IBM Connect for iSeries 1.1

Java Connector Sample

September, 2001

Contents

License and disclaimer

This material contains IBM copyrighted sample programming source code ("Sample Code"). IBM grants you a nonexclusive license to compile, link, execute, display, reproduce, distribute and prepare derivative works of this Sample Code. The Sample Code has not been thoroughly tested under all conditions. IBM, therefore, does not guarantee or imply its reliability, serviceability, or function. IBM provides no program services for the Sample Code.

All Sample Code contained herein is provided to you "AS IS" without any warranties of any kind. THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU. IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES FOR ANY USE OF THE SAMPLE CODE INCLUDING, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOSS OF PROGRAMS OR OTHER DATA ON YOUR INFORMATION HANDLING SYSTEM OR OTHERWISE, EVEN IF WE ARE EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Top

Overview

The basic functions that a Java connector application must perform are:

- Retrieve the input fields passed to it by the flow manager
- Convert the input field into the form that is expected by the application
    - Process the input fields or pass them to another back-end application for processing.
- Return feedback to the flow manager

> **Note:** This sample program assumes a purchase order request. The order response does not include output fields. For this reason, this document focuses on input fields only. Future sample programs will illustrate the use of output fields when appropriate.

Top

Java connector skeleton

The Java connector application is a Java class that implements the JavaConnectorInterface. When the iSeries Connect flow manager determines that the request received from the marketplace must be processed by your Java connector application, it invokes the run() method of the JavaConnectorInterface that your Java connector application instantiates.

When coding a Java connector, use this class skeleton as your starting point:

```
import com.ibm.connect.flowmanager.interfaces.*;
import com.ibm.connect.flowmanager.metadata.Field;
import java.util.Enumeration;import java.util.Vector;

public class JavaConnectorSample1 implements JavaConnectorInterface
{
  public JavaConnectorResult run(ConnectorParm parms)
  {
    JavaConnectorResult result = new JavaConnectorResult();
    result.setReturnCode(0);
    return result;
  }
}
```

- **The run() method** is called by the flow manager. The run() method is the heart of your Java connector application. It must implement the logic of accessing the input and output fields provided in the ConnectorParm object.

- **The ConnectorParm object** provides a set of interfaces that your program can use to:

    - Instantiate a set of Field objects to represent the set of input and output fields defined in the input and output field specification. You define the input and output fields from the request and response documents (for example, cXML document) that your Java connector application will use, in the PCML or XML document specified in the Application Connector.

      > **Note:** This sample Java connector program only uses input fields.

    - Navigate through the input request document to retrieve the values for this particular request.
    - Convert the input values (all Strings) to other Java types (for example, ints and Floats) expected by the

back end application.

- **The JavaConnectorResult object** is returned by the run() method in your Java connector application to indicate the success or failure of its invocation.

Top

Input request static tree

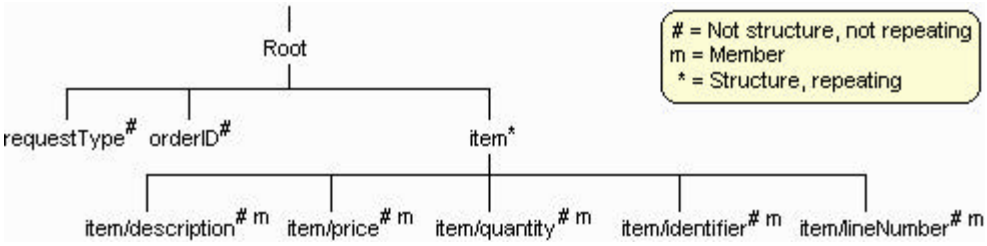
The main goal of your Java connector application is to process information received from a marketplace. To isolate the Java connector application from the different protocols that may come in from different marketplaces, the incoming request information is mapped from the protocol-specific format into the ConnectorParm object.

It is convenient to draw the incoming request static tree (as opposed to the runtime tree) to understand how to implement the navigation through the request document. Each node in the tree represents an input field in our sample application. The position of the field (node) in the static tree and its characteristics are important because the rules that you must follow to navigate the runtime tree and retrieve the field values depend on them.

Note: The terms *node* and *field* are used synonymously.

The static tree that represents the order request in the sample application is shown below.

Figure 1. Static tree for an order request



It is through this view that the Java connector application accesses the values in the request document.

This table summarizes the characteristics of the sample Java connector application's input fields.

Field Name	Structure	Repeating	Member
requestType	no	no	no
orderID	no	no	no
item	yes	yes	no
item/description	no	no	yes
item/quantity	no	no	yes
item/price	no	no	yes
item/identifier	no	no	yes
item/lineNumber	no	no	yes

Top

Define the interface between the flow manager and Java connector application

This interface definition to your application is provided in the form of an application connector document (AppConnector). The AppConnector is created by using the application connector editor (click **Application Connector**) under the Deployment tab in the iSeries Connect configuration tool.

