# Build a J2C application for a CICS COBOL copybook: Different input and output

### 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
- COBOL programming language
- CICS ECI server technology

### Learning objectives

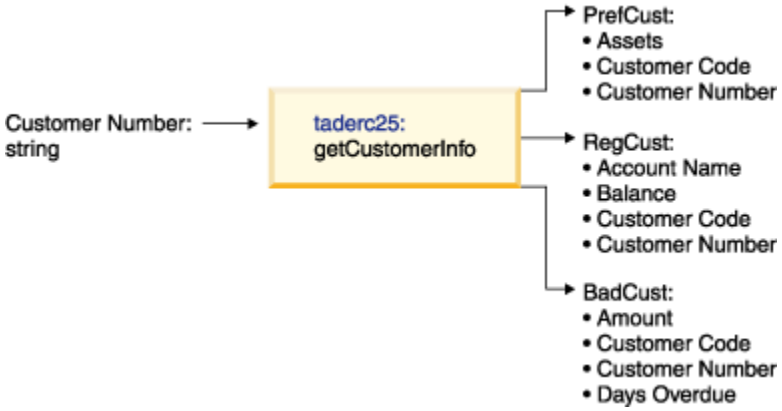
This tutorial is divided into several exercises that must be completed in sequence for the tutorial to work properly. This tutorial teaches you how to use the J2C Java bean wizard to connect to a CICS ECI server. While completeing the exercises, you will:

- Use the J2C Java bean wizard to create a J2C application that interfaces with a CICS transaction using an External Call Interface (ECI).
- Create a Java method, `getCustomerInfo`, which accepts a customer number. Depending on the customer's classification, preferred customer, regular customer or bad customer, the program returns different output information about the customer.
- Create a JSP to deploy the application on a WebSphere application server.

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 interfaces with a CICS transaction using an External Call Interface (ECI). The service is built from a CICS COBOL function, `getCustomerInfo`, which accepts a customer number. Depending on the customer's classification, preferred customer, regular customer or bad customer, the program returns different output information about the customer.




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

- **Connection to a CICS ECI server**: In this tutorial, your application interacts with a CICS program on a server. Specifically, you need to set up a CICS transaction gateway on a machine to access the server. You also need to perform some setup work on the CICS server machine, where you want the CICS to run. These steps are not covered.
- **A copy of the COBOL file taderc25.cbl.** You may locate this file in your product installation directory: `\rad\eclipse\plugins\com.ibm.j2c.cheatsheet.content_6.0.0\Samples\CICS\taderc25`. If you wish to store it locally, you can copy the code from here:

**taderc25.cbl**

```
       identification division.
        program-id. TADERC25.
        environment division.
        data division.
        working-storage section.
        01 tmp pic a(40).
        01  ICOMMAREA.
            02  ICustNo    PIC X(5).
            02  Ifiller    PIC X(11).
        01  GENCUST.
            02  GCUSTCODE PIC X(4).
            02  GFILLER PIC X(40).
        01  PREFCUST.
            02  PCUSTCODE PIC X(4).
            02  PCUSTNO    PIC X(5).
            02  ASSETS  PIC S9(6)V99.
        01  REGCUST.
            02  RCUSTCODE PIC X(4).
            02  RCUSTNO    PIC X(5).
            02  ACCOUNTNAME PIC A(10).
            02  BALANCE PIC S9(6)V99.
        01  BADCUST.
            02  BCUSTCODE PIC X(4).
            02  BCUSTNO    PIC X(5).
            02  DAYSOVERDUE PIC X(4).
            02  AMOUNT PIC S9(6)V99.
```

```
LINKAGE SECTION.
01 DFHCOMMAREA.
   02 inputfield pic x(50).
procedure division.
start-para.
    move DFHCOMMAREA to ICOMMAREA.
    IF ICustNo EQUAL '12345'
       move 'PREC' to PCUSTCODE
       move ICustNo to PCUSTNO
       move 43456.33 to ASSETS
       move PREFCUST TO DFHCOMMAREA
    ELSE IF ICustNo EQUAL '34567'
            move 'REGC' to RCUSTCODE
            move ICustNo to RCUSTNO
            move 'SAVINGS' TO ACCOUNTNAME
            move 11456.33 to BALANCE
            move REGCUST TO DFHCOMMAREA
         ELSE
            move 'BADC' to BCUSTCODE
            move ICustNo to BCUSTNO
            move '132' to DAYSOVERDUE
            move -8965.33 to AMOUNT
            move BADCUST TO DFHCOMMAREA
*        END-IF.
    END-IF.
    EXEC CICS RETURN
    END-EXEC.
```

