# XML For the Enterprise

## Providing An XML Interface To A CICS Application

Teodoro Cipresso
Anthony Flusche
Gary Mazo

IBM Silicon Valley Laboratory

Second Edition
February, 2003

## Table of contents

## Acknowledgements

Many thanks to Andy Krasun, Michael Connor, Stephen Hancock and John Lawrence for reviewing this paper and providing invaluable comments.

## Introduction

With the advent of Web Services, users and owners of many mainframe applications that used to rely on just binary interfaces for communication have been trying to harness XML as a new means of information exchange. This approach presents unique opportunities and challenges to the programmers who are trying to efficiently adapt business applications in order to process and produce XML documents with minimal disruptions to the existing system infrastructure. An example of one such infrastructure would be a bank Call Support Center in which operators utilize 3270 terminals to access a mainframe CICS application that retrieves and updates customer and account information. Creating new points of access to such a system may include the addition of a Web-based interface, an automated Voice Response Unit (VRU), or a Web service capability.

In this paper, we will describe how IBM WebSphere® Studio Enterprise Developer™ (WSED) tools help you in modernizing your Enterprise assets and adapting them to process and produce XML messages.

## Business Application Sample

Throughout this paper, we will use an existing CICS application that is included in the samples for IBM WebSphere® Studio Enterprise Developer™. First, you will go through installing and running this application and then you will learn a how to enable this application to process and produce XML messages using IBM WebSphere® Studio Enterprise Developer™.

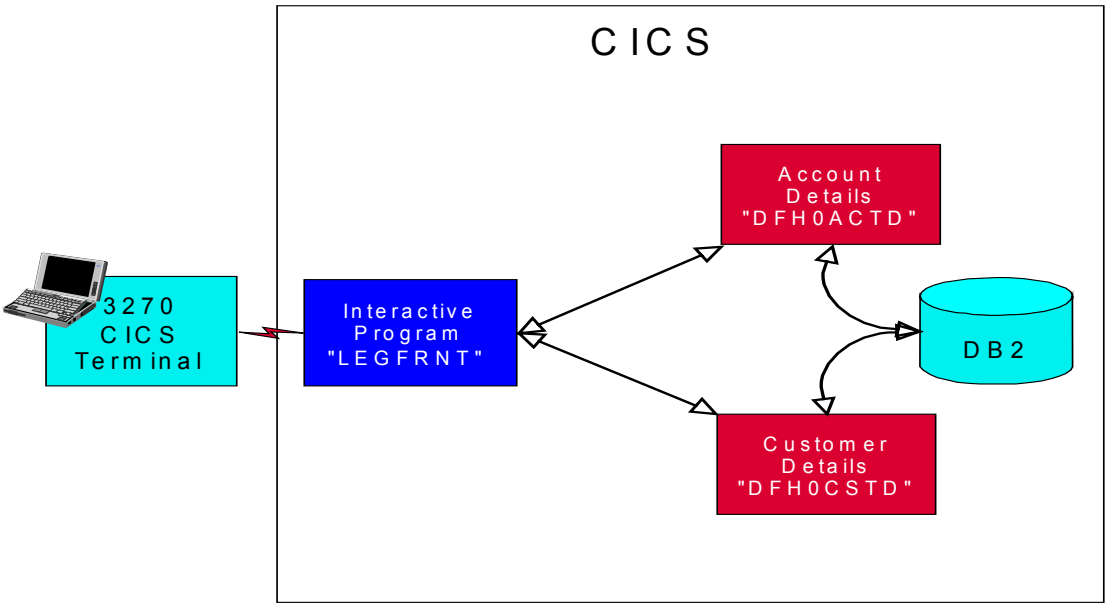


**Figure 1, Existing CICS Application**

This CICS application consists of an interactive program, LEGFRNT, which calls two CICS programs, DFH0ACTD and DFH0CSTD. They, in turn, access the DB2 table to retrieve customer and account information. This information is exchanged in a binary form via a CICS COMMAREA and the results are displayed on a 3270 terminal. Screens that illustrate the interaction are shown later in this paper. (See Step **(8)** on page 9).
The rest of this paper assumes that you have some basic familiarity with operations of a CICS environment in OS/390 or z/OS.

## Parts of the existing business application

Here are the required programs for the current application:

- DFH0ACTD (Account Details sample program)
- DFH0CSTD (Customer Details sample program)
- LEGFRNT (CICS front end program for executing the business programs)
- LEGMAP (CICS BMS map for front end program)
- DFH$EDB2 (creates DB2 tables for the sample programs)
- DFH$ESQL (DB2 bind for the sample programs DFH0ACTD and DFH0CSTD)
- XML$CEDA (creates CICS table entries).

These COBOL programs are included in WSED samples. You can find the complete list of pre-requisite software in Appendix A. Pre-requisite software on page 31.

## Setting up and running the existing business application

Follow the steps below to set up and run the application.

### Step (1) Preparing your datasets

To set up your application, you allocate the following partitioned data sets (PDSs or PDSEs) then transfer the associated members. The source files for these programs are shipped in the Samples directory on the WSED Installation CD. You can use the IDE for z/OS tools to transfer the source files as members to the appropriate datasets on z/OS.

| Data set name | Data set members | Data set characteristics |
|---|---|---|
| XML.COBOL | COBOL source code:<br>• DFH0ACTD<br>• DFH0CSTD<br>• LEGFRNT | RECFM: FB LRECL: 80<br>BLKSIZE: any (e.g. 4000) |
| XML.CNTL | JCL:<br>• DFH$EDB2<br>• DFH$ESQL<br>• XML$CEDA | RECFM: FB LRECL: 80<br>BLKSIZE: any (e.g. 4000) |
| XML.LOAD | Load modules | RECFM: U LRECL: 0<br>BLKSIZE: 32760 |
| XML.OBJECT | Object decks | RECFM: FB LRECL: 80<br>BLKSIZE: any (e.g. 4000) |
| XML.DBRMLIB | DB2 sample data | RECFM: FB LRECL: 80<br>BLKSIZE: any (e.g. 4000) |
| XML.BMS | LEGMAP (BMS map) | RECFM: FB LRECL: 80<br>BLKSIZE: any (e.g. 4000) |

### Step (2) Configuring DB2

Use the DB2 sample program DSNTIAD (which is shipped with DB2) to create the DB2 tables "ACCOUNT" and "CUSTOMER" which are needed by the sample programs. Use DFH$EDB2 as a template for creating the tables. Create the tables by running the DB2 sample program DSNTIAD. The DSN SYSTEM (…) parameter is the name of your DB2 subsystem. This DB2 subsystem should be connected to your target CICS. You should replace DSTNIAxx with the correct name that corresponds to the release level of your DB2 subsystem, for example, DSTNIA61 for DB2 6.1. Note that starting with release 6 of DB2, DSNTIAD is shipped as source and a load module; in prior releases, it is available as source only. The sample JCL to assemble the DSNTIAD source can be found in DB2.SDSNSAMP(DSNTIJTM) as one of the steps in the control file:

```
//SYSTSIN DD *
      DSN SYSTEM(…)
      RUN PROGRAM(DSNTIAD) PLAN(DSNTIAxx)
      END
//SYSIN  DD    DSN=DFH$EDB2,DISP=SHR
```

### Step (3) Assembling BMS maps

To create the layout of the CICS front end to the sample programs, you assemble the BMS map LEGMAP using the procedure DFHMAPT supplied with CICS. The resultant COBOL copybook is referenced in the LEGFRNT program.

**Step** (4) **Pre-compiling the existing programs**

Since the sample programs contain EXEC SQL statements they must be pre-compiled. Use the DB2 pre-compiler DSNHPC to pre-compile the sample programs DFH0ACTD and DFH0CSTD.

**Step** (5) **Compiling and link-editing the sample**

Compile and Link-Edit the sample and front end programs DFH0ACTD, DFH0CSTD and LEGFRNT using the procedure IGYWCL. Be sure to include the CICS compile option. Ensure that the resulting load modules are in a load data set visible to the CICS RPL.

**Step** (6) **Binding the DB2 tables**

To allow the sample programs to access the DB2 tables use the sample DFH$ESQL to perform a DB2 bind for the programs DFH0ACTD and DFH0CSTD. The contents of DFH$ESQL should be as follows:

```
DSN SYSTEM(…)
  BIND PACKAGE(EBUSCOL) -
       OWNER(DAVIN22) -
       QUALIFIER(DAVIN22) -
       MEMBER(DFH0CSTD) -
       LIBRARY('DAVIN22.DBRMLIB.DATA') -
       ACTION(REP) -
       ISOLATION(CS)   -
       VALIDATE(BIND)  -
       DYNAMICRULES(BIND)  -
       ENABLE(CICS)

  BIND PACKAGE(EBUSCOL) -
      OWNER(DAVIN22) -
      QUALIFIER(DAVIN22) -
      MEMBER(DFH0ACTD) -
      LIBRARY('DAVIN22.DBRMLIB.DATA') -
      ACTION(REP) -
      ISOLATION(CS)   -
      VALIDATE(BIND)  -
      DYNAMICRULES(BIND)  -
      ENABLE(CICS)

 BIND PLAN(EBUSPLAN) -
      OWNER(DAVIN22) -
      QUALIFIER(DAVIN22) -
      ACTION(REP) -
      PKLIST(EBUSCOL.*)  -
      ISOLATION(CS)   -
      VALIDATE(BIND)  -
      DYNAMICRULES(BIND)  -
      ENABLE(CICS)
   END
 COMMIT;
```

Note that you should use your DB2 subsystem ID as the parameter to the DSN SYSTEM directive. Also replace the highlighted fields with your system's high-level qualifier.

**Step** (7) **Configuring CICS**
Define various resources to CICS. A sample XML$CEDA is provided to assist with this:

- A transaction named LEGF
- The programs DFH0ACTD, DFH0CSTD and LEGFRNT
- The BMS map LEGMAP
- A DB2ENTRY for LEGF that connects it to sample plan EBUSPLAN
- A DB2 transaction for LEGF.

**Step** (8) **Running the application**
To start the application bring up a CICS terminal and run transaction LEGF. The following screen should appear. Note: the following instructions illustrate transaction 1. The procedure for transaction 2 is the same.

```
  _____ _____ _____  _____
 / ____|_   _/ ____|/ ____|
| |      | || |     | (___
| |      | || |      \___ \
| |____ _| || |____ ____) |
 \_____|_____\_____|_____/

PLEASE CHOOSE A TRANSACTION:

 1. CUSTOMER DETAIL SCREEN
 2. ACCOUNT DETAIL SCREEN


 ENTER YOUR CHOICE ====>
```

If you enter 1 as your choice the following will appear on the screen:

```
     CUSTOMER DETAIL SCREEN

CUSTOMER NUMBER: 00001

     FIRST NAME:

      LAST NAME:

        ADDRESS:

           CITY:

          STATE:

        COUNTRY:
```

Change 00001 to 00004 and press Enter. The following screen will appear:

```
    CUSTOMER DETAIL SCREEN

CUSTOMER NUMBER: 00004

     FIRST NAME: JOHN

      LAST NAME: ROYSON

        ADDRESS: 15 MISTYVIEW

           CITY: ROANOKE

          STATE: TX

        COUNTRY: US
```

Note that this program is not a complete CICS application and in order to return to the initial screen you will need to restart the LEGF transaction.