The input used to create this application connector document is a document provided by you that defines a set of input and output fields. iSeries Connect supports three techniques for defining these fields: a PCML document, an XML document, or an ACDFieldSet document. In the sample Java connector application, a PCML document is used as input to define the set of input fields required by the Java connector program, JavaConnectorSample1. The sample PCML document JavaConnectorSample1.pcml is shown below:

```
<pcml version="1.0">
<!-- PCML use to define ACD for JavaConnectorSample1  -->
 <program name="JavaConnectorSample1" path="/xxx">
        <data name="incount" type="int" length="4" usage="input" />
        <data name="orderID" type="char" length="6" usage="input" />
        <data name="requestType" type="char" length="6" usage="input" />
        <struct name="item"  usage="input" count="JavaConnectorSample1.incount" >
          <data name="lineNumber" type="int" length="4" usage="input" />
          <data name="identifier" type="char" length="16" 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>
 </program>
</pcml>
```

Note: In the case of the Java connector, the PCML file is only used to define the input and output fields to the application, and not to call the program. The <program> tag is required, but the "path" attribute (which defines the actual program to be called) is only used by the PCML connector. The path attribute is displayed here only because it is required by the syntax.

The Field object

The ConnectorParm object is passed to your Java connector program in the run() method:

```
public JavaConnectorResult run(ConnectorParm parms)
```

Before retrieving a value from the request document, the connector program must acquire the associated Field object from the ConnectorParms. If the field is not a member of a structure, get the Field object using the getInputField() method. To obtain the Field objects for members of a structure, use the getInputFieldList() method.

The field "name" attribute is used by your Java connector application to retrieve input values. The `struct` and `repeating` attributes are important to determine how your Java connector program must retrieve an input value. They are the attributes that indicate whether or not the element is a structure and whether the element repeats in the tree. Struct elements are structures and data elements are not. Elements which have a count attribute in the PCML are considered repeating; those without a count attribute are not. A repeating field can appear a number of times in the request document.

In the example, there are three fields that are not members of a structure: requestType, orderID, and item. To acquire their associated Field object:

```
Field itemFld = parms.getInputField("item");
```

In the example, there are five Field object members of the item structure (under "item"), regardless of how many times the item structure may be repeated in a particular request document. To acquire their associated Field object:

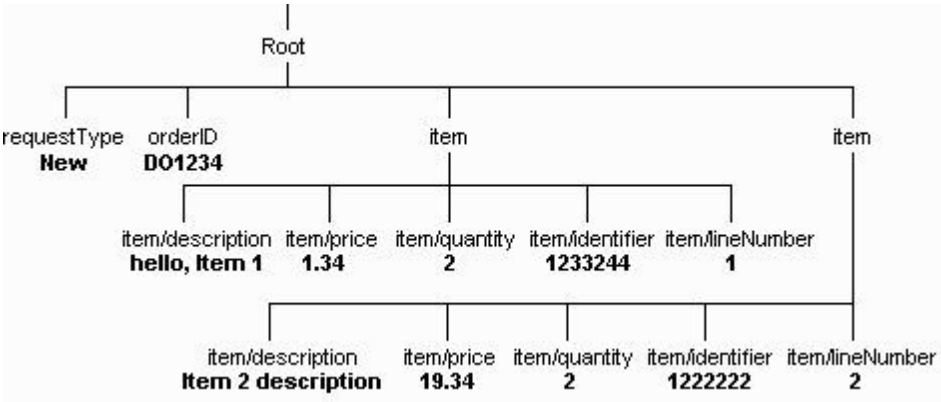
```
Vector memberflds = parms.getInputFieldList(itemFld);
```

Top

Navigate the runtime tree

At runtime, the tree looks like the following:

Figure 2. Runtime tree of the Java connector sample



The runtime tree is populated from the input request document. Only nodes at the end of a branch have values. You must navigate the tree until you arrive at the field that contains the value you want to extract.

iSeries Connect introduces the concept of a cursor, which is a pointer to a node in the tree. The cursor sets the starting navigation point in the runtime tree. It is implemented by the class MapCursor.

To extract the value of a field in the input request document, your Java connector application must have a cursor that points to the node above it. For example, to extract the requestType value from the request document, the connector program must have a cursor that points to the root node. For convenience, the ConnectorParms object always contains a default cursor that points to the root node.

Cursors are very important when the tree includes repeating structures. In the example, the request document contains two items. Since "item" is also a structure, each item node has identical nodes below it. When the connector program retrieves a value for item/price, it must have a way to specify which item/price it wants. It does this by positioning a cursor on the item node above the item/price node that contains the value that it wants to retrieve.

Follow these rules to navigate the runtime tree to the location of the field to specify which value you want to extract:

1. Build the static tree of Field objects as shown in The Field object.
2. Acquire a cursor associated with the node above the field you want to get to (only for structures).
3. Bind the corresponding Field object (the one you want to get to) to the cursor acquired in step 2.

Acquiring a cursor for "item" (step 2)

There are four steps that the connector program must perform to acquire a cursor to a particular node:

Tip: Acquire a cursor to a node that is a structure.

1. It has a MapCursor object that represents a cursor to the node above it. Because the default cursor (which always points to the root node) is available, any node can be reached through a number of recursions. This step is not necessary for nodes under the root. Because "item" is under the root, this step is skipped.

2. It has a Field object the represents the node/input field. See The Field object.
3. It binds the node to the cursor:

   ```
   parms.bindInfieldToCursor(itemFld);
   ```

4. It calls the getFieldAsStructure() method on the ConnectorParms object. This returns an array of MapCursor objects if the node is a repeating node, a single MapCursor object otherwise.

   ```
   MapCursor[] cursorlist = parms.getFieldAsStruct(itemFld);
   ```

Once the connector program has acquired the cursor, it is free to access the nodes below it.

