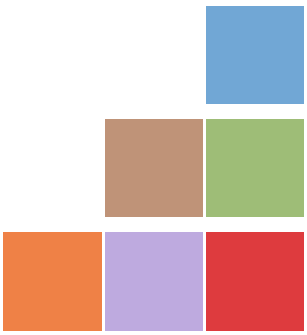
COBOL-IT® Developer Studio
Getting Started
COBOL-IT Web Services
Version 2.0



Contents

Getting Started with COBOL-IT Web Services

Acknowledgment

Contact Information:

The Lawn
22-30 Old Bath Road
Newbury, Berkshire, RG14 1QN
United Kingdom
Tel: +44-0-1635-565-200

Dependencies

Dependencies and Comments

Dependency	Comment
"C" compiler	The COBOL-IT Compiler requires a "C" compiler. While most Linux>Unix installations will include a "C" compiler, many Windows installations will not. Windows users can download the Visual Studio from www.microsoft.com.
COBOL-IT Compiler Suite	The COBOL-IT Compiler Suite, Standard Edition can be downloaded at the COBOL-IT Online Portal. For access to the COBOL-IT Online Portal, please contact your sales representative at sales@cobol-it.com.
Java Runtime Environment (JRE)	The COBOL-IT Developer Studio Kepler build can be run with the Java Runtime Environment (JRE) Version 1.6 or greater. The COBOL-IT Developer Studio Neon build can be run with the JRE Version 1.8 or greater.
Eclipse	Eclipse is included with the download of Developer Studio.

The COBOL-IT Developer Studio requires that the COBOL-IT Compiler Suite already be installed on the host platform, and that a "C" compiler be installed on the host platform.

The COBOL-IT Developer Studio is an Eclipse plug-in, and as such, requires that Eclipse be installed on the host platform. Eclipse, in turn, requires that a Java Runtime Environment (JRE) be installed on the host platform.

The COBOL-IT Developer Studio Distribution

For Windows-based installations, the COBOL-IT Developer Studio, Enterprise Edition can be downloaded from the COBOL-IT online portal with a login and password provided by your sales representative.

The COBOL-IT Developer Studio, Enterprise Edition is available with Subscription. The COBOL-IT Developer Studio, Enterprise Edition provides functionality with the installation of several Perspectives:

- Developer Studio Perspective in which users set up and build COBOL projects, using a locally installed version of the COBOL-IT Compiler Suite Enterprise Edition. The Developer Studio Perspective additionally provides access to Code Coverage and Profiling Tools.
- Debugger Perspective providing access to a feature-rich COBOL debugger both locally, and on Remote Systems
- Remote Systems Perspective, allowing use of Compiler, Runtime, and Debugger functionalities installed on remote servers.

- Git and RSEGit Perspectives, providing users with full access to the Git/Github Source Code Control System.
- Data Displayer Perspective, providing access to a tool for browsing and modifying data in indexed, sequential and relative files.
- Web Services Perspective, providing the ability to generate intermediate artifacts, bridge program, and input strings which, when URL-encoded can be used to access entry points in a COBOL-IT Web Service program.
- Planning Perspective, providing access to the Mylyn Task Manager.
- For more information about the usage of Git/RSEGit, Data Displayer, Mylyn Task Manager, and Code Coverage, see the Getting Started with the Developer Studio- The Utilities Manual.
- Using the COBOL-IT Developer Studio requires a license for both the COBOL-IT Compiler Suite Enterprise Edition, and COBOL-IT Developer Suite.

Overview

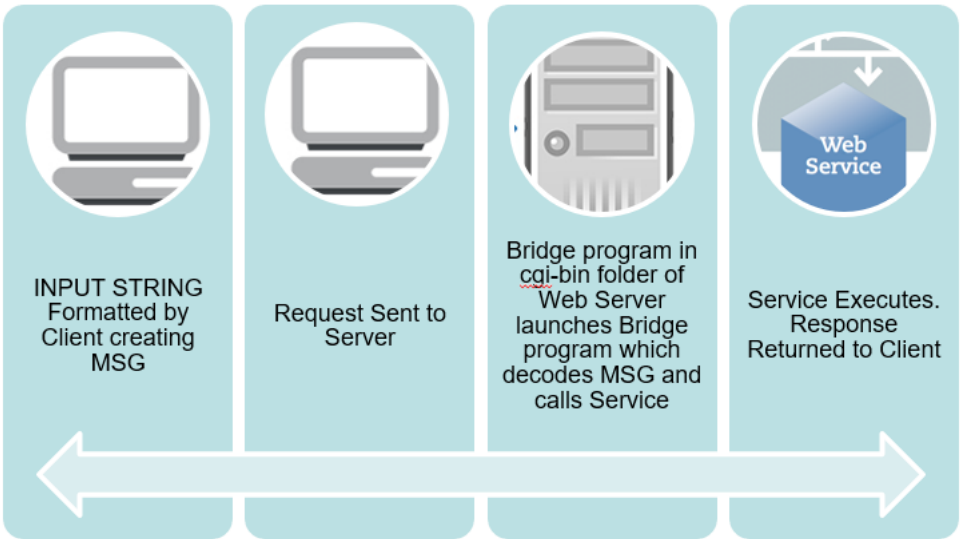
The COBOL-IT Web Services Perspective in the Developer Studio provides a platform that allows the user to convert a COBOL-IT Web Service program into a callable Web Service. The RESTful Web Service functions, POST, GET, PUT, and DELETE, map well to common COBOL operations. A single COBOL program can have entry points mapped to each of these functions. In our samples, we will be using indexed files as the target of the COBOL operations.

RESTful Web Service function	COBOL operation
POST	OPEN [OUTPUT/I-O FILE], WRITE
GET	OPEN [INPUT FILE], READ
PUT	OPEN [I-O FILE], REWRITE
DELETE	OPEN [I-O FILE], DELETE RECORD

The COBOL-IT Web Services Perspective provides two outputs which allow the COBOL-IT Web Service program to be converted into a callable Web Service. These are the INPUT STRING, and the BRIDGE PROGRAM. The COBOL-IT Web Services Perspective supports three different configurations, the Wildfly Application Server Configuration, the Apache Web Server Configuration and the xbind configuration.

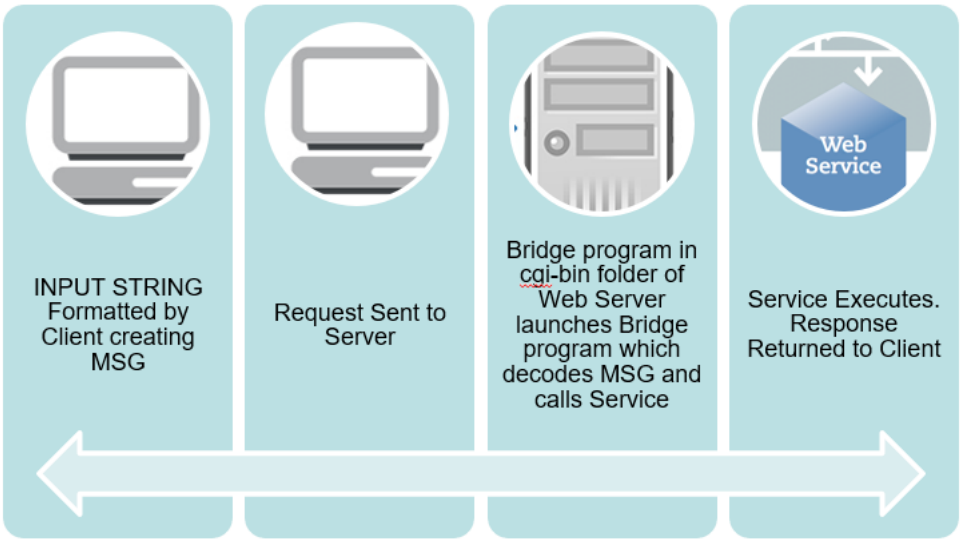
The Apache Web Server Configuration

In the Apache Web Server Configuration, the Bridge Program is executed from within a script located in the cgi-bin folder of the Web Server. In our example, the shell is identified as: form action=http://localhost/cgi-bin/xholidays41.sh in the html file executing on the client. The shell script is required because COBOL-IT must be able to locate the license file, and the COBOL-IT environment must be setup.



The WildFly (JBoss) Configuration

In the WildFly Application Server Configuration, the Bridge Program is executed from within a script located in the … folder of the Application Server. In our example, the shell is identified as: form action=http://… in the html file executing on the client. The shell script is required because COBOL-IT must be able to locate the license file, and the COBOL-IT environment must be setup.



The xbind Configuration

In the xbind Configuration, the Bridge Program is executed directly by the xbind listener. In our example, the listener is launched from a shell script.

```
./xbind 9735 xholidays &
```

and identified as:
form action="`http://localhost:9735`" in the html file executing on the client. The shell script is required because COBOL-IT must be able to locate the license file, and the COBOL-IT environment must be setup.

The BRIDGE PROGRAM

The COBOL-IT Web Services Perspective uses templates to generate the bridge program, which can be customized. The default templates that are provided correspond to the two different configurations supported for COBOL-IT Web Services. Note that the MESSAGE is the INPUT-STRING, which has been URL-encoded.

Template	Application
sysin_bridge_template.xml	Apache Web Server configuration ACCEPTS MESSAGE from SYSIN
xbind_bridge_template.xml	xbind configuration MESSAGE passed through Linkage

The generated bridge program is designed to decode the URL-encoded string, and call the appropriate entry-point. In our sample COBOL-IT Web Service Program, there are four entry-points, called ‘postholiday’, ‘getholiday’, ‘putholiday’, and ‘deleteholiday’. Each of the entry-point requires that an INPUT STRING be generated for it. Then, in our samples, we will see how that INPUT STRING is, first encoded, by the Client, then decoded by the Bridge program, which will then CALL the appropriate Entry point in the COBOL-IT Web Service Program.

The INPUT STRING

The INPUT STRING is generated in XML format, and includes such information as the name of the COBOL-IT Web Service Program, the name of the entry-point in the COBOL-IT Web Service Program that is to be CALL'ed, and the parameter names in the USING clause for the CALL'ed entry point, with values, where applicable. When populating a record for purposes of a WRITE, all fields might contain values. In an indexed file, when retrieving a record, for purposes of a READ, a REWRITE, or a DELETE, only the KEY field is required.

A sample INPUT STRING for a POST in holidays.cbl, with `ENTRY 'postholiday' USING hol-linkage.`

```
<PROGRAM name="holidays"><ENTRY name="postholiday"><PARAM name="hol-linkage"><hol-
linkage><hol-rec><hol-id>0</hol-id><hol-name>Christmas</hol-name><hol-dt><hol-
wkday>Wednesday</hol-wkday><hol-mon>December</hol-mon><hol-day>25</hol-day><hol-
yr>2019</hol-yr></hol-dt><hol-cur-dt>char*</hol-cur-dt></hol-rec><hol-io-
msg>char*</hol-io-msg></hol-linkage></PARAM></ENTRY></PROGRAM>
```

The COBOL-IT Web Service Program Entry Points

The key elements, for purposes of generating the INPUT STRING and the Bridge Program are the Entry-points. A quick examination of the way our sample COBOL-IT Web Service Program is constructed shows. There is a bit more to it, but basically, the program CALLs the entry point using the linkage section, which consists of the elements of the holiday record, and at the end, a io-message, which conveys the file status result of the key operation. Each of these entry-points must have its own INPUT STRING generated for it, as the INPUT STRING contains the name of the entry point to be called.

```
       PROCEDURE DIVISION.
      *
       ENTRY 'postholiday' USING hol-linkage.
           OPEN OUTPUT holidaysIX.
           WRITE holiday-record.
           MOVE holiday-io-msg TO hol-io-msg.
           CLOSE holidaysIX.
           GOBACK.
      *
       ENTRY 'getholiday' USING hol-linkage.
           OPEN INPUT holidaysIX.
           READ holidaysIX KEY IS holiday-name.
           MOVE holiday-io-msg TO hol-io-msg
           CLOSE holidaysIX.
           GOBACK.
      *
       ENTRY 'putholiday' USING hol-linkage.
           OPEN I-O holidaysIX.
           READ holidaysIX KEY IS holiday-name.
           REWRITE holiday-record.
           MOVE holiday-io-msg TO hol-io-msg.
           CLOSE holidaysIX.
           GOBACK.
      *
       ENTRY 'deleteholiday' USING hol-linkage.
           OPEN I-O holidaysIX.
           READ holidaysIX KEY IS holiday-name.
           DELETE holidaysIX RECORD.
           MOVE holiday-io-msg TO hol-io-msg.
           CLOSE holidaysIX.
           GOBACK.
      *
       UPDATE-HOL-IO-MSG.
           INITIALIZE hol-io-msg.
           STRING "HOLIDAY-STATUS:" DELIMITED BY SIZE,
                  holiday-status DELIMITED BY SIZE,
                  INTO holiday-io-msg.
      *
```

The COBOL-IT Web Services Perspective

How it Works

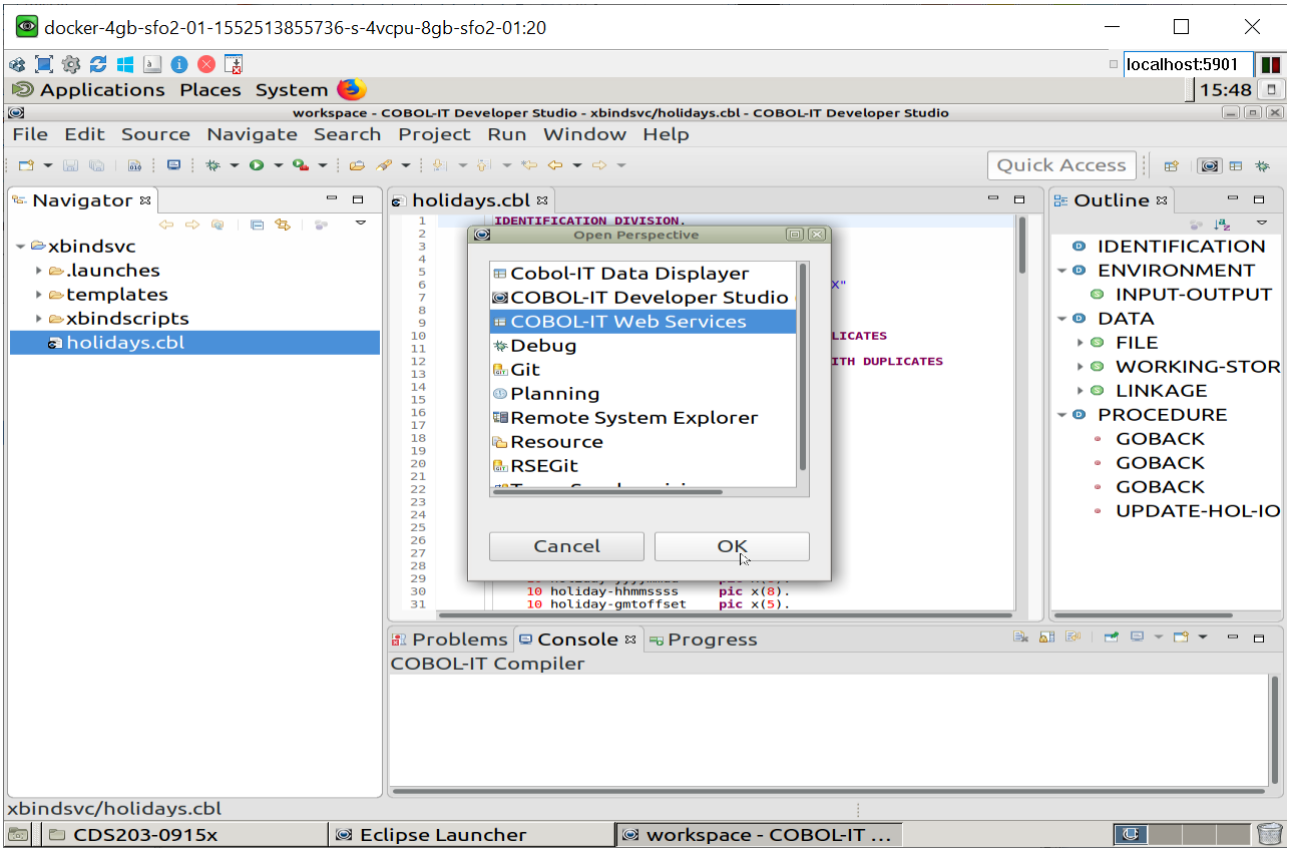
In the Web Services Perspective, the user selects a Source File, which contains separate ENTRY points- which can contain separate Linkage Sections- for performing functions associated with the POST, GET, PUT, and DELETE functions.

To prepare the source file as a Web Service in the COBOL-IT Web Services Perspective, the user performs the following:

- Select a COBOL-IT Setup Script.
- Select a Source file from the Developer Studio Project.
- Generate an XSD file from the Source file. This process parses the program, and records the names of the entry points, and for each entry point, the linkage section items. This information is stored in an XML format. Note that this parsing process can be done at the command line, using the command:
  - cobc -linkage-desc=[programname].xsd [programname].cbl
- Select a templates folder. This folder will store default templates that are used, as well as customized templates created by the user.
- After the XSD file is generated, it is parsed, and COBOL-IT mapping can be done. On the mapping interface, the user associates an entry point that has been recorded in the XSD file with one of the functions, POST, GET, PUT, DELETE.
- The user saves the mappings, and then generates a bridge program.
- Depending on the environment, the user runs a script in which the bridge program and web service program are compiled together. When using the xbind solution, the programs are compiled with the -b compiler option to create a single shared object. When using the Apache Web Server solution, the programs are compiled to create an executable and the executable is copied into the cgi-bin directory of the Web Server.
- In our samples, we use Firefox to run HTML files, with different charactericis for the xbind and Apache web server solutions. In both cases, you can run POST, GET, PUT, or DELETE functions, and view the data passing from the Client to the Server, and back to the Client, with information provided during the transaction.