## XML-enabled Business Application

In order to allow XML documents to flow through to the existing business programs, the source of those programs is passed through the XML Enablement tool of the IBM WebSphere® Studio Enterprise Developer™ (WSED). The tool generates a set of COBOL programs called "XML converters" (Inbound and Outbound) based on the original binary interface. The tool also generates a template COBOL program called "Converter driver" that illustrates how to invoke the converters. In this paper we will show how to augment the driver with EXEC CICS statements to call the existing business application in concert with calling the XML converters. An interactive menu-driven 3270 front-end program facilitates local testing of the new application. The diagram below shows the structure of the modernized application.

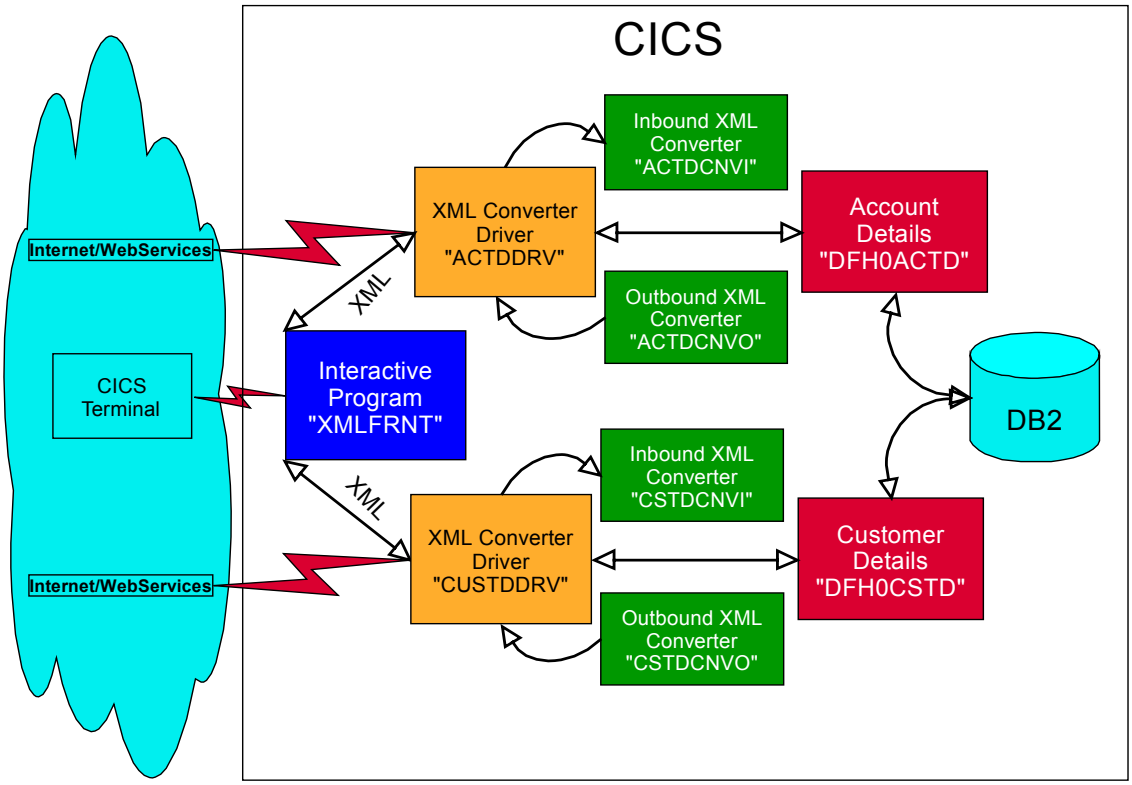


**Figure 2, Existing Business CICS Application Enabled for XML**

Parts of the application are described in details later in the paper.

The rest of the paper describes the process that you follow to replace the binary interface of two sample COBOL CICS programs with an XML interface.

## Using WSED XML Enablement Tool

This chapter describes how to use the XML Enablement tool in WSED to generate XML Converters and a Driver template. Before you read this chapter you should familiarize yourself with the basic concepts of WSED, including its IDE for z/OS feature.

Required programs for this sample:

- **DFH0ACTD** (Account Details sample program)
- **DFH0CSTD** (Customer Details sample program).

These COBOL programs are shipped with WSED samples.

During development of your application, you will also use the following programs and datasets:

- **ACTDCNV(I,O)** - account details inbound/outbound XML converters that are generated by the XML Converter generator. The **inbound** XML converter is a COBOL program that processes an incoming XML document and converts contents of its elements into a COBOL data structure. The inbound converter uses high-performance XML parsing capabilities of the latest IBM Enterprise COBOL compiler and runtime to efficiently parse the inbound XML document. During the parse, the inbound XML converter converts XML data and stores it in the existing application's COBOL data structure (in case of a CICS application this data structure is a CICS COMMAREA). The conversion and moving of data is based on proprietary algorithms that provide high efficiency in transforming character data from the XML document into appropriate COBOL data. The **outbound** XML converter is a COBOL program that takes the results of the execution of the existing transactional program and converts COBOL data into an XML message. That message will be returned to the client. In case of an error during execution of the transaction, an XML-based error message will be returned
- **CSTDCNV(I,O)** - customer details inbound/outbound XML converters that are generated by the XML Converter generator
- **ACTDDRV** - account details XML converter driver that is generated by the XML Converter generator. The XML converter **driver** is a COBOL program that shows the invocation sequence for the inbound converter, the existing program and the outbound converter
- **CSTDDRV** - customer details XML converter driver that is generated by the XML Converter generator
- **XMLFRNT** - CICS front-end program for executing sample programs
- **XMLMAP** - CICS BMS map for front-end program
- **DFH$EDB2** - creates DB2 tables for the sample programs
- **DFH$ESQL** - DB2 bind for the sample programs DFH0ACTD and DFH0CSTD
- **XML$CEDA** - creates CICS table entries.

Follow the steps outlined below to create, set up and run your XML-enabled business application.

### **Step** (1) **Locating your existing application**

You locate the sources for DFH0ACTD and DFH0CSTD on your z/OS system. In order to do that, you can use IDE for z/OS tools in WSED. You define and connect to the remote z/OS system. For this example, let's assume that the system you connect to is called ADSE. Once you are connected to ADSE, locate the sources for DFH0ACTD and DFH0CSTD. Let's assume that they are located in a PDS called XML.COBOL under the high level qualifier (HLQ) DAVIN22.

**Step** (2) **Creating a WSED project**

You create a local container for the generated XML Converters and the Driver template. This local container is called "Simple Project". In WSED, switch to the Resource perspective and invoke the New Project wizard. Select "Simple" as the type of the project you want to create. Follow the wizard to create a project called "XML Account Test" (Fig. 3.)

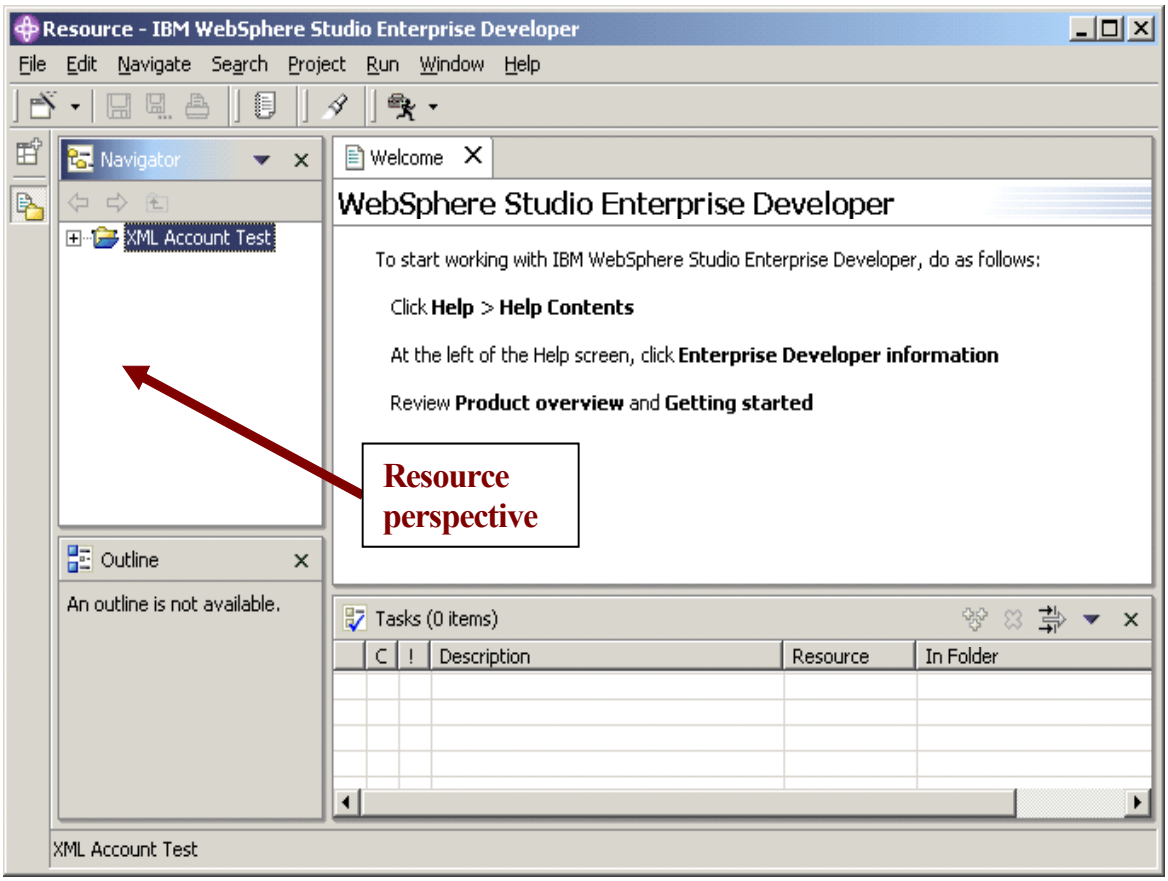


**Figure 3, XML Account Test Project**

**Step** (3) **Moving your existing application to WSED**

You use Copy and Paste operations on the source COBOL files for DFH0ACTD and DFH0CSTD to copy them into your local XML Account Test Project. (Fig. 4) Hint: You use the "Go Into" and "Refresh" actions to "go into" the XML.COBOL PDS represented as a Folder in the Resource perspective.