Bind a field to a cursor for "item/price" (step 3)

It has been shown how the connector program can use cursors to identify a specific node in the tree for any input request. Now it is time to get the value. But before retrieving the value, the connector must "bind" the node it is interested in to the cursor, which must be positioned on the node above. This operation is performed with 2 steps.

1. It has a Field object that represents the node/input field. Call the getInputFieldList() method on the ConnectorParms object to get the Field object that represents the node/input field under "item". In the JavaConnectorSample1 sample, all Field objects (built static tree) are obtained at the beginning of the program:

   ```
   Vector memberFlds = parms.getInputFieldList(itemFld);
   ```

2. Call the bindInfieldToCursor() method on the ConnectionParms object:

   ```
   parms.bindInfieldToCursor(itemPriceFld,cursorlist[i]);
   ```

Although there is no specific reason to do this, it is necessary to prepare the object to retrieve the value. All items at the top level can be bound to the default cursor which is contained within the ConnectorParms object.

Top

Retrieve the value of a field

When the connector program has identified to the specific non-structure node which value it wants to extract, and has bound it to a cursor, it can retrieve the value from the request document. Depending on the data type of the field, it uses one of the following methods on the ConnectorParms object:

- getFieldAsInt
- getFieldAsByte
- getFieldAsDouble
- getFieldAsFloat
- getFieldAsBigDecimal
- getFieldAsByteArray
- getFieldAsLong
- getFieldAsShort
- getFieldAsBoolean
- getFieldAsString

These methods return an array of the objects. If there was only one value, the array contains only one item:

```
float [] itemPriceVal = parms.getFieldAsFloat(itemPriceFld);
```

Top

Install the JConSam1 package

1. Map a network drive to the root directory of your iSeries integrated file system.
2. Download the self-extracting, executable file jconsam1_110.exe.
3. Run the file to install the package. During installation, select to install the package to a unique directory in the integrated file system on your iSeries by modifying the location to save files.

   For example, if F: is your mapped drive to the integrated file system, it is recommended you specify `F:\connsam11` as the target directory. This directory is created for you if it does not already exist.

   Note: The directory connsam11 is referenced throughout this document as this unique directory that you specify.

When the installation is complete, you will have all the files for the Java connector application in a subdirectory called JConSam1. Here are the files:

- JavaConnectorSample1.java: Source code for the Java connector application.
- JavaConnectorSample1.class: Compiled Java class.
- JavaConnectorSample1.pcml: PCML file that defines the input fields for the sample JavaConnectorSample1 Java connector program.

In addition, the package includes the application connector and process flow documents that were generated using the iSeries Connect configuration tool. Here are the sample application connector and process flow documents:

- JavaConnectorSample1.AppConnector
- JavaConnectorSample1.ProcessFlow

Top

Configure iSeries Connect for JavaConnectorSample1

To configure iSeries Connect for the Java connector sample, perform the following steps:

1. Create a B2B instance. This table lists the field values that you should use:

Field	Value
B2B instance name:	JConSam1
Supplier URL for marketplace:	aribaorders
HTTP port	4083 (must be an unused port on your system)
Supplier DUNS	123222888
Supplier Id	123222888
Supplier Id Domain	DUNS
Supplier Logon Information	
Supplier Log-on Id	123222888
Domain	DUNS
Password/Shared Secret:	secret
Buyer Organization Id	321222888
Buyer Organization Id Domain	DUNS

2. Create the application connector and process flow documents necessary for this Java connector. You may use the ones included with the sample or see Appendix A for information about creating your own.

   If you choose to deploy the process flow using the AppConnector and ProcessFlow documents supplied with this sample, copy these sample files from the /ConnSam11/JConSam1 directory to the /QIBM/UserData/Connect110/JConSam1/Connector directory:
   - JavaConnectorSample1.AppConnector
   - JavaConnectorSample1.ProcessFlow

