How to use Web Services with z/VSE

Last formatted on: Monday, May 02, 2016

Ingo Franzki
ifranzki@de.ibm.com

Alina Glodowski
alina.glodowski@de.ibm.com

Disclaimer

Contents

1. Introduction

http://

This document describes how to use Web Services with z/VSE. z/VSE provides a SOAP Engine as part of the VSE Connectors component that supports the SOAP protocol (Simple Object Access Protocol).

With the z/VSE SOAP Engine you can:

- Provide Web Services to someone outside of z/VSE. In this case, an (existing or newly written) CICS program is web service enabled, so that it can be provided as a Web Service. In this scenario z/VSE acts as a SOAP Server.

- Use Web Services that someone outside of z/VSE is providing. In this case a CICS program calls a Web Service through the z/VSE SOAP Engine. In this scenario z/VSE acts as a SOAP Client.

This document covers both scenarios, and provides a step by step description for setting up all required subsystems, creating all required programs, as well as testing the Web Services.

Starting z/VSE 5.2 there are two supported version of SOAP of the z/VSE Engine:

- **SOAP Engine version 1**: Available since VSE/ESA 2.7. Supports SOAP encoding only.
- **SOAP Engine version 2**: It includes literal encoding and array support.

This article assumes that you already have some basic knowledge about the Web Services technology and its surrounding protocols and formats. The following terms and abbreviations will be used within this article:

• XML	- Extensible Markup Language
• WSDL	- Web Service Description Language
• TCP/IP	- Transmission Control Protocol/Internet Protocol
• HTTP	- Hypertext transport Protocol
• SOAP	- Simple Object Access Protocol
• CWS	- CICS Web Support

For more information about Web Services and z/VSE SOAP Engine, please also see the z/VSE e-business Connectors, User's Guide SC34-2629-02 and the documentation links at the very end of this document. It also describes the programming interfaces of the z/VSE SOAP Engine, which will be referenced in this document.

Note: Before you start using the z/VSE SOAP Engine please make sure you have applied all existing PTFs related to SOAP, CICS Web Support and TCP/IP.

You will find a list of available PTFs here:

http://www.ibm.com/systems/z/os/zvse/support/connectors.html
http://www.ibm.com/systems/z/os/zvse/support/tcpip.html
http://www.ibm.com/systems/z/os/zvse/support/ipv6vse.html
http://www.ibm.com/systems/z/os/zvse/support/cics.html

Download and Install CICS2WS

In order to use Web Services with z/VSE it's recommended to use a tool called CICS2WS. This toolkit helps you to perform Web Service development with z/VSE. It is free of charge and can be downloaded from the z/VSE homepage:
http://www.ibm.com/systems/z/os/zvse/downloads/#cics2ws

CICS2WS Tool supports both z/VSE SOAP Engine (version 1 and version 2). The tool works with both scenarios: z/VSE acts as a Server and z/VSE acts as a Client.

Note: The CICS2WS Toolkit is provided 'as-is', no support, no warranty. For questions or in case of problems, please send an email to zvse@de.ibm.com.

Download and install the CICS2WS Tool on your PC or workstation. When you start the CICS2WS Toolkit the first time, it will show a help page that shows how to download additional Java libraries required by CICS2WS. You need to download the following libraries and copy them to the originally empty *lib* folder:

Axis: Version: 1.2.1 or higher
http://axis.apache.org/axis/java/releases.html

After downloading *axis-bin-X_X_X.zip* you have to extract following libraries from the *lib* folder:

axis.jar
wsdl4j-1.5.1.jar
commons-discovery-0.2.jar
commons-logging-1.0.4.jar
log4j-1.2.8.jar
jaxrpc.jar
saaj.jar

axis-ant.jar is not necessary for the CICS2WS Application Toolkit.

