Creating a J2C Application to process an IMS transaction with input and output data containing arrays

Time required

To complete this tutorial, you will need approximately **30 minutes**. If you decide to explore other facets of the J2C Java bean wizard while working on the tutorial, it could take longer to finish.

Prerequisites

In order to complete this tutorial end to end, you should be familiar with:

- J2EE and Java programming
- Basic IMS Transaction Manager (IMS TM) concepts

Learning objectives

This tutorial is divided into several exercises that must be completed in sequence for the tutorial to work properly. This tutorial teaches you how to use the J2C Java Bean wizard to create a Java bean that runs a transaction in IMS. While completing the exercises, you will:

- Use the J2C Bean wizard to create a J2C Java bean that runs an IMS transaction.
- Create a Java method for the bean, `runInOut.java`, to run the IMS transaction.
- Create a test proxy Java class, `TestInOutProxy.class`, to build the input message for the IMS transaction, invoke the J2C Java bean method that runs the IMS transaction, then display the output data returned by the IMS transaction.

  NOTE: The test Java class was created for an English locale; you may have to make modifications in the code for other locales.

When you are ready, begin Exercise 1.1: Selecting the resource adapter

Exercise 1.1: Selecting the resource adapter

This tutorial will lead you through the detailed steps to generate a J2C application that processes an IMS transaction with input and output data containing arrays.

Before you can begin this tutorial, you must first obtain the required resources:

- **Information about your IMS environment**: In this tutorial, your application interacts with an application program in IMS. You need to obtain information such as the host name and port number of IMS Connect and the name of the IMS datastore where the transaction will run. Contact your IMS systems administrator for this information. In addition, you need to perform some setup work in IMS if you want to run this sample. This information is provided below.
- **A copy of the COBOL file `InEqualsOut.cbl`** You may locate this file in your product installation directory: `\rad\eclipse\plugins\com.ibm.j2c.cheatsheet.content_6.0.0 \samples\IMS\InOutArray`. If you wish to store it locally, you can copy the code from here: InEqualsOut.cbl.

```
 IDENTIFICATION DIVISION.
 program-id. pgm1.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 DATA DIVISION.

 LINKAGE SECTION.

01  IN-OUT-MSG.
    05  WS-LL                      PIC S9(3) COMP VALUE +0.
    05  WS-ZZ                      PIC S9(3) COMP VALUE +0.
    05  WS-TRCD                    PIC X(5).
    05  INDX                       PIC 99.
    05  WS-CUSTOMER     OCCURS 1 TO 8 TIMES
        DEPENDING ON INDX.
        15  WS-CUST-NUMBER         PIC  X(5).
        15  WS-CUST-NAME           PIC  X(20).
    05  WS-FUNC-CODE               PIC  X(6).

 PROCEDURE DIVISION.
```

- **A clean workspace.**

This tutorial uses COBOL data structures to describe the IMS transaction input and output messages. The input and the output messages are identical and contain an array of customer elements, followed by a single field containing a function code. The array can have a maximum of eight elements, but for this tutorial only three elements are input to the IMS application program and only four elements are returned by the IMS application program.

The IMS transaction that is used by this tutorial is *not* one of the IMS Installation Verification Programs. It uses DFSDDLT0, an IMS application program that issues calls to IMS based on control statement information. The DFSDDLT0 control statements for this tutorial are provided below. However, you must configure your environment for DFSDDLT0 and provide the necessary JCL if you wish to run the tutorial. This tutorial uses SKS2 as the transaction code for the DFSDDLT0 application.

DFSDDLT0 control statements

```
S11 1 1 1 1    TP     1
L        GU
E      OK
```

```
E  Z0088 DATA  SKS2 03CN001Cathy Tang          CN002Haley Fung         X
                CN003Steve Kuo           123456
WTO IC4JINOU: Single segment received from JITOC
L        GN
E      QD
WTO IC4JINOU: End of input segments from JITOC
L        ISRT  JITOC53
L  Z0113 DATA  TRNCD04CN001Cathy T.            CN002Haley F.           X
                CN003Steve K.            CN004Kevin F.            65432X
               1
E      OK
WTO IC4JINOU: Single segment inserted - 3 elements !!!!!!!!!!!!
L        GU
```

Selecting the resource adapter

Switching to the J2EE Perspective

If the J2EE icon,  , does not appear in the top right tab of the workspace, you need to switch to the J2EE perspective.

