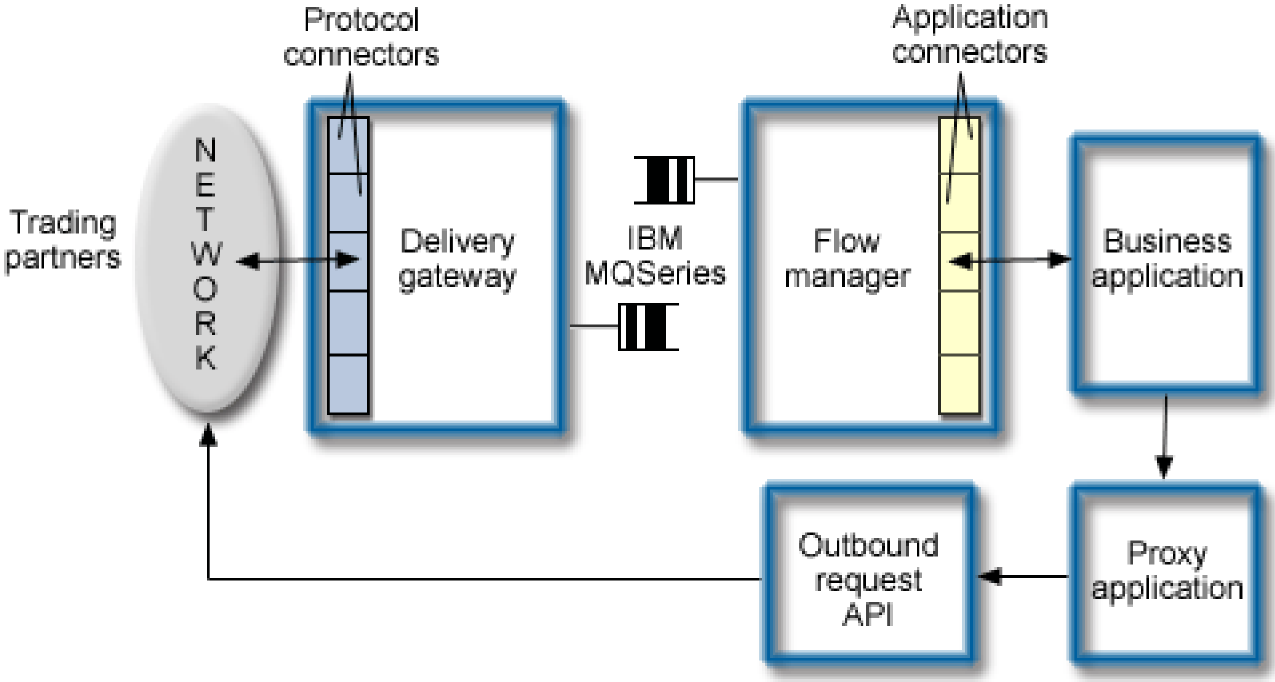
iSeries Connect provides a component called the **outbound message handler** that supports outbound requests—that is, requests that originate from iSeries Connect rather than from remote trading partners.

For conceptual information about outbound requests, see "Outbound request processing in the delivery gateway" on page 12.

To configure outbound message handling, perform these steps:

1. **Decide how outbound requests are initiated.** You have two options:
   - **From a step in your business process flow.** A process flow that is deployed in the flow manager uses a Java application connector instance. The process flow contains a step that builds the required outbound request fields and calls the outbound request API. It passes the field information that the delivery gateway uses to process the outbound request.
   - **From your business application.** Data from the request has been sent (through a flow manager Java connector instance) to the business application for order processing. The business application calls a proxy application. The proxy application gathers information (through parsing a message or querying the application) that is required for processing the outbound request. The proxy application then calls the outbound request API and sends the required information.

     The figure below shows the processing flow when an outbound request is initiated by the business application after an incoming request is received.



2. “Set up outbound request initiation”.
3. If necessary, “Grant authorities for outbound message handler” on page 101.
4. “Configure delivery methods for outbound message handler” on page 102.

Classpath requirements

These JAR files should be in your classpath to support calling the OutboundRequest API:

```
/QIBM/ProdData/Connect200/Gateway/gateway.jar
/QIBM/ProdData/Connect200/Gateway/gatewayAPI.jar
/QIBM/ProdData/Connect200/Tools/Runtime/util.jar
/QIBM/ProdData/Connect200/Classes/xalan220.jar
/QIBM/ProdData/Connect200/Classes/loggingapi.jar
/QIBM/ProdData/Connect200/Classes/logging.jar
/QIBM/ProdData/Connect200/Classes/log.jar
/QIBM/ProdData/Connect200/Classes/comibmconnect.jar
/QIBM/ProdData/Connect200/Classes/xerces321.jar
/QIBM/ProdData/Connect200/Classes/config.jar
/QIBM/ProdData/Connect200/Classes/bridge.jar
/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar
```

Set up outbound request initiation

To initiate an outbound request, you must construct an OutboundRequest API object and set the required fields. You then call the initiateRequest() API method (which throws an OutboundMessageException API exception if an error occurs).