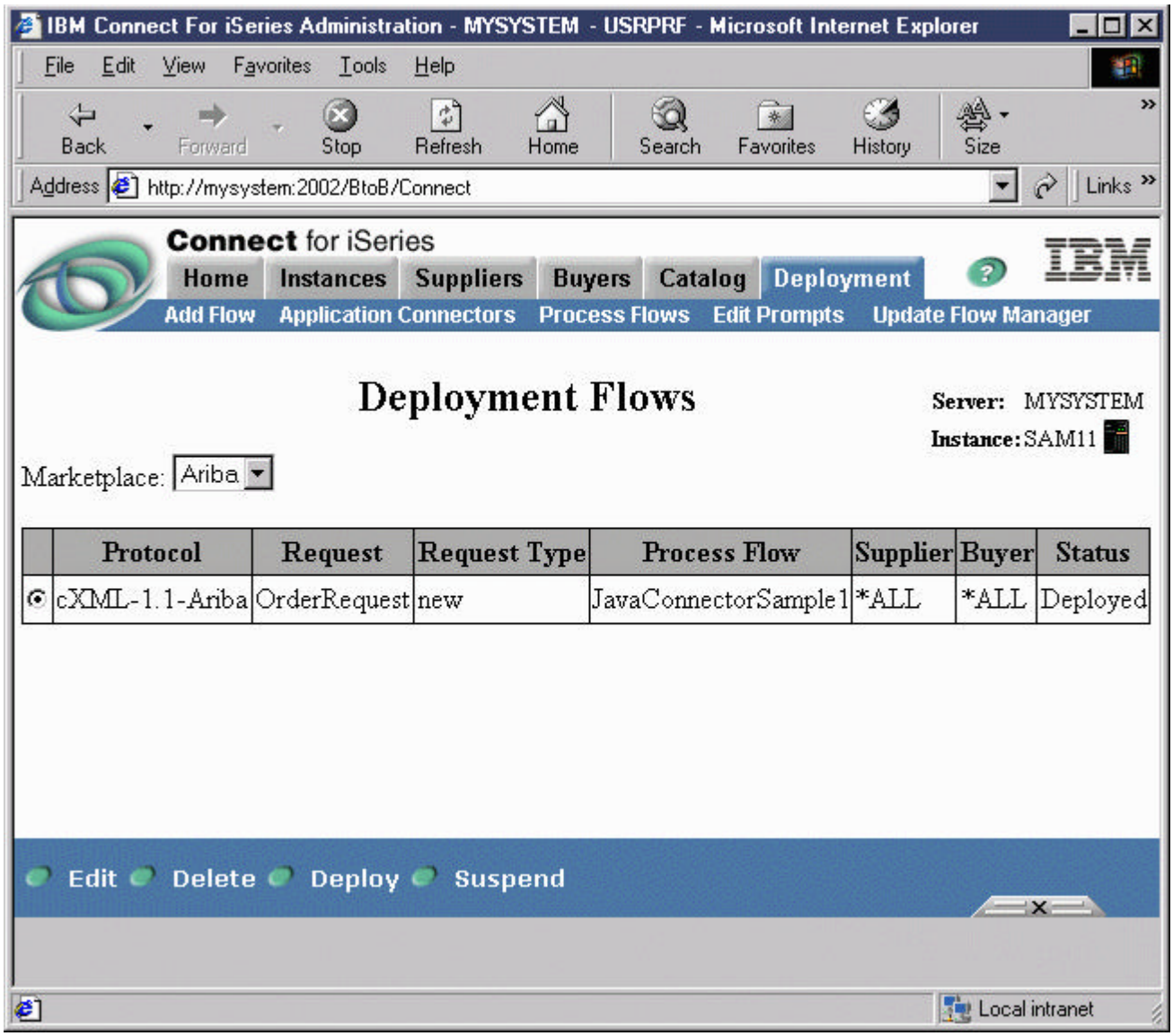
3. Deploy the process flow using the Deployment function in the iSeries Connect configuration tool.

   a. Click **Instances**, and then select your B2B instance name (JConSam1).
   b. Click the **Deployment** tab, and then click **Add Flow**.
   c. Select the protocol to use. Click **Next**.
   d. Select **OrderRequest/new** and **JavaConnectorSample1** from the drop-down list.
   e. Click **Next**.
   f. Select **All Suppliers** and **All Buyers**.
   g. Click **Finish** to deploy this flow.

   In the iSeries Connect configuration tool, the completed deployment look similar to Figure 3. Note that the status is "Deployed."

Note: If deployment is successful, the file RuntimeMeta.xml is created in the /QIBM/UserData/Connect110/JConSam1/Connector directory.

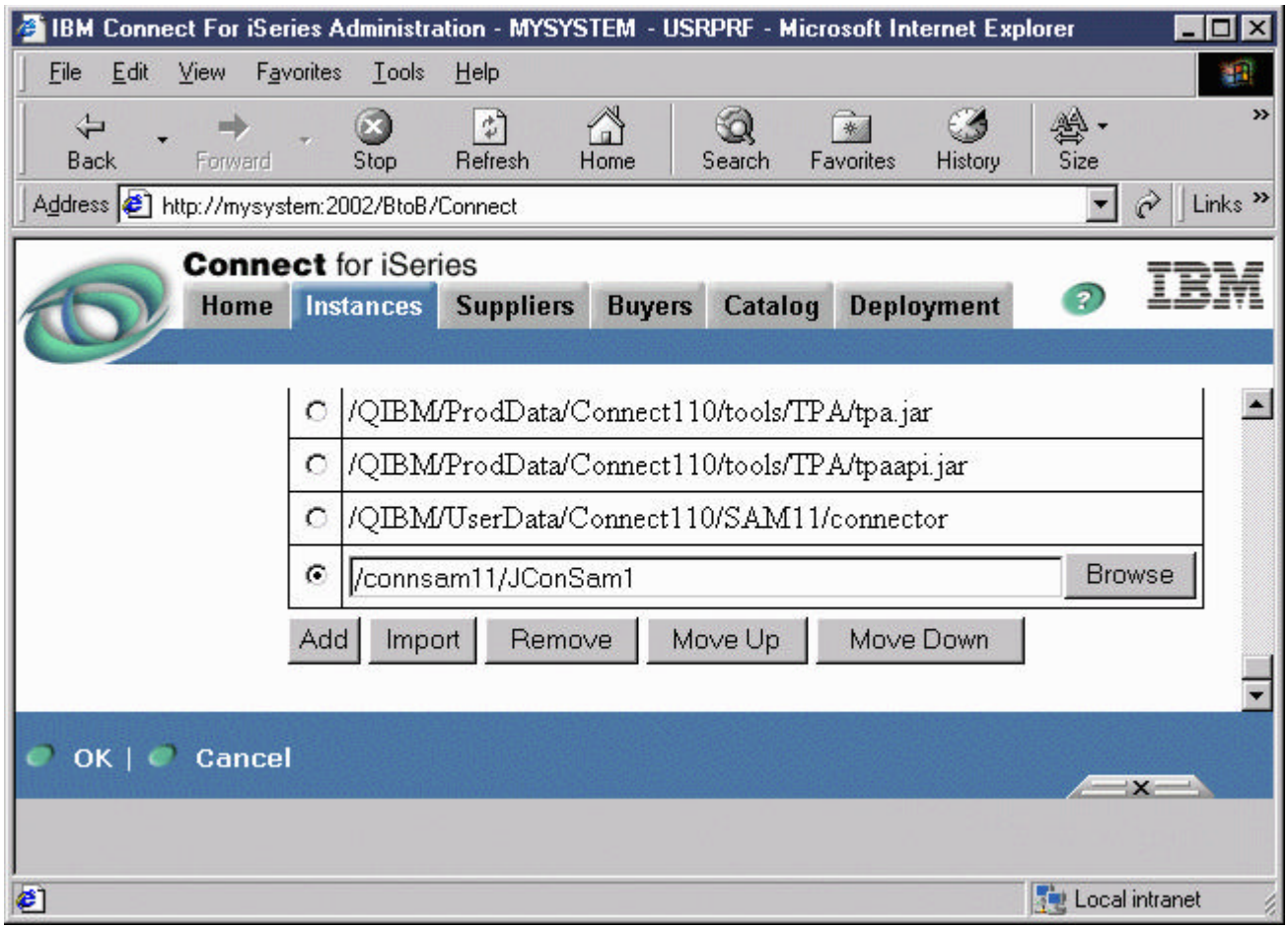
Figure 3. Deployed flow for the Java connector sample