1. From the menu bar, select **Window** > **Open Perspective** > **Other**. The Select Perspective window opens.
2. Select **J2EE**.
3. Click **OK**. The J2EE perspective opens.

Connecting to the IMS server

1. In the J2EE perspective, select **File** > **New** > **Other**.
2. In the New page, select **J2C** > **J2C Java Bean**. Click **Next**
   Note: If you do not see the J2C option in the wizard list, you need to Enable J2C Capabilities.
   1. From the menu bar, click **Window** > **Preferences**.
   2. On the left side of the Preferences window, expand Workbench.
   3. Click **Capabilities**. The Capabilities pane is displayed. If you would like to receive a prompt when a feature is first used that requires an enabled capability, select **Prompt when enabling capabilities**.
   4. Expand Enterprise Java.
   5. Select **Enterprise Java**. The necessary J2C capability is now enabled. Alternatively, you can select the Enterprise Java capability folder to enable all of the capabilities that folder contains. To set the list of enabled capabilities back to its state at product install time, click **Restore Defaults**.
   6. To save your changes, click **Apply**, and then click **OK**. Enabling Enterprise Java capabilities will automatically enable any other capabilities that are required to develop and debug J2C applications.
3. In the **Resource Adapters Selection** page, , select either the J2C 1.0 or J2C 1.5 IMS resource adapter. For this tutorial select **IMS Connector for Java (IBM: 9.1.0.1.1)**. Click **Next**.
4. In the **Connection Properties** page, de-select the Managed Connection check box and select **Nonmanaged connection**. (For this tutorial, you will use a non-managed connection to directly access IMS.) Accept the default Connection class name of `com.ibm.connector2.ims.ico.IMSManagedConnectionFactory`. In the blank fields, enter all the required connection information. Required fields, indicated by an asterisk (*), include the following:
   For TCP/IP connection:
   - **Host name**: (Required) The IP address or host name of IMS Connect.
   - **Port Number**: (Required) The number of the port used by the target IMS connect.

   For local option connection:
   - **IMS Connect name**: (Required) The name of the target IMS connect.

   For both:

- **Data Store Name**: (Required) The name of the target IMS datastore.

You may obtain the connection information from your IMS system administrator. When you have provided the required connection information, click **Next**.

Now you are ready to begin Exercise 1.2: Setting up the Web project and Java Interface and Implementations .

Exercise 1.2: Setting up the Web project and Java Interface and Implementations

Before you begin, you must complete Exercise 1.1: Selecting the resource adapter.

Exercise 1.2 steps you through the creation of a J2C application. In this exercise you will

- Create a J2C Java Bean
- Create a dynamic Web project

Creating a J2C Java bean

All work done in the workbench must be associated with a project. Projects provide an organized view of the work files and directories, optimized with functions based on the type of project. In the workbench, all files must reside in a project, so before you create the J2C Java bean, you need to create a project to put it in.

1. In the New J2C Java Bean page, type the value `InOutArray` in the **Java Project Name** field.
2. Click the **New** button beside the **Java Project Name** field to create the new project.

Creating a dynamic Web project

1. In the New Source Project Creation page, select **Web project**, and click **Next**.
2. In the New Dynamic Web Project page, click **Show Advanced**.
3. Ensure that the following values are selected:
    - **Name**: `InOutArray`
    - **Project location**: accept default
    - **Servlet version**: `2.4`
    - **Target server**: `WebSphere Application Server v6.0`
    - **EAR Project**: `InOutArrayEAR`
    - **Context Root**: `InOutArray`
4. Click **Finish**.
5. A dialog box may appear asking if you would like to switch to the Dynamic Web perspective. Click **Yes**.
6. On the J2C Java Bean Output Properties page:
    - Type `sample.ims` in the **Package Name** field.
    - Type `InOut` in the **Interface Name** field.
    - Type `InOutImpl` in the **Implementation Name** field.
7. Click **Next**.