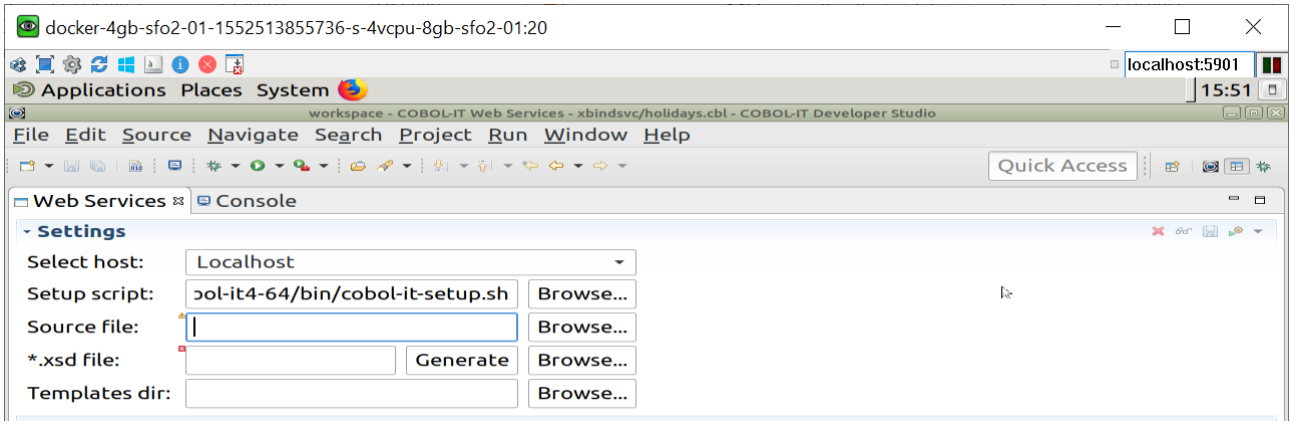
Open the Web Services Perspective

From the Open Perspective dialog screen, select COBOL-IT Web Services. Click OK to open the COBOL-IT Web Services Perspective.

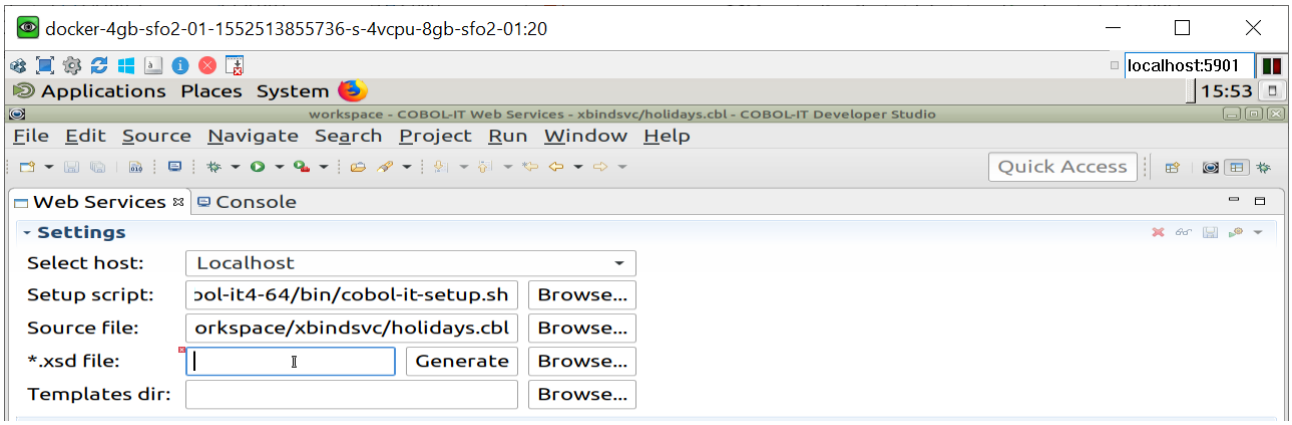


Setup the Environment

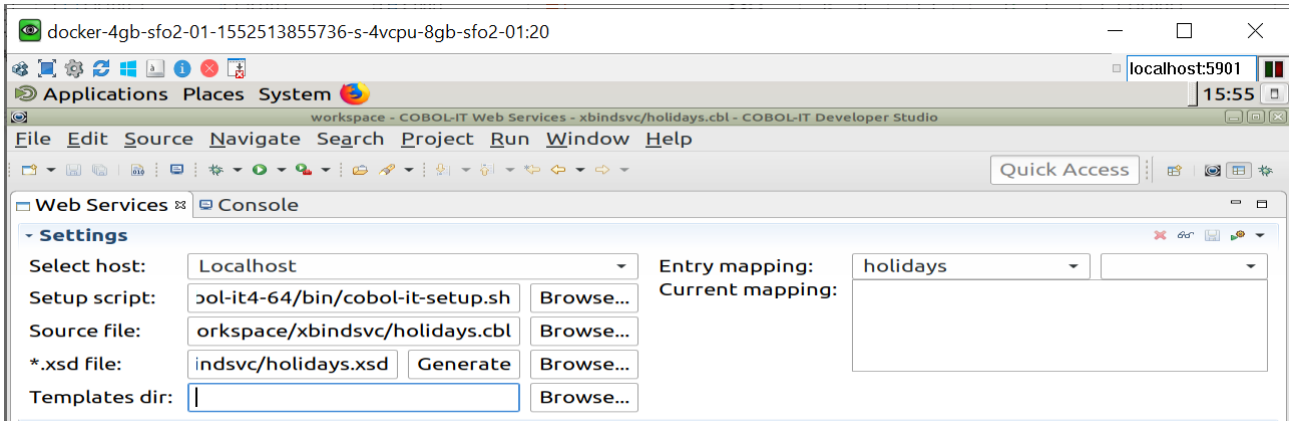
Setup script: Select the cobol-it-setup shell script in the bin directory of your COBOL-IT installation.



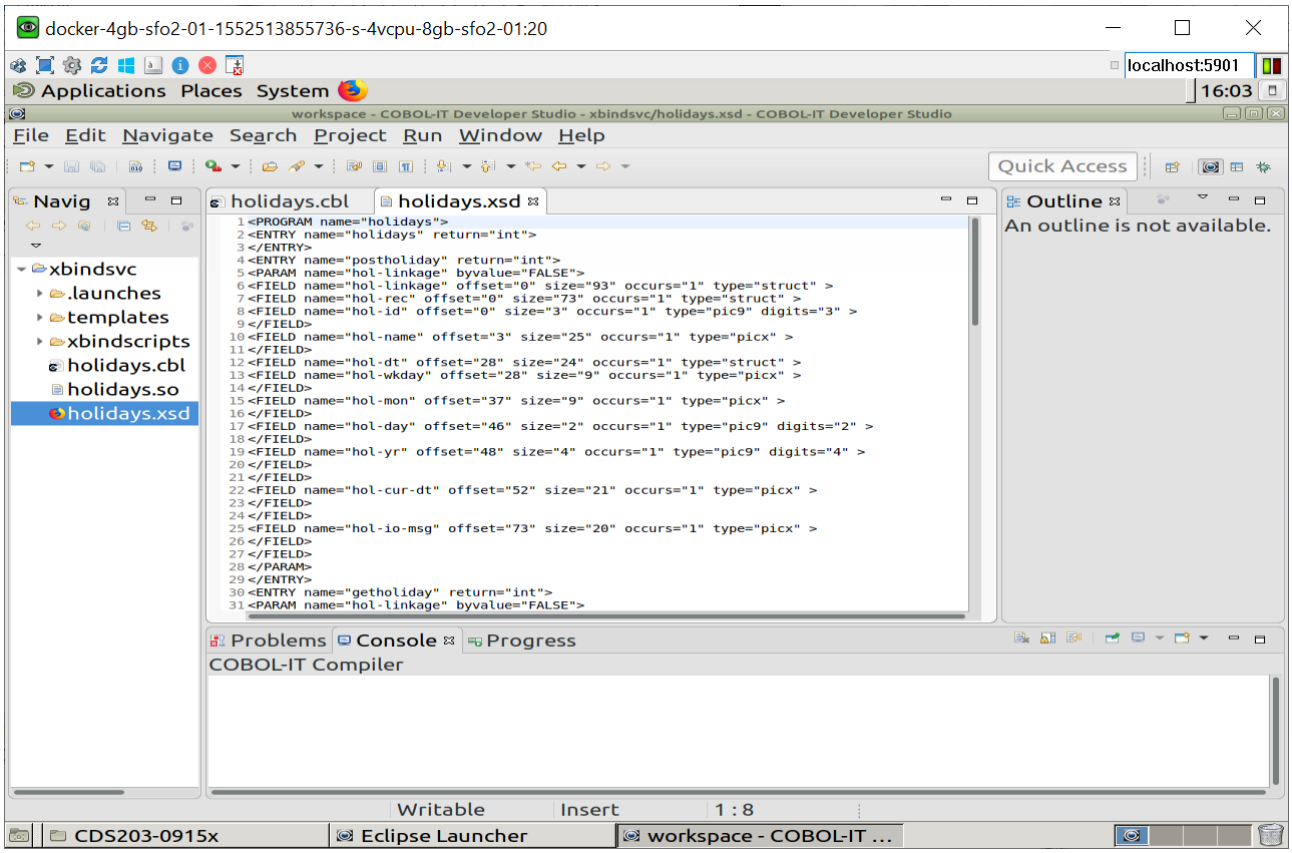
Source file: Select a source file from your project directory. In our example, we are using holidays.cbl, for which the source is included in Appendix A.



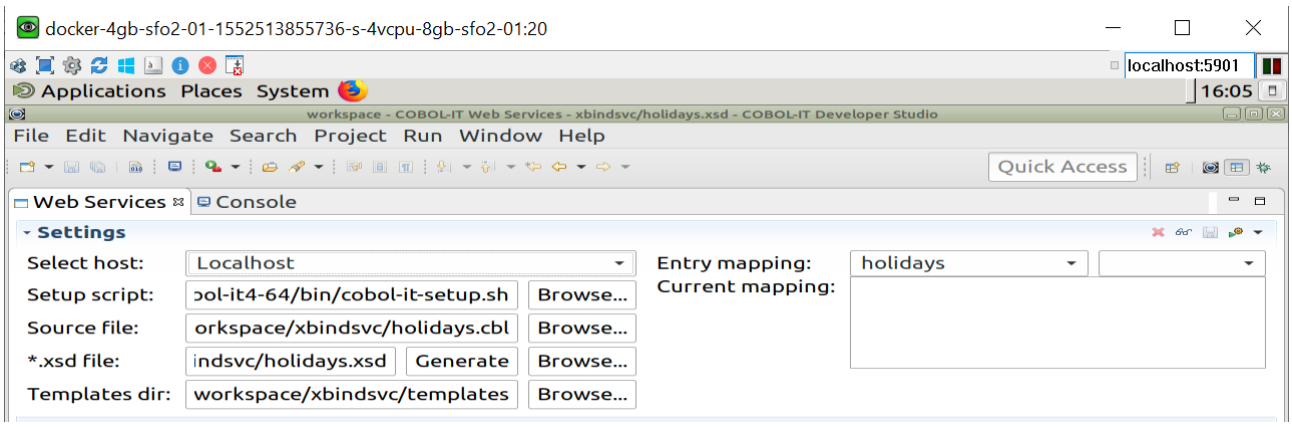
*.xsd file: Click on the Generate button or use the Browse button to locate and select the .xsd file. The xsd file will be generated and parsed in your project folder. During the parsing, entry-points will be detected and fields in the Entry mapping: interface will be populated.



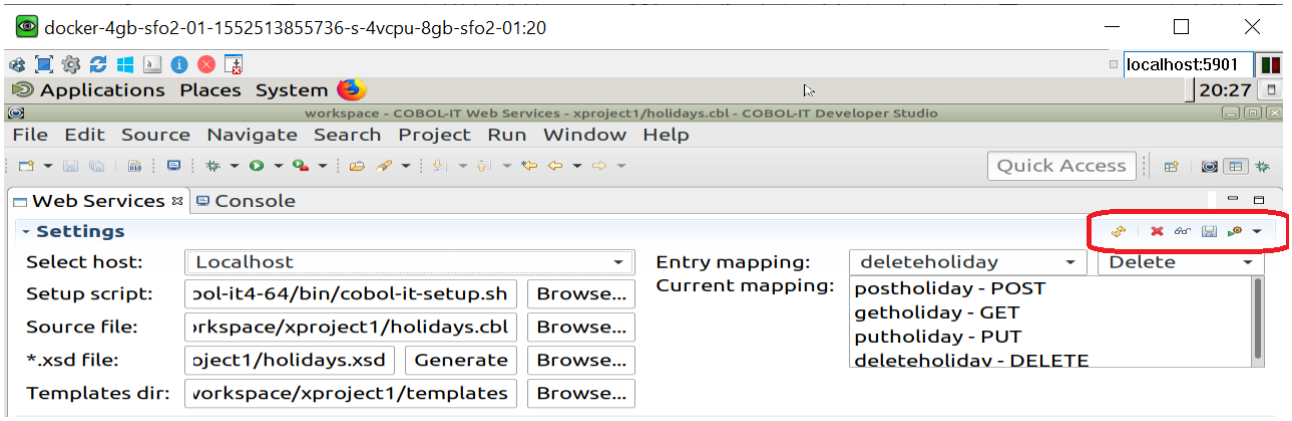
The xsd file:



Templates dir: In our project, we have added a folder called “templates”. In the Settings interface, use the Browse button to locate the template folder, and select it. Templates are used to generate the bridge program. When you generate a bridge program, you will have the option of selecting one of the default templates, and your selection will be stored in the templates folder. You may also store customized templates in your template folder, and will be able to select them for use in generating your bridge program as well.



The COBOL-IT Web Services Toolbar



Reset Settings

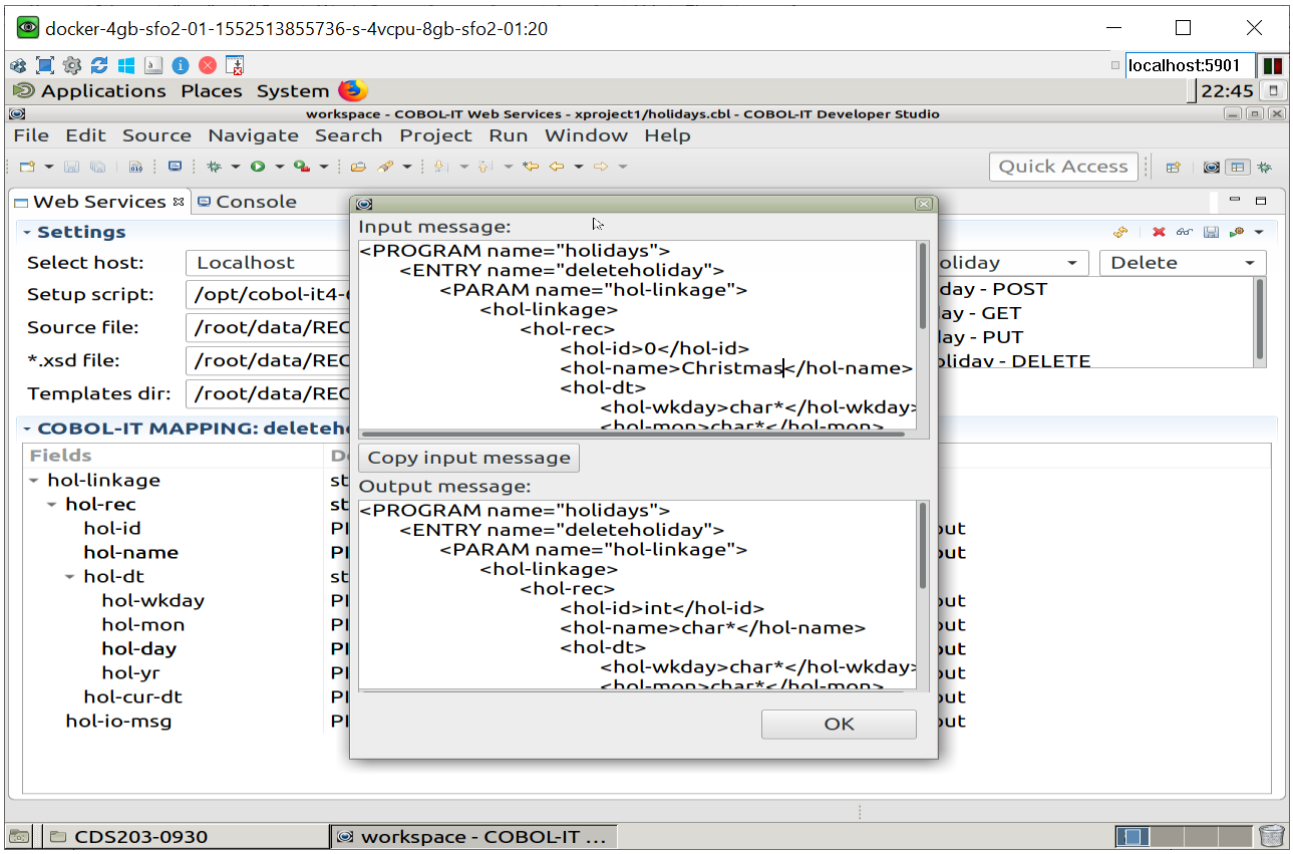
The Reset Settings function has the effect of clearing all of the Settings and Entry-mapping selections. If you have changed the linkage section in one of the entry points in your source file, but have not regenerated an XSD file, or if you have made modifications to your Entry-mapping, but have not generated a new bridge program, they you can get in a situation where some of the elements of your solution are out of sync. It is important to be aware of the dependencies between the different elements in the Web Services Perspective. There are times when best practice is to clear the Settings, and re-enter the different fields.