Initiating the request from a business process flow

Here is the recommended scenario for initiating an outbound request from a step in your business process flow:

1. Define an additional connector step in the business process flow for the purchase order request. You must use a Java connector instance.
2. Construct an OutboundRequest object.

3. Set the required fields. You have two options:
   - Manually define the fields as parameters on the API call or through methods on the OutboundRequest object.
   - Automatically define the fields with a request token. A request token is usually generated on an incoming request during delivery gateway processing. It is stored in the sendable message header. You can retrieve this information with a flow manager connector that is defined to handle the outbound request processing.

   For more information on the required fields, see ″Sending Outbound Messages″ in the iSeries Connect Customization Guide .
4. Call the initiateRequest() method.
5. Define a new connector instance to support the outbound request. The connector maps fields from the back-end application into the outbound request.

   For example, for a StatusUpdateRequest, the delivery gateway sends a shell document in the request to the flow manager. Here is a portion of the shell document:

   ```
   <Request>
       <StatusUpdateRequest>
           <DocumentReference/>
           <Status/>
       </StatusUpdateRequest>
   </Request>
   ```

   You must map fields in your application to the DocumentReference and Status fields to complete the shell document. Here is an example of the same portion of code that has been filled in with data from the application:

   ```
   <Request>
       <StatusUpdateRequest>
           <DocumentReference
               payloadID="0c300508b7863dcclb_14999"/>
           <Status code="200" text="OK" xml:lang="en-US">Forwarded to supplier</Status>
       </StatusUpdateRequest>
   </Request>
   ```

Initiating the request from the proxy application

If you want a proxy application to initiate the request, the steps are the same as except that your application calls the OutboundRequest API. For more information on creating the proxy application (including code examples), see ″Sending Outbound Messages″ in the iSeries Connect Customization Guide .

Grant authorities for outbound message handler

Because of security restrictions in accessing the delivery gateway, the user profile under which the outbound message handler is initiated may need additional authorities. In these cases, you do not need to make any changes:

- The outbound request is initiated by a step in your business process flow that is running under the iSeries Connect instance user profile.
- The user profile under which your proxy application runs has *ALLOBJ authority.

If your user profile does not have *ALLOBJ authority, it is not recommended that you grant it this authority if you do not otherwise need it. Instead, you can authorize your user profile to the QIBM_QBEG_OUTBOUNDMESSAGES function through iSeries Navigator.

Note: If you are running a split-system configuration (that is, the delivery gateway is on a system that is remote to the flow manager system), both systems must have the same user profile defined if the API is initiated from the flow manager system. For example, if the user profile FRED calls the outbound request API from the flow manager system, a user profile FRED must also exist on the delivery gateway system.

To authorize your user profile for outbound message handling, perform these steps:

1. Under your system in iSeries Navigator, expand **Users and Groups**, and select **Users**.
2. In the user profile list that appears in the main window, double-click your user profile.
3. In the Properties dialog, click **Capabilities**.
4. Select the **Applications** tab.
5. In the **Access for** field, select **Host applications**.
6. In the resulting display, expand **QIBM_QBEX_CONNECT** and then **QIBM_QBEG_GATEWAY**.
7. Click the checkbox that corresponds to the **QIBM_QBEG_OUTBOUNDMESSAGES** function, and click **OK**
8. Click **OK**.

Configure delivery methods for outbound message handler

To set up the delivery methods, perform these steps in the iSeries Connect configuration tool:

1. On the Manage B2B Instances page, select your instance.
2. Click the **Providers** tab.
3. Select the provider and click **Protocol Association**.
4. In the Provider Protocol Associations page, click **Properties**.
5. Select the **Message Delivery Method Setup** tab.
6. Specify the values that are required by the protocol you are using. Here is an example when cXML is the protocol:
   - **Outbound message address (URL)**: Specify a URL provided by your trading partners to which they require messages sent. If you are using pull technology, you do not need to specify anything in this field.
   - **Protocol to use**: This value depends on the requirements of your protocol and trading partners:
     - To use push technology (which is when the delivery gateway sends or ″pushes″ the requests to the remote trading partners), type `HTTP` or `HTTPS` in the field.
     - To use pull technology (which is when the delivery gateway sends the request to a local mailbox database and the trading partners retrieve or ″pull″ the messages, type `Mailbox`. (This value is case-sensitive.) Note, cXML currently does not support this method.
7. Click **OK**.

Chapter 8. Manage iSeries Connect

See these topics for information about managing the iSeries Connect product and associated objects:

"Start and stop iSeries Connect"
Provides information on stopping and starting iSeries Connect and other system objects.