- **A clean workspace.**

## 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 CICS ECI 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 page, under **View by**, select **JCA version**. Expand 1.5, and select **ECIResourceAdapter (IBM:6.0.0)** . Click **Next**.
4. In the Connection Properties page, select **Nonmanaged connection** check box. (For this tutorial, you will use the non-managed connection to directly access the CICS server, so you do not need to provide the JNDI name.) Accept the default Connection class name of `com.ibm.connector2.cics.ECIManagedConnectionFactory`. In the blank fields, provide connection information. Required fields, indicated by an asterisk (*), include the following:
    - **Server name**: (Not required) The name of the CICS Transaction Gateway server.
    - **Connection URL***: (Required) The server address of the CICS ECI server
    - **Port number**: (Not required) The number of the port that is used to communicate with the CICS Transaction Gateway. The default port is 2006.
    - **User name**: (Not required) The user name for the connection.
    - **Password**: (Not required) The password for the connection.

    You may obtain the connection information from your CICS Server 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 Implementation.

## Exercise 1.2: Setting up the Web Project and Java Interface and Implementations

Before you begin, you must complete Exercise 1.1: Using the J2C Java bean Wizard to select 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

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 `Taderc25Sample` in the **Project Name** field.
2. Click the **New** button beside the **Project Name** field to create the new project.
3. In the New Source Project Creation page, select **Web project**, and click **Next**.
4. In the New Dynamic Web Project page, click **Show Advanced**.
5. Ensure that the following values are selected:
    - **Name**: Taderc25Sample
    - **Project location**: accept default
    - **Servlet version**: 2.4
    - **Target server**: WebSphere Application Server v6.0
    - **EAR Project**: Taderc25SampleEAR
    - **Context Root**: Taderc25Sample
6. Click **Finish**.
7. A dialog box may appear asking if you would like to switch to the Dynamic Web perspective. Click **Yes**.
8. In the New J2C Java Bean page, ensure that the following values appear:
    - In the **Package name** field, type `sample.cics`
    - In the **Interface name** field, type `CustomerInfoMO`.
    - In the **Implementation name** field, type `CustomerInfoMOImpl`.
9. Click **Next**.

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

# Exercise 1.3: Creating the Java method

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

Exercise 1.2 steps you through the creation of a Java method, `getCustomerInfo`. In this exercise you will

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

## Creating a Java method

You will now create a Java method that will use the COBOL importer to map the data types between the COBOL source and the data in your Java method.

1. In the Java Method page, click **Add**.
2. In the **Java method name** field, type `getCustomerInfo` for the name of the operation. Click **Next**.

#### Creating the Input parameter Data Mapping

In this step, you will import the taderc25.cbl (COBOL) file that is needed to create your application. The taderc25.ccp file is located in <RSDP_installdir>\rad\eclipse\plugins\com.ibm.j2c.cheatsheet.content_6.0.0 \Samples\CICS\taderc25, where <RSDP_installdir> is the directory where this product is installed. The COBOL file contains the program that runs on the CICS server. It has the definition of the structure to be passed to the CICS server via the communications area (COMMAREA). This structure represents the customer records being returned from the CICS application. Before you can work with a file, you must import it from the file system into the workbench.