4. Configure the path to your Java connector application:

    a. In the Manage B2B Instances page, select your instance and click **Properties**.
    b. Click the **Flow Manager** tab.
    c. At the end of the existing Java virtual machine classpath, add the path where JavaConnectorSample1.class. The sample uses the /ConnSam11/JConSam1 directory.

Figure 4. Update the flow manager classpath



5. Start your instance:

    a. Click the **Instances** tab.
    b. Select **JConSam1**.
    c. Click **Start**.
    d. Select all the instance components, and click **Start**.

Top

Test JavaConnectorSample1

Use the Drive Connect application to simulate an OrderRequest/new using your instance as configured for the Java connector sample. The Drive Connect application is integrated into the iSeries Connect configuration tool.

1. Click the **Instances** tab, and ensure that your instance is still selected.
2. Click **Test Drive Connect**.
3. Click **Next**.
4. Select **OrderRequest** on the Request Type page.
5. Click **Next**.
6. On the Request Options page, if your instance is set up for a supplier with a DUNS of 123222888 and a buyer with DUNS 321222888, select **Yes, use the following file for the test:**. There is no need to change the default value of "/QIBM/ProdData/Connect110/Samples/Driver" for directory and "OrderRequest.xml" for file.
7. Click **Next**.

This page shows you the XML that is used to run the request. You can edit it here to change the DUNS numbers, as well as other things. If you followed the field values you were given for creating the instance, JConSam1, there is no reason to change anything in order for the sample to run.

8. Click **Next**.

   A summary of the request information is displayed. If everything matches the instructions given above, select **Send Request**.

   If the OrderRequest is successful, you receive the following response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.006/cXML.dtd">
  <cXML payloadID="9804549191052@MYSYSTEM.IBM.COM"
        timestamp="2001-01-25T20:35:19+00:00" version="1.1.007">
    <Response>
      <Status code="200" text="OK"/>
    </Response>
  </cXML>
```

9. Additionally, the JavaConnectorSample1 application writes its output to a spool file in the flow manager job. To print or display the output:

   a. End the flow manager:
      1. Select your instance (JConSam1)
      2. Click **Stop**
      3. Select **IBM Connect Flow Manager**
      4. Click **Stop**.

   b. The spool file is under the B2B instance user profile. Enter the command Work with Spool Files (WRKSPLF) command on the iSeries command line:

```
WRKSPLF JCONSAM1
```

      Here are sample contents of the spool file:

```
Loading PARSEACC
PARSEACC loaded
Loading DATAACC
DATAACC loaded
-----------------------------------------------
In JavaConnectorSample1 connector application:
running java connector code!
-----------------------------------------------
Order ID = DO1234
Request type = new
Total number of items in the order: 1
----
Item identifier = 1233244
Item line number not found
Item quantity = 2
Item price = 1.34
Item description = hello
.................................
```

Top

Compile a Java connector application

To compile the Java connector application if you change it, follow these steps:

1. Start Qshell Interpreter with the STRQSH command.
2. Set your the classpath environment variable. On the Qshell command line, enter this command as one continuous line:

   ```
   export -s CLASSPATH=.:/QIBM/ProdData/Connect110/Classes/flowmanagerapi.jar
   ```

3. Change to the JConSam1 directory:

   ```
   cd /connsam11/JConSam1
   ```

4. Compile the application with the javac utility:

   ```
   javac JavaConnectorSample1.java
   ```

Top

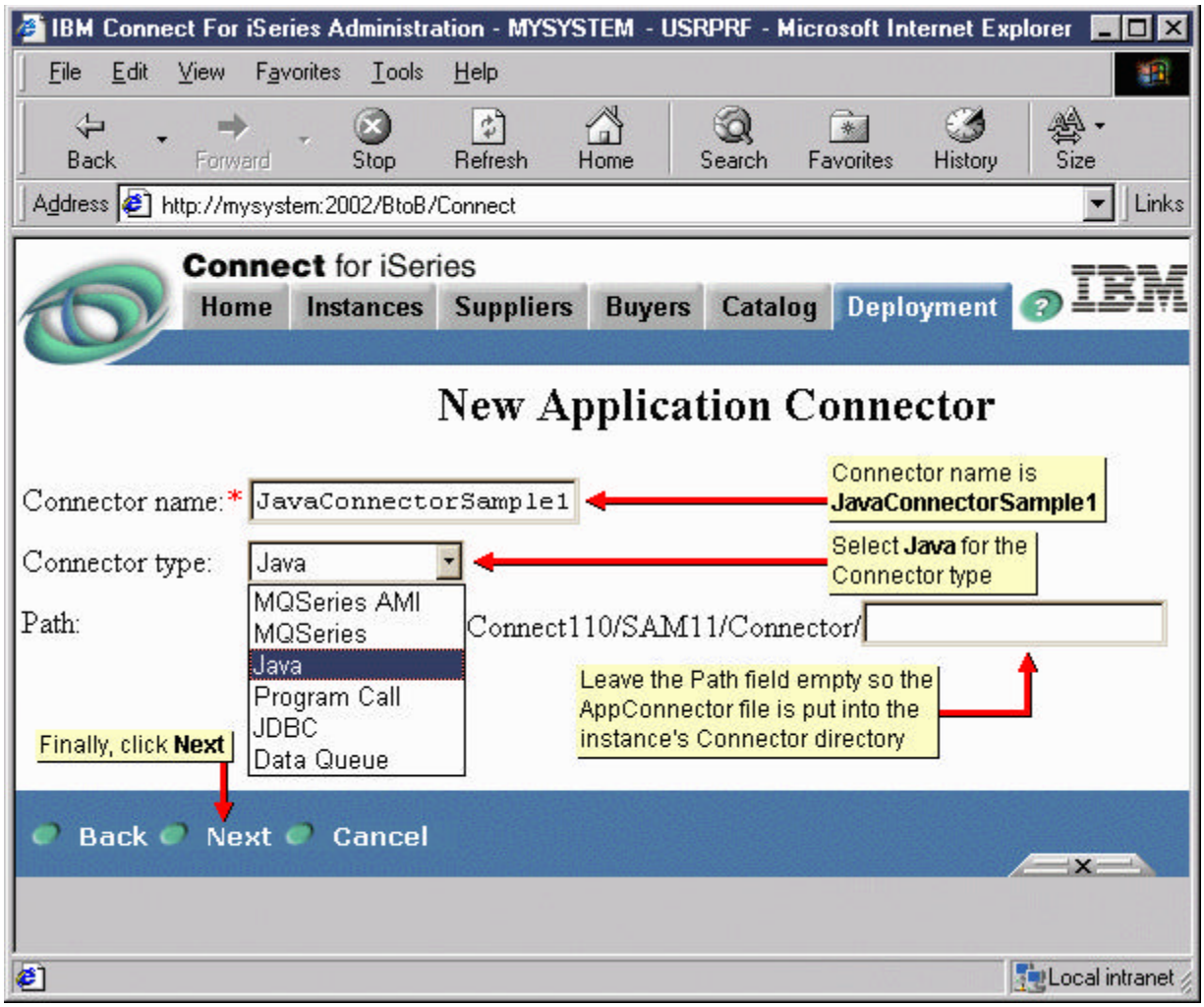
Appendix A: Create the AppConnector and ProcessFlow documents

Create the AppConnector and ProcessFlow documents with the iSeries Connect configuration tool:

1. On the Manage B2B Instances page, select the **JConSam1** instance from the list of B2B instances.
2. Click the **Deployment** tab.
3. Click **Application Connectors**.
4. Select **Create new Application Connector document.**
5. Click **Next**.
6. Enter these following values on the New Application Connector page:
   - **Class Name**: JavaConnectorSample1
   - **Input field Template**: JavaConnectorSample1.pcml
   - **Output field Template**: JavaConnectorSample1.pcml

   There are no output fields to define, so click **Next** until you reach the end of the wizard. Click **Finish**.

Figure 5. Application connector for the Java connector sample



Note: When this finishes successfully, the following message displays in the message area at the bottom of the configuration tool, "The file JavaConnectorSample1.AppConnector was successfully created.".

Under the Deployment tab, follow these steps:

1. Click **Process Flows**.
2. Select **Create a new process flow document.**
3. Click **Next**.
4. Enter theses values for this New Process Flow:
    - **Flow Name:** JavaConnectorSample1
    - **Protocol:** cXML/Ariba/1.1
    - **Request:** OrderRequest/new
    - **First Step Name:** JavaConnectorSample1

    Click **Next**.

5. Enter these values for the Insert New Step page:
    - **Step Name:** JavaConnectorSample1
    - **Step Type:** Connector
    - **Connector:** /connsam11/JConSam1/JavaConnectorSample1.AppConnector