Now you are ready to begin Exercise 1.3: Creating a Java Method.

Exercise 1.3: Creating a Java method

Before you begin, you must complete Exercise 1.2: Setting up the Web Project and Java Interface and Implementations .

Exercise 1.3 leads you through the creation of a Java method. In this exercise you will

- Create a Java method
- Create the input and output data mapping between COBOL and Java

Creating a Java method

1. In the Project Explorer view, expand the project **InOutArray** in Dynamic Web Projects.
2. Right-click on InOutImpl.java in JavaSource and select **Source** > **Add method to J2C Java bean**.
3. In the Java Method page click **Add**.
4. In the **Java method name** field, type `runInOut` for the name of the method.
5. Click **Next**.

Creating the input data mapping between COBOL and Java

1. Beside the **Input** type field of the Java Method page, click **New**.
2. In the Data Import page, ensure that the **Choose mapping** field is **COBOL_TO_JAVA**. Click **Browse** beside the **Cobol file** field.
3. Browse to find the file location of the `InEqualsOut.cbl` file (You can find a copy in your product installation folder: `\rad\eclipse\plugins\com.ibm.j2c.cheatsheet.content_6.0.0\Samples\IMS\InOutArray`).
4. Click **Open**.
5. Click **Next**.
6. In the COBOL Importer page, click **Show Advanced**.
    - Select the following options:

Parameter	Value
Platform Name	Z/OS
codepage	037
Floating point format name	IBM 390 Hexadecimal
External decimal sign	EBCDIC
Endian name	Big
Remote integer endian name	Big
Quote name	DOUBLE
Trunc name	STD
Nsymbol name	DBCS

    - Click **Query** to load the data.
    - A list of data structures is shown. Select IN-OUT-MSG in the **Data structures** field.
    - Click **Next** .
7. In the Saving Properties page,
    - Select **Default** for **Generation Style**.
    - Click **Browse** beside the **Project Name** and choose the Web project **InOutArray**.
    - In the **Package Name** field, type `sample.ims.data`.
    - In the **Class Name** field, accept the default INOUTMSG. Click **Finish**.
8. In the Java Method page, select **Use input type for output**.
9. Click **Finish**.
10. Click **Finish** to complete the definition of the method.
11. In the Java Methods page, click **Finish**.

Now you are ready to begin Exercise 1.4: Creating a Java proxy class to test your application.

Exercise 1.4: Creating a Java test class to test your application

Before you begin, you must complete Exercise 1.3: Creating a Java method.

Exercise 1.4 leads you through the creation of a Java test class to test your application. In this exercise you will

- Create a Java test class.
- Edit the class using the code supplied below.
- Run the test class to test your application.

Creating a Java test class

1. Expand the InOutArray project, expand the **Java Resources** section and select the **sample.ims** package.
2. Right click and select **New**. Select the  class option to create a new Java class.
3. In the Java class name, type `TestInOutProxy`. Note that the TestInOutProxy class is provided as an example only; you may need to change the transaction code to your IMS machine specifications. Consult your IMS administrator for the transaction code. You can locate this statement `input.setWs__trcd("SKS7 ");` in the code to make the changes.
4. Ensure that the **Source Folder** field contains `InOutArray/JavaSource` and that the **Package name** field contains sample.ims.data.
5. Click **Finish**.
6. Double-click **TestInOutProxy** to open the file in the Java editor.
7. Copy all the code provided below, and paste it into the TestInOutProxy.java class. Replace any existing code in the editor.