If your source files are located in the local file system on your PC, you can use the File->Import operation provided in the Workbench to import your source files into your local project.

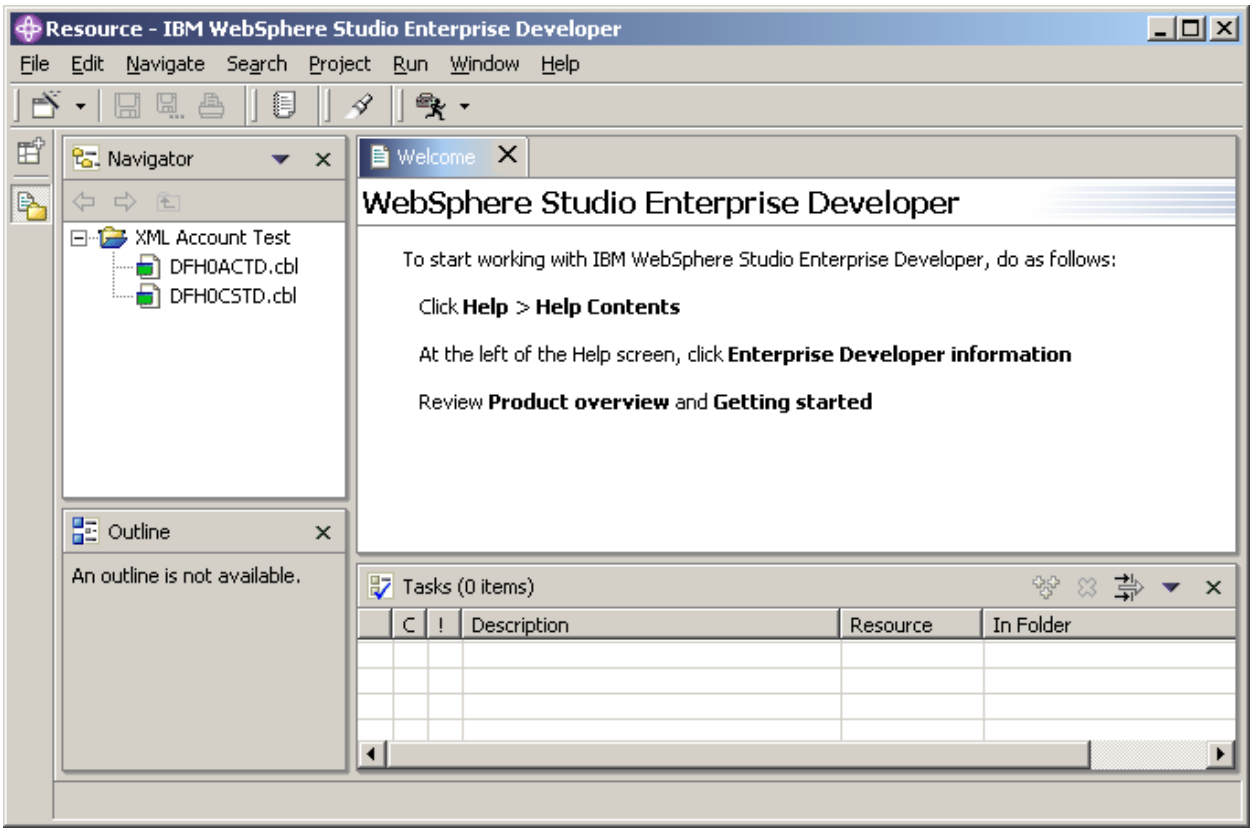


**Figure 4, XML Account Test Project With Source**

Note that the IDE appends .CBL to the source file name. This way you can use file type-specific WSED tools like a language-sensitive editor.

**Step (4) Invoking the XML Converter generator wizard**

Invoke XML Converter generator for the source program DFH0ACTD.cbl. To do that, you select DFH0ACTD.cbl in the Navigator view and invoke the pop-up menu by pressing the right mouse button. (Fig. 5)

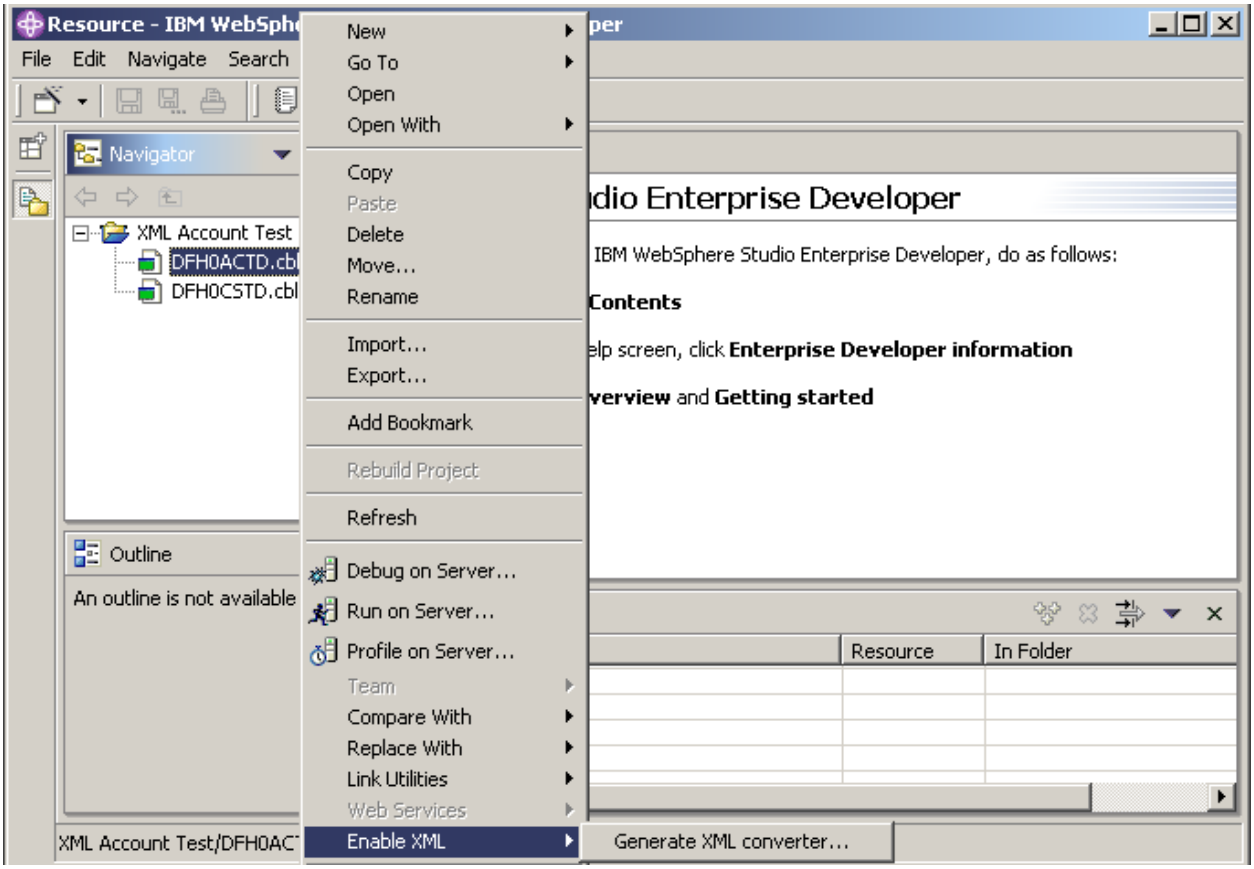


**Figure 5, Invoking XML Converter generator**

## Step (5) Specifying input and output files for the wizard

You use the first page of the XML Converter wizard to select input and output files for the Converter, Driver template, and the XML Schema. (Fig.6) Enter ACTDCNV.cbl in the Converter File Name field and ACTDDRV.cbl in the Converter Driver Filename. You don't need to modify the rest of the specified defaults on the first page. Hint: The XML Schema is automatically generated that contains the description of the names and types of XML elements. These elements can appear in an XML document that our program will process and generate. For more information on XML Schema visit http://www.w3.org/XML/Schema

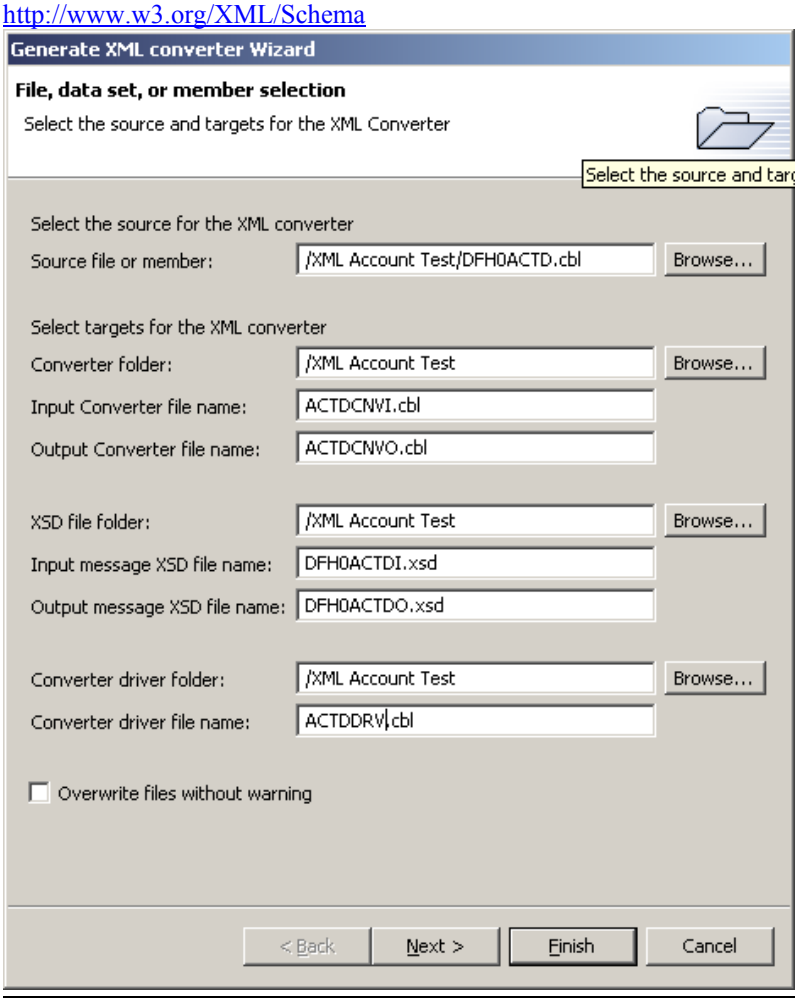


**Figure 6, File Selection**

On this page, the fields are as follows:

- Source file: Specify where your existing COBOL program is located
- Converter folders: Specify folder(s) where the wizard will generate converter program(s)
- Converter file names: Specify the name(s) you want to give your converter file(s)
- XSD file folders: specify where the wizard will generate the XML Schema file(s)
- XSD file names: Specify the name(s) you want to give your XML Schema file(s)
- Converter driver folder: Specify folder where the wizard will generate the driver program
- Converter driver file name: Specify the name you want to give your converter driver file
- Overwrite files without warning: Select if you want to overwrite existing output files

## Step (6) Specifying the generation options

Use the second page of the XML Converter wizard to specify generation options for your Converter and Driver programs. Enter ACTDCNV in the Program Name entry field and choose an appropriate inbound code page. You don't need to modify the rest of the specified defaults on this page. (Fig. 7)

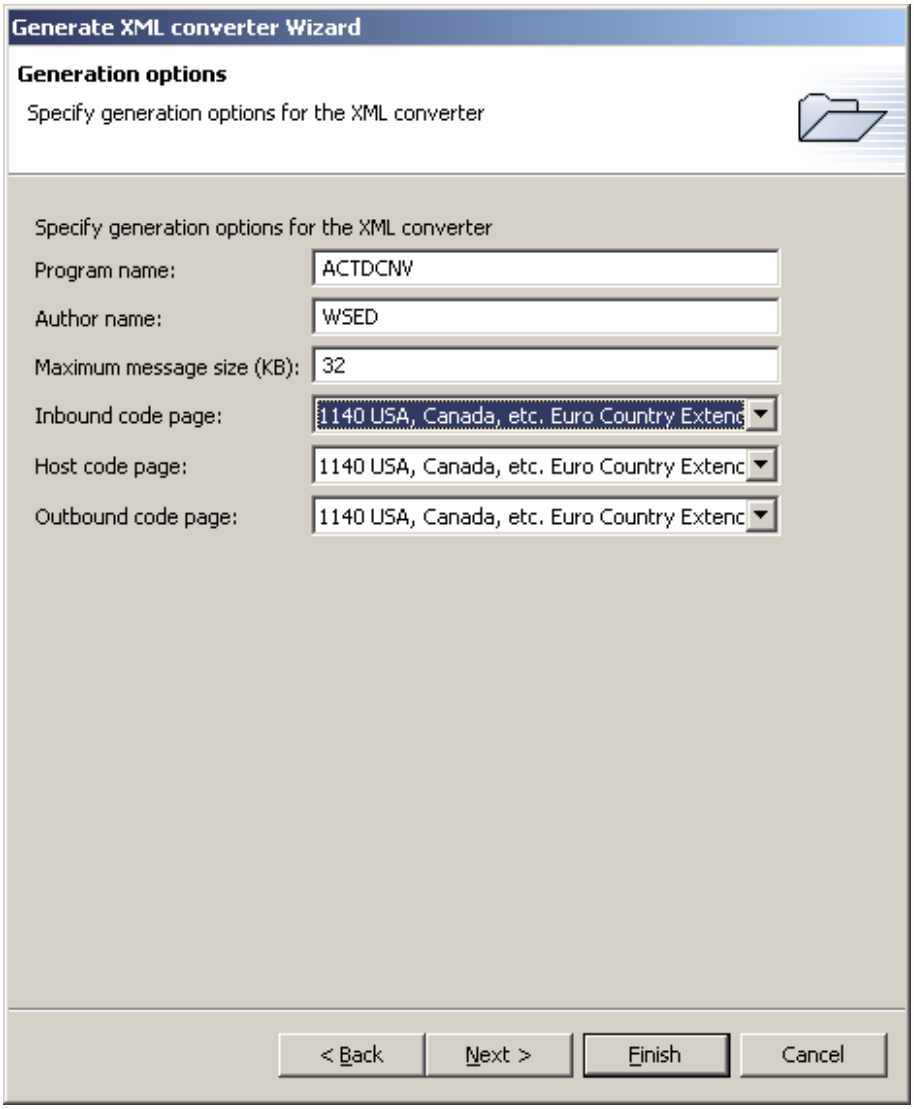


**Figure 7, Generation Options Selection**

On this page, the fields are as follows:

- Program name – specify the "stem" value for the program names in the PROGRAM-ID paragraph of the IDENTIFICATION DIVISION in the COBOL programs that this wizard will generate. For example, if you enter ACTDCNV, the wizard will generate ACTDCNVI for the inbound converter program name, ACTDCNVO for the outbound converter, and ACTDCNVD for the converter driver.
- Author name – specify the value for the AUTHOR paragraph
- Maximum message size – specify the maximum size of the XML message that will need to be allocated when processing and generating the XML message.
- Code pages - specify code page(s) for the encoding of the inbound and outbound XML documents, and the code page for the host data.

Note: The default values for this page are taken from the default preference values stored in the WSED Workbench. You can modify those defaults by visiting the Preference page for the wizard. The process for modifying the preferences is described in "Appendix B. Modifying COBOL Generator preferences" on page 32.

**Step (7) Specifying input and output data structures**

You use the third page of the XML Converter wizard to specify the input and output data structures for which you want to generate the equivalent XML-based interface. From the two pull-down combo boxes, select DFHCOMMAREA as both input and output structure. (Fig. 8)

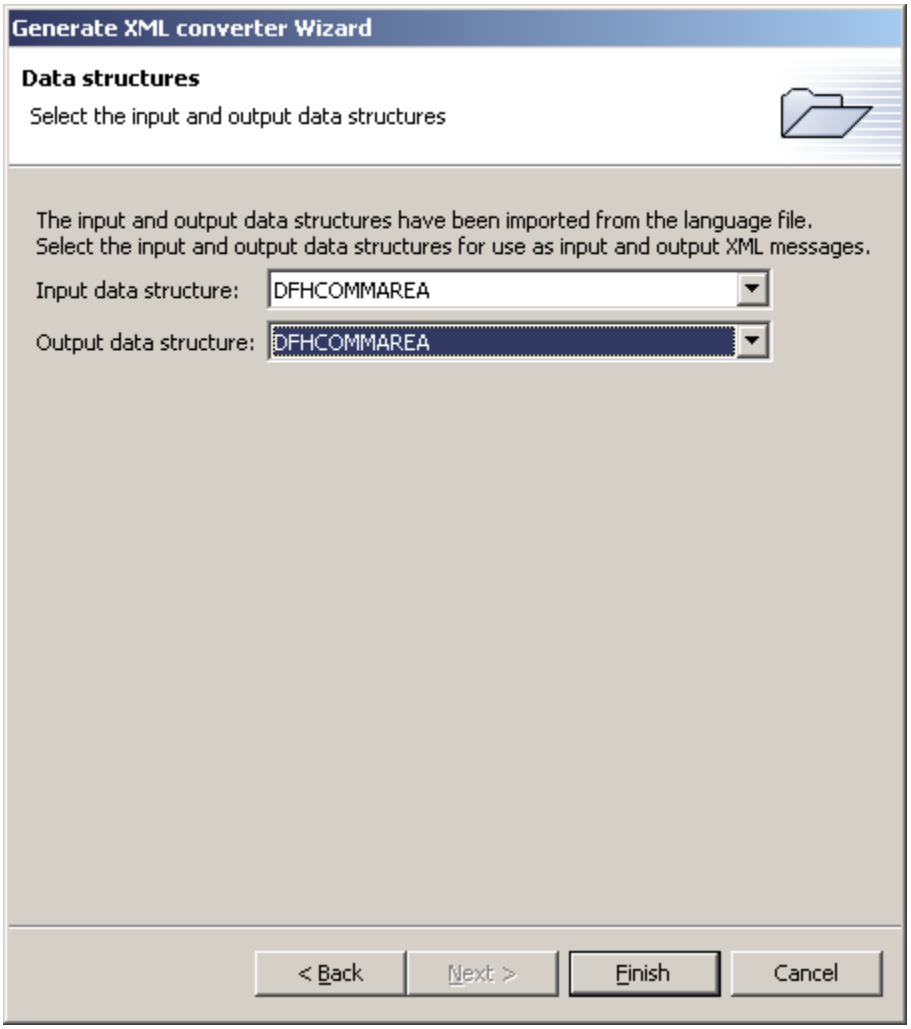


**Figure 8, Data Structure Selection**

On this page, the fields are as follows:

- `Input data structure` – select the level 01 data from an available list of level 01 COBOL data structures extracted from your existing application program. This data structure will be used to generate inbound XML converters.
- `Output data structure` – same as above but used to generate outbound XML converters.

Note: Some of the attributes for COBOL data items and their treatment by the generated converters depend on COBOL compiler options. An example of such options is the TRUNC(BIN) option that causes the z/OS COBOL compiler to treat all binary items as if they all were native binary items. You can modify these options using the COBOL Importer preference page described in Appendix C. Modifying COBOL Importer preferences on page 34.

## Step (8) Generating code

You press Finish to complete the generation process. After the wizard processing completes, you will notice that your Converter and Driver files are generated and displayed in the Navigator view. (Fig. 9)