From the Insert New Step page, map the request input fields to the fields your application connector expects (see instructions below to add a mapping). Here are all the mappings you need to do for this sample (**Fields** are on the left

of arrow, **Map Into** values are on the right):

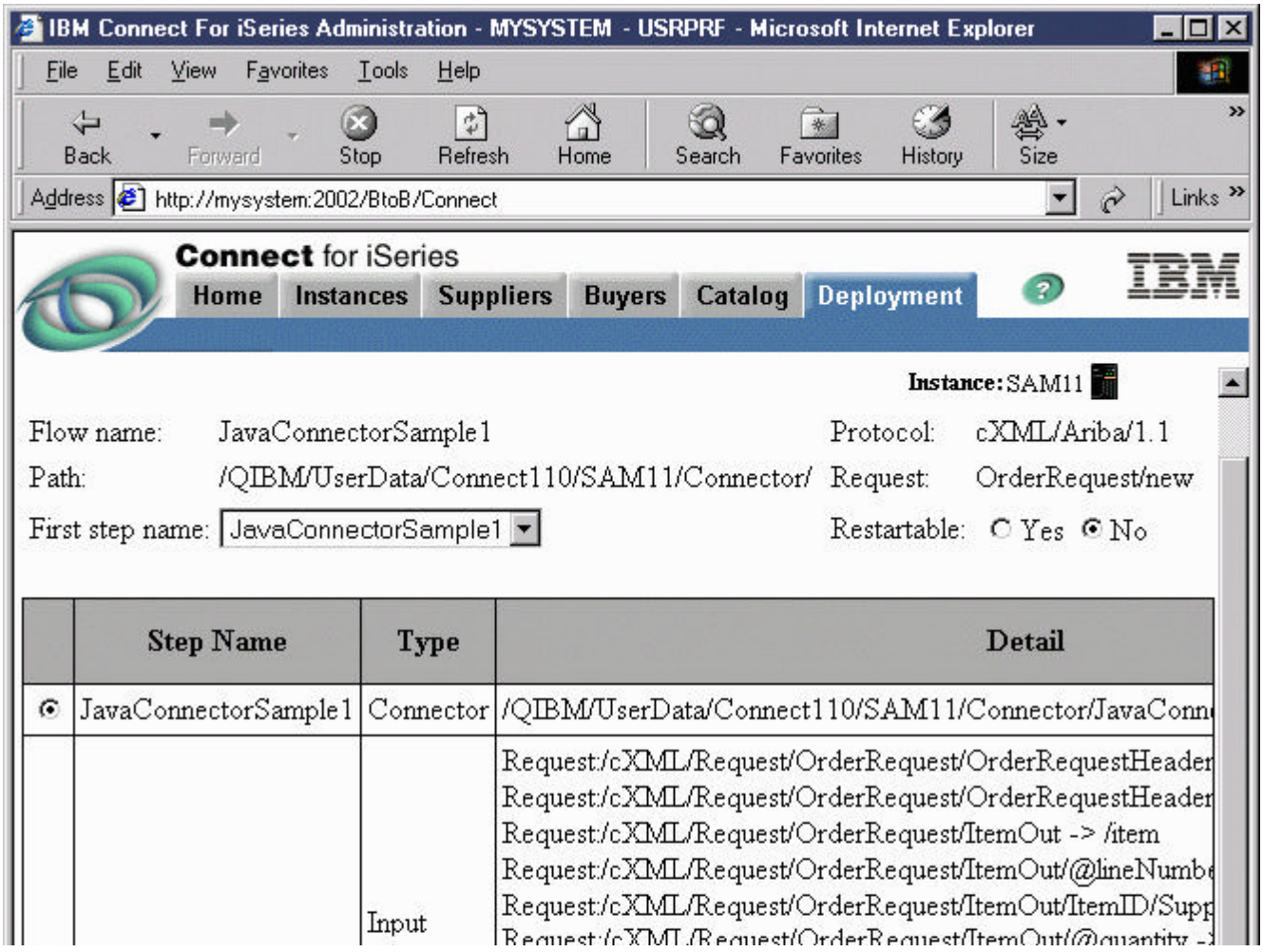
- Request:/cXML/Request/OrderRequest/OrderRequestHeader/@orderID --> /orderID
- Request:/cXML/Request/OrderRequest/OrderRequestHeader/@type --> /requestType
- Request:/cXML/Request/OrderRequest/ItemOut --> /item
- Request:/cXML/Request/OrderRequest/ItemOut/@lineNumber --> /item/lineNumber
- Request:/cXML/Request/OrderRequest/ItemOut/ItemID/SupplierPartID --> /item/identifier
- Request:/cXML/Request/OrderRequest/ItemOut/@quantity --> /item/quantity
- Request:/cXML/Request/OrderRequest/ItemOut/ItemDetail/UnitPrice/Money --> /item/price
- Request:/cXML/Request/OrderRequest/ItemOut/ItemDetail/Description --> /item/description

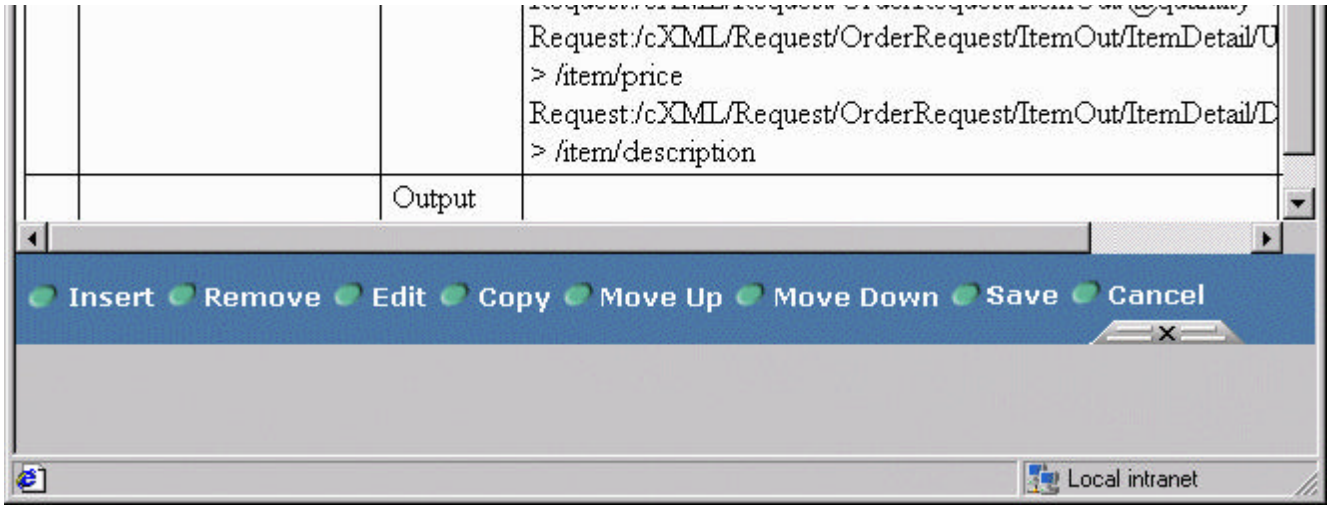
For each mapping listed above, you have to go through the following steps in the configuration tool on the Insert New Step page:

1. Click **Add Mapping** to go to the Edit Input Mapping page.
2. Scroll to the right to see the **Map Into** list. Select the field you want to map. A list of possible Fields are directly above. Scroll down the list until you find the field to map and select it.
3. Scroll left and click **Insert Field**.
4. Click **OK** on the Edit Input Mapping page to save your mapping.
5. Click **Add Mapping** to add the next mapping from the list above.

When done mapping, your JavaConnectorSample1 step appears on the page with the list of inputs and no outputs as shown in Figure 6.

Figure 6. Process flow for Java connector





When you are done mapping fields, click **OK** on the Insert New Step page, and then click **Save**.

Note: When this finishes successfully, the message "The file JavaConnectorSample1.ProcessFlow was successfully created" displays in the message area.

Appendix B: Common problems

This section lists some of the common problems that you may encounter during your tests.

Authentication error. Status code 401

- **Symptom:** OrderRequest fails with the following response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.006/cXML.dtd">
  <cXML payloadID="9806140231182@MYSYSTEM.IBM.COM"
        timestamp="2001-01-27T16:47:03+00:00" version="1.1.007">
    <Response>
      <Status code="401" text="Unauthorized">Plug-In Error 42:
            Authentication failed for marketplace Ariba Network,
            protocol cXML, protocol subtype Ariba, protocol version 1.1,
            user ID 923222888, and domain DUNS.>
    </Response>
  </cXML>
```