ws-jaxme: Version: 0.4
http://www.apache.org/dyn/closer.cgi/ws/jaxme/

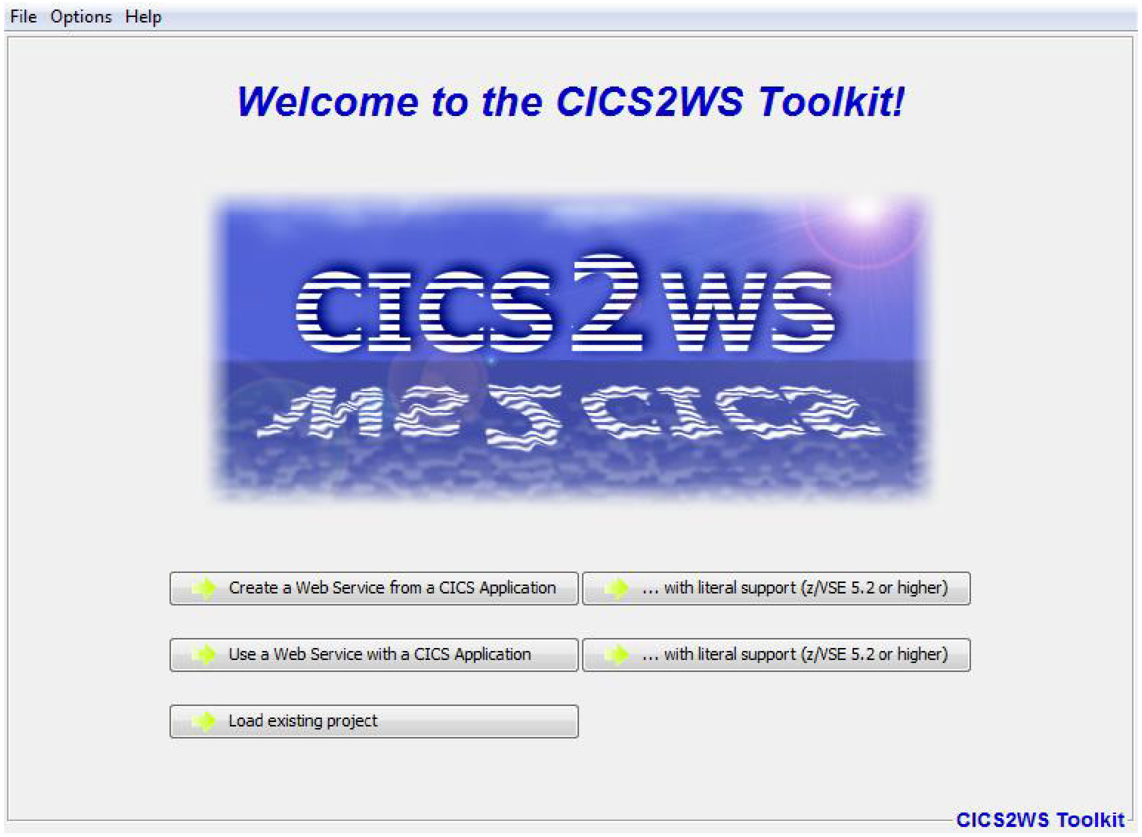
After downloading *ws-jaxme-0.4-bin.tar.gz* you have to extract following library from the lib folder:

jaxmexs.jar

The further libraries are not used for the CICS2WS Application Toolkit.

Note: IBM is not allowed to re-distribute those libraries as part of the CICS2WS Toolkit. That's why you need to download them separately.

When you start the CICS2WS Toolkit the next time, it should come up with the following title screen:


File Options Help
Welcome to the CICS2WS Toolkit!
CICS2WS
Create a Web Service from a CICS Application
... with literal support (z/VSE 5.2 or higher)
Use a Web Service with a CICS Application
... with literal support (z/VSE 5.2 or higher)
Load existing project
CICS2WS Toolkit

For z/VSE as a SOAP Server, we will use buttons "Create a Web Service from a CICS Application" and "… with literal support (z/VSE 5.2 or higher)" in the next steps (see chapter 2).

For z/VSE as a SOAP Client, we will use buttons "Use a Web Service with a CICS Application" and "… with literal support (z/VSE 5.2 or higher)" in the next steps (see chapter 3).

"Load Existing Project" button is used to work with the Project created and saved earlier.