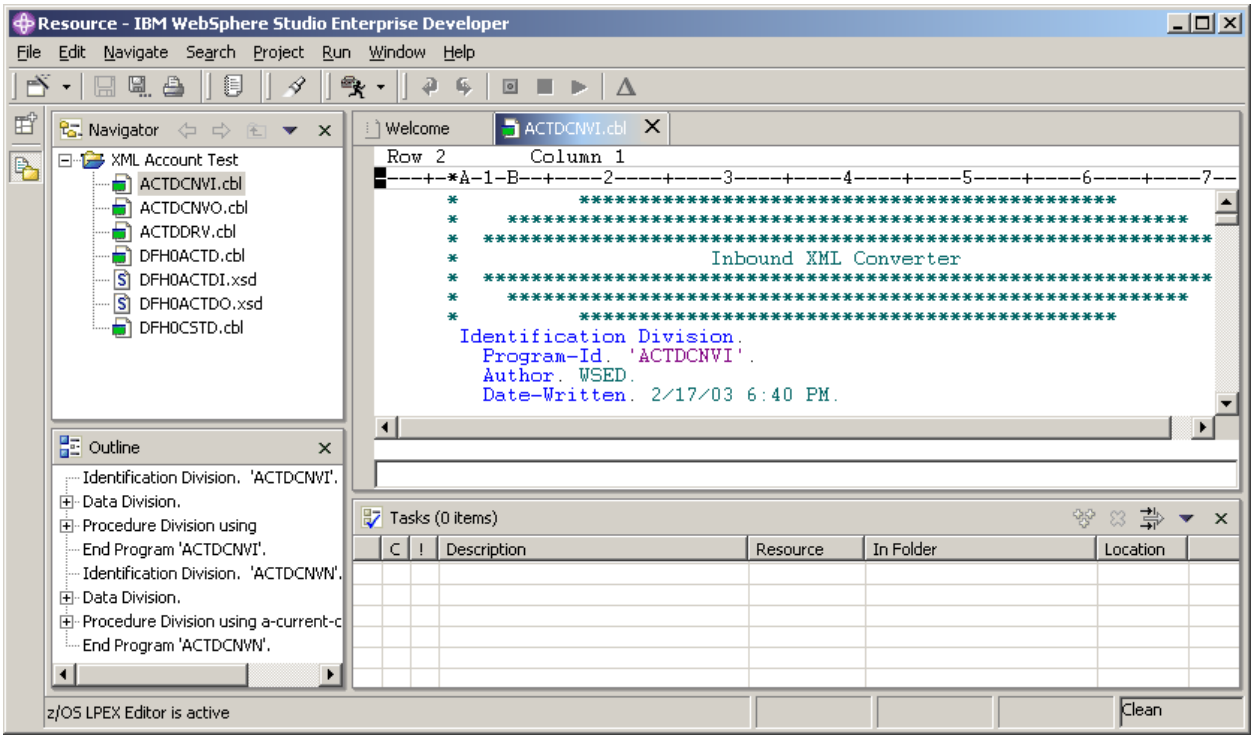


**Figure 9, Generated Converters and Driver template for DFH0ACTD**

The inbound converter uses the new Enterprise COBOL language, and specifically, the new XML PARSE verb to parse XML documents and convert XML into COBOL data:

```
...
xml parse a-input-xml (1:a-input-xml-len)
       processing procedure a-xml-handler
       thru a-general-logic-exit
      on exception
        perform a-unregister-exception-handler
        perform a-signal-condition
      not on exception
        perform a-unregister-exception-handler
        move zero to a-converter-return-code
  end-xml
...
```

The outbound converter converts the output COBOL data from the existing program into an output XML message:

```
...
move corresponding DFHCOMMAREA
 to a-xml-response
...
```

## Step (9) Generating additional converters

You use the XML Converter wizard as described in steps 4 through 8 above to generate Converters and Driver template for DFH0CSTD. You will notice that Converter and Driver files got generated and displayed in the Navigator view. (Fig. 10)

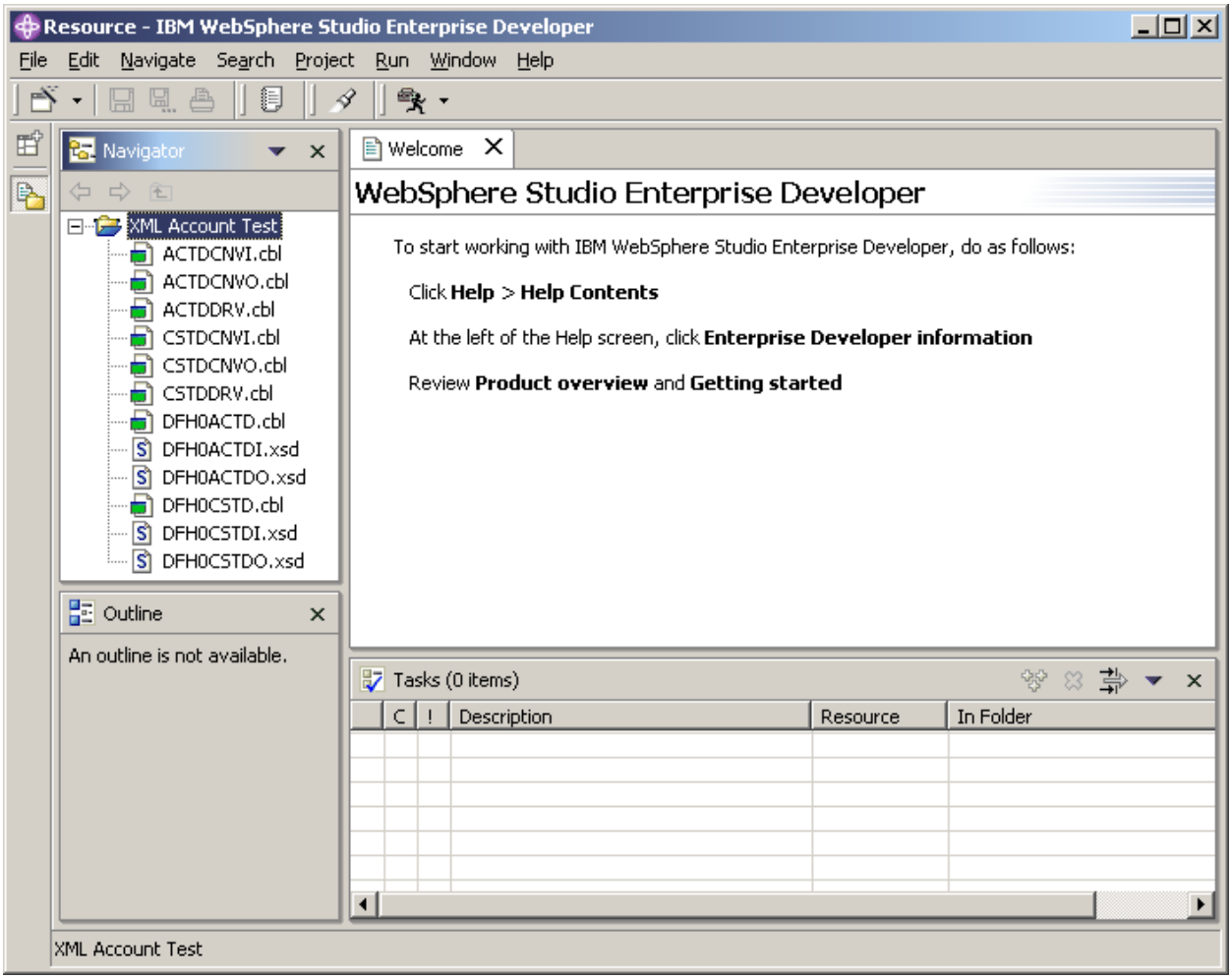


**Figure 10, Generated Converters and Driver template for DFH0CSTD**

**Step (10) Modifying the converter driver program**

You can now modify the converter driver template that got generated by the tool. This is needed to correctly invoke the inbound converter, the existing application, and the outbound converter using the EXEC CICS language. The driver template also provides error-handling mechanisms that can be modified to suit your needs. Modify the mainline section of the ACTDDRV driver program as follows (the necessary changes are highlighted in bold):

```
 Process opt,lib,codepage(01140),CICS
*          ******************************************
*    ************************************************
*  **************************************************************
*                   XML Converter Driver Program
*  **************************************************************
*    ************************************************
*          ******************************************
 Identification Division.
   Program-Id. 'ACTDDRV'.
   ...
 Data Division.
  Working-Storage Section.
  Local-Storage Section.

   ...

* ************************************
* ** Business Program Binary Interface **
* ************************************
 01   BUSINESS-DATASTRUCT.
 05  CUSTNO     PIC S99999 .
 05  ACCTNO     PIC S99999 .
 05  BALANCE    PIC S9999V99 .
  Linkage Section.

* ************************************
* ** New Business Program XML Interface **
* ************************************
  1 DFHCOMMAREA.
   2 a-interface-xml-text-len   pic 9(9) binary.
   2 a-interface-xml-text       pic x(2048).
 Procedure Division using DFHCOMMAREA.
  Mainline Section.
* + ------------------------- +
* | Enable Exception Handler   |
* + ------------------------- +
     perform a-register-exception-handler
* + ---------------------------- +
* | Execute Inbound XML Converter   |
* + ---------------------------- +
     move 'N' to a-exception-occurred
     perform a-inbound-conversion
* + ------------------------------------------+
* | Send XML Error Message If Exception Occurred |
* + ------------------------------------------+
     if a-exception-occurred = 'Y'
       move a-message-buffer to a-error-description
       move a-failure-message-number to a-error-message-number
```

```
            move a-converter-return-code to a-error-code
            move length of a-failure-response
              to a-interface-xml-text-len
            move a-failure-response
              to a-interface-xml-text(1:a-interface-xml-text-len)
            perform a-unregister-exception-handler
            exec cics return
            end-exec
          end-if
      * + ------------------------------ +
      * | Execute Current Business Program |
      * + ------------------------------ +
          exec cics link
             program ('DFH0ACTD')
             commarea (BUSINESS-DATASTRUCT)
          end-exec
      * + ------------------------------ +
      * | Execute Outbound XML Converter   |
      * + ------------------------------ +
          perform a-outbound-conversion
      * + ---------------------------------------- +
      * | Send XML Error Message If Exception Occurred |
      * + ---------------------------------------- +
          if a-exception-occurred = 'Y'
          perform a-populate-failure-response
          perform a-unregister-exception-handler
          exec cics return
          end-exec
          .
        a-register-exception-handler.
          set a-routine to entry 'ACTDCNVH'
          set a-token to address of a-failure-data
          call 'CEEHDLR' using a-routine a-token a-feedback-code
          if not CEE000 of a-feedback-code
            display 'Failed To Register Exception Handler'
              msg-no of a-feedback-code
            stop run
          end-if
          .
        a-unregister-exception-handler.
          call 'CEEHDLU' using a-routine a-feedback-code
          if not CEE000 of a-feedback-code
            display 'Failed To Unregister Exception Handler'
              msg-no of a-feedback-code
            stop run
          end-if
          .
        a-inbound-conversion.
          call 'ACTDCNVI'
            using
              BUSINESS-DATASTRUCT
              a-interface-xml-text-len
              a-interface-xml-text
              omitted
      *       a-optional-feedback-code
            returning
              a-converter-return-code
          .
```

```
        a-outbound-conversion.
           call 'ACTDCNVO'
             using
               BUSINESS-DATASTRUCT
               a-interface-xml-text-len
               a-interface-xml-text
               omitted
      *        a-optional-feedback-code
             returning
               a-converter-return-code
           .
       End Program 'ACTDDRV'.
      *           ******************************************
      *     ********************************************************
      *   ************************************************************
      *                          Exception Handler
      *   ************************************************************
      *     ********************************************************
      *           ******************************************
       Process NOCICS
       Identification Division.
         Program-Id. 'ACTDCNVH'.
         ...
       Data Division.

        ...

       End Program 'ACTDCNVH'.
```