Clear Mapping

When you have entered all of your Settings, and made your mapping choices, and SAVEd your mapping choices, you will see that the WSV file has been created in your project directory. The WSV file contains all of the mapping choices you have made, laid upon the information in the XSD file. This information is used in both the generation of the bridge program, and the generation of the input strings. The Clear Mapping function keeps the Settings intact, but removes all of the current mappings that you have made. Reset your mappings, and SAVE them, to generate a new WSV file.

Show Structure I-O messages

The Show Structure I-O messages interface opens the interface for generating Input Strings. In the Input Message area, you should validate the program name of the COBOL-IT Web Service, and the Entry Point for which you are generating an input string. The interface allows you to overwrite the “int” castings with integers, and the “char*” castings with character strings. When you have completed this exercise, click on the “Copy input message” button to copy the data to the Clipboard. Then, Click “OK” to close the dialog window. In our samples, we create empty textfiles to hold our input strings, and store them in the Project.



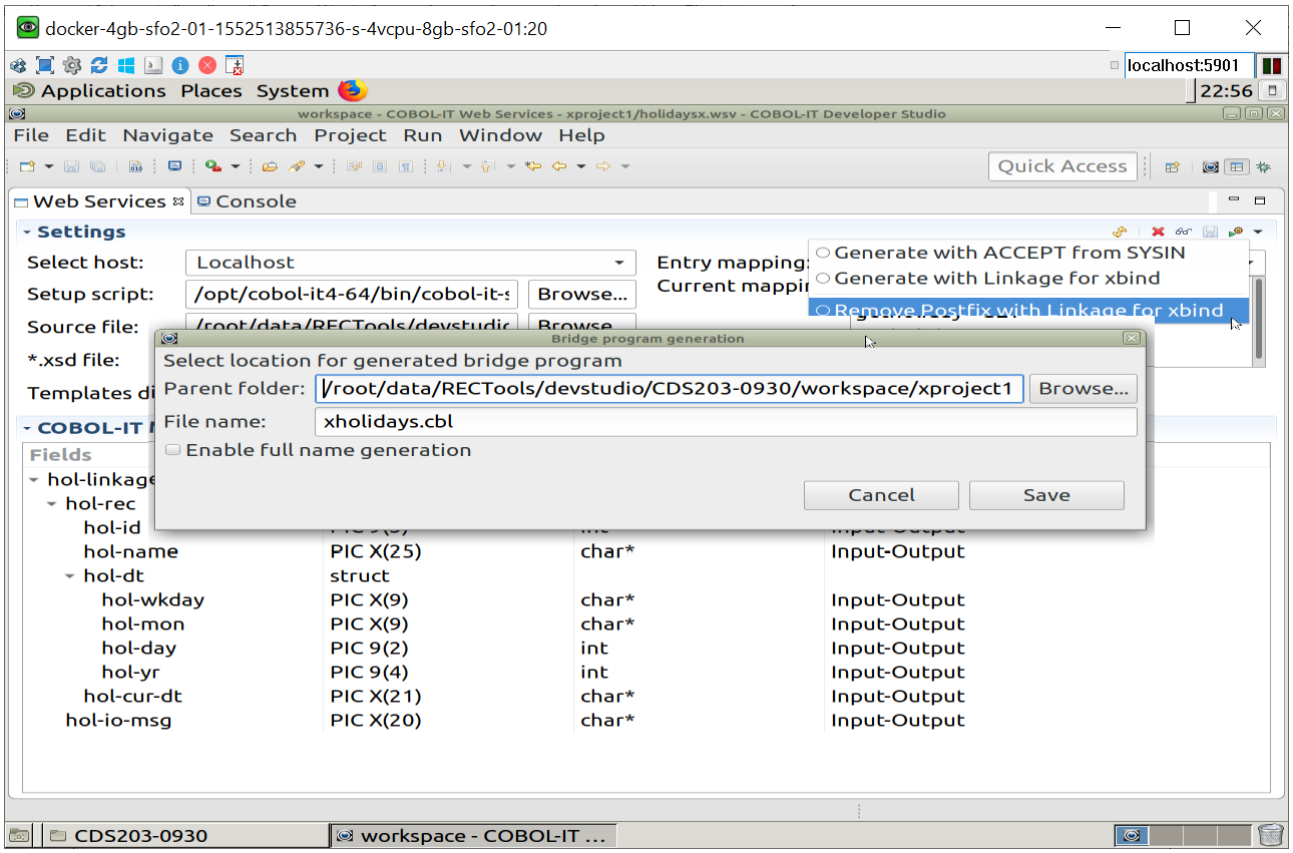
Save

The Save button has the effect of SAVEing all of your Mappings in a WSV file that is stored in your project.

Generate bridge program

Select the Generate bridge program button to open a drop-down box, displaying default templates, and customized templates that are stored in the templates folder.

After you have made your selection, you will see the default name and location of the generated program, which you may change or accept. The default name and default program-id of the generated program is the name of the COBOL-IT Web Service program, pre-pended with an “x”.

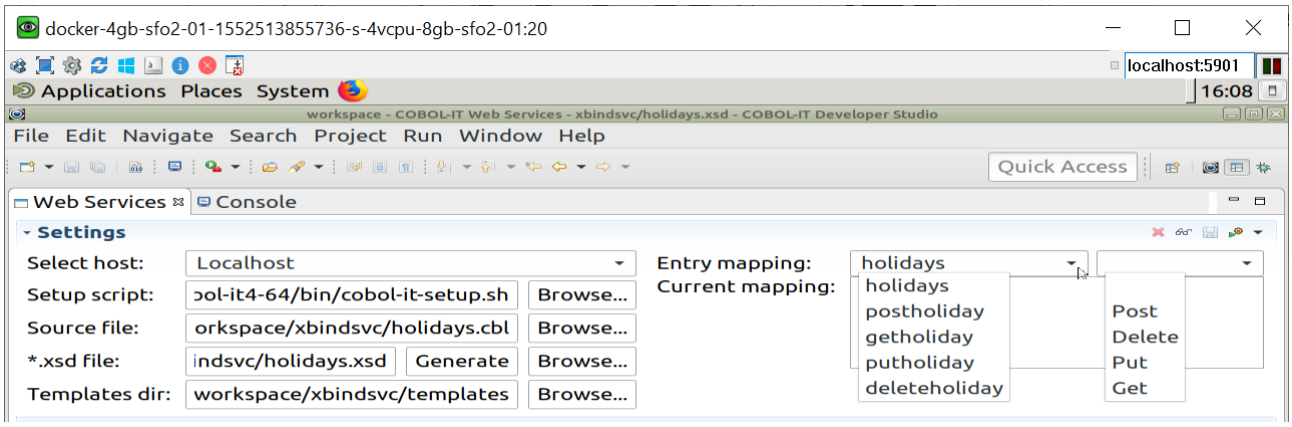


Entry Mapping

The Entry Mapping interface consists of two dropdown boxes. The left-most dropdown box lists the names of the entry points in the COBOL-IT Web Service program. The right-most dropdown box lists the names of the functions, POST, GET, PUT, DELETE. It is necessary to match the name of an entry point to the name of a function, in order to generate the input string that can provide access to that function.

In our sample program, there are 5 entry points. They are:

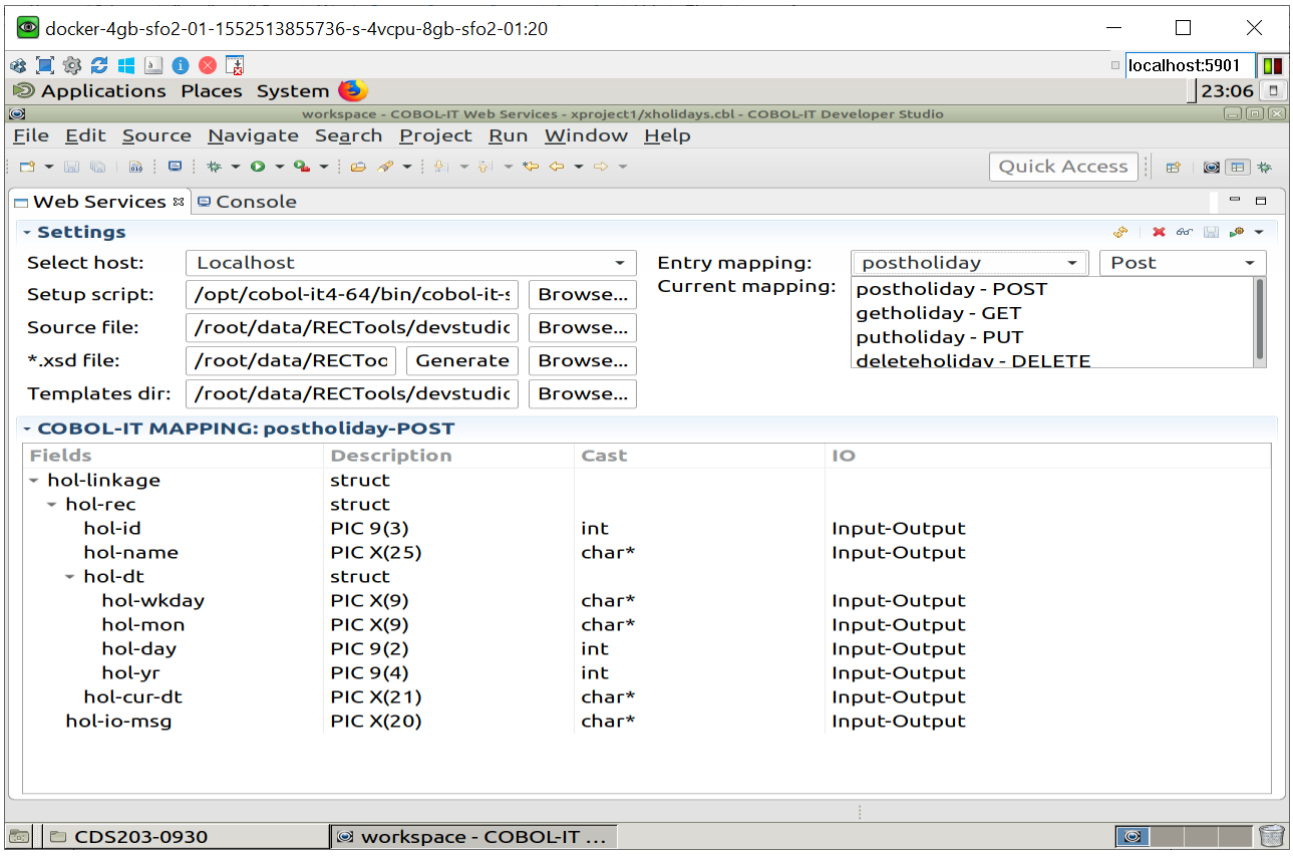
holidays, representing the PROCEDURE DIVISION entry point.
postholiday, representing ENTRY 'postholiday' USING hol-linkage.
getholiday, representing ENTRY 'getholiday' USING hol-linkage.
putholiday, representing ENTRY 'putholiday' USING hol-linkage.
deleteholiday, representing ENTRY 'deleteholiday' USING hol-linkage.



In our sample program, there are 4 ENTRY statements, which contain code that is designed to perform the associated service. That is:

postholiday	Performs an OPEN OUTPUT, and a WRITE
getholiday	Performs an OPEN INPUT, and a READ
putholiday	Performs an OPEN I-O, a READ and a REWRITE
deleteholiday	Performs an OPEN I-O, a READ and a DELETE RECORD

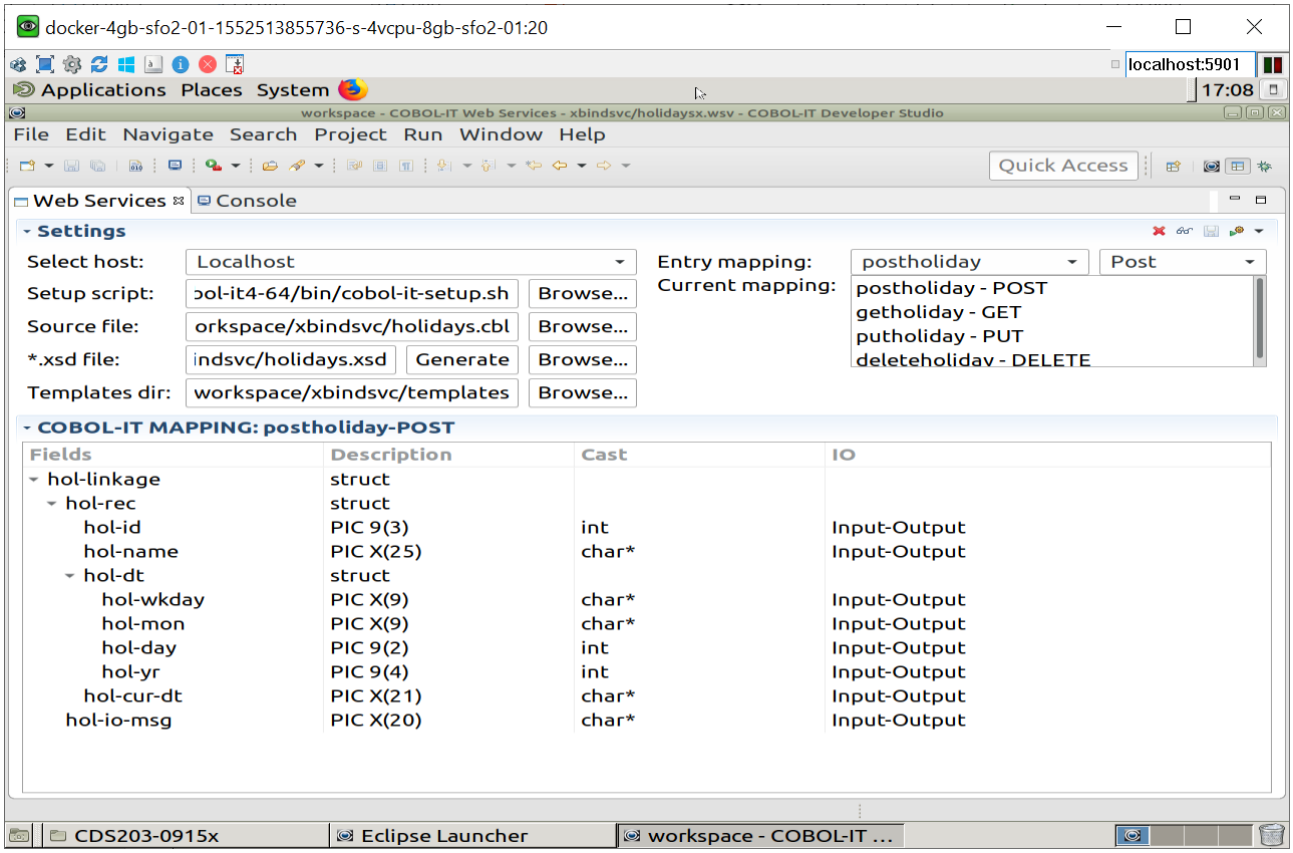
Use the left-most Entry mapping dropdown box to select the name of the entry point in the program. Use the right-most Entry mapping dropdown box to select the function with which that ENTRY statement should be associated.



Update, Save the Mapping Interface for each function

As you associate an ENTRY point with a function, the COBOL-IT mapping for that ENTRY statement is displayed. The Cast and IO functions are pre-filled with defaults, which can be changed.

The Save button is on the Settings Toolbar. When you Save your Entry mappings, a file with extension .wsv is generated into your Project folder. This file is used in the generation of the bridge program. As a result, you must first Save your settings, then generate your bridge program.