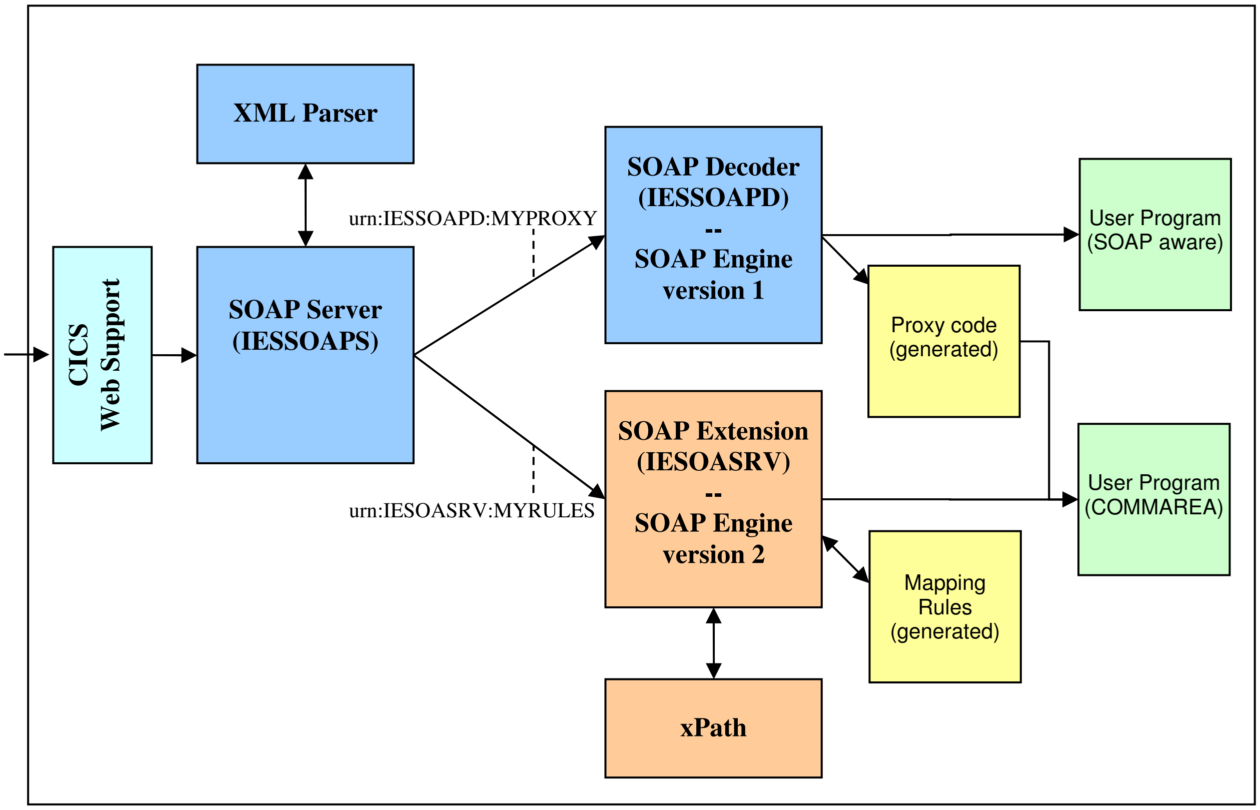
2. Providing Web Services with z/VSE – z/VSE acts as a SOAP server

This chapter describes step by step how to setup CICS TS and the z/VSE SOAP Engine and how to web service enable a CICS program in order to provide it as a Web Service.

Some of the steps below are setup tasks that only have to be done once, when you web service enable your very first program. Other steps are tasks that you must repeat for every program that you want to provide as a Web Service.

2.1. Overview

The figure below shows the modules that are involved when a Web Service request is processed.


XML Parser
urn:IESSOAPD:MYPROXY
CICS Web Support
SOAP Server (IESSOAPS)
SOAP Decoder (IESSOAPD) -- SOAP Engine version 1
User Program (SOAP aware)
Proxy code (generated)
SOAP Extension (IESOASRV) -- SOAP Engine version 2
urn:IESOASRV:MYRULES
User Program (COMMAREA)
Mapping Rules (generated)
xPath

The SOAP Server part of the z/VSE SOAP Engine is based on CICS Web Support (CWS). CWS acts as a HTTP server used for incoming SOAP requests. CWS will pass the requests to the z/VSE SOAP Engine which will perform any further processing.

When CWS receives a HTTP request on a TCPIPSERVICE, it uses the URL to determine which program to call. For Web Service requests, the URL must look like

```
http://hostname:port/cics/CWBA/IESSOAPS
```

This URL tells CICS to call program IESSOAPS (which is the server part of the z/VSE SOAP Engine) under the CICS supplied transaction CWBA. IESSOAPS basically is a regular CICS program that uses EXEC CICS WEB commands to retrieve the HTTP content (i.e. the SOAP request), processes it and sends a SOAP response back as HTTP response.

