BEA WebLogic Java Adapter for Mainframe™

CrossPlex Sample

Release 5.1
Document Date: August 2002

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA WebLogic Java Adapter for Mainframe CrossPLex Sample

Part Number	Date	Software Version
870-001037-006???	August 2002	5.1

Contents

CHAPTER

A CrossPlex Sample

The following CrossPlex Sample is included to demonstrate how users can use existing mainframe applications most effectively by integrating CrossPlex with WebLogic Java Adapter for Mainframe.

About CrossPlex

The seamless interoperability of CrossPlex with the proven BEA WebLogic E-Business Platform provides a reliable infrastructure for Business-to-Customer (B2C) and Business-to-Business (B2B) e-business applications. Seamless integration with SofTouch CrossPlex addresses customer requirements for reuse of existing 3270 mainframe applications without the need for mainframe coding changes.

CrossPlex is not a screen scraper. CrossPlex operates at the datastream level, offering customers transparent access to mainframe applications; Business-to-Business (B2B) market, buyers and sellers can be connected 24/7 and with the speed and convenience of the internet.

Integrating WebLogic Java Adapter for Mainframe with CrossPlex provides:

- Integration of multiple mainframe applications into one GUI-based presentation with no change to existing mainframe application code
- Connectivity to mainframe applications in a familiar format
- Integration of mainframe applications to technologies and protocols such as XML, HDML, Java, DHTML, PDAs, WAP phones, POS systems and more
- Speed, security, scalability, and reliability
- Future-proofing by allowing integration to any technology - present and future

About this Sample

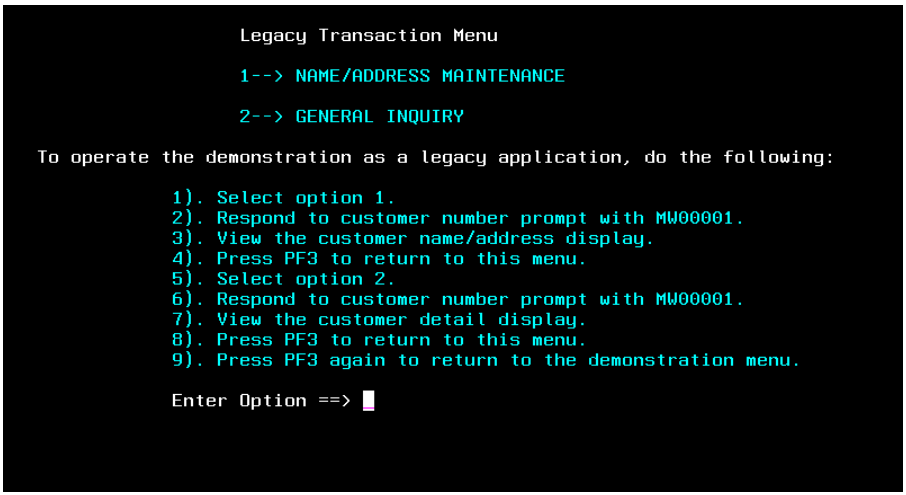
This sample shows how to develop a single service servlet-based application that invokes a CrossPlex script on the mainframe when you are using WebLogic Server. You may use similar techniques to interface to other third-party products. Because CrossPlex requires the use of a record header that should not be presented on a browser page, some DataView manipulation will be required.

This is a very simple CICS application involving transaction entry, navigation through a menu, entry of a customer number and finally one data display. The goal of this example is to web enable this application using WebLogic JAM and CrossPlex so that a single customer number entry will perform all back-end navigation and return the data display to a browser.

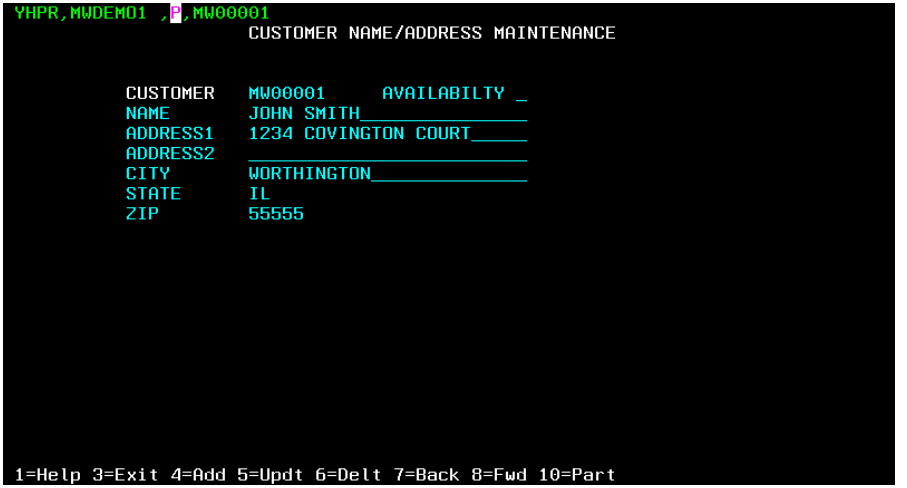
The Back-end Application

A sample CICS application is provided with the CrossPlex installation. It operates as follows:

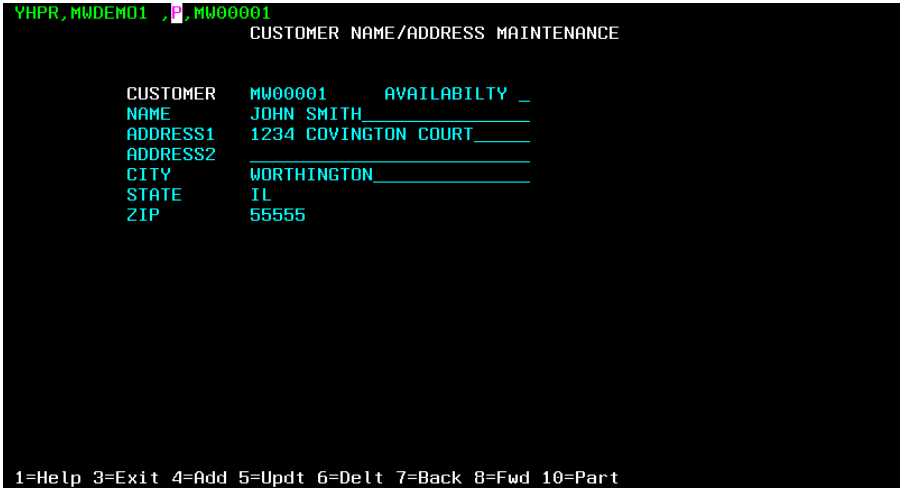
1. At a blank 3270 screen, enter `yhpr,mwdemo` in row 1, column 1. This will produce the following display:



2. Enter a "1" in the Option field and press Enter. The following screen will display.



3. Enter `MW0001` in the Customer field and press Enter. The following name and address information displays.



Press PF3 to return to the Menu. Press PF3 again to exit.

Before You Begin

Before you begin, verify that the following prerequisites have been completed

1. Verify that the following software has been properly installed:
   - BEA WebLogic Serverr
   - BEA WebLogic Java Adapter for Mainframe
   - SofTouch CrossPlex

   Refer to the corresponding installation guide for information about product installation requirements and processes.
2. Verify that the environment and the WebLogic JAM software components have been properly configured. Refer to the BEA WebLogic Java Adapter for Mainframe *Configuration and Administration Guide* for more information.
3. Verify that the appropriate mainframe application is available. Consult your mainframe system administrator for assistance with issues involving your mainframe application.
4. Review the steps to develop a java application as described in "Basic Programming Techniques" in the *BEA WebLogic Java Adapter for Mainframe Programming Guide*.

Setting up the Sample

To implement WebLogic JAM with CrossPlex, complete the following tasks.

Task 1: Create a CrossPlex Script

A CrossPlex script provides the business logic to execute one or more 3270 transactions running on the mainframe. Transactions in any VTAM system, such as CICS or IMS, can be accessed. When a script executes in CrossPlex, it usually requires some input data, such as a customer number and part number. This input data is passed from your application in a container called a record definition.

During execution, a script selects and optionally reformats data from the screen displays of the executed 3270 transactions. This selected data is returned to your application in a record definition.

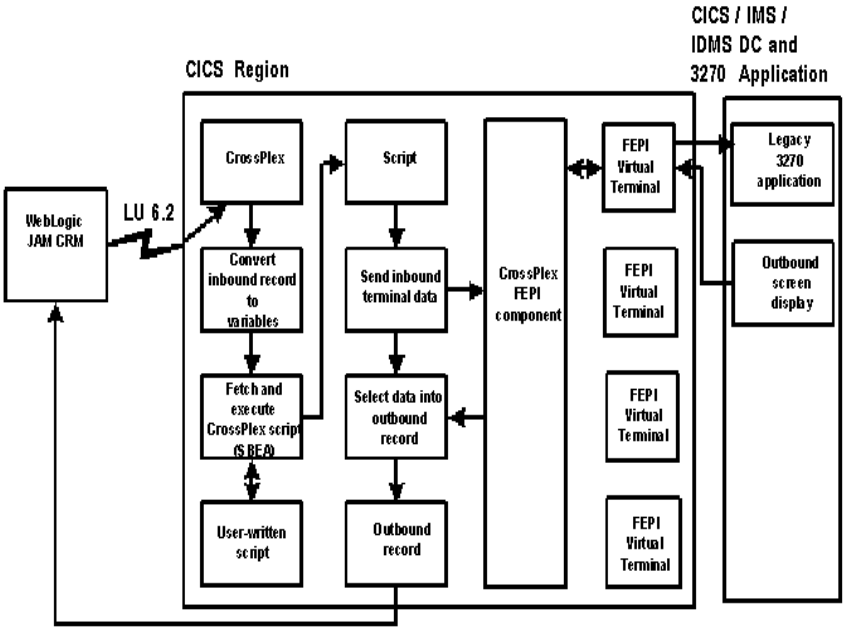
Note: Record definitions do not necessarily conform to any known data record in a file. A record definition is simply a description of a series of data fields being passed to and from a script.

The CrossPlex development system creates record definitions. An online editor is used to define each field in the record, along with its length and type (alpha, numeric, binary, packed). A single record definition may be used for data passing to and from the mainframe, or two definitions may be used.

The CrossPlex development system also creates a COBOL copybook, using a record definition as input. The generated copybook is stored in a PDS member where it can be copied into your application program as needed.

Figure 1 illustrates the processing flow from the JAM front end to retrieve data from one or more mainframe transactions.