1. In the **Specify the input/output type** field of the Java Method page, click **New**.
2. In the Data Import page, ensure **Choose mapping** field is **COBOL_TO_JAVA**.
3. Click **Browse** beside the **Cobol file name** field.
4. Locate the taderc25.cbl file in the file system, and click **Open**.
5. Click **Next**.
6. In the COBOL Importer page, select a communication data structure
    - Select Win32 for **Platform Name**.
    - Select ISO-8859-1 for **Code page**
    - Click **Query**.
    - Select **ICOMMAREA** for **Data structures**. Click **Next**
7. In the Saving properties page
    - Select **Default** for **Generation Style**.
    - Click **Browse** to choose the Web project **Taderc25Sample**
    - In the **Package Name** field, enter sample.cics.data
    - In the **Class Name** field, the default value is ICOMMAREA; replace it with `InputComm`. Click **Finish**.

#### Creating the multiple possible outputs for the output parameter

1. In the **Specify the input/output type** in the Java Method page, click **New** beside the Output type area.
2. In the Data Import page, ensure that the **Choose mapping** field is **COBOL_MPO_TO_JAVA**.
3. Click **New** beside the multiple possible output area.
4. Click **Browse** beside the **Cobol file name** field, and locate the file location of the `taderc25.cbl` file. Click **Open**.
5. Click **Next**.
6. In the COBOL Importer page, select a communication data structure.
    - Select Win32 for **Platform Name**.
    - Select ISO-8859-1 for **Code page**
    - Click on the Query button to select **Data structures**
    - Select **PREFCUST**, **REGCUST** and **BADCUST** for **Data structures**.

- Click **Finish**.

7. In the Specify data import configuration properites page page, you will see the three data types listed.
8. Click **Next**.

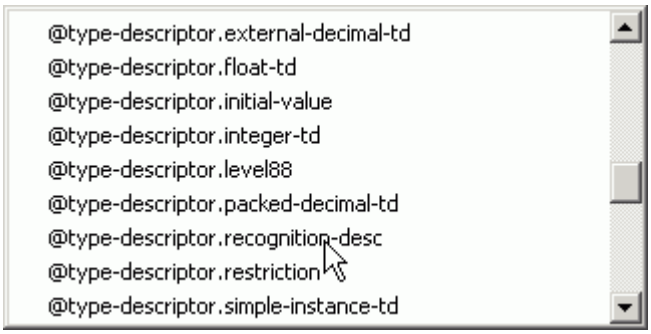
**Specifying the saving properties**

1. In the Saving Properties page, you will see default values set for each of the customer type record.
2. In the Specify the Saving properties page, under the Data Binding section
   - Accept Java project **Taderc25Sample**.
   - In the **Package Name** field, type sample.cics.data.
   - In the **Class Name** field, type OutputComm.
3. In the COBOL To Java Save Properties For "PREFCUST"
   - Accept Project name**Taderc25Sample**.
   - In the **Package Name** field, type sample.cics.data.
   - In the **Class Name** field, type PrefCust.
4. In the COBOL To Java Save Properties For "REGCUST"
   - Click **Browse** to choose the Web project **Taderc25Sample**.
   - In the **Package name** field, type sample.cics.data.
   - In the **Class Name** field, type RegCust.
5. In the COBOL To Java Save Properties For "BADCUST"
   - Click **Browse** to choose the Web project **Taderc25Sample**.
   - In the **Package name** field, type sample.cics.data.
   - In the **Class Name** field, type BadCust.
6. Click **Finish**.You will see that OutputComm contains PrefCust, RegCust and BadCust in the output type.
7. On the Java Method page, click **Finish** to complete the operation.
8. In the Java methods page,
   - Type the COBOL program id (`TADERC25`) in the **functionName** field.
   - Enter `50` in the **commareaLength** field.
   - Select the value `SYNC_RECEIVE(1)` in the **interactionVerb** field.
   - Enter `-1` in the **replyLength field**

   Click **Finish**.