"Manage instances" on page 104
Lists considerations for managing your instances (and other resources) with the iSeries Connect configuration tool.

"Monitor B2B transactions" on page 105
Describes the Search Transaction History function

"Troubleshoot iSeries Connect" on page 106
Shows you how to find error information and how to troubleshoot request errors. This topic also provides information about program temporary fixes (PTFs) and getting support for iSeries Connect.

"iSeries Connect backup and recovery" on page 114
Lists the directories and objects you should include in your backup plan.

Start and stop iSeries Connect

See these topics for information about starting and stopping iSeries Connect and its prerequisite products.

Start the prerequisite programs

1. To start the QSERVER subsystem, run this command:
   ```
   STRSBS QSERVER
   ```
2. To start the OS/400 Host Servers, run this command:
   ```
   STRHOSTSVR *ALL
   ```

   Note: If you receive the message ″Host server daemon jobs unable to communicate using IPX,″ ignore it. This message only means that you do not have the IPX servers installed on your system—which is not a requirement for running iSeries Connect.
3. If you are using WebSphere Advanced 4.0 Application Server, start the subsystem by typing the following command:
   ```
   STRSBS SBSD(QEJBADV4/QEJBADV4)
   ```

   To verify that the WebSphere Application Server QEJBADMIN and QEJBMNTR jobs are started, run this command:
   ```
   WRKACTJOB
   ```

   The jobs are located under the QEJBADV4 subsystem.
4. If you are using WebSphere Single Server 4.0, start the subsystem by typing the following command:
   ```
   STRSBS SBSD(QEJBAES4/QEJBAES4)
   ```

   To verify that the WebSphere Application Server DEFAULT_SE job is started, run this command:
   ```
   WRKACTJOB
   ```

   The jobs are located under the QEJBAES4 subsystem.
5. To start the HTTP Server administrative instance, run this command:
   ```
   STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
   ```

Note: If your system runs with a coded character set identifier (CCSID) value of 65535, see "Change the coded character set identifier (CCSID)" on page 36 for critical information.

6. To check the status of the IBM HTTP Server administrative server, run this command:

   ```
   WRKACTJOB
   ```

   Under the QHTTPSVR subsystem, you should see two jobs in SIGW status. These jobs are named ADMIN.
7. To start the MQSeries subsystem, run this command:

   ```
   STRSBS SBSD(QMQM/QMQM)
   ```
8. To start the user work subsystem, run this command:

   ```
   STRSBS SBSD(QSYS/QUSRWRK)
   ```

Start iSeries Connect

In the iSeries Connect configuration tool, start one or more instances. In the Manage B2B Instances page, select your instance and click **Start**.

Stop iSeries Connect

In the iSeries Connect configuration tool, stop all running instances. In the Manage B2B Instances page, select an instance and click **Stop**. Repeat until all instances are stopped.

Stop the IBM HTTP Server administrative server

To stop the IBM HTTP Server administrative server, run this command from the iSeries command line:

```
ENDTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
```

Occassionally, you may need to stop and restart the HTTP Server administrative server. Do not use the restart function of HTTP Server to do this. Follow these steps instead:

1. Stop the HTTP Server with the End TCP/IP Server (ENDTCPSVR) command.
2. Use the Work with Active Jobs (WRKACTJOB) command to ensure the HTTP Server has stopped.
3. Start the HTTP Server with the Start TCP/IP Server (STRTCPSVR) command.

Manage instances

Manage your iSeries Connect instances with the iSeries Connect configuration tool. In addition to managing instances, the iSeries Connect configuration tool allows you to manage the following:

- registered providers
- registered partners
- catalogs
- application connectors
- process flow models
- protocols

The first page of each tab provides options for making changes to the configuration settings.

Considerations for managing iSeries Connect:

- **Instances**

  To change your instance configuration, use the Instance Properties pages (click **Properties** on the Manage Instances page). Note that if you change the protocol for your instance, protocol associations to registered providers and partners are lost. Run the Protocol Association wizard to associate the providers and partners with the new protocol. Changing protocols for your instance also affects any process flows that you have deployed. You must deploy a new set of process flows that support your new protocol.

- **Providers and partners**

  The iSeries Connect provider and partner services provide the ability to register an exit program. An exit program is additional code that can be connected to exit points for creating, updating or deleting providers or partners. Through a registered exit program, you can integrate the iSeries Connect provider and partner services with other applications. For example, the iSeries Connect WebSphere Commerce Suite (WCS) extensions use this mechanism to map providers and partners to WCS merchants and shoppers (customers).

  If you change provider or partner data in WCS or other application that is registered as an exit program, the changes are not automatically propagated to iSeries Connect. You must then update your provider and partner data with one of these methods:

  – Use the iSeries Connect configuration tool to manually update the provider and partner data.