When the SOAP server has received the SOAP message, which is XML, it calls the XML parser to parse it. It then analyses and verifies the SOAP message. Based on the URN (the method namespace (URN), it is determined which z/VSE SOAP Engine version is used:

urn:**IESSOAPD**:MYPROXY
urn:**IESOASRV**:MYRULES

z/VSE SOAP Engine version 1.
In case of urn:**IESSOAPD**:MYPROXY the next program to call is IESSOAPD, the SOAP decoder program. The SOAP decoder looks into the SOAP message and extracts information about the method to execute and its parameters. It also deserializes the parameters from its XML representation into a z/VSE specific representation. Based on the second part of URN, MYPROXY, it then determines what program to call next. This can either be a user program that is SOAP aware (i.e. it uses the z/VSE SOAP Engines programming interface directly), or a proxy program, that has been generated by the CICS2WS Toolkit. The Proxy program then translates the input parameters into a COMMAREA and calls the user program.

z/VSE SOAP Engine version 2.
In case of urn:**IESOASRV**:MYRULES the next program to call is IESOASRV, the new SOAP decoder program. The SOAP decoder looks into the SOAP message and extracts information about the method to execute and its parameters. It also deserializes the parameters from its XML representation into a z/VSE specific representation. In this case second part of URN, MYRULES is used by IESOASRV for correct converting SOAP data types and SOAP messages to user program data types and copybook. MYRULES is an assembler code, that has been generated by the CICS2WS Toolkit starting version 2.6.

When the user program returns, it goes back the same way. This time the output parameters are serialized by IESSOAPD/ IESOASRV and a SOAP response message is created. IESSOAPS finally sends the response back via HTTP.

2.2. Setup CICS Web Support

Note: CWS is only available on CICS Transaction Server (CICS TS). If you want to web service enable programs running in CICS/VSE 2.3, you need to configure and run a separate CICS TS region with CWS, and use remote program definitions (MRO) to call your CICS/VSE programs from CICS TS.

CWS is also used for other types of TCP/IP communication with CICS, i.e. CICS Transaction Gateway (ECI) and the 3270 Bridge. If you have already set up CWS for those types of communication, you probably have done most of this step already.

Please refer to the CICS Transaction Server Internet Guide, SC34-5765 for detailed information about CICS Web support.

Here are the tasks required to setup CICS Web support:

1. Enable CICS Web Support in the CICS System Initialization Table (SIT):
   - ISC=YES enable intersystem communication
   - TCPIP=YES enable TCP/IP protocol
2. Configure and enable codepage conversion in CICS. Use the IBM provided skeleton DFHCNV (ICCF library 59)
3. Configure and enable the CICS Web Error program. Use the IBM provided skeleton DFHWBEP (ICCF library 59)
4. CICS Web support needs to be able to obtain the hostname of the local IP address it is working under. With TCP/IP for VSE use the TCP command DEFINE NAME to define a name for the local IP address (e.g. DEFINE NAME,NAME=name,IPaddr=n.n.n.n). If you forget to do this step, you will get the following message at CICS startup:

```
DFHSO0117 applid Unable to determine the TCP/IP host name.
  Language Environment return code X'00000458', return code X'00000000'.
  TCP/IP services are unavailable.
```

5. You need to adjust the CICS startup job, to specify the ID of the TCP/IP stack with which CICS should work. You specify the ID using the // OPTION SYSPARM statement (e.g. // OPTION SYSPARM=nn).
6. After CICS has been started, check if TCPIP support is active. Use CEMT INQ TCPIP to check if it is OPEN.
7. Define and install a TCPIPSERVICE using CEDA DEFINE TCPIPSERVICE similar to the example shown below:

```
  CEDA  DEFine TCpipservice(SOAP)
    TCpipservice   : SOAP
    Group          : VSESPG
    Description  ==> SERVICE FOR SOAP
    Urm          ==> DFHWBADX
    Portnumber   ==> 80          1-65535
    Certificate  ==>
    STatus       ==> Open        Open | Closed
    SSl          ==> NO          Yes | No |    Clientauth
    Attachsec    ==> Local       Local | Verify
    TRansaction  ==> CWXN
    Backlog      ==> 00009       0-32767
    TSqprefix    ==>
    Ipaddress    ==>
    SOcketclose  ==> No          No | 0
```

Use CEMT INQ TCPIPSERVICE to check if it is OPEN.