Figure 1 Processing Flow from JAM to Mainframe Transactions

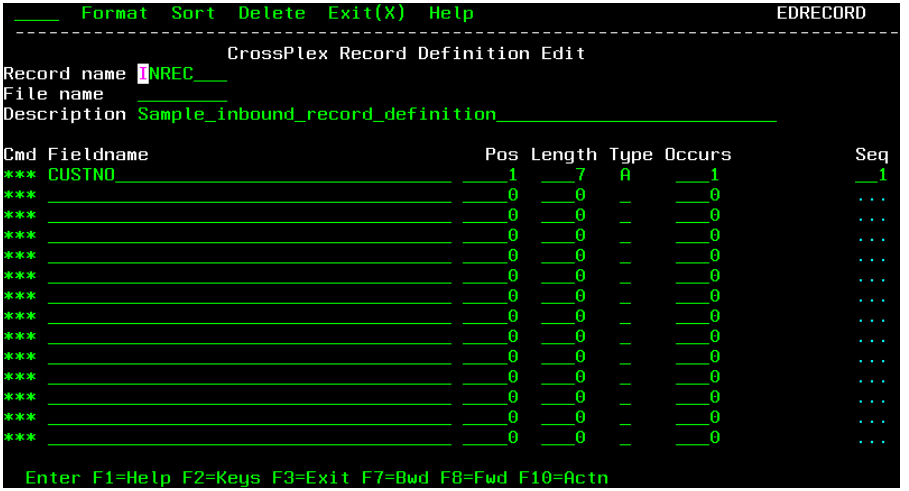


Step 1: Prepare Record Definition for the Mainframe

To prepare a record definition for the mainframe, assign a record name and description, then define each data field to be passed to the CrossPlex script. The process of defining a record definition is described in detail in the *CrossPlex Middleware Programmer's Guide*.

The mainframe application is a simple name/address display that requires a customer number as input. In this example, the record definition to and from the mainframe are different, though the same record definition can be used for both. Figure 2 shows how the record sent to the mainframe looks.

Figure 2 Illustration of a Record Sent to the Mainframe



The data required by the mainframe transaction is `CUSTNO`, a seven-byte alphanumeric field beginning in position one of the record.

This record definition is provided with the CrossPlex installation. It is in the Record Definition Directory with the name `INREC`.

Step 2: Create a Copybook of the Record Definition Sent to the Mainframe

The copybook command of the CrossPlex development menu is used to generate a COBOL copybook of a record definition. Store the generated copybook in a PDS member where you can easily copy it to your development system. For a complete description of the process of creating a COBOL copybook from a record definition, refer to the *CrossPlex Middleware Programmer's Guide*.

Continuing with the same example, a COBOL copybook generated from the previously illustrated record definition, `INREC`, appears as shown in Listing 1:

Listing 1 `INREC` Example

```
*****************************************************************
*       INREC     - Sample record definition sent to the m/f*
*****************************************************************
```

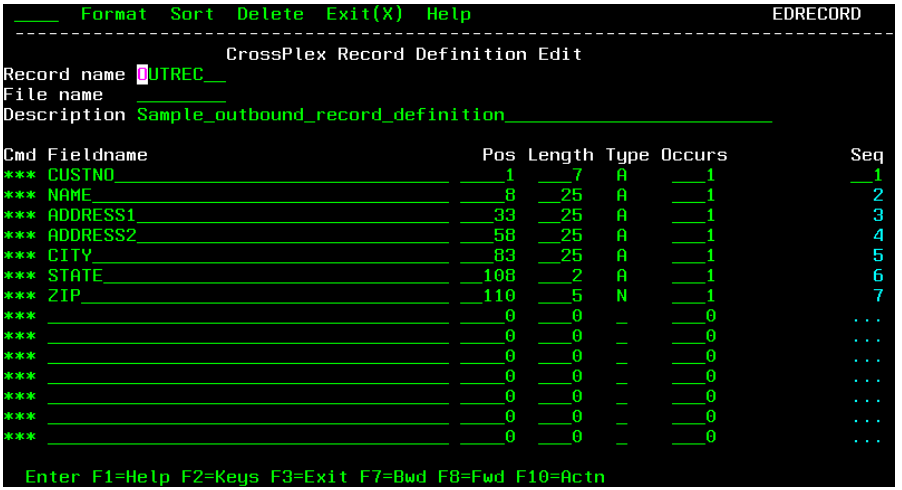
```
01   INREC-START.
     05     INREC-CUSTNO     PIC X(007).
```

Step 3: Create a Record Definition and Copybook Sent From the Mainframe

If the data sent from the mainframe is to use a different record format from the data sent to the mainframe, repeat Steps 1 and 2 to prepare the record definition and copybook.

For this example, the record definition and copybook appears as in Figure 3.

Figure 3 Record Definition for Data Sent From the Mainframe



This record definition is provided with the CrossPlex installation. It is in the Record Definition directory with the name `OUTREC`.

Listing 2 OUTREC Example

```
*****************************************************************
*      OUTREC - Sample record definition sent from the m/f      *
*****************************************************************
  01   OUTREC-START.
       05    OUTREC-CUSTOMER    PIC X(007).
       05    OUTREC-NAME        PIC X(025).
```

```
05    OUTREC-ADDRESS1    PIC X(025).
05    OUTREC-ADDRESS2    PIC X(025).
05    OUTREC-CITY        PIC X(025).
05    OUTREC-STATE       PIC X(002).
05    OUTREC-ZIP        PIC 9(005).
```

Step 4: Prepare the CrossPlex Script

Scripts can be coded using the CrossPlex script editor, or they may be coded on any external editor and imported into the CrossPlex control file. The CrossPlex script language and the process of creating a script are described in the *CrossPlex Middleware Programmer's Guide*.