The .wsv file

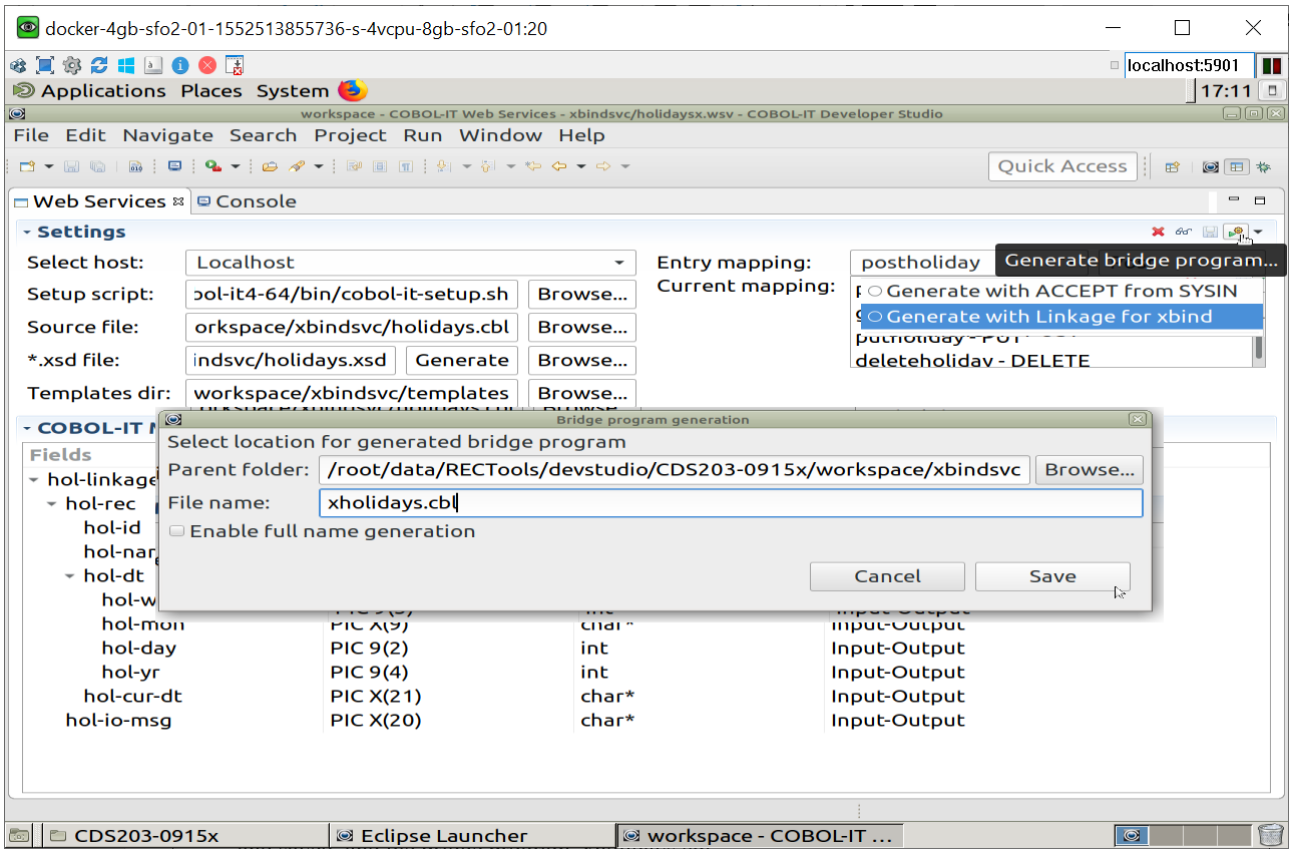
The .wsv file contains all of the mappings you have saved.

```
<MODEL sourceFile="/root/data/RECTools/devstudio/CDS203-0930/workspace/xproject1/holidays.cbl"
xsdModel="/root/data/RECTools/devstudio/CDS203-0930/workspace/xproject1/holidays.xsd">
	<ENTRY name="postholiday" function="Post">
		<FIELD name="hol-linkage">
			<FIELD name="hol-rec">
				<FIELD name="hol-id" cast="int" io="Input-Output"/>
				<FIELD name="hol-name" cast="char*" io="Input-Output"/>
				<FIELD name="hol-dt">
					<FIELD name="hol-wkday" cast="char*" io="Input-Output"/>
					<FIELD name="hol-mon" cast="char*" io="Input-Output"/>
					<FIELD name="hol-day" cast="int" io="Input-Output"/>
					<FIELD name="hol-yr" cast="int" io="Input-Output"/>
				</FIELD>
				<FIELD name="hol-cur-dt" cast="char*" io="Input-Output"/>
			</FIELD>
			<FIELD name="hol-io-msg" cast="char*" io="Input-Output"/>
		</FIELD>
	</ENTRY>
	<ENTRY name="getholiday" function="Get">
		<FIELD name="hol-linkage">
			<FIELD name="hol-rec">
				<FIELD name="hol-id" cast="int" io="Input-Output"/>
				<FIELD name="hol-name" cast="char*" io="Input-Output"/>
				<FIELD name="hol-dt">
					<FIELD name="hol-wkday" cast="char*" io="Input-Output"/>
					<FIELD name="hol-mon" cast="char*" io="Input-Output"/>
					<FIELD name="hol-day" cast="int" io="Input-Output"/>
					<FIELD name="hol-yr" cast="int" io="Input-Output"/>
				</FIELD>
				<FIELD name="hol-cur-dt" cast="char*" io="Input-Output"/>
			</FIELD>
			<FIELD name="hol-io-msg" cast="char*" io="Input-Output"/>
		</FIELD>
	</ENTRY>
	<ENTRY name="putholiday" function="Put">
		<FIELD name="hol-linkage">
			<FIELD name="hol-rec">
				<FIELD name="hol-id" cast="int" io="Input-Output"/>
				<FIELD name="hol-name" cast="char*" io="Input-Output"/>
				<FIELD name="hol-dt">
					<FIELD name="hol-wkday" cast="char*" io="Input-Output"/>
					<FIELD name="hol-mon" cast="char*" io="Input-Output"/>
					<FIELD name="hol-day" cast="int" io="Input-Output"/>
					<FIELD name="hol-yr" cast="int" io="Input-Output"/>
				</FIELD>
				<FIELD name="hol-cur-dt" cast="char*" io="Input-Output"/>
			</FIELD>
			<FIELD name="hol-io-msg" cast="char*" io="Input-Output"/>
		</FIELD>
	</ENTRY>
	<ENTRY name="deleteholiday" function="Delete">
		<FIELD name="hol-linkage">
			<FIELD name="hol-rec">
				<FIELD name="hol-id" cast="int" io="Input-Output"/>
				<FIELD name="hol-name" cast="char*" io="Input-Output"/>
				<FIELD name="hol-dt">
					<FIELD name="hol-wkday" cast="char*" io="Input-Output"/>
					<FIELD name="hol-mon" cast="char*" io="Input-Output"/>
					<FIELD name="hol-day" cast="int" io="Input-Output"/>
					<FIELD name="hol-yr" cast="int" io="Input-Output"/>
				</FIELD>
				<FIELD name="hol-cur-dt" cast="char*" io="Input-Output"/>
			</FIELD>
			<FIELD name="hol-io-msg" cast="char*" io="Input-Output"/>
		</FIELD>
	</ENTRY>
</MODEL>
```

Generate Bridge Program

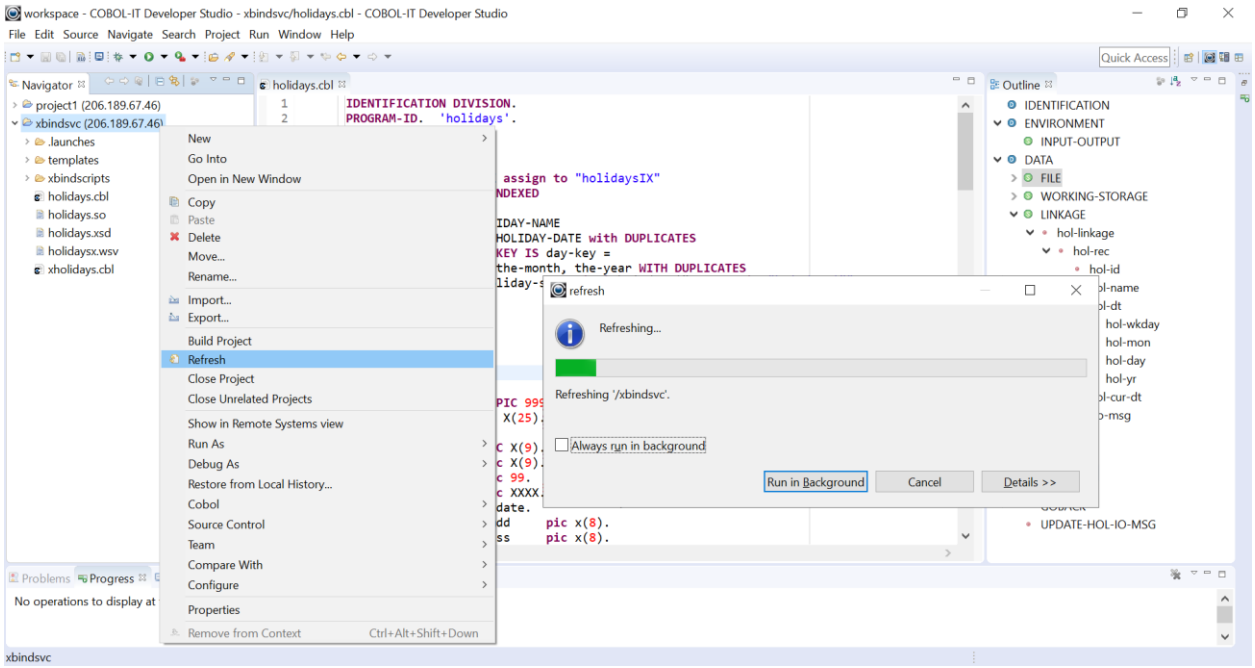
It is only necessary to generate a single bridge program for the multiple mappings that have been made. It is necessary, however, to generate separate bridge programs for the xbind solution and Apache Web Server solution.

As you drop down the Generate Bridge Program combo box, you will see the two default templates described at the top, and the customized templates, stored in the templates folder below them. In our sample, we are selecting the template which is designed to accept the incoming data in linkage, as is required when using xbind. The bridge program generator suggests a default name and location in which the program is generated. The default location is the project folder. The default name prepends an "x" to the name of the COBOL-IT Web Service program.
Click "Save" to generate the bridge program in the project folder.



Refresh your Developer Studio Project

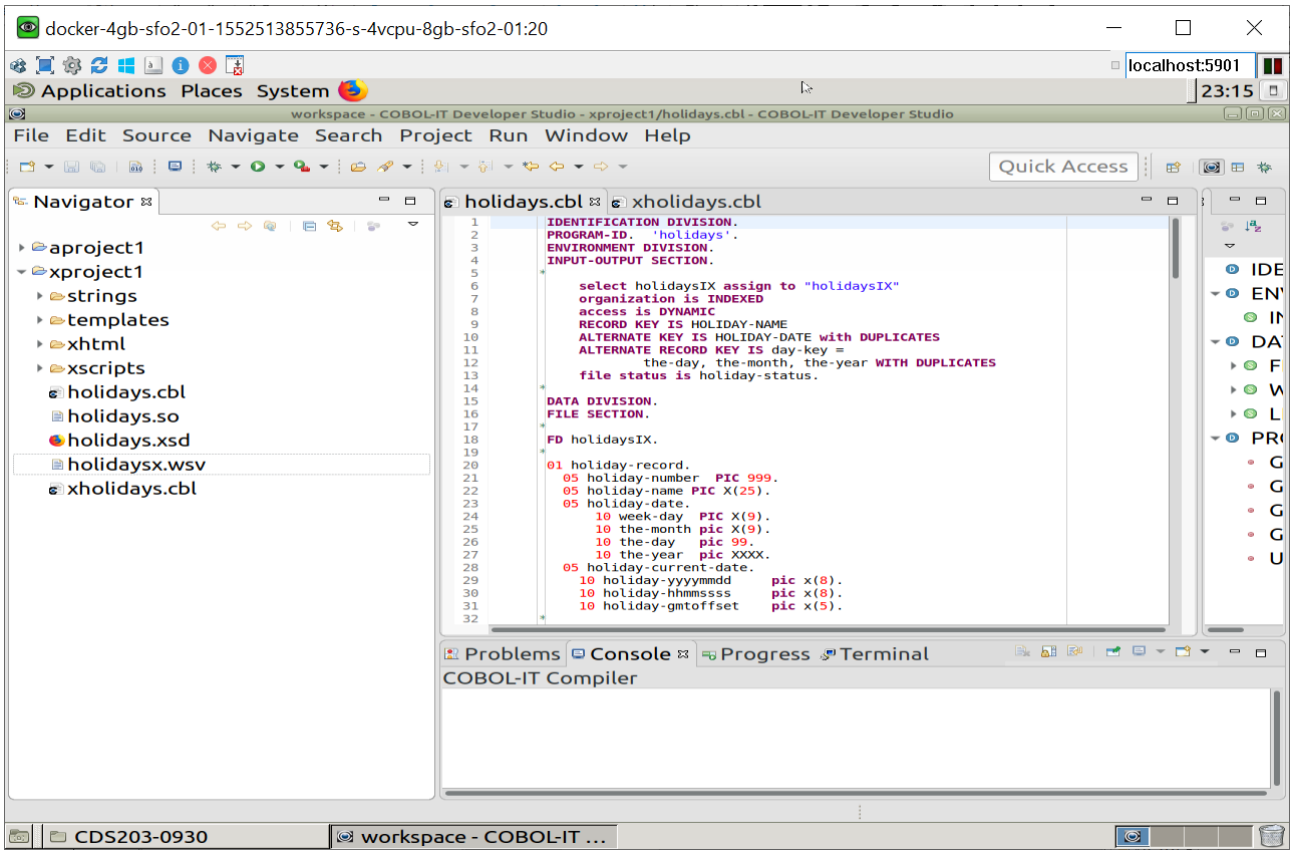
When using the Remote System Explorer, a remote project may need to be refreshed to display new files added. To refresh your project, right-click on the project name in the Developer Studio and select the Refresh function. Your project's artifacts now include the generated xsd file, the wsv file generated after the mapping was completed and saved, and the bridge program, xholidays.cbl.



A view of your COBOL-IT Web Services Project

A number of new files have been added to your Web Services Project.
Note: When using the Remote System Explorer, right-click on the Project folder, and select the Refresh function to make sure that your view of the folder is fully refreshed.

holidays.cbl	The original COBOL-IT Web Service source file
holidays.so	Created during the compilation that generated the XSD file
holidays.xsd	Created in the Settings interface with the Generate (*.xsd) function
holidays.wsv	Contains mapping information. Generated with the Save function on the Settings toolbar
xholidays.cbl	The generated bridge program

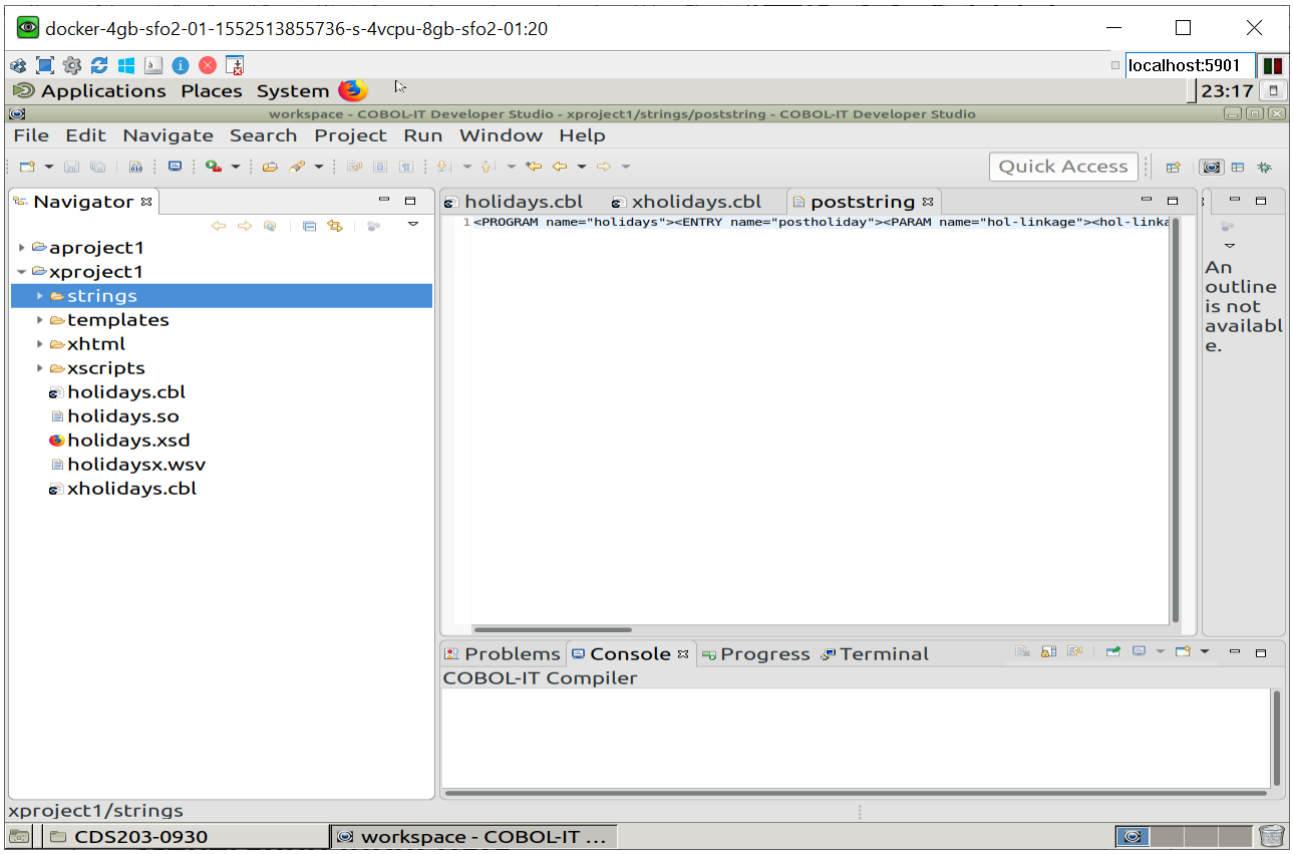


The Input String Generator

The COBOL-IT Bridge Program is designed to receive URL-encoded XML. The Input Generator produces a message, which, when sent to the Bridge Program, will be decoded, and the appropriate Entry point will be CALL'ed. That is, if the Input is being generated for a POST function, the message will be decoded and the Entry point associated with the POST function will be CALL'ed. There, the POST operation will take place, and the results will be returned to the bridge program, which will return them back to the Client.

Saving Input Strings in text files in your project

Before generating the Input Strings for our Web Services, we will create empty text files as place holders to save the generated strings. We will create an untitled text file called poststring and save it in the strings folder.