2.3. Create a CICS program that is to be web service enabled

When Web Service enabling an existing CICS program, you have 2 options how to do that:

- Modifying the program to interface with the z/VSE SOAP Engine directly.
- Use the CICS2WS Toolkit to generate a so called Proxy code that acts as a wrapper between the z/VSE SOAP Engine and your program
- Use CICS2WS Toolkit to generate a so called mapping rules that transfer data between SOAP and your program (z/VSE 5.2 only)

While interfacing directly with the z/VSE SOAP Engine gives you the greatest flexibility, it requires good knowledge about the programming interfaces of the z/VSE SOAP Engine. Please see Appendix C: for some details of the programming interfaces. Choosing this approach is recommended when very complex data structures are being used in the Web Service parameters.

This article will focus on the use of the CICS2WS Toolkit to generate the so called proxy/rules code that acts as a wrapper between the z/VSE SOAP Engine and your program. This approach assumes that your CICS program, that is to be web service enabled, can be called via EXEC CICS LINK and a user defined COMMAREA.

Requirements for programs to be able to get web service enabled

A CICS program that you want to web service enable (i.e. make it callable as a Web Service) must fulfill the following requirements:

- It can be written in any CICS supported programming language (e.g. Assembler, COBOL, PL/1, C, CICS/REXX)
- It must be callable via EXEC CICS LINK and a COMMAREA. It must return to the caller via EXEC CICS RETURN.
- It may access files and/or call other programs, but it must return back to the z/VSE SOAP Engine via EXEC CICS RETURN.
- It can NOT use any terminal specific functions, since there is no terminal assigned when called as Web Service.
- If the program does not run on CICS TS, you can use a remote program definition and use MRO to call a program residing in a different CICS region. The z/VSE SOAP Engine will only run in CICS TS.
- You must know the structure of the COMMAREA used by the program. I.e. you must have a copybook in COBOL, PL/1 or assembler that describes the structure of the COMMAREA. You will later import this copybook into the CICS2WS Toolkit.

In this chapter we will use two copybooks as samples.
Copybook 1 will be used to show how to create proxy code:

```
     03 CUSTOMER.
             05 CUSTOMERNR    PIC 9(9) BINARY.
             05 FIRSTNAME     PIC X(20).
             05 LASTNAME      PIC X(20).
           03 ACCOUNT.
             05 ACCOUNTNR     PIC 9(9) BINARY.
             05 AMOUNT        PIC 9(9) BINARY.
           03 SOURCEACCOUNT   PIC 9(9) BINARY.
           03 TARGETACCOUNT   PIC 9(9) BINARY.
```

Copybook 2 will be used to demonstrate array/literal style support starting z/VSE 5.2:

```
      3 EMPLOYEELIST.
               5 EMPLOYEE OCCURS 123.
                 7 ID   PIC 9(9) BINARY.
                 7 NAME PIC X(20).
                 7 BIRTHDAY PIC 9(9) COMP.
             3 EOVER40 PIC 9(9) COMP.
```