  – Use the iSeries Connect provider and partner application programming interfaces (APIs) to make the corresponding update in the provider or partner registry. For more information, see the iSeries Connect Customization Guide .

- **Catalog**

  If the source from which you imported your catalog has changed, those changes are not automatically made to your B2B catalog. Columns in the catalog that are mapped from a source database (DB2, Domino, or WebSphere Commerce Suite) cannot be edited. You must refresh your B2B catalog to incorporate these changes.

- **Application connectors and process flows**

  If you edit your application connector or process flow, you must redeploy it. For the changes to take effect, you must update the flow manager runtime data. Click **Update flow manager** under the Deployment tab, or use the Manage Instances page to stop and start the flow manager component for your instance.

- **Protocols**

  The iSeries Connect configuration tool allows you to do the following:

  – Create or edit protocols.

  – Add a protocol to iSeries Connect.

  – Add a protocol to an instance.

  For more information on creating protocols, see the iSeries Connect Customization Guide .

Monitor B2B transactions

The Search Transaction History utility is a browser-based implementation of the B2B Activity Monitor (a part of iSeries Navigator - Management Central). It lets you monitor B2B requests that are received by iSeries Connect. You can search requests by instance, status, request, and other search criteria.

To enable and use the search tool, follow these steps:

1. Get the appropriate fixes for the B2B Activity Monitor. See the product Web site for information about fixes.
2. In the iSeries Connect configuration tool, click the **Instances** tab, and then click **Properties**.
3. Click the **Tracing** tab.
4. On the Instance Properties - Tracing page, select **Log transactions to Management Central consolidated system**. Click **OK**.

Any new B2B transactions that occur are now logged. You can then use the Search Transaction History function to search the transactions. Note that transactions that occurred before you enabled transaction logging will not appear.

5. On the Manage B2B Instances page, make sure your instance is running. If not, start it by selecting the instance and clicking **Start**.
6. On the Manage B2B Instances page, click **Search Transaction History**.
7. On the Search Transaction History page, enter your search criteria and click **Search Now** to view the results.

Troubleshoot iSeries Connect

If you encounter problems while using iSeries Connect, see these topics for information about determining the source of the problem:

- "Find error information".
- "Troubleshoot request errors" on page 107.
- "Diagnose and solve iSeries Connect problems using trace" on page 108.
- "Apply program temporary fixes (PTFs) for iSeries Connect" on page 113.
- "Get support for iSeries Connect" on page 114.
- "Collect useful data for an authorized program analysis report" on page 114.

See iSeries Connect Support Information for additional service and support information.

Additionally, you may want to check the iSeries Connect newsgroup to post your problem or read postings from other users.

Find error information

If you are having problems with iSeries Connect, check these message queues or files for additional error information:

QSYSOPR message queue
Message queue that contains fatal errors that cause the delivery gateway or flow manager to stop running.

B2B message queue
Message queue that is created for each instance. The queue contains these messages:

- Informational messages, such as the time that the flow manager started and what parameters were used while starting.
- Warning messages for situations that are not normal, but would not cause problems, such as unrecognized commands.
- Error messages that explain information about specific errors. For example, an exit not completing successfully, or no flow definition found for a specific request.
- Fatal messages that describe problems that prevent the flow manager from continuing. For example, a user initialization exit failing or the queue manager not started.

Message files
Files that are created for the same messages that are written in the B2B message queue. Separate files are written for the flow manager and delivery gateway:

- **Delivery gateway** message files are name GW_Message_*date*.log (where *date* is the date and time that the file was created). The message files are located in the /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs directory (where *instance_name* is the name of your instance).

- **Flow manager** message files are named FM_Message_*date*.log (where *date* is the date that the file was created). The message files are located in the /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs directory (where *instance_name* is the name of your instance).

You can view these files in the Instance Properties - Tracing page of the iSeries Connect configuration tool.

The message file name has a date and time stamp that is switched at midnight. You may wish to periodically delete or archive old message log files.

Audit files
Extensible Markup Language (XML) files that are created in the iSeries integrated file system as records. These files describe the progress of each individual request. You can use the audit log to determine how far a specific request got and if it failed, the reason it failed. Separate audit records are written for the delivery gateway and the flow manager:

- **Delivery gateway** audit files are named GW_Audit_*date*.log (where *date* is the date and time that the file was created). The audit files are located in the /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs directory (where *instance_name* is the name of your instance).
- **Flow manager** audit files are named FM_Audit_*date*.log (where *date* is the date and time that the file was created). The audit files are located in the /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs directory (where *instance_name* is the name of your instance).

You can view these files in the Instance Properties - Tracing page of the iSeries Connect configuration tool.

The audit file name has a date and time stamp that is switched at midnight. You may wish to periodically delete or archive old audit log files.

Troubleshoot request errors

To troubleshoot request errors, you can find out how far a request got by using the audit data or determine what happens to requests if the delivery gateway or flow manager goes down.