- **Possible Cause:**
  - The supplier Logon Information in the instance configuration does not match the <To> <Credential> element in the OrderRequest.xml file:

```
<To>
   <Credential domain="DUNS">
      <Identity>923222888</Identity>
   </Credential>
</To>
```

  - The supplier Logon Information Password/shared Secret does not match the <SharedSecret> in the

<Sender> element of the OrderRequest.xml file:

```
<Sender>
   <Credential domain="AribaNetworkUserId">
      <Identity>admin@supplier.com</Identity>
      <SharedSecret>secret10</SharedSecret>
   </Credential>
```

Authorization error. Status code 403.

- **Symptom:** OrderRequest fails with the following response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.006/cXML.dtd">
<cXML payloadID="98061657938511@MYSYSTEM.IBM.COM"
      timestamp="2001-01-27T17:29:39+00:00" version="1.1.007">
   <Response>
      <Status code="403" text="Forbidden">Plug-In Error 38:
            Authorization failed for marketplace Ariba Network, protocol
            cXML, protocol subtype Ariba, protocol version 1.1, request
            OrderRequest, request type new, supplier 123222888:DUNS, and
            buyer 921222888:DUNS.  Return code is 0</Status>
   </Response>
```

- **Possible cause:** The buyer information in the instance configuration does not match the <From> <Credential> element in the OrderRequest.xml file:

```
<From>
   <Credential domain="DUNS">
      <Identity>921222888</Identity>
   </Credential>
```

Internal server error. Status code 500.

- **Symptom:** OrderRequest fails with the following response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cXML SYSTEM "http://xml.cXML.org/schemas/cXML/1.1.006/cXML.dtd">
<cXML payloadID="98063138525517@MYSYSTEM.IBM.COM"
      timestamp="2001-01-27T21:36:25+00:00" version="1.1.007">
   <Response>
      <Status code="500" text="Internal Server Error">FlowManager
            Error 2032: The application called by the connector returned
            an error.  The errorcode is: 2041 the errorstring is:  An
            error occurred while trying to instantiate and run the user
            java connector class JavaConnectorSample1. The error
            information is: java.lang.ClassNotFoundException:
            JavaConnectorSample1
```

- **Possible cause:** The flow manager cannot find the Java connector application class specified in the process flow:
  - The Java virtual machine classpath for the flow manager in the instance configuration is incorrect.
  - The class is not in the expected path.

Top