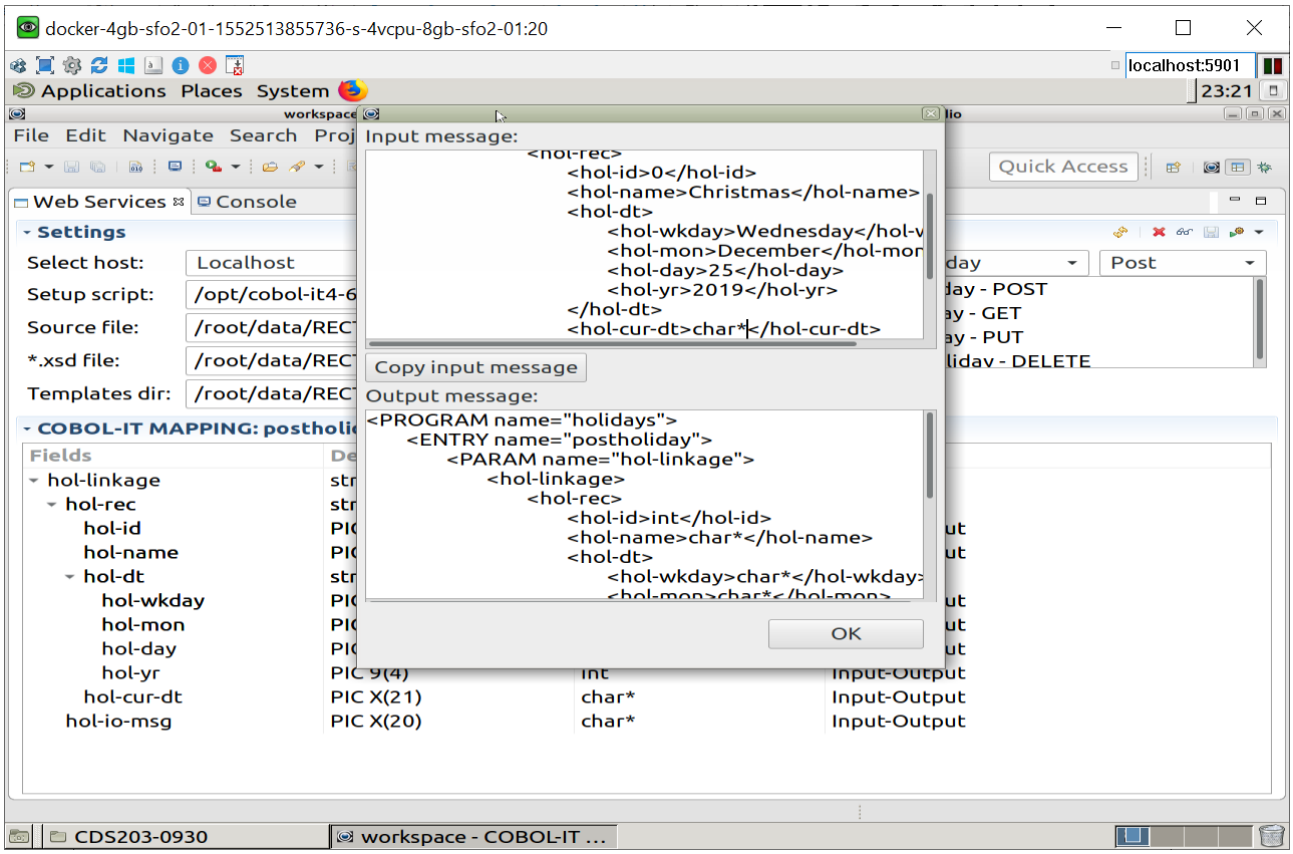


Generating input data

In this example, we will generate input data for a Post function. You will note that in the Entry Mapping interface, the “postholiday” entry point, which is associated with the Post function is selected. The message that is generated is structured such that the postholiday entrypoint will be CALL’ed, and passed data through the linkage section. For the separate cases, where you wished to GET, or PUT, or DELETE this record, you would require separate generations of input data, in which the “getholiday”, “putholiday”, or “deleteholiday” entry point, with their associated function, was selected.

Click on the “eyeglasses” icon to generate a structured IO message. In the message, you will see the castings that have been made. You can, for the purposes of testing, or for generating input data for use, overwrite a casting message with a data value. In our example, for our “holidays” application, we have entered a holiday name of “Christmas”, along with the subsequent date information of Wednesday, December 25, 2019. The hol-id of 0 will be replaced with a 1 when the record is posted. When the record is posted, we will also see a current date-time stamp, and a message will be generated reporting on the success or failure of the operation.

To capture the input message in your Clipboard, click on the “Copy input message” button.

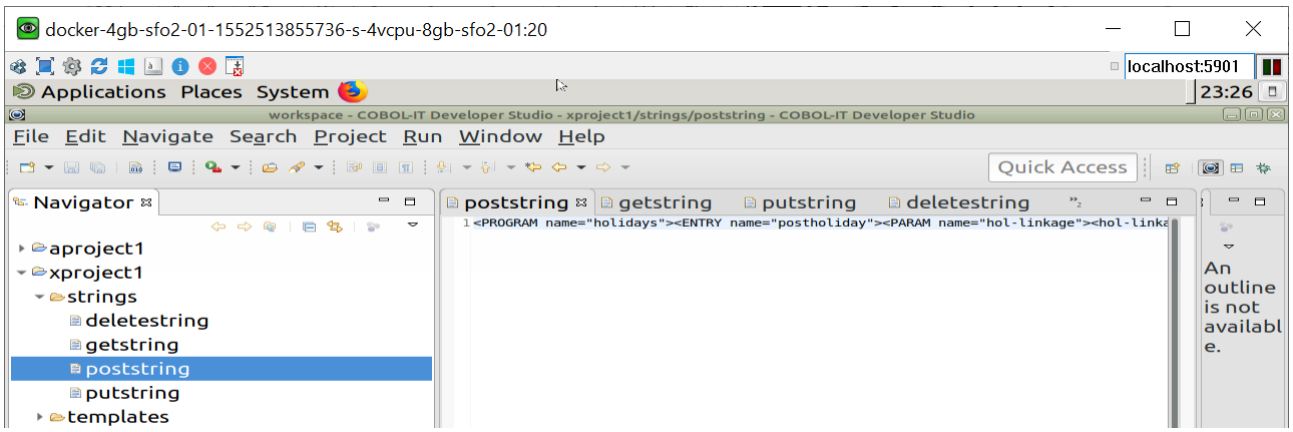


Click on the “Copy Input message” button to copy the INPUT STRING to the clipboard. Click OK.

Save the INPUT STRINGs in your project

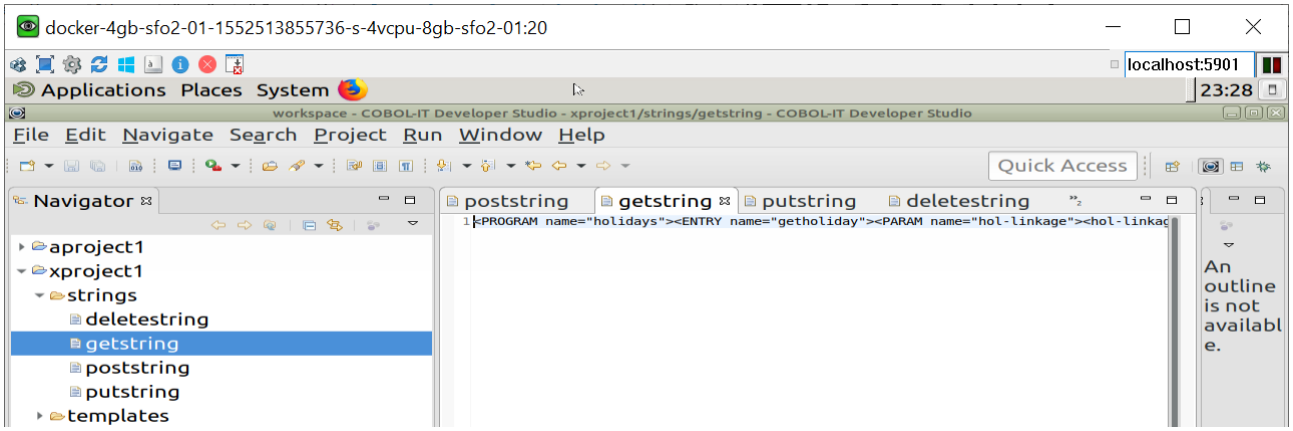
Paste the INPUT STRING into the empty text file you have created, and save the file.
Note that the INPUT STRING includes the program name that will be CALL’ed by the bridge program, the ENTRY point name- in this case “postholiday”- that will be called in the program, and descriptions of the USING data elements being passed in through the linkage section. When first generated, the INPUT STRING is not URL-encoded.

poststring



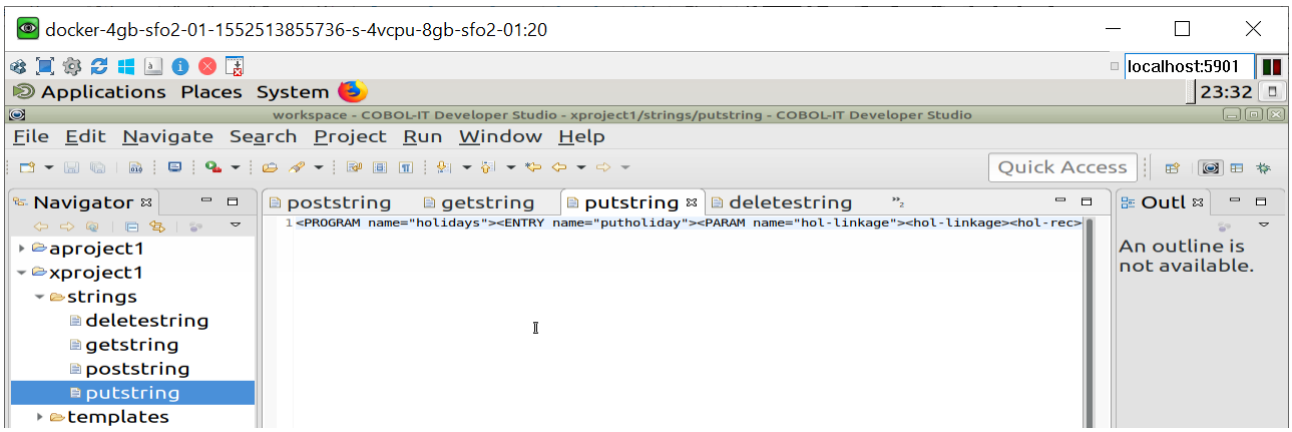
getstring

The INPUT STRING for getholiday only requires the entry of the primary key, in this case, the holiday-name, or hol-name as it is called in the Linkage Section. Note that the INPUT STRING includes the ENTRY point name- in this case “getholiday”- that will be called in the program, and descriptions of the USING data elements being passed in through the linkage section.



putstring

The INPUT STRING for putholiday only requires the entry of the primary key, in this case, the holiday-name, or hol-name as it is called in the Linkage Section.

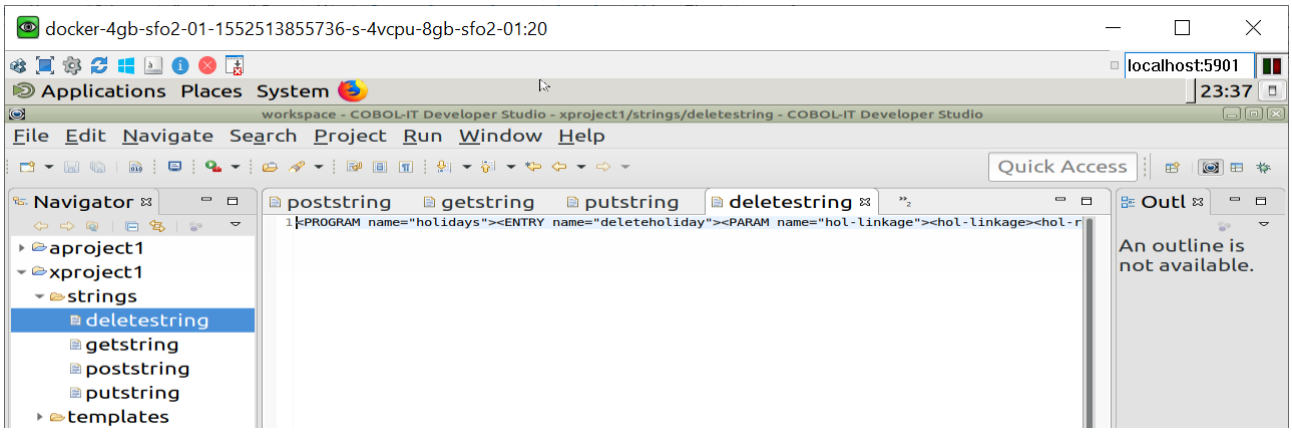


Note that the INPUT STRING includes the ENTRY point name- in this case “putholiday”- that will be called in the program, and descriptions of the USING data elements being passed in through the linkage section.

deletestring

The INPUT STRING for deleteholiday only requires the entry of the primary key, in this case, the holiday-name, or hol-name as it is called in the Linkage Section.

Note that the INPUT STRING includes the ENTRY point name- in this case “deleteholiday”- that will be called in the program, and descriptions of the USING data elements being passed in through the linkage section.



Summary

The COBOL-IT Web Services provides an interface which allows the user to parse a COBOL program with one or more entry points containing a USING clause that references a linkage section, and create the artifacts needed to generate INPUT STRINGs for each of the Entry points in the

program with USING clauses. The INPUT STRINGs are in XML format. In the natural course of electing a POST, GET, PUT, or DELETE function, this INPUT string will be URL-encoded. The BRIDGE PROGRAM is designed to decode the URL-encoded XML, populate the linkage items for a given entry point, and call the selected ENTRY point in the host program.
We have seen how to use the COBOL-IT Web Services Interface to produce the intermediate artifacts- the XSD and WSV file, and the final artifacts- the INPUT STRINGs and the BRIDGE PROGRAM. Next, we will take more detailed looks at Important Topics. Then, we will proceed to examine some test cases, in which scripts are run that make use of the INPUT STRINGs and BRIDGE PROGRAMs in different solution scenarios, most notably the Apache Web Server Solution, the xbind Solution which are brower-oriented solutions, and a curl Solution, which has implications for consuming COBOL-IT Web Services from a program.

Deeper Dives

Setting up the Environment

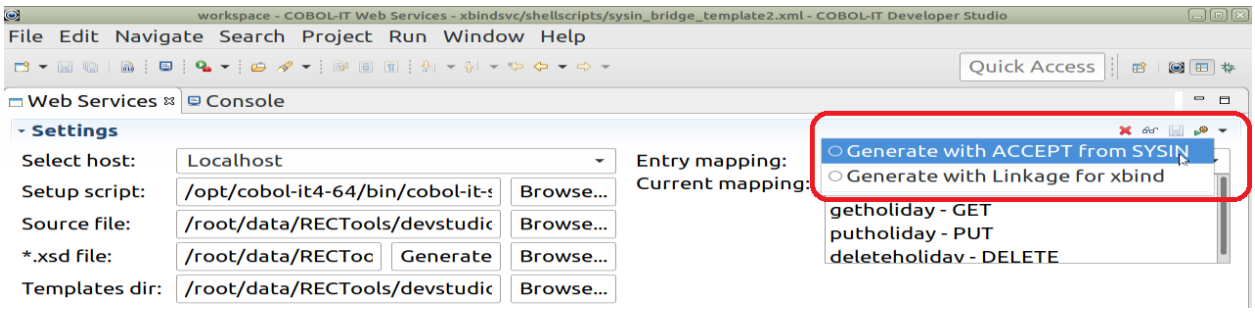
The Template Folder

The Default Templates

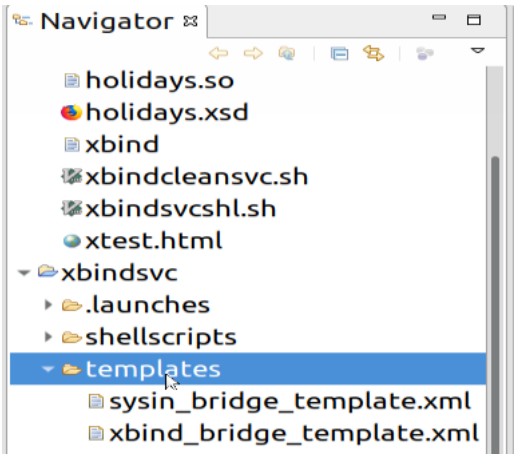
The default templates are “Generate with ACCEPT from SYSIN”, and “Generate with Linkage for xbind”. These templates differ in the way that they receive the URL-encoded input. When operating in an Apache Web Server environment, you will have an executable which is created with the compilation of the bridge program together with the COBOL-IT Web Services program. This executable is deployed to the cgi-bin folder of the Web Server, and receives the incoming data through an ACCEPT from SYSIN. When operating in a Browser environment, with the xbind listener, you will have a shared object deployed, which includes the compiled objects of both the bridge program and the COBOL-IT Web Service program. This shared object receives the incoming data through the Linkage Section of the bridge program, which is launched by the listener xbind.