Modify the mainline section of the CSTDDRV driver program as follows (the necessary changes are highlighted in bold):

```
 Process opt,lib,codepage(01140),CICS
*          ******************************************
*    ************************************************
*  ********************************************************
*                   XML Converter Driver Program
*  ********************************************************
*    ************************************************
*          ******************************************
 Identification Division.
   Program-Id. 'CSTDDRV'.
   ...
 Data Division.
  Working-Storage Section.
  Local-Storage Section.

      ...

* ************************************
* ** Business Program Binary Interface **
* ************************************
 01   BUSINESS-DATASTRUCT.
 05  CUSTNO     PIC S99999 .
 05  LASTNAME   PIC A(25) .
 05  FIRSTNAME  PIC A(15) .
 05  ADDRESS1   PIC X(20) .
 05  CITY       PIC A(20) .
 05  STATE      PIC A(10) .
 05  COUNTRY    PIC X(15) .
  Linkage Section.

* ************************************
* ** New Business Program XML Interface **
* ************************************
 1 DFHCOMMAREA.
   2 c-interface-xml-text-len   pic 9(9) binary.
   2 c-interface-xml-text       pic x(2048).
 Procedure Division using DFHCOMMAREA.
  Mainline Section.
* + ------------------------- +
* | Register Exception Handler |
* + ------------------------- +
     perform c-register-exception-handler
* + ----------------------------- +
* | Execute Inbound XML Transformer |
* + ----------------------------- +
     move 'N' to c-exception-occurred
     perform c-inbound-conversion

* + ---------------------------------------- +
* | Send XML Error Message If Exception Occurred |
* + ---------------------------------------- +
     if c-exception-occurred = 'Y'
```

```
           move c-message-buffer to c-error-description
           move c-failure-message-number to c-error-message-number
           move c-converter-return-code to c-error-code
           move length of c-failure-response
             to c-interface-xml-text-len
           move c-failure-response
             to c-interface-xml-text(1:c-interface-xml-text-len)
           perform c-unregister-exception-handler
           exec cics return
           end-exec
         end-if

      * + ------------------------------ +
      * | Execute Current Business Program |
      * + ------------------------------ +
         exec cics link
            program ('DFH0CSTD')
            commarea (BUSINESS-DATASTRUCT)
         end-exec
      * + ------------------------------ +
      * | Execute Outbound XML Transformer |
      * + ------------------------------ +
         perform c-outbound-conversion
      * + ---------------------------------------- +
      * | Send XML Error Message If Exception Occurred |
      * + ---------------------------------------- +
         if a-exception-occurred = 'Y'
         perform a-populate-failure-response
         perform c-unregister-exception-handler
         exec cics return
         end-exec
         .
       c-register-exception-handler.
         set c-routine to entry 'CSTDCNVH'
         set c-token to address of c-failure-data
         call 'CEEHDLR' using c-routine c-token c-feedback-code
         if not CEE000 of c-feedback-code
           display 'Failed To Register Exception Handler'
             msg-no of c-feedback-code
           stop run
         end-if
         .
       c-unregister-exception-handler.
         call 'CEEHDLU' using c-routine c-feedback-code
         if not CEE000 of c-feedback-code
           display 'Failed To Unregister Exception Handler'
             msg-no of c-feedback-code
           stop run
         end-if
         .
       c-inbound-conversion.
         call 'CSTDCNVI'
           using
             BUSINESS-DATASTRUCT
             c-interface-xml-text-len
             c-interface-xml-text
             omitted
      *      c-optional-feedback-code
```

```
            returning
              c-converter-return-code
          .
        c-outbound-conversion.
           call 'CSTDCNVO'
             using
               BUSINESS-DATASTRUCT
               c-interface-xml-text-len
               c-interface-xml-text
               omitted
      *        c-optional-feedback-code
             returning
               c-converter-return-code
          .
       End Program 'CSTDDRV'.
      *          ******************************************
      *    ************************************************************
      *  ****************************************************************
      *                        Exception Handler
      *  ****************************************************************
      *    ************************************************************
      *          ******************************************
       Process NOCICS
       Identification Division.
         Program-Id. 'CSTDCNVH'.
         ...

       End Program 'CSTDCNVH'.
```

## Step (11) Preparing your datasets

To set up your XML-enabled application, you allocate the following partitioned data sets then transfer the associated members. The original sources as well as generated programs are shipped in the XML4ESMP.ZIP file in WSED. You can use the IDE for z/OS tools to transfer the source files as members to the appropriate datasets on z/OS. Data set characteristics are described in "Setting up and running the existing business application" on page 7.

a. XML.COBOL
   - DFH0ACTD*
   - DFH0CSTD*
   - ACTDCNVI
   - ACTDCNVO
   - CSTDCNVI
   - CSTDCNVO
   - ACTDDRV
   - CSTDDRV
   - XMLFRNT

b. XML.CNTL*
   - DFH$EDB2*
   - DFH$ESQL*
   - XML$CEDA*

c. XML.LOAD*

d. XML.OBJECT*

e. XML.DBRMLIB*

f. XML.BMS*
   - XMLMAP

## Step (12) Configuring DB2*

Use the DB2 sample program DSNTIAD (which is shipped with DB2) to create the DB2 tables "ACCOUNT" and "CUSTOMER" which are needed by the sample programs. Use DFH$EDB2 as a template for creating the tables. Create the tables by running the DB2 sample program DSNTIAD. The DSN SYSTEM (…) parameter is the name of your DB2 subsystem. This DB2 should be connected to your target CICS. You should replace DSTNIAxx with the correct name that corresponds to the release level of your DB2 subsystem, for example, DSTNIA61 for DB2 6.1. Note that starting with release 6 of DB2, DSNTIAD is shipped as source and a load module; in prior releases, it is available as source only. The sample JCL to assemble the DSNTIAD source can be found in DB2.SDSNSAMP(DSNTIJTM) as one of the steps in the control file:

```
//SYSTSIN DD *
      DSN SYSTEM(…)
      RUN PROGRAM(DSNTIAD) PLAN(DSNTIAxx)
      END
//SYSIN  DD    DSN=DFH$EDB2,DISP=SHR
```

* These files/steps are not necessary if you installed and ran the existing application described earlier in this paper (See "Setting up and running the existing business application" on page 7).

**Step** (13) **Assembling BMS maps** *

To create the layout of the CICS front end to the sample programs, you assemble the BMS map XMLMAP using the procedure DFHMAPT supplied with CICS. The resultant COBOL copybook is referenced in the XMLFRNT program.

**Step** (14) **Pre-compiling the existing programs** *

Since the sample programs contain EXEC SQL statements they must be pre-compiled. Use the DB2 pre-compiler DSNHPC to pre-compile the sample programs DFH0ACTD and DFH0CSTD.

**Step** (15) **Compiling and link-editing the existing application**

Compile and Link-Edit the sample and front end programs DFH0ACTD*, DFH0CSTD* and XMLFRNT using the procedure IGYWCL. Be sure to include the CICS compile option. Ensure that the resulting load modules are in a load data set visible to the CICS RPL.

**Step** (16) **Compiling and link-editing the XML processing code**

Compile and Link-Edit the XML Converter and Converter driver programs ACTDDRV, ACTDCNVI, ACTDCNVO, CSTDDRV, CSTDCNVI, CSTDCNVO using the procedure IGYWCL. Please do not specify the CICS compiler option when building these programs as it is provided where appropriate in the source.

* These files/steps are not necessary if you installed and ran the existing application described earlier in this paper (See "Setting up and running the existing business application" on page 7).

## Step (17) Binding the DB2 tables *

To allow the sample programs to access the DB2 tables use the sample DFH$ESQL to perform a DB2 bind for the programs DFH0ACTD and DFH0CSTD. The contents of DFH$ESQL should be as follows:

```
DSN SYSTEM(…)
  BIND PACKAGE(EBUSCOL) -
       OWNER(DAVIN22) -
       QUALIFIER(DAVIN22) -
       MEMBER(DFH0CSTD) -
       LIBRARY('DAVIN22.DBRMLIB.DATA') -
       ACTION(REP) -
       ISOLATION(CS)   -
       VALIDATE(BIND)  -
       DYNAMICRULES(BIND)  -
       ENABLE(CICS)

  BIND PACKAGE(EBUSCOL) -
      OWNER(DAVIN22) -
      QUALIFIER(DAVIN22) -
      MEMBER(DFH0ACTD) -
      LIBRARY('DAVIN22.DBRMLIB.DATA') -
      ACTION(REP) -
      ISOLATION(CS)   -
      VALIDATE(BIND)  -
      DYNAMICRULES(BIND)  -
      ENABLE(CICS)

 BIND PLAN(EBUSPLAN) -
      OWNER(DAVIN22) -
      QUALIFIER(DAVIN22) -
      ACTION(REP) -
      PKLIST(EBUSCOL.*)  -
      ISOLATION(CS)   -
      VALIDATE(BIND)  -
      DYNAMICRULES(BIND)  -
      ENABLE(CICS)
   END
 COMMIT;
```

Note that you should use your DB2 subsystem ID as the parameter to the DSN SYSTEM directive. Also replace the highlighted fields with your system's high-level qualifier.

## Step (18) Configuring CICS

Define the various resources to CICS. A sample XML$CEDA is provided to assist with this:

- A transaction named XMLF
- The programs DFH0ACTD*, DFH0CSTD* and XMLFRNT
- The BMS map XMLMAP
- A DB2ENTRY for XMLF that connects it to sample plan EBUSPLAN
- A DB2 transaction for XMLF.

## Step (19) Running the application

To start the application bring up a CICS terminal and run transaction XMLF. The following screen should appear. Note: the following instructions illustrate transaction 1. The procedure for transaction 2 is the same.

```
 _____ _____ _____ _____  __   ____  __ _
/  ____|_   _/  ____|/  ____| \ \ / /  \/  | |
| |      | || |     | (___     \ V /| \  / | |
| |      | || |      \___ \     > < | |\/| | |
| |____ _| || |____ ____) |   / . \| |  | | |__
 \_____|_____\_____|_____/   /_/ \_\_|  |_|____|


 PLEASE CHOOSE AN XML ENABLED TRANSACTION:
1. DB2 BANKACCOUNT TABLES FOR THE CUSTOMER DETAILS.
2. DB2 BANKACCOUNT TABLES FOR THE ACCOUNT DETAILS.

  ENTER YOUR CHOICE ====>
```

If you enter 1 as your choice the following will appear on the screen:

```
<?xml version="1.0" encoding="ibm-1140"?><message>  <custno>1</custno>
<lastname>filler</lastname>  <firstname>filler</firstname>
<address1>filler</address1>
<city>filler</city>  <state>filler</state>
<country>filler</country></message>
```

The XML document above represents input to the transaction that will retrieve customer information from a DB2 table. When you press enter the request is executed and the output XML message will appear on the screen that contains the information retrieved from DB2.

```
<?xml version="1.0" encoding="ibm-1140"?><DFHCOMMAREA><custno>
00001</custno><lastname>WEAVER                    </lastname><firstname>RICK
</firstname
><address1>5 WEST KIRKWOOD     </address1><city>ROANOKE </city><state>TX
</state><country>US </country></DFHCOMMAREA>
```

**Step (20) Error reporting**

The XML Converters are able to report errors. You can verify that by performing the following test. Clear the terminal screen and run transaction XMLF and select option 1.