Note: In the CrossPlex documentation, scripts are also known as command streams and stream objects.

Prepare a script that navigates through a series of 3270 transactions in the same manner as a terminal operator. The script acts as a virtual operator, performing a log-on to the OLTP system, sending terminal data to the mainframe as if keyed on a keyboard, examining the returned screen display for correct execution, and selecting data from the screen if needed. Any number of transactions may be executed. The script language also provides a method of linking to a user program on the mainframe in order to perform direct retrieval of data that may not be available in a 3270 transaction display.

Continuing with the example of name/address data retrieval, a sample script is provided with the CrossPlex installation. It resides in the Command Stream directory with the name `MWDEMO`. It is similar to the script in Listing 3.

Listing 3 CrossPlex Script

```
* INVOKE LEGACY MENU
    Callcpx Msgarea (“YHPR,MWDEMO”);

* RESPOND TO MENU WITH OPTION 1
    Callcpx Rowcol (20033) “1”;

* RESPOND TO KEY PROMPT WITH CUSTOMER NUMBER
    Callcpx Rowcol (05023) &CUSTNO;

* SELECT NAME/ADDRESS FIELDS INTO MWDREC
    Select Record (“OUTREC”)
       ROWCOL(05023) RFIELD “CUSTOMER”
```

```
        ROWCOL(06023) RFIELD “NAME”
        ROWCOL(07023) RFIELD “ADDRESS1”
        ROWCOL(08023) RFIELD “ADDRESS2”
        ROWCOL(09023) RFIELD “CITY”
        ROWCOL(10023) RFIELD “STATE”
        ROWCOL(11023) RFIELD “ZIP”;
* PF3, RETURN to MENU
    Callcpx Aid(PF3);

* PF3, RETURN TO CLEAR SCREEN
    Callcpx Aid(PF3);
```

Note: This example illustrates row/column addressing of screen data. CrossPlex also provides a method of assigning screen field names to avoid specific row/column references

Step 5: Test and Debug the Script

You can fully test and debug the script that executes on the mainframe without connecting it to your front-end application. CrossPlex provides a variety of execution and debugging tools to ensure the back-end portion of your application is operating properly.

When you are satisfied that the script is doing what you want and the returned data is correct, proceed to prepare the front-end of your application and connect the two together.

The process of testing and debugging a script is described in the *CrossPlex Middleware Programmer's Guide*.

HANDLING THE MAINFRAME SIGN-ON

Most VTAM systems require the user to sign on in the target region when first connecting. You must also sign on when connecting to a target region with CrossPlex. This sign-on requirement can be handled in any one of the following ways:

- Interact with a user sign-on transaction in the script.

  The most common situation, especially for CICS, requires that your script handle the sign-on. Many users have CICS configured so that upon the first connection, the terminal is presented with a sign-on panel that may have been customized for the installation. If this is the case, the first CALLCPX command

of the script returns the sign-on screen to the script and a subsequent CALLCPX must send a valid user ID and password. The mainframe sign-in is discussed in the *CrossPlex Middleware Programmer's Guide*.

- Let CrossPlex perform a short-form sign-on.

  Supplying a valid user ID and password in the CrossPlex header will cause CrossPlex to perform a short-form sign-on before sending the first transaction data from the script.

  Note: This case is valid for CICS systems only, and is installation dependent.

  The short-form CICS sign-on may be disabled, depending on the user's CICS configuration. This case is discussed in the *CrossPlex Middleware Programmer's Guide*.

- Perform a mass log-on at CICS startup.

  With this technique, several FEPI virtual terminals are logged-on when CICS is first started and they remain active until CICS is recycled. If this is done, scripts do not need to be concerned with doing a sign-on at all. This topic is discussed in the *CrossPlex Web Enabling Guide*.

Task 2: Use the eGen Application Generator to Create a Base Application

Copy the CrossPlex COBOL copybooks to your development system. These copybooks include the copybook for the CrossPlex header (CSMF), the script invocation record definition (in this case INREC), and the script result record definition (in this case OUTREC). This sample generates four DataView classes from these three copybooks by merging them in the correct pattern. Table 1 lists the four DataView classes created from the three copybooks.

Table 1 Merge Pattern for DataView Classes

Purpose	Copybook(s) used	Combined Copybook Name
Initial form for presentation on browser	`INREC`	`INREC`
Record sent to mainframe	`CSMF + INREC`	`INREC-H`

Purpose	Copybook(s) used	Combined Copybook Name
Result returned from mainframe	CSMF + OUTREC	OUTREC-H
Result presented to user	OUTREC	OUTREC

You must manually create a combined copybook that includes the record definitions and the header files.

When the application calls CrossPlex to retrieve data from the mainframe, it must pass a 256-byte header (CSMF), followed by the record area (INREC) to the mainframe. The data selected in the script will be returned in the record area (OUTREC) from the mainframe, which occupies the same memory address as the record to the mainframe, immediately following the header.

The CrossPlex header is described in the *CrossPlex Middleware Programmer's Guide*. The COBOL version called XPLXCBL is available.

In addition to the required fields listed in *Standardized Message Format*, two additional fields must be supplied by your application:

Table 2 Additional Standardized Message Format Fields

XP-EXECUTING-SCRIPT	The name of the CrossPlex script to execute.
XP-INBOUND-RECORD	The name of the record definition sent to the mainframe.
XP-MODE	Operating mode. Must contain CMDR to execute a script with a record definition as input.