Customizing Templates

Templates are identified in the Bridge Program Generator by their “TEMPLATE name”. The first of the default templates has a template name of “Generate with ACCEPT from SYSIN”.
<TEMPLATE name="Generate with ACCEPT from SYSIN">
<ID-DIVISION>
…..
This is what you see in the Bridge Program Generator dropdown box:



Note that the Template name is different than the file name. In this case, we see our two default templates, which will be generated into the templates folder after being used.
In the Navigator Window, you can see the actual file names differ from the more descriptive Template names seen above:



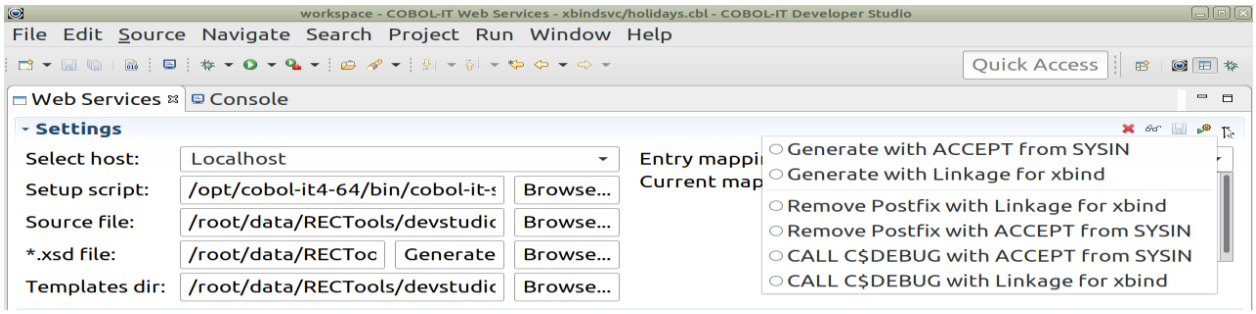
To create a Customized template, first generate the custom templates into your template folder. Then, make your changes to the default template. Before you save the new template, change the Template Name. To change the template name, open the template in a Text Editor. On the first line of the default xbind Template, you will see:

<TEMPLATE name="Generate with Linkage for xbind">

Changes to the default template could included Debugging DISPLAY UPON SYSERR statements, or a CALL "C$DEBUG" statement, or the addition of a procedure that is not included in the default template. After you have made your changes, return to the first line, create a new TEMPLATE name, and save the file to a new file name in your templates folder. Note that if you make changes to the default template, and do not change the Template name, your changes will be overwritten the next time the default template is used.

Selecting a customized template

When you have created customized templates, and identified them with descriptive Template names, and stored the customized templates in your templates folder, you will be able to select your customized templates for the generation of your bridge program:



As with the default templates, the default behavior will be to generate a program that prepends the letter 'x' to the source file name. As we have seen, in our sample, the source file 'holidays.cbl' is paired with the bridge program 'xholidays.cbl'.

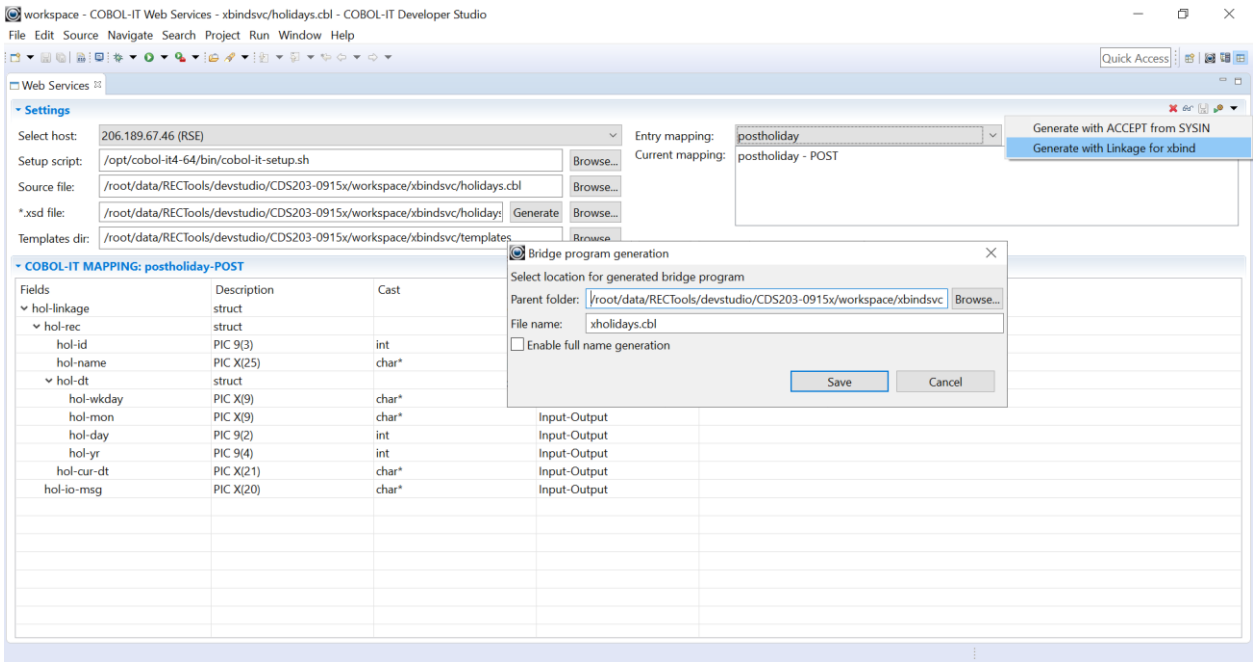
Entry Mapping

Generating the Bridge Program

The Bridge Program is generated from within the COBOL-IT Web Services perspective after the generation of the XSD which records the Entry Points, and Linkage Sections for each of the Entry Points, and after the Mappings have been completed. By default there are two options:
( ) Generate with ACCEPT from SYSIN
( ) Generate with Linkage for xbind
However, the Generate bridge program interface will also detect the presence of customized templates in the templates folder, and allow them to be selected.
The two default options reflect the two ways in which the COBOL-IT bridge program receives the message.

About the xbind solution

xbind takes a port number and a module name as input. The program will bind itself to the specified port where it waits for incoming connections. When a message is received, xbind will fork a sub process of itself to handle that message while the main process will continue to listen for new messages. The word 'bind' in this context means that incoming connections are queued in an input queue in case that they arrive faster than the program can process them.
xbind is a COBOL-IT utility, and uses the COBOL-IT runtime API to launch the named program.
There are a few important considerations when using this implementation:
First, xbind launches the COBOL-IT bridge program, and passes the message through to the Linkage Section of the bridge program. As a result, when working with xbind, you must use a bridge program generator that receives the message through the Linkage Section.



Second, xbind requires that the COBOL-IT license be installed, and that the full COBOL-IT setup script be executed prior to being launched, and as it makes use of COBOL-IT libraries.

Finally, xbind is designed to launch a shared object. As a result, when building the Web Service, you must compile the bridge program, and web service program together using the -b compiler flag.

>cobc -b xholidays.cbl holidays.cbl

This creates a single shared object from the two programs, and provides the ability to of the bridge program to CALL an ENTRY point in the web service program, which is required, if you design your COBOL-IT Web Service to be able to handle calls to each of the POST, GET, PUT and DELETE functions.

When implementing through an HTML page, you will see that the HTML page is written with : form action=http://localhost:9735 … to trigger interaction with xbind, which must be previously launched to listen on port 9735. You will also notice that the target program is run in the background. Programs being deployed as Web Services are best run in the background, as they will not be associated with a Terminal, and DISPLAYs can create unpredictable behaviors.

In our shell script:
./xbind 9735 xholidays &

About the Apache Web Server solution

Useful Commands

The Apache HTTP server is the most widely-used web server in the world. In order to use the COBOL-IT Web Services solution, it is helpful to have some guidelines about installing the Web Server, and verifying that it is running correctly. Some useful commands include:

$ sudo ufw app list	Lists applications that have registered their profiles with UFW upon installation. Verify that OpenSSH is listed as an Available application.
$ sudo ufw allow OpenSSH	Allow SSH connections
$ sudo ufw enable	Enable the firewall. Type “y” at the prompt.
$ sudo ufw status	Verify that OpenSSH is ALLOWed from Anywhere

Commands to install the Apache Web Server:

$ sudo apt-get update	Updates the local package to reflect the latest changes.
$ sudo apt-get install apache2	Install Apache and all required dependencies
$ sudo ufw app list	List existing application profiles. The goal is to enable access to Apache through the firewall.
$ sudo ufw allow ‘Apache Full’	Allow incoming traffic for the Apache Full profile

$ sudo ufw status	Verify that Apache Full ALLOWs HTTP traffic from Anywhere
$ sudo systemctl status apache2	Verify that Apache is running

$ http://your-servers-public-IP-address	Request a page from Apache. If you don't know it:
	$ sudo apt-get install curl
	$ curl -4 icanhazip.com

Enter your IP address into your browser's address bar. You should see the default Apache web page.	

Some other useful commands:

$ sudo systemctl stop apache2	Stops the web server
$ sudo systemctl start apache2	Starts the web server when it is stopped
$ sudo systemctl restart apache2	Stop and then start the web server
$ apache2 -v	Version information on apache Ex: Server version Apache/2.4.18 (Ubuntu)
$ curl http://127.0.0.1	Output in HTML printed on screen

Running an executable in the cgi-bin folder

When using a Web Server, such as Apache, the bridge program and Web Service program must be compiled together, creating an executable, which must be launched from within a shell script in the cgi-bin directory. There are a few important considerations when using this implementation:

The Web Server (Apache, for example), must be up and running.

Where your program requires Writing or Reading a file, the Web Server must have privileges to access those files in those directories. In our samples, we have the set the COB_FILE_PATH to /tmp and the COB_ERROR_FILE to /tmp/coberrorfile. This was done for expediency- we could be sure that the Web Server would have privileges to read and write the /tmp folder.

The shell script you are executing in the cgi-bin folder should locate the COBOL-IT license using the COBOLIT_LICENSE environment variable. The shell script should also duplicate the settings made in the running of the cobol-it-setup script. Finally, none of the previous environments established in your development environment will be recognized. So, if you require runtime environment variables for the COB_ERROR_FILE, COB_FILE_PATH, or COB_EXTFH functionalities for example, they must be included in the shell script executing in the cgi-bin folder.

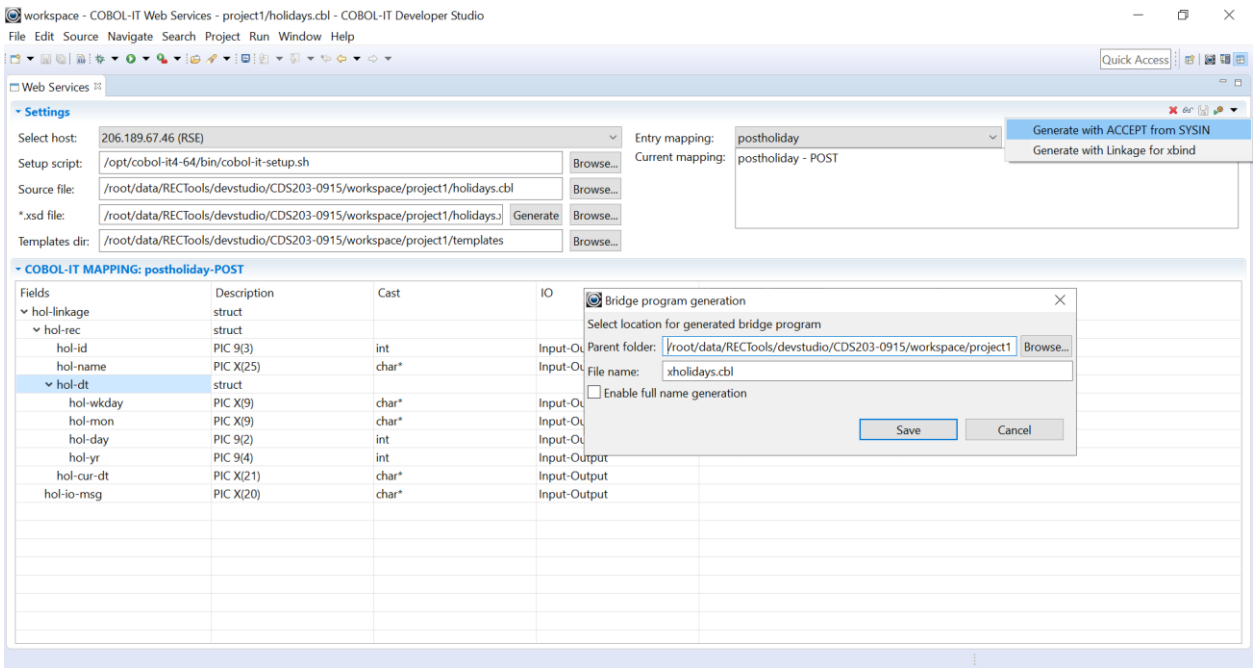
The shell script that you are executing in the cgi-bin folder should be set with executable privileges for all. >chmod 755 cit.sh or >chmod 777 cit.sh works. The executable that you have copied into the cgi-bin folder, and which is referenced by the shell script must also be executable by all.

The shell script that we are executing has a first line of :
#!/bin/bash

There are a number of scenarios whereby if you omit this line, or mistype it, the effect can be that your shell script will not work correctly.

Your executable should not perform any DISPLAYs, and it would be a common standard to require that it be run in the background.

First, the COBOL-IT bridge program and web service program are compiled together, creating an executable. The message is received via an ACCEPT from SYSIN statement in the bridge program. As a result, when working with a Web Server, such as Apache, you must use a bridge program generator that receives the message through ACCEPT from SYSIN.



About the Wildfly (JBoss) solution

Useful Commands

WildFly (JBoss), is an open-source application server, that is developed by Red Hat.
WildFly is written in Java, and is based on pluggable subsystems, which provide a maximum amount of agility on a lightweight framework

Detailed instructions are available for download on the how to install WildFly on your Linux platform. The Download instructions include instructions on installing pre-requisites (OpenJDK), downloading, installing and configuring WildFly. This includes instructions on making modifications to the wildfly configuration file. The launch.sh shell script is copied into the /opt/wildfly/bin folder and made executable. The systemd unit file is copied to the /etc/system/system folder, and the daemon is reloaded.

This is a short list of useful commands, which can be performed by a user with sudo privileges.

Commands to install WildFly:

$ sudo systemctl start wildfly	Starts the WildFly service
$ sudo systemctl status wildfly	Check the output for a status of active (running)
$ sudo systemctl enable wildfly	Enables the service to be automatically started at boot time
$ sudo ufw allow 8080/tcp	Allows traffic on port 8080
$ sudo /opt/wildfly/bin/add-user.sh	Create a user who will be able to connect using the administration console. When prompted, select a for Management User. Then follow prompts to create a user-name and password.
$ http://< your domain or ip address .:8080	Opens the default WildFly page
$ http://localhost:9990/console	The WildFly administration console. Sign in using the user-name and password you have created.
$ sudo systemctl restart wildfly	Restart the wildfly service if you have made changes to wildfly.service, wildfly.conf or launch.sh files. See documentation for details.

spring-resteasy.war, and the XProgram bean

There are a few important considerations when using this implementation:

The Wildfly Application Server must be up and running. User must have sudo privileges and must have access to the Wildfly Web Console.

Our spring-resteasy sample is accessible through github at: https://github.com/wildfly/quickstart/tree/17.0.1.Final/spring-resteasy. In the spring-resteasy quickstart example, we have added an XProgram bean. The bean is initialized with a program execution working directory and with a path to a shell script to execute. The process method of the bean accepts a string as an argument, executes the configured command line program and returns the program execution output. In the java source code, you will find:

```
<!-- Xprogram bean -->
<bean id=”xProgramBean”
class=”org.jboss.as.quickstarts.resteasyspring.XProgramBean”>
 <!—Program execution working directory -->
<constructor-arg index=”1” type =”java.lang.String”
value=/vagrant/cobol/webservices/xholidays/run.sh” />
</bean>
```

As a result, you should create the full path “/vagrant/cobol/webservices/xholidays” on your server. In the xholidays folder, create “run.sh”, as seen below. run.sh must first locate the COBOL-IT license, then run the cobol-it-setup script, and finally run the xholidays executable, which has been created with the command:

```
>cobc -x xholidays.cbl holidays.cbl
When debugging, use:
>cobc -g -ftraceall -x xholidays.cbl holidays.cbl
```

When setting COB_FILE_PATH, append a slash “/” to the end of the path name.
Locating files in the /tmp/ folder ensures that the Application Server will have privileges to Create files, Read Files, Write Files, and Delete files in the named folder.

