

Create a J2C application for an IMS transaction containing arrays introduction

This tutorial describes how to use the J2C Java Bean wizard to build a simple web application that processes an IMS transaction with input and output data containing arrays.

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 `TestInOutProxy.class` Java class was created for an English locale; you may have to make modifications in the code for other locales.

Time required

This tutorial should take approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Skill level

Experienced

Audience

This tutorial is intended for users who are familiar with Enterprise Information systems (EIS) and IMS in particular.

System requirements

To complete this tutorial, you need to have the following tools and components installed:

- ✎ **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.
- ✎ **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_7.0.0\samples\IMS\InOutArray`. If you want to store it locally, you can copy the code from here: [../resources/inequalsout.html#inequalsout](http://resources/inequalsout.html#inequalsout)
- ✎ **A clean workspace.**

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

Lessons in this tutorial

- ✎ [Lesson 1.1: Select a resource adapter](#)
This lesson leads you through the detailed steps to select and configure the resource adapter to connect to the IMS server.
- ✎ [Lesson 1.2: Set up a Web project and Java Interface and Implementations](#)
Lesson 1.2 steps you through the creation of a Web project and Java Interface and Implementations
- ✎ [Lesson 1.3: Create a Java method](#)

Lesson 1.3 leads you through the creation of a Java method.

✎ [Lesson 1.4: Create a Java test class to test your application](#)

Lesson 1.4 leads you through the creation of a Java test class to test your application.

Lesson 1.1: Select a resource adapter

This lesson leads you through the detailed steps to select and configure the resource adapter to connect to the IMS server.

This tutorial uses COBOL data structures to describe the IMS transaction input and output messages. The input and the output messages are identical, and they 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
WT0 IC4JIN0U: Single segment received from JITOC
L      GN
E      QD
WT0 IC4JIN0U: 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
WT0 IC4JIN0U: Single segment inserted - 3 elements !!!!!!!!!!!!!!!
L      GU

```

Connecting to the IMS server

1. If the J2EE icon, , does not appear in the top right tab of the workspace, you need to switch to the J2EE perspective. From the menu bar, select Window > Open Perspective > Other. The Select Perspective window opens.
2. Select J2EE.
3. Click OK. The J2EE perspective opens.
4. In the J2EE perspective, select File > New > Other.
5. 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.

- a. From the menu bar, click Window > Preferences.
- b. On the left side of the Preferences window, expand Workbench.
- c. 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.
- d. Expand Enterprise Java.
 - e. 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.
 - f. 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.
6. 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: 9.1.0.2.3). Click Next.
 7. In the Connection Properties page, de-select Managed Connection 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:
 - a. **For TCP/IP connection:**
 - i. **Host name:** (Required) The IP address or host name of IMS Connect.
 - ii. **Port Number:** (Required) The number of the port used by the target IMS connect.
 - b. **For local option connection:**
 - i. **IMS Connect name:** (Required) The name of the target IMS connect.
 - c. **For both:**
 - i. **Data Store Name:** (Required) The name of the target IMS datastore.
 8. You may obtain the connection information from your IMS system administrator. When you have provided the required connection information, click Next.

Lesson 1.2: Set up a Web project and Java Interface and Implementations

Lesson 1.2 steps you through the creation of a Web project and Java Interface and Implementations

Before you begin, you must complete Lesson 1.1. In this lesson you will

- ✍ Create a J2C Java bean
 - ✍ Create a dynamic Web project
1. 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. In the New J2C Java Bean page, type the value `InOutArray` in the Project Name field.
 2. Click the New button beside the Project Name field to create the new project
 3. Now you will create a dynamic Web project. In the New Source Project Creation page, select Web project, and click Next.
 4. In the New Dynamic Web Project page, ensure that the following values are selected:
 - a. Project name: `InOutArray`
 - b. Project contents: accept default
 - c. Target runtime : WebSphere Application Server v6.1
 - d. Configurations: accept default
 - e. Add project to an EAR: checked
 - f. EAR Project name: `InOutArrayEAR`
 5. Click Next.
 6. On the Project Facets page, accept the defaults.
 7. Click Next.
 8. On the Web Modules page, accept the defaults.
 9. Click Finish.
 10. A dialog box may appear asking if you would like to switch to the Dynamic Web perspective. Click Yes.

11. On the J2C Java Bean Output Properties page:
 - a. In the Package Name field, click Browse and select the InOutArray project. Click OK.
 - b. Type `sample.ims` in the Package Name field.
 - c. Type `InOut` in the Interface Name field.
 - d. Type `InOutImpl` in the Implementation Name field.
 - e. Leave the Save session as ant script unchecked.
12. Click Finish.

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

Lesson 1.3: Createa Java method

Lesson 1.3 leads you through the creation of a Java method.

Before you begin, you must complete Lesson 1.2. In this lesson you will

- ✍ Create a Java method
 - ✍ Create the input and output data mapping between COBOL and Java
1. **First you will create a Java method:** 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.
 6. **Next you will create the input data mapping between COBOL and Java:** Beside the Input type field of the Java Method page, click New.
 7. In the Data Import page, ensure that the Choose mapping field is COBOL to Java. Click Browse beside the COBOL file field.
 8. Browse to find the file location of the InEqualsOut.cbl file. (You can find a copy in your product installation folder: `..\common\plugins\com.ibm.j2c.cheatsheet.content_6.0.0\Samples\IMS\InOutArray`).
 9. Click Open.
 10. Click Next.
 11. In the COBOL Importer page, click Show Advanced.
 - a. Select the following options:

Table 1. COBOL Importer Parameter Settings

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

- b. Click Query to load the data.
 - c. A list of data structures is shown. Select IN-OUT-MSG in the Data structures field.
 - d. Click Next.
12. In the Saving Properties page,
 - a. Select Default for Generation Style.
 - b. Click Browse beside the Project Name and choose the Web project InOutArray.
 - c. In the Package Name field, type `sample.ims.data`.
 - d. In the Class Name field, accept the default INOUTMSG. ClickFinish.
 - e. Leave the Save session as Ant script unchecked.
13. In the Java Method page, select Use input type for output.

14. Click Finish.
15. Click Finish to complete the definition of the method.
16. On the Java Method page, click Finish.
17. In the Java Methods page, click Finish .

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

Lesson 1.4: Create a Java test class to test your application

Lesson 1.4 leads you through the creation of a Java test class to test your application.

Before you begin, you must complete Lesson 1.3. In this lesson you will

- ✍ Create a Java test class.
 - ✍ Edit the class using the code supplied below.
 - ✍ Run the test class to test your application.
1. **First you will create a Java test class:** Expand the InOutArray project, expand Java Resources, 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:

```

/*
 * 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 until
// 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 elements
// to be sent to the IMS application program. In this
// case three elements are sent.
// -----
Inoutmsg_ws__customer[] customers = new Inoutmsg_ws__customer[3];

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

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

Inoutmsg_ws__customer aCustomer3 = new Inoutmsg_ws__customer();
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: " + input.getIndx());

// -----
// 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 set
// 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 message is: " + input.getSize());
System.out.println("\nAdjusted size of INDX is: " + input.getIndx());

// -----
// 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: "
            + ((IMSDFSMessageException
        )
    }
    else
    {
        System.out.println(
            "\nIMS Connector exception is: " +
        )
    }
}
}
}
}

```

8. Press Ctrl-S to save the changes.
9. **Next you will test the application:** Expand the InOutArray project and the sample.ims package.
10. Right click on the TestInOutProxy.java class and expand Run , and select Run As < Java Application.
11. 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.

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