The record definition from the mainframe is named in a SELECT statement within the script.

Listing 4 shows the COBOL version of the header copybook.

Listing 4 COBOL Version of Header Copybook

```
*****************************************************************
*                                                               *
*          XPLXCBL - CROSSPLEX STANDARDIZED MESSAGE FORMAT      *
```

```
*                       COBOL VERSION                             *
*                                                                 *
*******************************************************************

01  XP-COMMAREA.
     05  XP-COMMAND                PIC X(4).
     05  XP-RESPONSE               PIC S9(8) COMP.
         88  XP-NO-ERROR           VALUE 0000.
         88  XP-ANY-ERROR          VALUES 0001 THRU 9999.
         88  XP-COMM-LENGTH        VALUE 0001.
         88  XP-ABEND              VALUE 0002.
         88  XP-FILE-CLOSED        VALUE 0003.
         88  XP-NO-INSCREEN        VALUE 0004.
         88  XP-NO-OUTSCREN        VALUE 0005.
         88  XP-STORAGE-ERROR      VALUE 0006.
         88  XP-PROG-UNAVAIL       VALUE 0007.
         88  XP-NO-FILE            VALUE 0008.
         88  XP-EXPIRED            VALUE 0009.
         88  XP-UNMATCHED          VALUE 0010.
         88  XP-FIELD-ERROR-IN     VALUE 0011.
         88  XP-NO-SCREEN-FIELDS   VALUE 0012.
         88  XP-FIELD-ERROR-OUT    VALUE 0013.
         88  XP-NO-DATA            VALUE 0014.
         88  XP-INLENGTH-TOOLONG   VALUE 0015.
         88  XP-INVALID-AID        VALUE 0016.
         88  XP-MSG-TOOLONG        VALUE 0017.
         88  XP-XLATE-ERR          VALUE 0018.
         88  XP-END-SESSION        VALUE 0019.
         88  XP-END-TRANSMISSION   VALUE 0020.
         88  XP-CONNECT-FAILED     VALUE 0021.
         88  XP-NOT-INBOUND        VALUE 0022.
         88  XP-NOT-OUTBOUND       VALUE 0023.
         88  XP-SIGNON-FIELDS      VALUE 0024.
         88  XP-SIGNON-FAILURE     VALUE 0025.
         88  XP-INVALID-COMMAND    VALUE 0026.
         88  XP-FIELD-TABLE-ERR    VALUE 0027.
         88  XP-EDIT-ERROR-IN      VALUE 0028.
         88  XP-STREAM-FAILED      VALUE 0029.
         88  XP-PROFILE-REQUEST-INVALID VALUE 30.
         88  XP-PROFILE-REQUEST-FAILED VALUE 31.
         88  XP-TEMPLATE-FAILED    VALUE 32.
         88  XP-SCREEN-FILE-FULL   VALUE 33.
         88  XP-NOTAUTH            VALUE 34.
         88  XP-RECORD-NOT-FOUND   VALUE 35.
         88  XP-STREAM-NAME-OMMITED VALUE 36.
         88  XP-RECORD-NAME-OMMITED VALUE 37.
         88  XP-EXEC-LINK-FAILED   VALUE 1002.
     05  XP-EXCEP-DATA.
         10  XP-EXCEP-ROWCOL       PIC S9(4) COMP.
```

```
    10  XP-EXCEP-LENGTH       PIC S9(4) COMP.
    10  XP-FLD-ERR            PIC S9(4) COMP.
    10  XP-EXCEP-MSG-FIELD    PIC S9(4) COMP.
    10  XP-EXCEP-FEPI         PIC X(4).
    10  XP-EXCEP-EIBRESP      PIC S9(8) COMP.
    10  XP-EXCEP-EIBRESP2     PIC S9(8) COMP.
05  XP-OPTIONAL-PARMLIST      POINTER.
05  XP-TARGET                 PIC X(8).
05  XP-POOL                   PIC X(8).
05  XP-AIDBYTE                PIC X(6).
05  XP-INSCREEN               PIC X(8).
05  XP-OUTSCREEN              PIC X(8).
05  XP-CURSOR.
    10  XP-CURSOR-ROW         PIC S9(4) COMP.
    10  XP-CURSOR-COL         PIC S9(4) COMP.
05  XP-SIGNON-USERID          PIC X(8).
05  XP-SIGNON-PASSWORD        PIC X(8).
05  XP-NODENAME               PIC X(8).
05  XP-FEPI-CONVID-BIN        PIC S9(18) COMP.
05  XP-FEPI-CONVID REDEFINES XP-FEPI-CONVID-BIN PIC X(8).
05  XP-DEBUG-QUEUE            PIC X(8).
05  XP-ASSOC-NAME             PIC X(8).
05  XP-MODE                   PIC X(4).
    88  XP-HTML               VALUE 'HTML'.
    88  XP-HTQS               VALUE 'HTQS'.
    88  XP-3270               VALUE '3270'.
    88  XP-CMDS               VALUE 'CMDS'.
05  XP-TRANSLATION-SCREEN     PIC X(8).
05  XP-IN-LENGTH              PIC S9(4) COMP.
05  XP-IN-LENGTH-HW REDEFINES XP-IN-LENGTH PIC X(2).
05  XP-AREA-LENGTH            PIC S9(4) COMP.
05  XP-AREA-LENGTH-HW REDEFINES XP-AREA-LENGTH PIC X(2).
05  XP-OUT-LENGTH             PIC S9(4) COMP.
05  XP-OUT-LENGTH-HW REDEFINES XP-OUT-LENGTH PIC X(2).
05  XP-TERM-OPTION            PIC X(1).
    88  XP-NOTERM             VALUE 'N'.
05  XP-USD                    PIC X(1).
    88  XP-USD-EXPECTED       VALUE 'Y'.
05  XP-USD-WAIT-TIME          PIC S9(4) COMP.
05  XP-SAVE-DENTITY           PIC X(8).
05  FILLER                    PIC X(16).
05  XP-LINK-PROGRAM           PIC X(8).
05  XP-SCREEN-ROWS            PIC 9(4) COMP.
05  XP-SCREEN-COLS            PIC 9(4) COMP.
05  XP-STREAM                 PIC X(8).
05  XP-EXECUTING-SCRIPT REDEFINES XP-STREAM PIC X(8).
05  XP-FEPI-TIMEOUT           PIC 9(4) COMP.
05  XP-TRANSLATION-TYPE       PIC X(1).
    88  XP-TRANSLATE-SCREEN   VALUE 'S'.
```

```
    88  XP-TRANSLATE-TEMPLATE VALUE 'T'.
    88  XP-TRANSLATE-COMMANDS VALUE 'C'.
    88  XP-TRANSLATE-USER     VALUE 'U'.
05  XP-DEBUG-STATUS           PIC X(1).
    88  XP-DEBUG-INACTIVE     VALUE '0'.
    88  XP-DEBUG-ACTIVE       VALUE '1'.
05  XP-ALT-AREA-PTR           POINTER.
05  XP-ALT-AREA-LENGTH        PIC 9(8) COMP.
05  XP-ALT-DATA-LENGTH        PIC 9(8) COMP.
05  XP-ALT-AREA-USE           PIC X(1).
    88  XP-ALT-AREA-INPUT     VALUE X'01'.
    88  XP-ALT-AREA-OUTPUT    VALUE X'02'.
    88  XP-ALT-AREA-IO        VALUE X'03'.
05  XP-RECORD-DEFINITION      PIC X(8).
05  XP-INBOUND-RECORD REDEFINES XP-RECORD-DEFINITION
                              PIC X(8).
05  XP-DATASTREAM-LOG
    REDEFINES XP-RECORD-DEFINITION PIC X(8).
05  XP-ECHO-MODE              PIC X(1).
    88  XP-ECHO               VALUE 'Y'.
    88  XP-NOECHO             VALUE 'N'.
05  XP-CONNECTION-TYPE        PIC X(1).
    88  XP-CONNECTION-TYPE-FEPI VALUE 'F'.
    88  XP-CONNECTION-TYPE-BRIDGE VALUE 'B'.
05  XP-DATA-FILE              PIC X(8).
05  XP-DEVL-TRANCODE          PIC X(4).
05  XP-FEPI-ERR-STREAM        PIC X(8).

05  XP-FILLER                 PIC X(21).
05  XP-MESSAGE-AREA.
```