Determining how far a failed request was processed by using the audit data

To determine how far a request has progressed, look at the logged audit data. Each audit record has the following general format:

```
<Record>
  <AuditVersion> Audit version </AuditVersion>
  <Timestamp> Timestamp </Timestamp>
  <AuditPoint> Audit point </AuditPoint>
  <AuditType> Audit type </AuditType>

  ... Additional audit data ...

</Record>
```

where this information is shown:

- **Audit version** identifies the version of Connect for iSeries that created the audit record. For example, 2.0.
- **Timestamp** is the date and time that the audit record was created.
- **Audit point** and **audit type** are used to identify the different points in Connect for iSeries processing at which an audit record can be created.

Audit records with an audit type of FRAMEWORK are used to record the general flow of a request and are written at these audit points (where *x* is a number that identifies the flow that made the audit record and *y* is a number that indicates how many times within a flow that a particular audit point occured):

- GW_START: In the delivery gateway after a request is received.
- GW_SEND_*x*_*y*: In the delivery gateway before a request is sent to the flow manager.
- FM_START_*x*_*y*: In the flow manager after a request is received from the delivery gateway.
- FM_SEND_*x*_*y*: In the flow manager before a response is sent to the delivery gateway.
- GW_RECEIVE_*x*_*y*: In the delivery gateway after a response is received from the flow manager.
- FM_END_*x*_*y*: In the flow manager after a flow is complete.
- GW_END: In the delivery gateway before a response is sent to the originator.

Audit records with an audit type of DATA are used to record detailed information about a request and are written at these audit points (where *x* is a number that identifies the flow that made the audit record and *y* is a number that indicates how many times within a flow that a particular audit point occured):

- GW_INBOUND_*x*_*y*: In the delivery gateway before a request is sent to the flow manager, but after the request is converted to XML.
- GW_OUTBOUND_*x*_*y*: In the delivery gateway after a response is received from the flow manager.
- GW_RESPONSE_*x*_*y*: In the delivery gateway before a response is sent to the originator.

Audit records with an audit type of STEP_*x*_*y*_*stepcount*_*stepname* (where *x* is a number that identifies the flow that made the audit record, *y* is a number that indicates how many times within a flow that a particular audit point occured, *stepcount* is the number of the step defined in your process flow, and *stepname* is the name of a step that you defined in your process flow) are used to record each step in a flow and are created in the flow manager after each step in a flow.

A unique ID is associated with each request that you can use to match audit records for the same request.

Determining what happens to requests if the delivery gateway goes down:

If the delivery gateway goes down while requests are still in progress, the flow manager continues to process them. The delivery gateway listens to responses that are sent to the queue; however, these responses are not removed from the queue, because the delivery gateway is not active. When the delivery gateway restarts, any responses are taken off of the queue. Since the original requester information is no longer available, these responses are discarded.

Determining what happens to requests if the flow manager goes down:

If the flow manager goes down, the delivery gateway will timeout while waiting for the response and it generates a response to the originator to indicate that a timeout has occurred. The delivery gateway attempts to recall any requests that have not been started by the flow manager.

Diagnose and solve iSeries Connect problems using trace

Trace is intended to aid IBM Service in diagnosing and solving product problems and is not generally intended for use by customers. However, a service representative may ask you to gather trace logs to help detemine the source of a problem.

This topic provides information about these traces:

- Delivery gateway and flow manager (See 109)
- PCML (See 109)
- JDBC (See 109)
- Configuration application (See 110)

Tracing the delivery gateway and the flow manager

Turn tracing on and off for the delivery gateway and the flow manager by following these steps:

1. “Start the iSeries Connect configuration tool” on page 39.
2. Click the **Instances** tab.
3. On the Manage Connect Instances page, select your instance and click **Properties**.
4. Click the **Tracing** tab. Specify the trace settings you want. Click **Yes**, the click **OK** to enable tracing to enable tracing any time after the delivery gateway or flow manager has started.

To display trace files for the component you are tracing, click **View Trace File** in the appropriate component’s tracing settings.

You can also view the trace files themselves. If you select to turn trace on for the delivery gateway or flow manager, the trace files are written to the /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs directory (where *instance_name* is the name of your instance). For the delivery gateway, the trace files are named with the format GW_Trace*n*.log (where *n* is a number). For the flow manager, the trace files are named with the format FM_Trace*n*.log (where *n* is a number).

> **Note:** Trace records are written to a file. The names of trace files contain a number, with the number 1 representing the newest trace file, for example, FM_Trace1.log. When a trace file reaches the maximum file size, it is closed and the number in the file name is incremented by one. Thus, FM_Trace1.log becomes FM_Trace2.log, and new trace records are written to FM_Trace1.log. When the maximum number of files is reached, the oldest file is deleted.