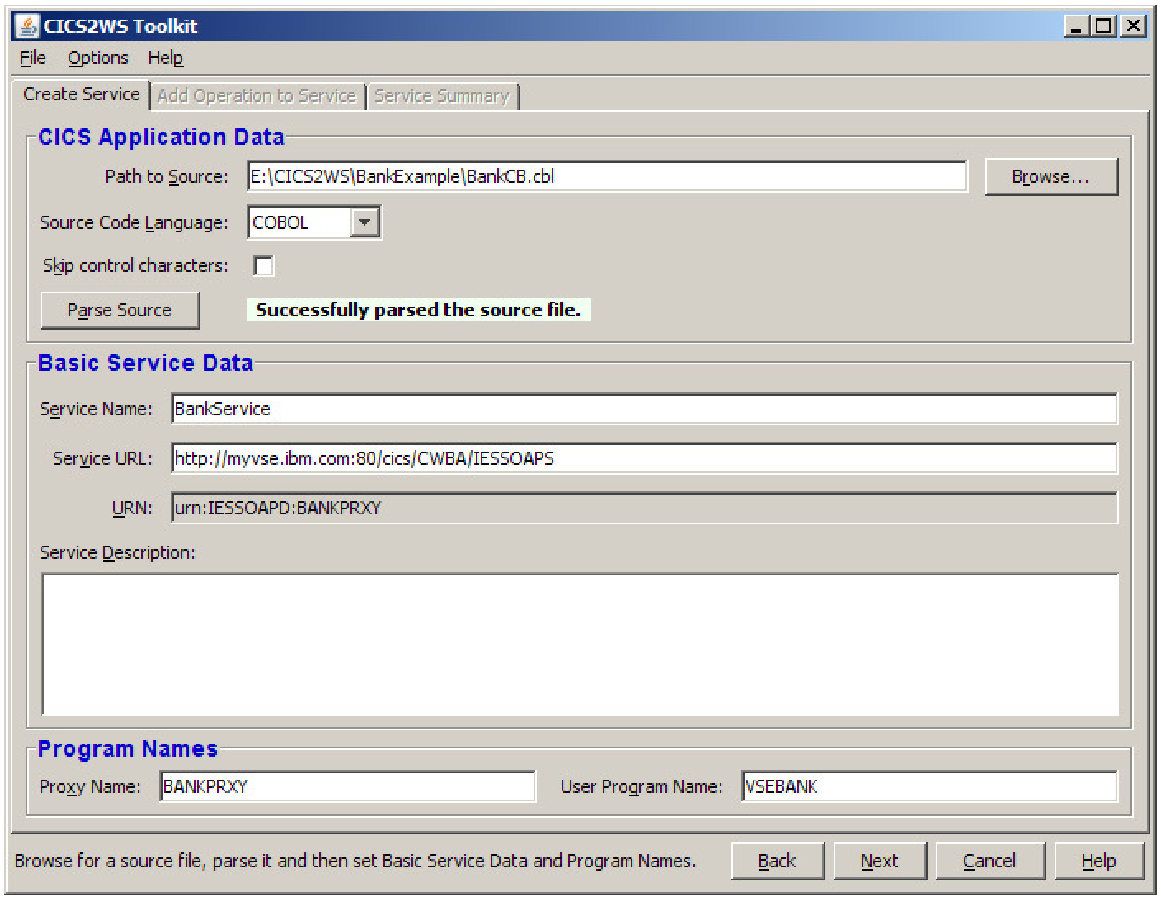
For z/VSE as a SOAP Server, we will use upper buttons in CICS2WS tool main screen “Create a Web Service from a CICS Application” and “… with literal support (z/VSE 5.2 or higher)” in the next steps.

2.4. Create a Web Service from a CICS Application

To use a z/VSE SOAP Engine version 1 button "Create a Web Service from a CICS Application" should be chosen. In this case CICS2WS Tool will generate based on the provided copybook a proxy program and wsdl file.

Import Copybook and create the Web Service to provide

In this step, we use the Copybook1 as described in chapter 2.3 and import it into the CICS2WS Toolkit. CICS2WS can read copybooks in COBOL, PL/1 and Assembler language. You may need to download the copybook from z/VSE in order to import it into the CICS2WS Toolkit. You can download the copybook using standard file transfer functions, like FTP.