Step 1: Prepare the eGen script

In Listing 5, the DataViews are generated from the combined copybooks.

1. Create header files (`INREC-H.cbl` and `OUTREC-H.cbl`.)
2. Merge the header files (`INREC-H.cbl` and `OUTREC-H.cbl`) with the record definitions (`INREC.cbl` and `OUTREC.cbl`.)

   You may need to adjust your level numbers after merging the records.

Listing 5 results in the script that creates the DataView.

Listing 5 Basic eGen script

```
view InrecRecord from INREC.cbl
view InrecHdrRecord from INREC-H.cbl
view OutrecRecord from OUTREC.cbl
view OutrecHdrRecord from OUTREC-H.cbl
```

Step 2: Add Service Entry

Add the single line service entry in Listing 6 for the CrossPlex operation. to the script This entry specifies the DataView.

Listing 6 Service Names Associated with Input and Output Views

```
service DoIt accepts InrecHdrRecord returns OutrecHdrRecord
```

Step 3: Add Page Declarations in the eGen script

This application requires two pages: one to invoke the operation and another to present the results. Note that the full records (with header) are mentioned, even though these are not displayed. The custom code written later in the scenario specifies this display.

Listing 7 Page Declaration Associating Display Buttons with Services

```
page page1 "Invoke Operation" {
view InrecHdrRecord
       buttons {
               "doit" service(DoIt) shows resultPage
       }
}
page resultPage "Results of Operation" {
       view OutrecHdrRecord
       buttons {
             // No buttons on this page.
       }
}
```

Step 4: Add Servlet Name

Add the servlet name to the end of the eGen script and save it as `crossplex.egen`.

Listing 8 Add Servlet Name

```
servlet DoItServlet shows page1
```

Add the servlet name to the WebLogic Server `web.xml`. As shown in Listing 8, `DoItServlet` is the servlet name to be registered as a URL in the WebLogic Server `web.xml` file. (Every servlet requires a URL to be registered this way. Refer to WebLogic Server documentation about deploying servlets for more specific information.) In this example, the page "page1" is to be displayed when the servlet "DoItServlet" is invoked.

Step 5: Generate the Java Source Code

As in Listing 9, invoke the eGen Application Generator to create the application that is then compiled. This process makes class files (`*.class`) available for servlet customizing. `CLASSPATH` should include the WebLogic Server subdirectories and the `jam.jar` file; otherwise, the compile fails. You can create a script file containing the eGen command line, along with the `javac` command to make the invocation easier.