Tracing PCML

If you encounter problems with a PCML connector, you can use the iSeries Connect configuration tool to enable PCML tracing.

1. In the Instances tab, select your instance and click **Properties**.
2. In the Instance Properties page, click the **Flow Manager** tab.
3. Under **Java Virtual Machine: System Properties**, add a system property with this value:
   ```
   com.ibm.connect.flowmanager.connector.TracePCML=true
   ```

The trace output is written to a QPRINT file under the QBEFSRVR job for that instance. Refer to the *IBM Toolbox for Java* documentation for debugging PCML and the com.ibm.as400.data.PcmlMessageLog class for more information about the PCML trace.

Note that PCML tracing logs output for all applications that use PCML, including the iSeries Connect framework. Therefore, the trace logs may contain information about applications other than your PCML connector.

Tracing JDBC

Both the IBM Developer Kit for Java and IBM Toolbox for Java JDBC drivers provide user tracing. You turn on JDBC tracing by including a connection property on the JDBC application connector JDBC URL. Refer to the documentation for those JDBC drivers for more information.

The IBM Developer Kit for Java JDBC driver provides an additional, more comprehensive JDBC trace. See the JDBC Driver FAQ for more information. You can enable this tracing for the iSeries Connect flow manager to trace all flow manager and JDBC connector activity.

Perform these steps in the iSeries Connect configuration tool to enable JDBC tracing:

1. In the Instances tab, select your instance and click **Properties**.
2. In the Instance Properties page, click the **Flow Manager** tab.
3. Under **Java Virtual Machine: System Properties**, add two new properties:
   - `jdbc.db2.trace`
     For the values of the jdbc.db2.trace system property, refer to the documentation for the QIBM_JDBC_TRACE_LEVEL environment variable.
   - `jdbc.db2.trace.config`
     This system property specifies the location for the trace output. Refer to the information on the QIBM_JDBC_TRACE_CONFIG environment variable for more information about the format of the jdbc.db2.trace.config system property.

Note that JDBC tracing logs output for all applications that use the IBM Developer Kit for Java JDBC driver, including the iSeries Connect framework. Therefore, the trace logs may contain information about applications other than your JDBC connector.

Tracing the configuration application

Enable tracing for the configuration application by doing the following:

1. Click the **Home** tab.
2. Click **Preferences**.
3. Click **Application** and select the trace levels.

   If you select to enable trace for the configuration application, the configuration trace files are written to the /QIBM/UserData/Connect200/Commerce/Servlet/Logs directory. The trace files are named config_trace*n*.log. You can select to enable any of these levels of trace: informational, warning, error, or fatal error.
4. Click **OK** to automatically enable or disable the trace.

   Note: The configuration application is not associated with a particular instance.

WebSphere Commerce Suite extensions runtime return codes

Use the information in this table to help diagnose problems with the WebSphere Commerce Suite runtime services. See “WebSphere Commerce Suite considerations for iSeries Connect” on page 23 for additional troubleshooting information.

Return code	Description
5003	The JDBC connection to the database failed. Check the iSeries Connect servlet traces for additional information.
5015	The provider is not found in the iSeries Connect configuration. Make sure that the provider information in inbound XML request matches the configured provider protocol information.
5019	The partner is not found in the iSeries Connect configuration. Make sure that the partner information in inbound XML request matches the configured partner protocol information.
5031	The product number in Quote or Purchase Order is not found in the iSeries Connect catalog (for WebSphere Commerce Suite) defined for the provider.
5033	The WebSphere Commerce Suite Shopper password is not found in the iSeries Connect configuration.

Return code	Description
5041	An unexpected response from WebSphere Commerce Suite. Make sure that the WebSphere Commerce Suite WebSphere Application Server instance is started. If the problem persists, contact IBM Service.
5046	An internal error getting the partner or provider API object. Contact IBM Service.
5047	The PingServer command to WebSphere Commerce Suite server failed. Make sure that the WebSphere Commerce Suite instance is active. If the problem persists, contact IBM Service.
5081	An internal error processing Address for order item shipping or billing information occurred. Contact IBM Service.
5082	A WebSphere Commerce Suite Address entry representing Purchase Order Request's shipping or billing address not created or found.
5083	An unexpected response from WebSphere Commerce Suite occurred. Make sure that the WebSphere Commerce Suite WebSphere Application Server instance is started. If the problem persists, contact IBM Service.
5101	An internal error occurred. No order items are found in the Purchase Order Request. Contact IBM Service.
5102	No order items were found in the Purchase Order Request. Check the XML document received from the iSeries Connect delivery gateway.
5103	The login to WebSphere Commerce Suite for shopper or WebSphere Commerce Suite administrator failed. Make sure WebSphere Commerce Suite is active and that the shopper password or WebSphere Commerce Suite administrator ID and password information entered in the iSeries Connect configuration is correct.
5106	The WebSphere Commerce Suite OrderDisplay command failed. Check the iSeries Connect flow manager trace and WebSphere Commerce Suite logs for additional information.
5107	The WebSphere Commerce Suite B2BVerify-OrderProcess failed. Check the iSeries Connect flow manager trace and WebSphere Commerce Suite logs for additional information.
5108	The WebSphere Commerce Suite Shopper password was not found in the iSeries Connect configuration.
5109	An unrecognized response from WebSphere Commerce Suite occurred. Make sure the WebSphere Commerce Suite instance is started. If the problem persists, contact IBM Service.
5110	An Order Request type is not valid. Valid request types for Order Request include: new, update, and cancel.
5111	An error processing Address Item for purchase order request shipping or billing information occurred. Examine the iSeries Connect file manager trace for additional information.