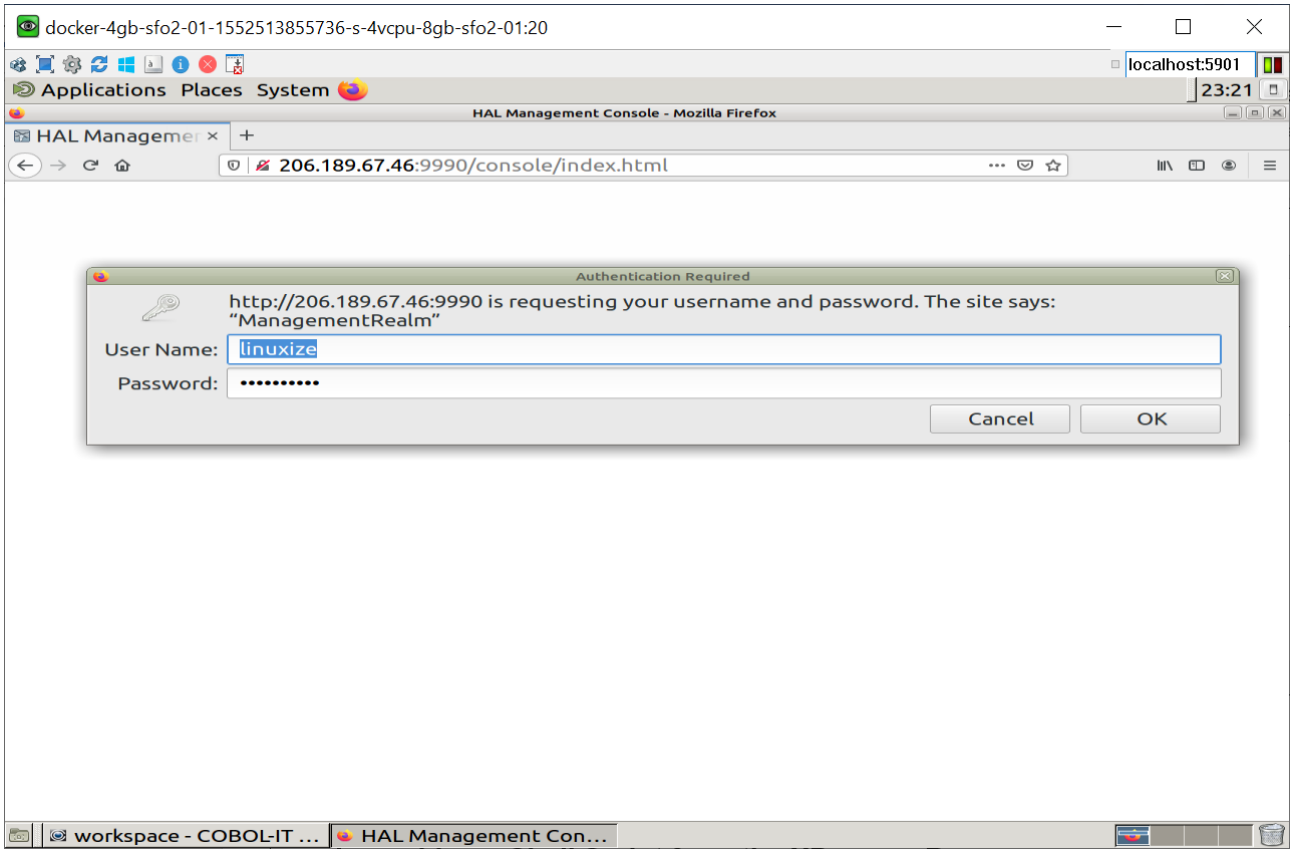
run.sh

```
#!/bin/bash
export COBOLIT_LICENSE=/opt/cobol-it4-64/citlicense.xml
COBOLITDIR=/opt/cobol-it4-64
PATH=$COBOLITDIR/bin:${PATH}
LD_LIBRARY_PATH="$COBOLITDIR/lib:${LD_LIBRARY_PATH:=}"
DYLD_LIBRARY_PATH="$COBOLITDIR/lib:${DYLD_LIBRARY_PATH:=}"
SHLIB_PATH="$COBOLITDIR/lib:${SHLIB_PATH:=}"
LIBPATH="$COBOLITDIR/lib:${LIBPATH:=}"
COB="COBOL-IT"
COB_ERROR_FILE=/tmp/jcoberrplus
export COB_FILE_PATH=/tmp/
```

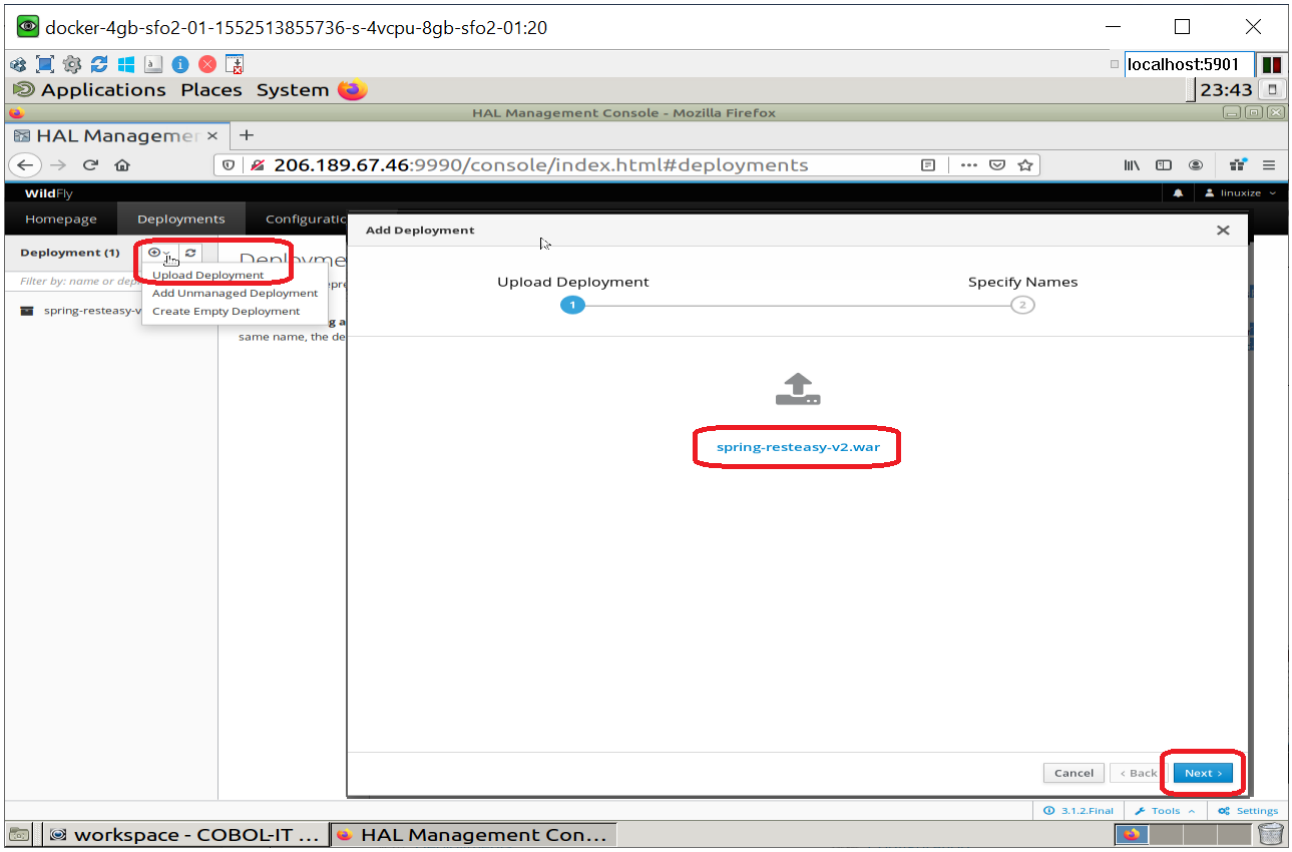
export COB_FILE_TRACE=Y
export COB COBOLITDIR LD_LIBRARY_PATH PATH DYLD_LIBRARY_PATH SHLIB_PATH LIBPATH COB_ERROR_FILE
./xholidays

Deploying spring-resteasy.war to the Server

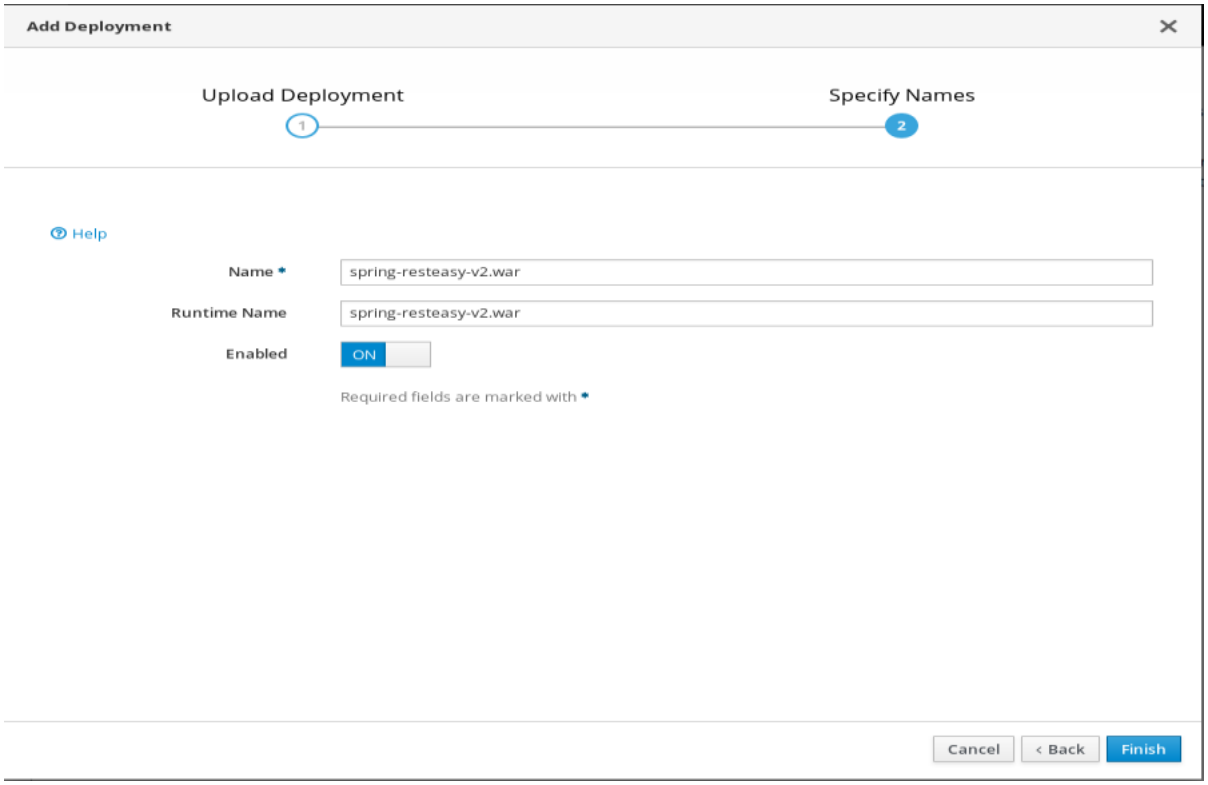
In order to run this sample, you will have to deploy spring-resteasy.war to the Server.
Enter http:// < your ip address >:9990/console/index.html into your browser. You will be prompted for the User and Password you have created:



This will open the HAL Management Console. On the HAL Management Console, select the "Deployments" tab to open the Deployments Dialog Window. From the Dropdown box, select "Upload Deployment". In the Add Deployment dialog screen, use the "Drag a File" link to open a File Browse interface. Locate your file, and click "Open" in the Browse interface to prepare your file for deployment. Click "Next".



Provide your application with a name and set Enable to On. Click Finish. Your application is deployed, and enabled, and appears in the Navigator Window on the left.



Launching a Shell Script from the XProgramBean

Where your program requires Writing or Reading a file, the Web Server must have privileges to access those files in those directories. In our samples, we have the set the COB_FILE_PATH to /tmp/ and the COB_ERROR_FILE to /tmp/jcoberrplus. This was done for expediency- we could be sure that the Application Server would have privileges to read and write the /tmp folder.

The shell script you are executing should locate the COBOL-IT license using the COBOLIT_LICENSE environment variable. The shell script should also duplicate the settings made in the running of the cobol-it-setup script. Finally, none of the previous environments established in your development environment will be recognized. So, if you require runtime environment variables for the COB_ERROR_FILE, COB_FILE_PATH, or COB_EXTFH functionalities for example, they must be included in the run.sh shell script.

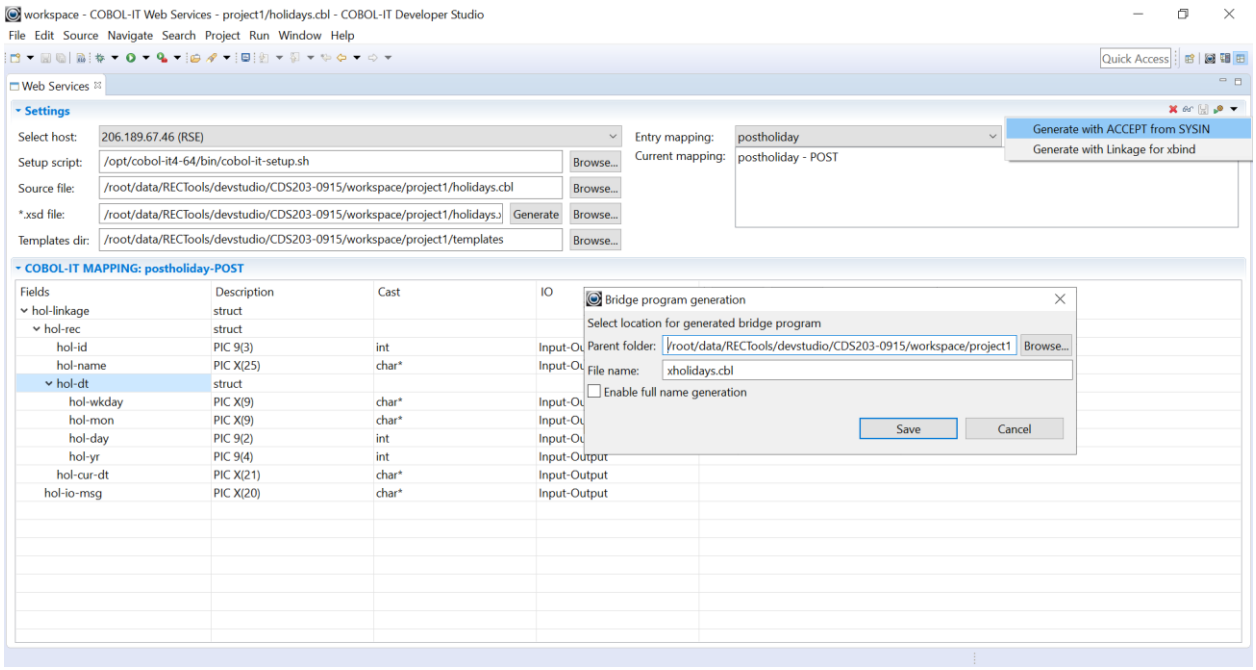
The run.sh shell script should be set with executable privileges for all. >chmod 755 run.sh or >chmod 777 run.sh works. The executable that you have copied into the xholidays folder, and which is referenced by the shell script must also be executable by all.

The shell script that we are executing has a first line of :
#!/bin/bash
There are a number of scenarios whereby if you omit this line, or mistype it, the effect can be that

your shell script will not work correctly.

Your executable should not perform any DISPLAYs, and it would be a common standard to require that it be run in the background.

First, the COBOL-IT bridge program and web service program are compiled together, creating an executable. The message is received via an ACCEPT from SYSIN statement in the bridge program. As a result, when working with an Application Server, such as WildFly, you must use a bridge program generator that receives the message through ACCEPT from SYSIN.



About the xbind solution

Debugging COBOL-IT Web Services

When running COBOL-IT Web Services, you have a number of debugging options:

- When you build your shared library or executable, depending on the environment in which you are running the Web Service, you can change the compile options so that they include debugging and file-tracing debugger options. In our example, we compile with -g -ftraceall, and set COB_ERROR_FILE = coberr.txt in our shell script. This will have the effect of generating a statement trace in the bridge program and web service program, as they are executed.
- You can enhance a template with Debugging DISPLAY…. UPON SYSERR statements. The output will be written to the COB_ERROR_FILE.

- You can enhance your shell script by setting the runtime environment variable COB_DEBUG_ID to 12345 (for example) and create a template that includes a CALL C$DEBUG statement. When the CALL C$DEBUG statement is executed, the runtime will pause, and, in the COBOL-IT Developer Studio, you can initiate a Debug Attach session, and resume debugging in the COBOL-IT Developer Studio.

Appendix A holidays.cbl Web Service

holidays.cbl COBOL Source Code

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  'holidays'.
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
*
     select holidaysIX assign to "holidaysIX"
     organization is INDEXED
     access is DYNAMIC
     RECORD KEY IS HOLIDAY-NAME
     ALTERNATE KEY IS HOLIDAY-DATE with DUPLICATES
     ALTERNATE RECORD KEY IS day-key =
             the-day, the-month, the-year WITH DUPLICATES
     file status is holiday-status.
*
 DATA DIVISION.
 FILE SECTION.
*
 FD holidaysIX.
*
 01 holiday-record.
   05 holiday-number  PIC 999.
   05 holiday-name PIC X(25).
   05 holiday-date.
       10 week-day  PIC X(9).
       10 the-month pic X(9).
       10 the-day   pic 99.
       10 the-year  pic XXXX.
   05 holiday-current-date.
     10 holiday-yyyymmdd     pic x(8).
     10 holiday-hhmmssss     pic x(8).
     10 holiday-gmtoffset    pic x(5).
*
 01 holiday-record-2.
   05 holiday-number-2   PIC 999.
   05 holiday-name-2 pic X(25).
   05 holiday-date-2 pic X(24).
   05 holiday-current-date-2  pic X(21).
*
 WORKING-STORAGE SECTION.
*
 77 holiday-status pic xx.
 77 ws-holiday-number PIC 999 VALUE 0.
 77 ws-dummy  pic x.
 77 holiday-io-msg PIC x(20).
*
 LINKAGE SECTION.
 01 hol-linkage.
  03 hol-rec.
   05 hol-id PIC 999.
```

```
   05 hol-name PIC X(25).
   05 hol-dt.
     10 hol-wkday  PIC X(9).
     10 hol-mon  PIC X(9).
     10 hol-day  PIC 99.
     10 hol-yr   PIC 9(4).
   05 hol-cur-dt PIC x(21).
  03 hol-io-msg PIC X(20).
*

*
 PROCEDURE DIVISION.
*
 ENTRY 'postholiday' USING hol-linkage.
     DISPLAY "IN postholiday" UPON SYSERR.
*
     OPEN OUTPUT holidaysIX.