**Adding the recognition pattern tag to the generate Java output data mapping file**

Since the output coming back can be any one of the data types, the only way to match it is to have some pattern predefined in the data stream. The match method is to check the recognition pattern.




1. To add the recognition pattern for PrefCust:
   a. Open the PrefCust.java file in a java editor.
   b. Navigate to the **getPcustcode()** method
   c. In the method comment area , add the tag `@type-descriptor.recognition-desc pattern="PREC"` or you can use the content assist by pressing CTRL-space and navigate down the list to find the tag and then enter "PREC" as the pattern.
   d. Save the changes and the code PrefCust.java will be regenerated.

e. Navigate to the match method to make sure the change is there.

```
/**
         * @generated

         */
        public boolean match(Object obj) {
                if (obj == null)
                        return (false);
                if (obj.getClass().isArray()) {
                        byte[] currBytes = buffer_;
                        try {
                                byte[] objByteArray = (byte[]) obj;
                                buffer_ = objByteArray;
                                if (!("PREC".equals(getPcustcode().to
                                        return (false);
                        } catch (ClassCastException exc) {
                                return (false);
                        } finally {
                                buffer_ = currBytes;
                        }
                } else
                        return (false);
                return (true);
        }
```

2. To add the recognition pattern for RegCust:
   a. Open the RegCust.java file in the Java editor.
   b. Navigate to the getRcustcode() method
   c. In the method comment area , add the tag `@type-descriptor.recognition-desc pattern="REGC"` or you can use the content assist by pressing CTRL-space and navigate down the list to find the tag and then enter "REGC" as the pattern.
   d. Save the changes and the code RegCust.java will be regenerated.
   e. Navigate to the match method to make sure the change is there.

```
/**
         * @generated

         */
        public boolean match(Object obj) {
                if (obj == null)
                        return (false);
                if (obj.getClass().isArray()) {
                        byte[] currBytes = buffer_;
                        try {
                                byte[] objByteArray = (byte[]) obj;
                                buffer_ = objByteArray;
                                if (!("REGC".equals(getRcustcode().to
                                        return (false);
                        } catch (ClassCastException exc) {
                                return (false);
                        } finally {
                                buffer_ = currBytes;
                        }
                } else
                        return (false);
                return (true);
        }
```

3. To add the recognition pattern for BadCust
    a. Open the BadCust.java file in the Java editor.
    b. Navigate to the getBcustcode() method
    c. In the method comment area , add the tag `@type-descriptor.recognition-desc pattern="BADC"` or you can use the content assist by pressing CTRL-space and navigate down the list to find the tag and then enter "BADC" as the pattern.
    d. Save the changes and the code BadCust.java will be regenerated.
    e. Navigate to the match method to make sure the change is there.

```
public boolean match(Object obj) {
                if (obj == null)
                        return (false);
                if (obj.getClass().isArray()) {
                        byte[] currBytes = buffer_;
                        try {
                                byte[] objByteArray = (byte[]) obj;
                                buffer_ = objByteArray;
                                if (!("BADC".equals(getBcustcode().to
                                        return (false);
                        } catch (ClassCastException exc) {
                                return (false);
                        } finally {
                                buffer_ = currBytes;
                        }
                } else
                        return (false);
                return (true);
        }
```

Now you are ready to begin Exercise 1.4: Deploying the application.