Return code	Description
5112	A duplicate Order request was received. Check the QABECORDMP table in the iSeries Connect database collection for a matching transaction ID.
5115	A time out in processing a WebSphere Commerce Suite command during order request processing occurred. Check WebSphere Commerce Suite for errors.
5116	A WebSphere Commerce Suite Administrator ID and password was not defined in the iSeries Connect instance. This information is required for processing WebSphere Commerce Suite order update or order cancel requests.
5117	An order update or cancel request was received and no previous order was found in iSeries Connect. Check the QABECORDMP table in the iSeries Connect database collection for a matching transaction ID.
5118	An error processing a B2BOrder request occurred. Check the iSeries Connect flow manager trace and WebSphere Commerce Suite logs for additional information.
5551	An internal error (invalid parameters) occurred. Contact IBM Service.
5552	An internal error (already logged on to WebSphere Commerce Suite) occurred. Contact IBM Service.
5553	The logon to WebSphere Commerce Suite failed. Make sure that WebSphere Commerce Suite is active and that the registered user exists in WebSphere Commerce Suite.
5554	An internal error (an exception occurred in the WebSphere Commerce Suite logon attempt) occurred. Contact IBM Service.
5555	An internal error (an exception occurred in the WebSphere Commerce Suite logon attempt) occurred. Contact IBM Service.
5556	An internal error (an exception occurred in the WebSphere Commerce Suite logon attempt) occurred. Contact IBM Service.
5557	An invalid user register type was detected. WebSphere Commerce Suite shoppers must be registered users and the WebSphere Commerce Suite Administrator ID must have CSR level authority.
5200	A partner cookie was not received in the PunchoutSetup request.
5201	A PostBack URL was not received in the PunchoutSetup request.
5202	A Punchout edit or inspect request is missing the product reference number or ship to reference number.
5203	Product items were not included in the punchout edit or inspect request.
5204	An unknown Punchout setup request type was received. Types expected are: new, edit, or inspect.
5205	An exception occurred during the PunchoutSetup processing. Contact IBM Service.

Return code	Description
5207	An internal error while processing the PunchoutSetup request occurred. Contact IBM Service.
5208	An unexpected response from WebSphere Commerce Suite was received. Make sure that the WebSphere Commerce Suite WebSphere Application Server instance is started. If the problem persists, contact IBM Service.
5209	An exception occurred during PunchoutSetup processing. Contact IBM Service.
5210	An internal error occurred while processing the PunchoutSetup request. Contact IBM Service.
5212	An internal error occurred while getting a generic requisitioner for the PunchoutSetup request. Contact IBM Service.
5300	The provider cookie provided on the Quote request does not match the previous punchoutsetup request.
5301	A WebSphere Commerce Suite shopper password was not found in the iSeries Connect configuration.
5302	A provider cookie or quote number was not received in the new Quote extrinsic data.
5303	The extrinsics data was not received in the new Quote request.
5304	An internal error getting the WebSphere Commerce Suite command object during quote processing occurred. Contact IBM Service.
5305	An internal error processing Quote occurred. Contact IBM Service.
5306	An unexpected response from WebSphere Commerce Suite was received. Make sure that the WebSphere Commerce Suite WebSphere Application Server instance is active. If the problem persists, contact IBM service.
5307	The Quote item is missing a quote reference number, product reference number, currency, or product description.

Apply program temporary fixes (PTFs) for iSeries Connect

If you cannot determine the cause of a problem, ensure that you have the latest fixes for iSeries Connect.

See Connect for iSeries Support Information for a current list of iSeries Connect PTFs.

You should also have the latest fixes for these products:

- IBM HTTP Server
- IBM MQSeries
- IBM WebSphere Application Server
- Lotus Domino
- IBM Toolbox for Java
- IBM DB2 Universal Database (UDB)
- IBM WebSphere Commerce Suite

See Connect for iSeries Support Information for links to support information for these products.

Get support for iSeries Connect

If you cannot determine the cause of a problem, contact IBM Support through the channels defined in your support contract. See IBM Support for technical documents and support information.