     IF holiday-status NOT = "00"
       DISPLAY"OPEN FAILED: ", holiday-status UPON SYSERR
     ELSE
       DISPLAY "OPEN SUCCESSFUL: ", holiday-status UPON SYSERR
       INITIALIZE holiday-record
       ADD 1 TO ws-holiday-number
       MOVE FUNCTION current-date TO holiday-current-date
       MOVE ws-holiday-number TO holiday-number
       MOVE hol-name TO holiday-name
       MOVE hol-dt TO holiday-date
*
       WRITE holiday-record
       IF holiday-status not = "00"
         DISPLAY "WRITE FAILED!: ", holiday-status UPON SYSERR
       ELSE
         DISPLAY"WRITE SUCCESSFUL!: ", holiday-status UPON SYSERR
         DISPLAY "Holiday Record: ", holiday-record UPON SYSERR
         MOVE holiday-number TO hol-id
         MOVE holiday-current-date TO hol-cur-dt
         PERFORM UPDATE-HOL-IO-MSG
         MOVE holiday-io-msg TO hol-io-msg
       END-IF
*
       CLOSE HOLIDAYSIX
       DISPLAY "CLOSE STATUS: ", HOLIDAY-STATUS UPON SYSERR
     END-IF.
*
     GOBACK.
*
 ENTRY 'getholiday' USING hol-linkage.
     DISPLAY "IN getholiday" UPON SYSERR.
*
     OPEN INPUT holidaysIX.
*
     IF holiday-status NOT = "00"
       DISPLAY"OPEN FAILED: ", holiday-status UPON SYSERR
     ELSE
       DISPLAY "OPEN SUCCESSFUL: ", holiday-status UPON SYSERR
       DISPLAY "OPEN Status: ", holiday-status UPON SYSERR
*
```

```
        MOVE hol-name TO holiday-name
        READ holidaysIX KEY IS holiday-name
        DISPLAY "READ Status: ", holiday-status UPON SYSERR
        IF holiday-status NOT = "00"
          DISPLAY "READ FAILED!" UPON SYSERR
        ELSE
          DISPLAY"READ SUCCESSFUL!" UPON SYSERR
          MOVE holiday-record TO hol-rec
        END-IF
        PERFORM UPDATE-HOL-IO-MSG
        MOVE HOLIDAY-IO-MSG TO HOL-IO-MSG
*
        CLOSE holidaysIX
        DISPLAY "CLOSE Status: ", holiday-status UPON SYSERR
      END-IF.

      GOBACK.
*
 ENTRY 'putholiday' USING hol-linkage.
      DISPLAY "IN putholiday" UPON SYSERR.
*
      OPEN I-O holidaysIX.
*
      IF holiday-status NOT = "00"
        DISPLAY"OPEN FAILED: ", holiday-status UPON SYSERR
      ELSE
        DISPLAY "OPEN SUCCESSFUL: ", holiday-status UPON SYSERR
        MOVE hol-name TO holiday-name
        READ holidaysIX KEY IS holiday-name
*
        IF holiday-status NOT = "00"
          DISPLAY "READ FAILED!: ", holiday-status UPON SYSERR
        ELSE
          DISPLAY"READ SUCCESSFUL! ", holiday-status UPON SYSERR
          MOVE holiday-record TO hol-rec
          MOVE FUNCTION CURRENT-DATE TO HOLIDAY-CURRENT-DATE
          REWRITE holiday-record
*
          IF holiday-status NOT = "00"
            DISPLAY "REWRITE FAILED: ", holiday-status UPON SYSERR
          ELSE
            DISPLAY"REWRITE SUCCESSFUL!: ", holiday-status
                                                    UPON SYSERR
            MOVE holiday-record TO hol-rec
          END-IF
        END-IF
        PERFORM UPDATE-HOL-IO-MSG
        MOVE HOLIDAY-IO-MSG TO HOL-IO-MSG
*
        CLOSE holidaysIX
        DISPLAY "CLOSE Status: ", holiday-status UPON SYSERR
      END-IF.

      GOBACK.
*
 ENTRY 'deleteholiday' USING hol-linkage.
      DISPLAY "IN deleteholiday" UPON SYSERR.
*
```

```
      OPEN I-O holidaysIX.
      IF holiday-status NOT = "00"
        DISPLAY "OPEN FAILED!: ", holiday-status UPON SYSERR
      ELSE
        MOVE hol-name TO holiday-name
        READ holidaysIX KEY IS holiday-name
        IF holiday-status NOT = "00"
          DISPLAY "READ FAILED: ", holiday-status UPON SYSERR
        ELSE
          DISPLAY "READ SUCCESSFUL: ", holiday-status UPON SYSERR
          DELETE holidaysIX RECORD
          IF holiday-status NOT = "00"
            DISPLAY "DELETE FAILED!: ", holiday-status UPON SYSERR
          ELSE
            DISPLAY "DELETE SUCCESSFUL!: ", holiday-status
                                                 UPON SYSERR
            MOVE HOLIDAY-RECORD TO HOL-REC
          END-IF
        END-IF
        PERFORM UPDATE-HOL-IO-MSG
        MOVE holiday-io-msg TO hol-io-msg
*
        CLOSE holidaysIX
        DISPLAY "CLOSE Status: ", holiday-status UPON SYSERR
      END-IF.
*
      GOBACK.
*
 UPDATE-HOL-IO-MSG.
      INITIALIZE hol-io-msg.
      STRING "HOLIDAY-STATUS:" DELIMITED BY SIZE,
             holiday-status DELIMITED BY SIZE,
             INTO holiday-io-msg.
*
```

The Apache Solution Shell Scripts

aclean.sh

```
rm /usr/lib/cgi-bin/xholidays
rm xholidays
rm /tmp/holidaysIX /tmp/holidaysIX.idx
rm /tmp/acoberrplus
/usr/bin/pkill -f firefox
```

abuild.sh

```
#/bin/bash
echo 'starting shellscript'
echo 'set -e'
set -e
echo 'run cit setup script'
export DEFAULT_CITDIR=/opt/cobol-it4-64
export COBOLITDIR=$DEFAULT_CITDIR
export PATH=$COBOLITDIR/bin:${PATH}
export LD_LIBRARY_PATH="$COBOLITDIR/lib:${LD_LIBRARY_PATH:=}"
export DYLD_LIBRARY_PATH="$COBOLITDIR/lib:${DYLD_LIBRARY_PATH:=}"
export SHLIB_PATH="$COBOLITDIR/lib:${SHLIB_PATH:=}"
export LIBPATH="$COBOLITDIR/lib:${LIBPATH:=}"
export COB="COBOL-IT"
echo COBOL-IT Environement set to $COBOLITDIR
echo 'building service ...'
cobc -g -ftraceall -x xholidays.cbl holidays.cbl
cp xholidays /usr/lib/cgi-bin/
echo 'all done'
```

xplus40.sh

```
#!/bin/bash
export COBOLIT_LICENSE=/opt/cobol-it4-64/citlicense-all.xml
COBOLITDIR=/opt/cobol-it4-64
PATH=$COBOLITDIR/bin:${PATH}
LD_LIBRARY_PATH="$COBOLITDIR/lib:${LD_LIBRARY_PATH:=}"
DYLD_LIBRARY_PATH="$COBOLITDIR/lib:${DYLD_LIBRARY_PATH:=}"
SHLIB_PATH="$COBOLITDIR/lib:${SHLIB_PATH:=}"
LIBPATH="$COBOLITDIR/lib:${LIBPATH:=}"
COB="COBOL-IT"
COB_ERROR_FILE=/tmp/acoberrplus
export COB_FILE_PATH=/tmp/
export COB COBOLITDIR LD_LIBRARY_PATH PATH DYLD_LIBRARY_PATH SHLIB_PATH LIBPATH
COB_ERROR_FILE
./xholidays
```

aallwithcurl.sh

```
#/bin/bash
curl --ipv4 --no-buffer -o /tmp/acurlpost --data-urlencode @/tmp/poststring
http://localhost/cgi-bin/xplus40.sh
curl --ipv4 --no-buffer -o /tmp/acurlget  --data-urlencode @/tmp/getstring
http://localhost/cgi-bin/xplus40.sh
curl --ipv4 --no-buffer -o /tmp/acurlput --data-urlencode @/tmp/putstring
http://localhost/cgi-bin/xplus40.sh
curl --ipv4 --no-buffer -o /tmp/acurldelete --data-urlencode @/tmp/deletestring
http://localhost/cgi-bin/xplus40.sh
curl --ipv4 --no-buffer -o /tmp/acurlget2 --data-urlencode @/tmp/getstring
http://localhost/cgi-bin/xplus40.sh
```

apost.sh

```
#!/bin/bash
echo 'starting firefox...'
firefox ./ahtml/appost.html
```

appost.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost/cgi-bin/xplus40.sh" target="result2"
method="post" id="myForm" accept-charset="ISO-8859-1">
<input type="submit" value="POST">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

aget.sh

```
#!/bin/bash
echo 'starting firefox...'
firefox ./ahtml/apget.html
```

apget.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost/cgi-bin/xplus40.sh" target="result2"
method="post" id="myForm" accept-charset="ISO-8859-1">
<input type="submit" value="GET">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

aput.sh

```
#!/bin/bash
echo 'starting firefox...'
firefox ./ahtml/aput.html
```

aput.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost/cgi-bin/xplus40.sh" target="result2"
method="post" id="myForm" accept-charset="ISO-8859-1">
<input type="submit" value="PUT">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

adelete.sh

```
#!/bin/bash
echo 'starting firefox...'
firefox ./ahtml/apdelete.html
```

apdelete.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost/cgi-bin/xplus40.sh" target="result2"
method="post" id="myForm" accept-charset="ISO-8859-1">
<input type="submit" value="DELETE">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

The Wildfly (JBoss) Solution Shell Scripts

jclean.sh

```
rm xholidays
rm /tmp/holidaysIX /tmp/holidaysIX.idx
rm /vagrant/cobol/webservices/xholidays/xholidays
echo 'rm jcoberrplus'
rm /tmp/jcoberrplus
/usr/bin/pkill -f firefox
```

jbuild.sh

```
#/bin/bash
echo 'starting shellscript'
echo 'set -e'
set -e
echo 'run cit setup script'
export DEFAULT_CITDIR=/opt/cobol-it4-64
export COBOLITDIR=$DEFAULT_CITDIR
export PATH=$COBOLITDIR/bin:${PATH}
export LD_LIBRARY_PATH="$COBOLITDIR/lib:${LD_LIBRARY_PATH:=}"
export DYLD_LIBRARY_PATH="$COBOLITDIR/lib:${DYLD_LIBRARY_PATH:=}"
export SHLIB_PATH="$COBOLITDIR/lib:${SHLIB_PATH:=}"
export LIBPATH="$COBOLITDIR/lib:${LIBPATH:=}"
export COB="COBOL-IT"
echo COBOL-IT Environement set to $COBOLITDIR
echo 'building service ...'
cobc -x -g -ftraceall -o xholidays xholidays.cbl holidays.cbl
cp xholidays /vagrant/cobol/webservices/xholidays/
echo 'all done'
```

run.sh

```
#!/bin/bash
export COBOLIT_LICENSE=/opt/cobol-it4-64/citlicense.xml
COBOLITDIR=/opt/cobol-it4-64
PATH=$COBOLITDIR/bin:${PATH}
LD_LIBRARY_PATH="$COBOLITDIR/lib:${LD_LIBRARY_PATH:=}"
DYLD_LIBRARY_PATH="$COBOLITDIR/lib:${DYLD_LIBRARY_PATH:=}"
SHLIB_PATH="$COBOLITDIR/lib:${SHLIB_PATH:=}"
LIBPATH="$COBOLITDIR/lib:${LIBPATH:=}"
COB="COBOL-IT"
COB_ERROR_FILE=/tmp/jcoberrplus
export COB_FILE_PATH=/tmp/
export COB_FILE_TRACE=Y
export COB COBOLITDIR LD_LIBRARY_PATH PATH DYLD_LIBRARY_PATH SHLIB_PATH LIBPATH
```

```
COB_ERROR_FILE
./xholidays
```

jallwithcurl.sh

```
#/bin/bash
curl -X POST -H "Content-Type: text/plain" --no-buffer -o /tmp/jcurlpost --data-
urlencode @/tmp/poststring http://localhost:8080/spring-resteasy-v2/xprogram
curl -X POST -H "Content-Type: text/plain" --no-buffer -o /tmp/jcurlget --data-
urlencode @/tmp/getstring http://localhost:8080/spring-resteasy-v2/xprogram
curl -X POST -H "Content-Type: text/plain" --no-buffer -o /tmp/jcurlput --data-
urlencode @/tmp/putstring http://localhost:8080/spring-resteasy-v2/xprogram
curl -X POST -H "Content-Type: text/plain" --no-buffer -o /tmp/jcurldelete --
data-urlencode @/tmp/deletestring http://localhost:8080/spring-resteasy-
v2/xprogram
curl -X POST -H "Content-Type: text/plain" --no-buffer -o /tmp/jcurlget2 --data-
urlencode @/tmp/getstring http://localhost:8080/spring-resteasy-v2/xprogram
```

jpost.sh

```
#/bin/bash
echo 'starting firefox...'
firefox ./jhtml/jpost.html
```

jpost.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost:8080/spring-resteasy-v2/xprogram"
target="result2" method="post" id="myForm" accept-charset="ISO-8859-1">
<input type="submit" value="POST">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

jget.sh

```
#/bin/bash
echo 'starting firefox...'
firefox ./jhtml/jget.html
```

jget.html

```
Message:<br>
```

```
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost:8080/spring-resteasy-v2/xprogram"
target="result2" method="post" id="myForm" accept-charset="ISO-8859-1">
<input type="submit" value="GET">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

jput.sh

```
#/bin/bash
echo 'starting firefox...'
firefox ./jhtml/jput.html
```

jput.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost:8080/spring-resteasy-v2/xprogram"
target="result2" method="post" id="myForm" accept-charset="ISO-8859-1">
<input type="submit" value="PUT">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

jdelete.sh

```
#/bin/bash
echo 'starting firefox...'
firefox ./jhtml/jdelete.html
```

jdelete.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost:8080/spring-resteasy-v2/xprogram"
```

```
target="result2" method="post" id="myForm" accept-charset="ISO-8859-1">
<input type="submit" value="DELETE">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

The xbind Solution Shell Scripts

xclean.sh

```
rm xholidays.so holidays.so
rm coberrorfile
echo 'pkill -f xbind, firefox'
/usr/bin/pkill -f xbind
/usr/bin/pkill -f firefox
killall -HUP xbind
```

xbuild.sh

```
#/bin/bash
echo 'starting shellscript'
echo 'set -e'
set -e
echo 'run cit setup script'
export DEFAULT_CITDIR=/opt/cobol-it4-64
export COBOLITDIR=$DEFAULT_CITDIR
export PATH=$COBOLITDIR/bin:${PATH}
export LD_LIBRARY_PATH="$COBOLITDIR/lib:${LD_LIBRARY_PATH:=}"
export DYLD_LIBRARY_PATH="$COBOLITDIR/lib:${DYLD_LIBRARY_PATH:=}"
export SHLIB_PATH="$COBOLITDIR/lib:${SHLIB_PATH:=}"
export LIBPATH="$COBOLITDIR/lib:${LIBPATH:=}"
export COB="COBOL-IT"
echo COBOL-IT Environement set to $COBOLITDIR
echo 'building service ...'
cobc -b -g -ftraceall -o xholidays.so xholidays.cbl holidays.cbl
echo 'all done'
```

startxbind.sh

```
#/bin/bash
echo 'starting shellscript'
echo 'set -e'
set -e
echo 'run cit setup script'
export DEFAULT_CITDIR=/opt/cobol-it4-64
export COBOLITDIR=$DEFAULT_CITDIR
export PATH=$COBOLITDIR/bin:${PATH}
export LD_LIBRARY_PATH="$COBOLITDIR/lib:${LD_LIBRARY_PATH:=}"
export DYLD_LIBRARY_PATH="$COBOLITDIR/lib:${DYLD_LIBRARY_PATH:=}"
export SHLIB_PATH="$COBOLITDIR/lib:${SHLIB_PATH:=}"
export LIBPATH="$COBOLITDIR/lib:${LIBPATH:=}"
export COB="COBOL-IT"
echo COBOL-IT Environement set to $COBOLITDIR
export COB_LIBRARY_PATH=.
export COB_FILE_PATH=.
export COB_ERROR_FILE=coberrorfile
echo 'launching xbind ...'
```

```
/opt/cobol-it4-64/bin/xbind 9735 xholidays &
```

xallwithcurl.sh

```
#/bin/bash
curl --ipv4 --no-buffer -o ./strings/xcurlpost --data-urlencode
@./strings/poststring http://localhost:9735
curl --ipv4 --no-buffer -o ./strings/xcurlget --get --data-urlencode
@./strings/getstring http://localhost:9735
curl --ipv4 --no-buffer -o ./strings/xcurlput --data-urlencode
@./strings/putstring http://localhost:9735
curl --ipv4 --no-buffer -o ./strings/xcurldelete --data-urlencode
@./strings/deletestring http://localhost:9735
curl --ipv4 --no-buffer -o ./strings/xcurlget2 --get --data-urlencode
@./strings/getstring http://localhost:9735
```

xpost.sh

```
#/bin/bash
echo 'starting firefox...'
firefox ./xhtml/xpost.html
```

xpost.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost:9735" target="result2" method="post" id="myForm"
accept-charset="ISO-8859-1">
<input type="submit" value="POST">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

xget.sh

```
#/bin/bash
echo 'starting firefox...'
firefox ./xhtml/xget.html
```

xget.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost:9735" target="result2" method="post" id="myForm"
accept-charset="ISO-8859-1">
```

```
<input type="submit" value="GET">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

xput.sh

```
#/bin/bash
echo 'starting firefox...'
firefox ./xhtml/xput.html
```

xput.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost:9735" target="result2" method="post" id="myForm"
accept-charset="ISO-8859-1">
<input type="submit" value="PUT">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

xdelete.sh

```
#/bin/bash
echo 'starting firefox...'
firefox ./xhtml/xdelete.html
```

xdelete.html

```
Message:<br>
<textarea rows="20" cols="90" maxlength="1000" name="msg" form="myForm">
</textarea>
<br>
<form action="http://localhost:9735" target="result2" method="post" id="myForm"
accept-charset="ISO-8859-1">
<input type="submit" value="DELETE">
</form>
<p>Result:</p>
<iframe name="result2" style="height:450px;width:643px;"></iframe>
```

www.cobol-it.com

June, 2020