# Exercise 1.4: Deploying the application

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

## Creating the TestECIMPO file

1. Expand the CustomerProj project, expand the Java Resources section and select the sample.cics. package.
2. Right click and select **New**. Select the class option to create a new Java class.
3. In the **Java class name** field, type `TestECIMPO`.
4. Open TestECIMPO in the Java editor.
5. Replace all the code in the editor with the following:

**TestECIMPO.java**

```
/*********************************************************************************
 * Licensed Materials - Property of IBM
 *
 * com.ibm.j2c.cheatsheet.content
 *
 * © Copyright IBM Corporation 2004. All Rights Reserved.
 *
 * Note to U.S. Government Users Restricted Rights:  Use, duplication or disclosu
 *********************************************************************************/
package sample.cics;

import sample.cics.data.*;
public class TestECIMPO
{



        public static void process(InputComm input)
        {

         System.out.println("processing....");
     try {
                //CustomerInfoMOImpl proxy = new CustomerInfoMOImpl();
                CustomerInfoMOImpl proxy = new CustomerInfoMOImpl();
        OutputComm output = proxy.getCustomerInfo (input);

                BadCust badCust = output.getBadCust();
                PrefCust prefCust = output.getPrefCust();
                RegCust regCust = output.getRegCust();

                if (regCust != null)
                {
                        System.out.println("Reg Customer");
                        System.out.println("account name:  " + regCust.getAccour
                        System.out.println("balance:  " + regCust.getBalance());
                        System.out.println("cust code:  " + regCust.getRcustcode
                        System.out.println("cust no:  " + regCust.getRcustno());
                }
                else if (prefCust != null)
                {
                        System.out.println("Pref Customer");
                        System.out.println("assets:  " + prefCust.getAssets());
                        System.out.println("cust code:  " + prefCust.getPcustcoc
                        System.out.println("cust no:  " + prefCust.getPcustno())
                }
```

```
        else if (badCust != null)
        {
                System.out.println("Bad Customer");
                System.out.println("amount:  " + badCust.getAmount());
                System.out.println("cust code:  " + badCust.getBcustcode
                System.out.println("cust no:  " + badCust.getBcustno());
                System.out.println("days overdue:  " + badCust.getDaysov
        }
        else
                System.out.println("No match");
}
catch (Exception exc)
{
        System.out.println (exc);
        exc.printStackTrace();
}

}

public static void testPrefCust()
{
   System.out.println("==========testPreCust=============");
try {
        InputComm input = new InputComm();
        String prefC = "12345";
        input.setICustNo (prefC);
        process(input);
}
catch (Exception exc)
{
        System.out.println (exc);
        exc.printStackTrace();
}


}

public static void testRegCust()
{
   System.out.println("==========testRegCust=============");
try {
        InputComm input = new InputComm();
        String regC = "34567";
        input.setICustNo (regC);
        process(input);
}
catch (Exception exc)
{
        System.out.println (exc);
        exc.printStackTrace();
}

}

public static void testBadCust()
{

        System.out.println("==========testBadCust=============");
try {

        InputComm input = new InputComm();
        String badC = "123";
```

```
                input.setICustNo (badC);
                process(input);

        }
        catch (Exception exc)
        {
                System.out.println (exc);
                exc.printStackTrace();
        }
        }


        public static void main (String[] args)
        {
                testPrefCust();
                testRegCust();
                testBadCust();

        }
}
```

**Step 9: Testing the Application**

1. Right-click TestECIMPO.java and select **Run** > **Application**.
2. The console should have the following output.

```
==========testPreCust==========
processing....
Pref Customer
assets:  43456.33
cust code:  PREC
cust no:  12345
==========testRegCust==========
processing....
Reg Customer
account name:  SAVINGS
balance:  11456.33
cust code:  REGC
cust no:  34567
==========testBadCust==========
processing....
Bad Customer
amount:  -8965.33
cust code:  BADC
cust no:  123
days overdue:  132
```

Congratulations! You have completed the CICS Taderc25 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 application that interfaces with a CICS transaction using an External Call Interface (ECI).
- Create a Java method, getCustomerInfo, which accepts a customer number. Depending on the customer's classification, preferred customer, regular customer or bad customer, the program returns different output information about the customer.
- Create a JSP to deploy the application on a WebSphere application server.