Listing 9 Generating the Java Source Code

```
egencobol emprec.egen
ls *.java
       BaseServlet.java
       InrecHdrRecord.java
       InrecRecord.java
       OutrecHdrRecord.java
       OutrecRecord.java
```

Task 3: Create Your Custom Application from the Generated Application

The preferred customizing method is to derive a custom class from the generated application. In this case, we will subclass the generated servlet code to both change record formats and manipulate CrossPlex header fields.

Start creating a new source file to extend the generated eGen code.

Step 1: Use Imports to Start Creating the Custom Application

In Listing 10, `BigDecimal` supports `COMP-3` packed data. `HttpSession` is available for saving limited state. `DataView` is the base for all generated data records.

Listing 10 Using Imports to Start Creating the Custom Application

```
import java.util.Hashtable;
import javax.servlet.http.HttpSession;
import com.bea.dmd.dataview.DataView;
import InrecRecord;
import InrecHdrRecord;
import OutrecRecord
import OutrecHdrRecord;
import com.bea.dmd.dataview.HashtableLoader;
import com.bea.dmd.dataview.HashtableUnloader;
import com.bea.dmd.dataview.PrefixChanger;
```

Step 2: Declare the New Custom Class

Listing 11 shows how to extend the generated servlet. Extension of the generated servlet enables regeneration of the base application without destroying customized code. Fields can be added to the copybook without disrupting the customized code.

Listing 11 Declaring the New Custom Class

```
public class customServlet
     extends DoItServlet
{
:
```

Step 3: Add Implementation for doGetSetup

In Listing 12, the `doGetSetup ( )` function is used to ensure that the user is presented with a form reflecting the `INREC` record.

Listing 12 Add Implementation for doGetSetup

```
public DataView doGetSetup(DataView dv, HttpSession s){
return new InrecRecord ();
}
```

Step 4: Create Implementation for doPostSetup

The `doPostSetup` method performs operations after a button has been pressed on the form, prior to the mainframe call. In Listing 13, the DataView passed in contains values entered into the form by the application user. This code moves the specified data into an `InrecHdrRecord`; then sets the header fields for the operation you wish to perform.

Listing 13 Create Implementation for doPostSetup

```
public DataView doPostSetup(DataView dv, HttpSession s)
    {
         InrecHdrRecord bhr = new InrecHdrRecord();
         try
         {

           // Move the contents, by using a Hashtable as an
           intermediate holder.
           Hashtable h = new HashtableUnloader(new PrefixChanger
```

```
                ("mwdrecStart.", "xpCommarea.")).unload(dv);
                new HashtableLoader().load(h,(bhr);
            // Load header fields.
            bhr.getXpCommarea().setXpCommand("EXEC");
            bhr.getXpCommarea().setXpTarget("THISCICS");
            bhr.getXpCommarea().setXpPool("POOLM2");
            bhr.getXpCommarea().setXpFepiConvidBin(0L);
            bhr.getXpCommarea().setXpMode("CMDR");
            bhr.getXpCommarea().setXpInLength((short) 300);
            bhr.getXpCommarea().setXpAreaLength((short) 1300);
            bhr.getXpCommarea().setXpExecutingScript("MWDEMO");
            bhr.getXpCommarea().setXpInboundRecord("INRECRECRD");

        }
        catch (Exception e)
        {
        }
        return bhr;
}
```

The meaning of each field in the CrossPlex header is described in the *CrossPlex Middleware Programmer's Guide*. For most executions, the following fields must contain meaningful data:

Table 3 CrossPlex Header Fields

COMMAND	Contains "EXEC" to execute a script, or "TERM" to terminate a session.
TARGET	Contains the FEPI target name of the VTAM region where transactions are to be executed.
POOL	Contains the FEPI pool name for this session.
ASSOC	Instead of TARGET and POOL, a CrossPlex Association can be named, which defines the target, pool and connection type (FEPI or BRIDGE).
MODE	Must contain "CMDR" if a record definition to the mainframe is used and a script is to be executed.
AREA-LENGTH	Contains the maximum length of MESSAGEAREA.
EXECUTING-SCRIPT	The name of the script to be executed.

INBOUND-RECORD	The name of the record definition sent to the mainframe.
MESSAGEAREA	Contains the record sent to the mainframe when CrossPlex is called and the record from the mainframe upon return.
USERID	To perform a short sign-on to the target region using FEPI, supply a valid user ID in this field.
PASSWORD	Valid password if USERID is present.
DEBUGQ	Name of a debug queue where execution trace records are to be written.
Upon return from CrossPlex, the following fields are supplied:	
NODENAME	The FEPI node name used by the mainframe session.
CONVID	The FEPI conversation ID assigned to the mainframe session.

On the first call to CrossPlex, all fields of the CSMF header must be completely initialized to their default values or filled with user data. The generated DataView code initializes with default values. Upon return, the header contains some fields provided by CrossPlex, such as the FEPI conversation ID. If subsequent calls to CrossPlex are made for the same session, these fields must not be re-initialized, since CrossPlex needs the FEPI conversation ID to continue the same session

Step 5: Create Implementation for doPostFinal

In Listing 14, the doPostFinal occurs after mainframe transmission, but prior to re-display in the browser. This example moves the result OutrecHdrRecord into an OutrecRecord prior to display.