TestInOutProxy.java

```
/*
 * Created on 4-Oct-2004
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package sample.ims;
import sample.ims.data.*;
import com.ibm.connector2.ims.ico.IMSDFSMessageException;

/**
 * @author ivyho
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class TestInOutProxy
{
        public static void main (String[] args)
        {
                try
                {
                        // -----------------------------------------------
                        // Create the formatHandler, then create the input
                        // message bean from the formatHandler.
                        // -----------------------------------------------
                        INOUTMSG input = new INOUTMSG();

                        int sz = input.getSize();
                        System.out.println("\nInitial size of input message

                        // -----------------------------------------------
```

```
// Don't set the length (LL) field yet... wait unt
// input message has been adjusted to reflect only
// the number of array elements actually sent.
// ---------------------------------------------
input.setWs__zz((short) 0);
input.setWs__trcd("SKS7 ");

// ---------------------------------------------
// Construct an array and populate it with the ele
// to be sent to the IMS application program.  In t
// case three elements are sent.
// ---------------------------------------------
Inoutmsg_ws__customer[] customers = new Inoutmsg_ws

Inoutmsg_ws__customer aCustomer1 = new Inoutmsg_ws_
aCustomer1.setWs__cust__name("Cathy Tang");
aCustomer1.setWs__cust__number("CN001");
customers[0] = aCustomer1;

Inoutmsg_ws__customer aCustomer2 = new Inoutmsg_ws_
aCustomer2.setWs__cust__name("Haley Fung");
aCustomer2.setWs__cust__number("CN002");
customers[1] = aCustomer2;

Inoutmsg_ws__customer aCustomer3 = new Inoutmsg_ws_
aCustomer3.setWs__cust__name("Steve Kuo");
aCustomer3.setWs__cust__number("CN003");
customers[2] = aCustomer3;

// ---------------------------------------------
// Set the array on the input message.
// ---------------------------------------------
input.setWs__customer(customers);
input.setIndx((short) 3);

System.out.println("\nInitial value of INDX is: " +

// ---------------------------------------------
// Reallocate the buffer to the actual size
// ---------------------------------------------
byte[] bytes = input.getBytes();
int size = input.getSize();
byte[] newBytes = new byte[size];
System.arraycopy(bytes, 0, newBytes, 0, size);

// ---------------------------------------------
// Set the bytes back into the format handler and s
// the length field of the input message, now that
// we know the actual size.
// ---------------------------------------------
input.setBytes(newBytes);
input.setWs__ll((short) size);
System.out.println("\nAdjusted size of input messag
System.out.println("\nAdjusted size of INDX is: " +

// ---------------------------------------------
// Set fields that follow the array after the input
// message has been adjusted.
// ---------------------------------------------
input.setWs__func__code("123456");

InOutImpl proxy = new InOutImpl();
```

```
                        INOUTMSG output = new sample.ims.data.INOUTMSG();
                        output = proxy.runInOut(input);

                        short outndx = output.getIndx();
                        System.out.println("\nOutput value of INDX is: " +

                        Inoutmsg_ws__customer outArray[] = output.getWs__cu

                        for (int i = 0; i < outndx; i++)
                        {
                                System.out.println(
                                        "\n"
                                                + outArray[i].getWs__cust_
                                                + outArray[i].getWs__cust_
                        }
                }
                catch (Exception e)
                {
                        if (e instanceof IMSDFSMessageException)
                        {
                                System.out.println(
                                        "\nIMS returned message: "
                                                + ((IMSDFSMessageExceptior
                        }
                        else
                        {
                                System.out.println(
                                        "\nIMS Connector exception is: " +
                        }
                }
        }
}
```

8. Press Ctrl-S to save the changes

Testing the Application

1. Expand the **InOutArray** project and the **sample.ims** package.
2. Right click on the TestInOutProxy.java class and expand the **Run** icon. Select **Run As** > **Java Application**
3. You should see the following output on the console:

```
Initial size of input message is: 217

Initial value of INDX is: 3

Adjusted size of input message is: 92

Adjusted size of INDX is: 3

Output value of INDX is: 4

Cathy T.            CN001

Haley F.            CN002

Steve K.            CN003

Kevin F.            CN004
```

Congratulations! You have completed the Input Output Array Tutorial.

Finish your tutorial by reviewing the materials in the Summary.

Summary

Completed learning objectives

If you have completed all of the exercises, you should now be able to

- Use the J2C Java Bean wizard to create a J2C Java bean that runs on IMS.
- Create a Java method for the bean, `runInOut.java`, to run the IMS transaction.
- Create a test Java class, `TestInOutProxy.class`, to test the J2C Java bean.