When the following screen appears alter one of the XML tags to make the XML document not well formed then press enter.

```
<?xml version="1.0" encoding="ibm-1140"?><message>  <custno>1</custno>
<lastname>filler</lastname>  <firstname>filler</firstname>
<address1>filler</address1>
<city>filler</city>  <state>filler</state>
<country>filler</country></message>
```

An XML document containing the error message appears:

```
<?xml version="1.0"?><failureResponse>
<errorMessageNumber>000000280</errorMessageNumber>
<errorCode>000000005</errorCode><errorDescription>IGZ0280S Inbound XML
conversion failed because an error return code of 5 was received from the XML
PARSE statement.  The error occurred at element "lastnamx" with the character
content "1  ".
</errorDescription></failureResponse>
```

## Appendix A. Pre-requisite software

The following software is required to develop and run sample applications described in this paper:

- CICS Transaction Server for OS/390 Version 1 Release 3 (program number 5655-147) or later
- IBM Database 2 Universal Database Server for OS/390 (DB2) Version 6 Release 1 (program number 5675-DB2) or later
- IBM Enterprise COBOL for z/OS and OS/390 Version 3 Release 1 (program number 5648-A25) or later
- IBM Language Environment for OS/390 Version 2 Release 10 (program number 5647-A01) or later with PTF for APAR PQ65085 (Available Sept. 2002)
- IBM WebSphere Studio Enterprise Developer (WSED) for Multiplatforms Version 5 Early Availability (EA) (program number 5724-B67) or later
- OS/390 R8/R9/R10 and z/OS V1R1 support for Unicode ™ is required for the XML converters generated by WSED General Availability release[1].

[1] OS/390 R8/R9/R10 and z/OS V1R1 support for Unicode ™ can be obtained free of charge at https://www6.software.ibm.com/dl/os390/unicodespt-p . This support is integrated into z/OS Version 1 Release 2 and later.

## Appendix B. Modifying COBOL Generator preferences

You access COBOL generator default preferences by selecting Window->Preferences from the Workbench menu bar. The following dialog will appear.

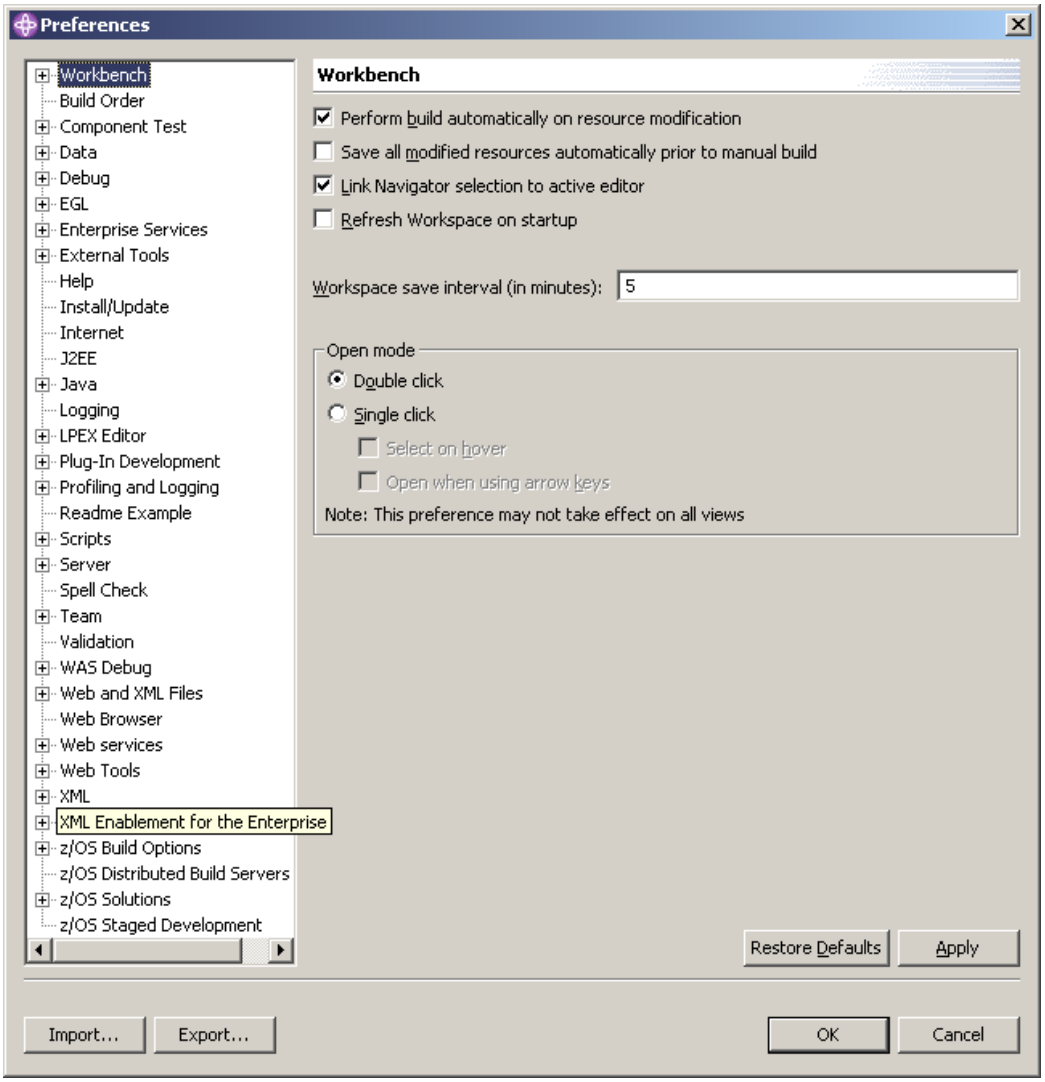


**Figure 11, Workbench preferences dialog**

You expand XML Enablement for the Enterprise section and then select the COBOL Generator page.

The following dialog will appear:

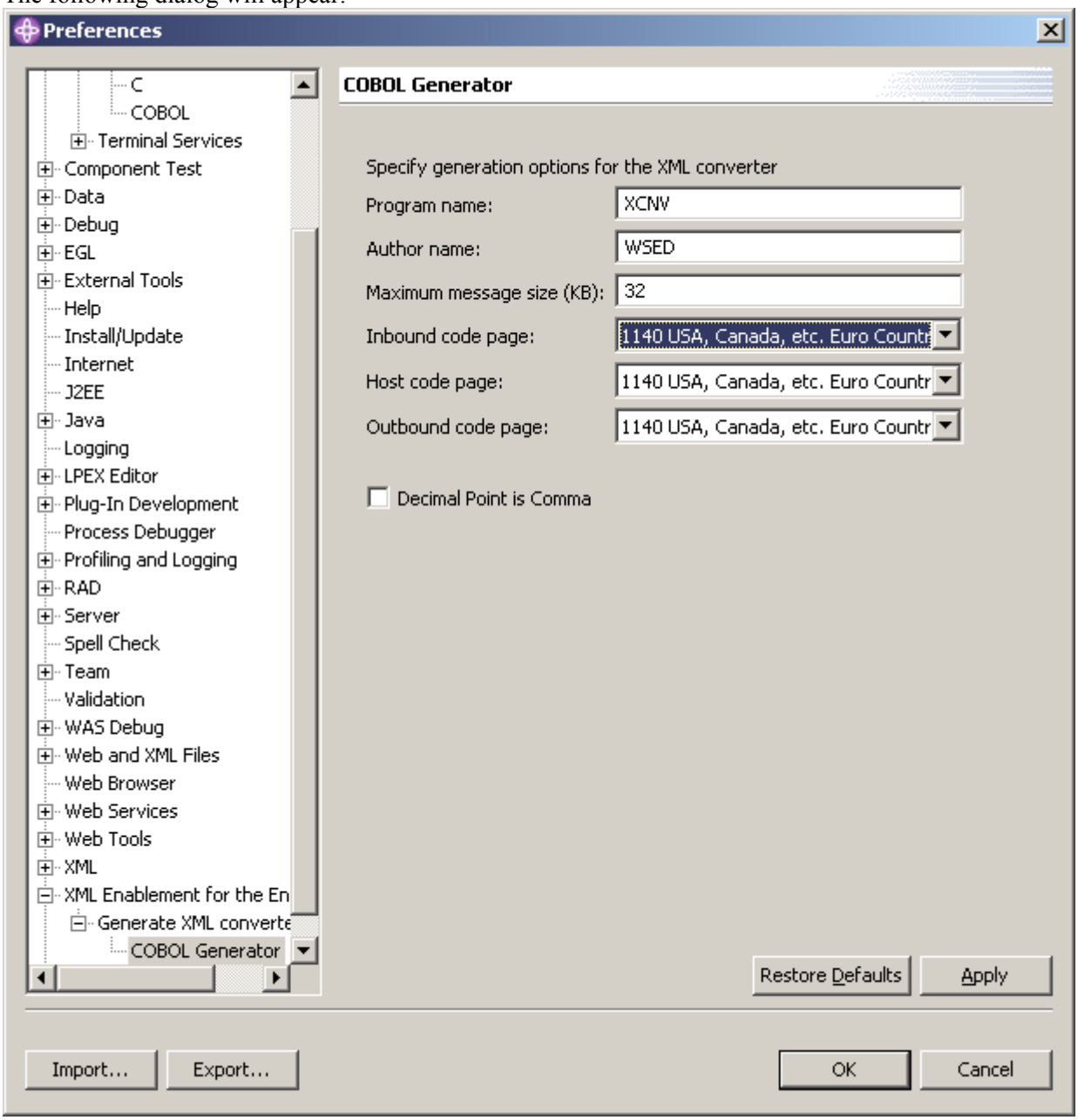


**Figure 12, COBOL Generator preferences**

Note that the entries on this page are very similar to the Options page of the COBOL converter generator wizard (See Step **(6)** on page 17). In fact, this preferences page is where the defaults for the wizard come from. The only difference is the "Decimal Point is Comma" selection which when specified, exchanges the functions of the period and the comma in PICTURE character strings and in numeric literals in a COBOL program.

## Appendix C. Modifying COBOL Importer preferences

You access COBOL importer default preferences by selecting Window -> Preferences from the Workbench menu bar. The following dialog will appear.