Collect useful data for an authorized program analysis report

To collect data for an authorized program analysis report (APAR), include a complete description of the problem, and send these log files, where *instance_name* signifies the name of your instance. If you have a split system, some of the logs files may exist on the delivery gateway system.

- /QIBM/UserData/HTTPA/admin/logs/jvmstdout.txt
- /QIBM/UserData/HTTPA/admin/logs/jvmstderr.txt
- /QIBM/UserData/Connect200/Connect_Registry.xml
- /QIBM/UserData/Connect200/Commerce/servlet/logs/config_trace*n*.log
- /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs/FM_trace *n*.log
- /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs/GW_trace *n*.log
- /QIBM/User/Data/Connect200/Commerce/*instance_name*/Logs/*instance_name*stdout.txt (WebSphere stdout log is only available if you are using WebSphere for the delivery gateway)
- /QIBM/User/Data/Connect200/Commerce/*instance_name*/Logs/*instance_name*stderr.txt (WebSphere stderr log is only available if you are using WebSphere for the delivery gateway)
- /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs/GW_Audit_ *timestamp*.log (each has its own unique time stamp)
- /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs/GW_Message_ *timestamp*.log (each has its own unique time stamp)
- /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs/FM_Audit_ *timestamp*.log (each has its own unique time stamp)
- /QIBM/UserData/Connect200/Commerce/*instance_name*/Logs/FM_Message_ *timestamp*.log (each has its own unique time stamp)

If you are encountering an exception, also send the text of the exception, including the stack trace.

iSeries Connect backup and recovery

To ensure that you can recover from catastrophic errors, you should periodically back up iSeries Connect. For more information on planning a backup and recovery strategy, see:

- V5R1 Backup, Recovery, and Availability
- V5R2 Backup, Recovery, and Availability

Backing up the product

To back up the iSeries Connect product, use the Save Licensed Program (SAVLICPGM) command. Here is an example:

```
SAVLICPGM LICPGM(5733CO2) DEV(*SAVF) SAVF(MYLIB/MYFILE)
```

Backing up user-defined data

Back up these user-defined iSeries Connect objects:

- ***INSTANCE_NAME* library**
  The library associated with your instance contains SQL collection and configuration objects for the instance, such as message queue and job description. You can use the Save Library (SAVLIB) command to create a backup. Here is an example:

  ```
  SAVLIB LIB(MYINST) DEV(*SAVF) SAVF(MYLIB/MYFILE)
  ```

  > **Note:** If you have migrated a 1.1 instance to version 2.0, the library name is not equivalent to the instance name. To determine the name of your instance library, view the /QIBM/UserData/Connect200/Commerce/*instance_name*/*instance_name* _registry.xml file (where *instance_name* is the name of your instance).
  >
  > Look for the B2BInstanceLibrary element. The path attribute contains the name of the instance library. Here is an example:
  >
  > ```
  > <B2BInstanceLibrary Path="MYINST">
  > ```

- **/QIBM/UserData/Connect200 directory**
  Contains user-defined data, such as global preferences data. Use the Save (SAV) command to create a backup of the directory. Here is an example:

  ```
  SAV DEV('/MYLIB.LIB/MYFILE.FILE') OBJ(('/QIBM/UserData/Connect200'))
  ```

- **/QIBM/UserData/Connect200/Commerce directory**
  Contains user-defined data, such as instance lists and theme data.
- **/QIBM/UserData/Connect200/Commerce/*instance_name* directory**
  Contains user-defined data, such as log files and process flows.
- **/QIBM/UserData/Connect200/Commerce/Protocols directory**
  Contains configuration files for supporting a user defined protocols.

Chapter 9. Reference

See these references for more information about iSeries Connect:

“API Javadoc”
Documents the iSeries Connect application programming interfaces (APIs).

“iSeries Connect Customization Guide”
Describes the iSeries Connect application programming interfaces (APIs) and contains information about creating user defined protocols.

“Samples for iSeries Connect”
Provides links to sample applications that demonstrate the features of iSeries Connect.

API Javadoc

The iSeries Connect API Javadoc documents the iSeries Connect application programming interfaces (APIs).

iSeries Connect Customization Guide

The iSeries Connect Customization Guide contains information about:

- Creating your own user defined protocol.
- Extending and customizing the iSeries Connect framework through application programming interfaces (APIs).

Samples for iSeries Connect

“Run the PCML verification sample” on page 85
The PCML verification sample ships with the Connect for iSeries product. The sample shows you how to configure an iSeries Connect instance from start to finish. Running the sample is a good place to start for learning how to configure iSeries Connect.

“Test your instance” on page 70
Use the Test Drive Connect utility in the iSeries Connect configuration tool to simulate requests from a B2B marketplace.

Connect for iSeries: Developer Resources
See this online resource for the most current sample applications for iSeries Connect.

Printed in U.S.A.