Listing 14 Create Implementation for doPostFinal

```
public DataView doPostFinal(DataView dv, HttpSession s)
{

       OutrecHdrRecord qhr = (OutrecHdrRecord) dv;
       int resp=qhr.getXPCommarea().getXpCommarea().getXpResponse();
       if (resp != 0  &&  resp != 12)
             throw new Error("Bad xp-response: " + resp);

       OutrecRecord qr = new OutrecRecord();
```

```
try
{
    // Move the contents, by using a Hashtable as an
    intermediate holder.
    Hashtable h = new HashtableUnloader(new
    PrefixChanger("xpCommarea.", "mwdrecStart."))
         .unload(dv);
    new HashtableLoader.load(h, qr);
    }
    catch (Exception e)
    {
    }

    return qr;
}
```

Task 4: Update the JAM Configuration and WebLogic Server web.xml

To update the WebLogic JAM configuration and WebLogic Server web.xml, perform the following steps.

Step 1: Update the jcrmgw.cfg.

Update the `jcrmgw.cfg` file with the remote service entries shown in Listing 15. The update is done through the Administration Console. Refer to the *BEA WebLogic Java Adapter for Mainframe Administration and Configuration Guide* for more information.

The WebLogic JAM gateway must be restarted for new services. The entries are used when the corresponding form button is pushed. The `doit` button triggers `DoIt`, which invokes `XPLXSBEA`. The service name must match values in the eGen script. In this example, the `RNAME` must match an actual CICS program name.

Listing 15 Remote Service Entries for Create/Read/Update/Delete

```
DoIt                RDOM="CICS410"
                    RNAME="XPLXSBEA"
```

Step 2: Update the WebLogic Server web.xml file.

Update the WebLogic Server `web.xml` file with the entries shown in Listing 16. The update may be done from a command line or through the Administration Console. Refer to the *BEA WebLogic Java Adapter for Mainframe Administration and Configuration Guide* for more information.

Listing 16 Update WebLogic Server web.xml File

```
weblogic.httpd.register.crossplex=customServlet
```

Task 5: Deploy Your Application

You now have a basic form, capable of receiving data entry, and some static HTML code for display. For a complete description of how to deploy a servlet, refer to the WebLogic Server documentation.

Running the Sample

Now you are ready to use the application. Figure 4 shows the default servlet with customized code displayed in an HTML facade. This type of servlet is useful for presentation, proof-of-concept, and as a test for development.

Figure 4 New Data Entry Servlet Display

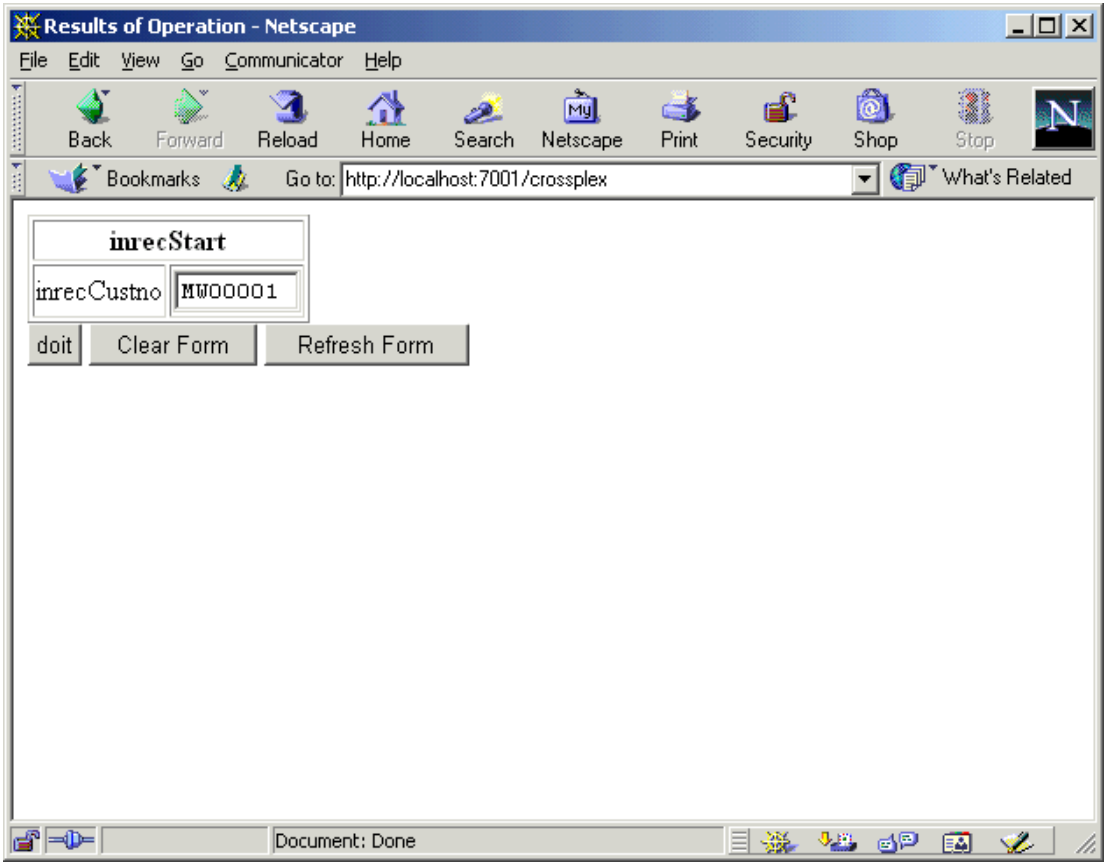


Figure 5 is an example of the page used for the front end of the new custom servlet.

Figure 5 New Data Entry Servlet Front End Page