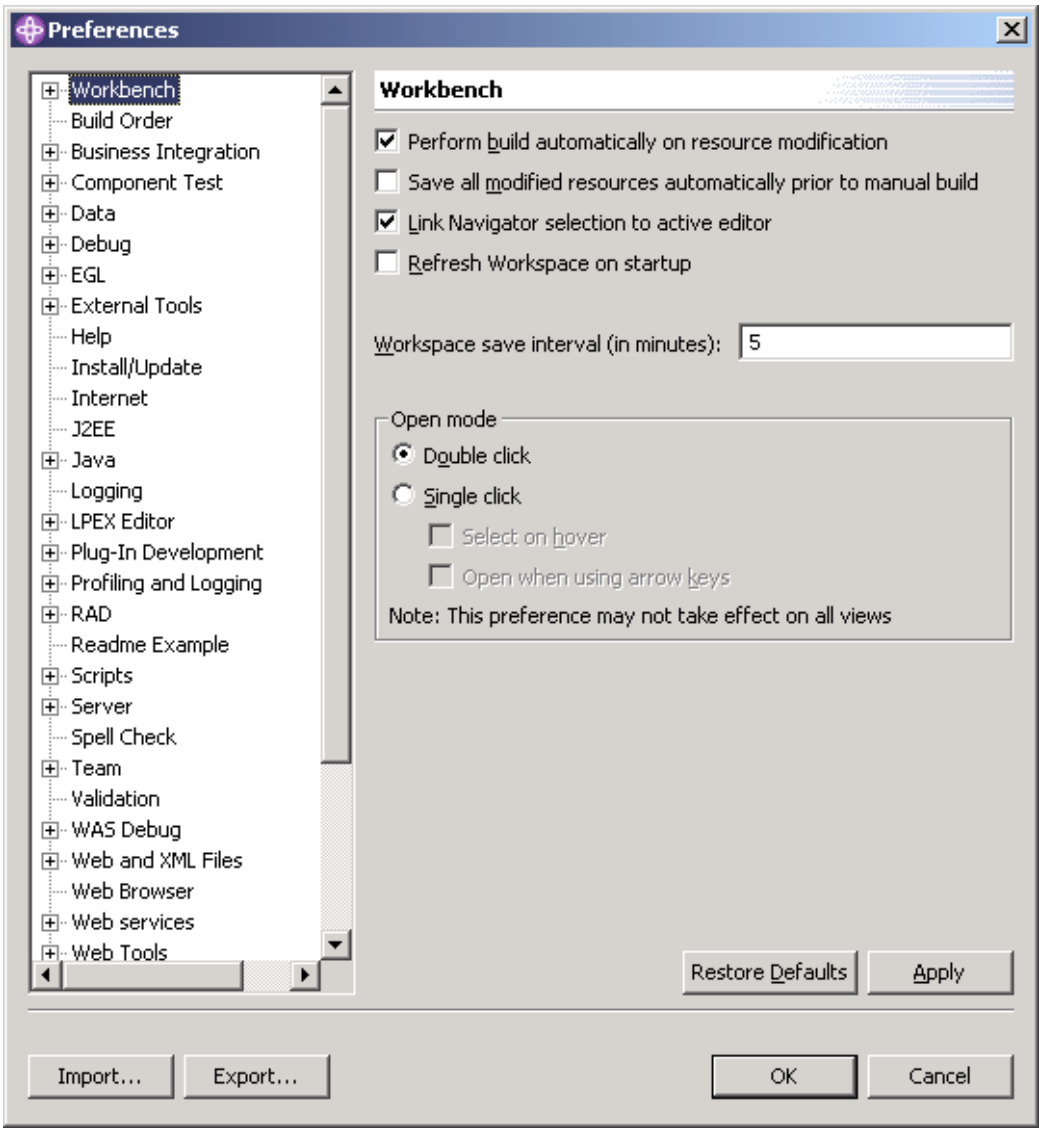


**Figure 13, Workbench preferences dialog**

You expand Business Integration section and then select Importers - COBOL page.

The following dialog will appear:

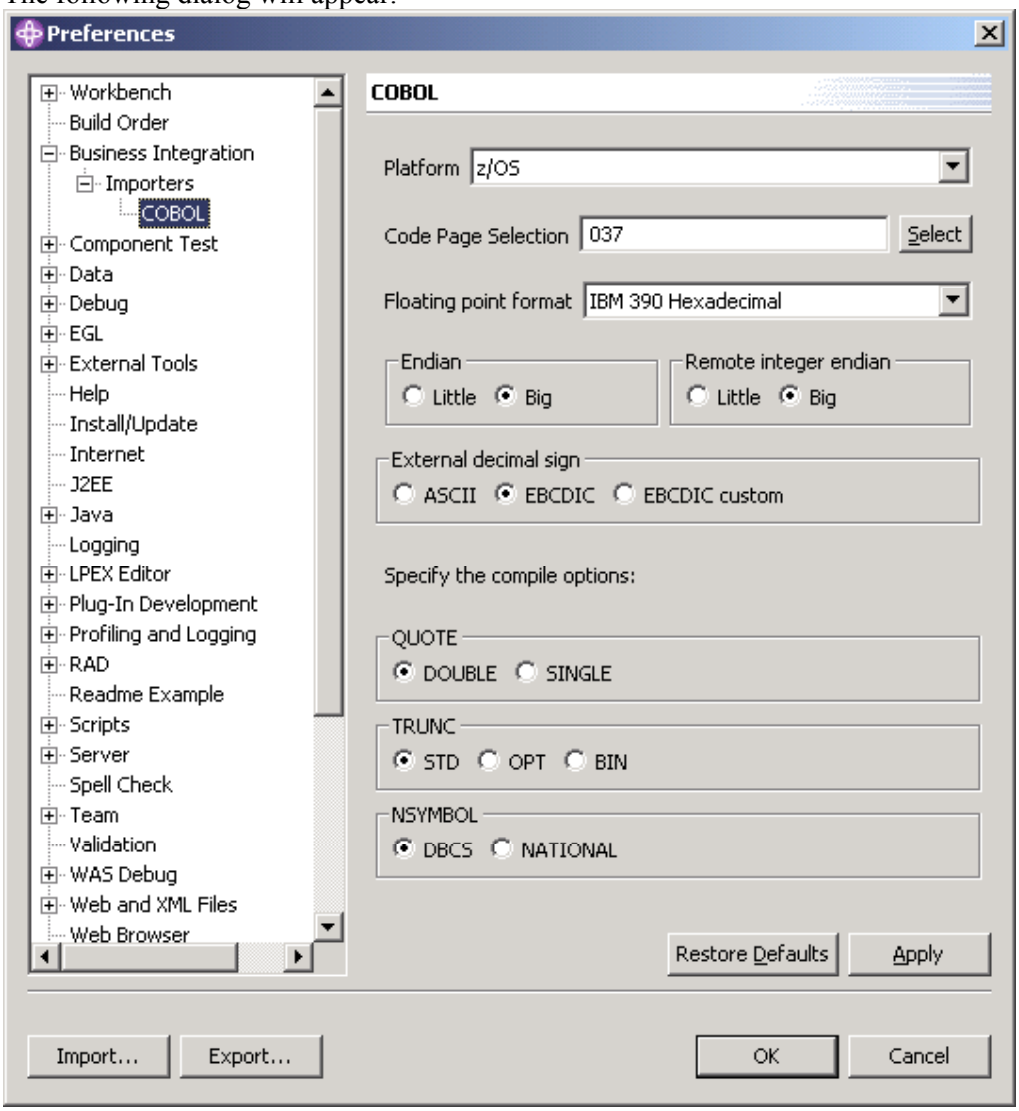


**Figure 14, COBOL Importer preferences**

This dialog allows you to change COBOL options that affect the way XML converters treat COBOL data items. Normally you should only modify the "Specify the compile options" section of this dialog, as the rest of the preferences do not apply to the XML converters.

## Appendix D. XML Converter Interface

Both the inbound and outbound XML converters are invoked via a call statement. Arguments to the converters are a mixture of input and output parameters whose contents may be changed upon return from invocation. The call signature of the converters is displayed below:

```
CALL 'CONV' USING
              DATA-STRUCTURE   (input)
              XML-MESSAGE-LEN  (input + output)
              XML-MESSAGE-TEXT (input + output)
             (FEEDBACK-CODE or OMITTED) (output)
             RETURING
              CONVERTER-RETURN-CODE (output)
```

The above COBOL code is an example of a call to a converter. Input and output properties for each argument are displayed in parenthesized italics. Since the structure of each argument is unique, it must not vary from the description here.

DATA-STRUCTURE is a piece of storage whose structure is identical to that of the data structure that was nominated as the inbound data structure when the converter was generated. During an inbound conversion, DATA-STRUCTURE will be populated with values from the input XML document provided in the arguments XML-MESSAGE-LEN and XML-MESSAGE-TEXT. In the case of an outbound conversion, DATA-STRUCTURE is used to populate an XML message, whose properties are placed in the XML-MESSAGE-LEN and XML-MESSAGE-TEXT arguments.

FEEDBACK-CODE is a 12 byte Language Environment Condition Token that can be omitted by using the "OMITTED" keyword on the call. Choosing to omit this argument will cause any error encountered by the converter to be signaled as a severe condition containing information about the error. On the other hand not omitting this argument will cause the converter to simply place a condition token representing the error into FEEDBACK-CODE without signaling a condition. The structure of FEEDBACK-CODE is displayed below.

```
       1 FEEDBACK-CODE.
         2 CONDITION-TOKEN-VALUE.
           COPY CEEIGZCT.
          3 CASE-1-CONDITION-ID.
           4 SEVERITY    PIC S9(4) BINARY.
           4 MSG-NO      PIC S9(4) BINARY.
          3 CASE-2-CONDITION-ID
                REDEFINES CASE-1-CONDITION-ID.
           4 CLASS-CODE  PIC S9(4) BINARY.
           4 CAUSE-CODE  PIC S9(4) BINARY.
          3 CASE-SEV-CTL PIC X.
          3 FACILITY-ID  PIC XXX.
         2 I-S-INFO      PIC S9(9) BINARY.
```

More detailed information about the structure and use of this condition token can be found in the Language Environment Programming Guide.

CONVERTER-RETURN-CODE is an output only argument, which will contain one of two classes of return codes upon completion of the call. If the converter encounters an error within its own facilities, that is, not an error from the XML PARSE statement, then the Language Environment message ID associated with the error will be placed in the argument. The second class of return codes is the codes returned from the XML PARSE statement. These will occur in the case where something was syntactically incorrect in the input XML document. Note that this second class of errors only occurs during an inbound conversion.