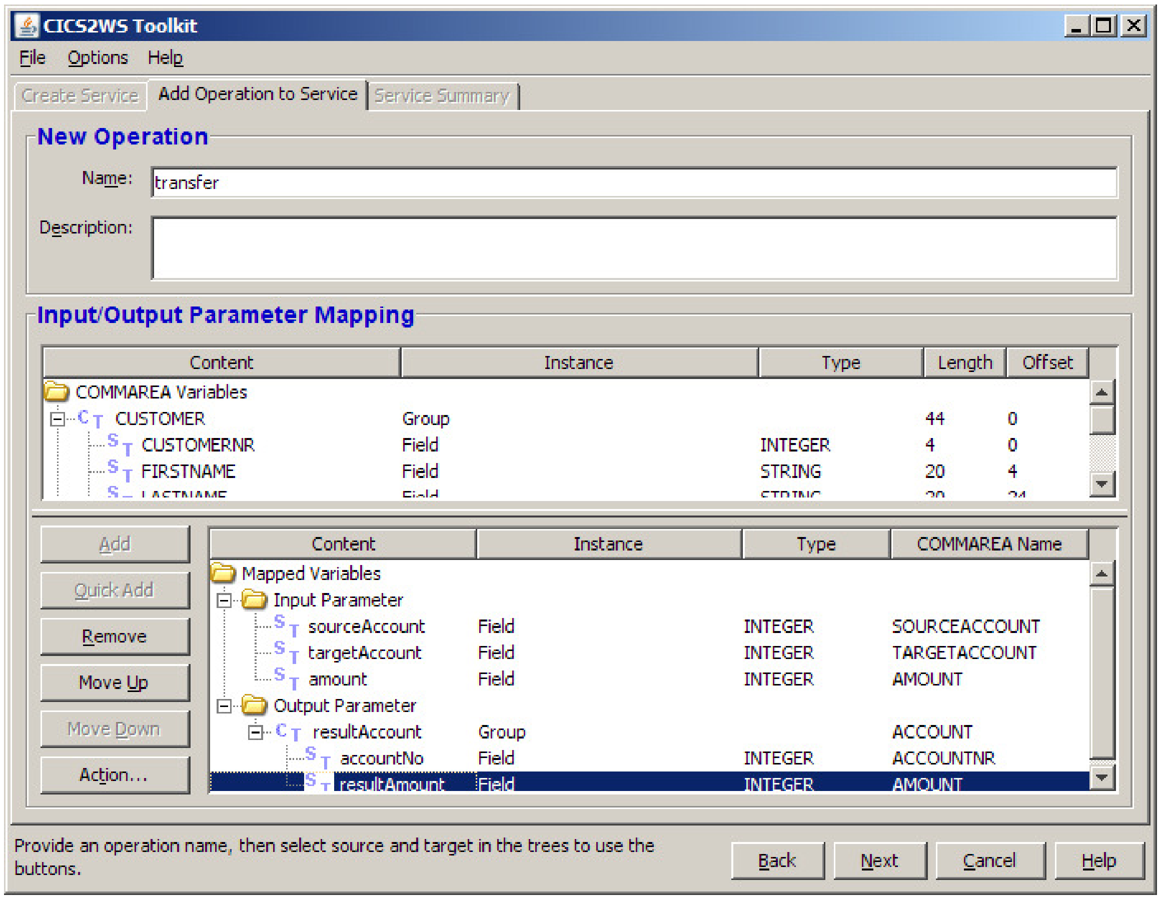
Path to Source: E:\CICS2WS\BankExample\BankCB.cbl
Source Code Language: COBOL
Skip control characters: ☐
Successfully parsed the source file.
Service Name: BankService
Service URL: http://myvse.ibm.com:80/cics/CWBA/IESSOAPS
URN: urn:IESSOAPD:BANKPRXY
Service Description:
Proxy Name: BANKPRXY
User Program Name: VSEBANK
Browse for a source file, parse it and then set Basic Service Data and Program Names.

In CICS2WS, press the "Browse…" button and choose the file that contains the copybook. The source code language is automatically detected based on the file extension. If it did not detect it correctly, choose the language manually. Then press the "Parse" button. In case you receive any errors, check the copybook source and correct the errors.

Once the copybook was imported successfully, you need to provide some basic information about the Web Service you want to provide. This includes the name of the service and the URL, where it will be hosted later on. The URL has a fixed format; you should only change the "server-name" and "port". The server-name is the hostname or IP address of your VSE system that will host the service. The port is the TCP/IP port you defined in the TCPIPSERVICE in chapter 2.2

Last, you specify the name of the proxy program that is to be generated, and the name of the program that implements the service and is to be web service enabled.

Press the "Next" button to continue to the next page.



Name: transfer

Description:

Content	Instance	Type	Length	Offset
COMMAREA Variables				
CUSTOMER	Group		44	0
CUSTOMERNR	Field	INTEGER	4	0
FIRSTNAME	Field	STRING	20	4

Content	Instance	Type	COMMAREA Name
Mapped Variables			
Input Parameter			
sourceAccount	Field	INTEGER	SOURCEACCOUNT
targetAccount	Field	INTEGER	TARGETACCOUNT
amount	Field	INTEGER	AMOUNT
Output Parameter			
resultAccount	Group		ACCOUNT
accountNo	Field	INTEGER	ACCOUNTNR
resultAmount	Field	INTEGER	AMOUNT

Provide an operation name, then select source and target in the trees to use the buttons.

You now define the Web Service operation (also called method) that you want to provide and define what input and output parameters it has.

Note: In general a Web Service can provide multiple operations. However, CICS2WS only supports one operation per Web Service. If you want to provide multiple operations, you need to create multiple Web Services.

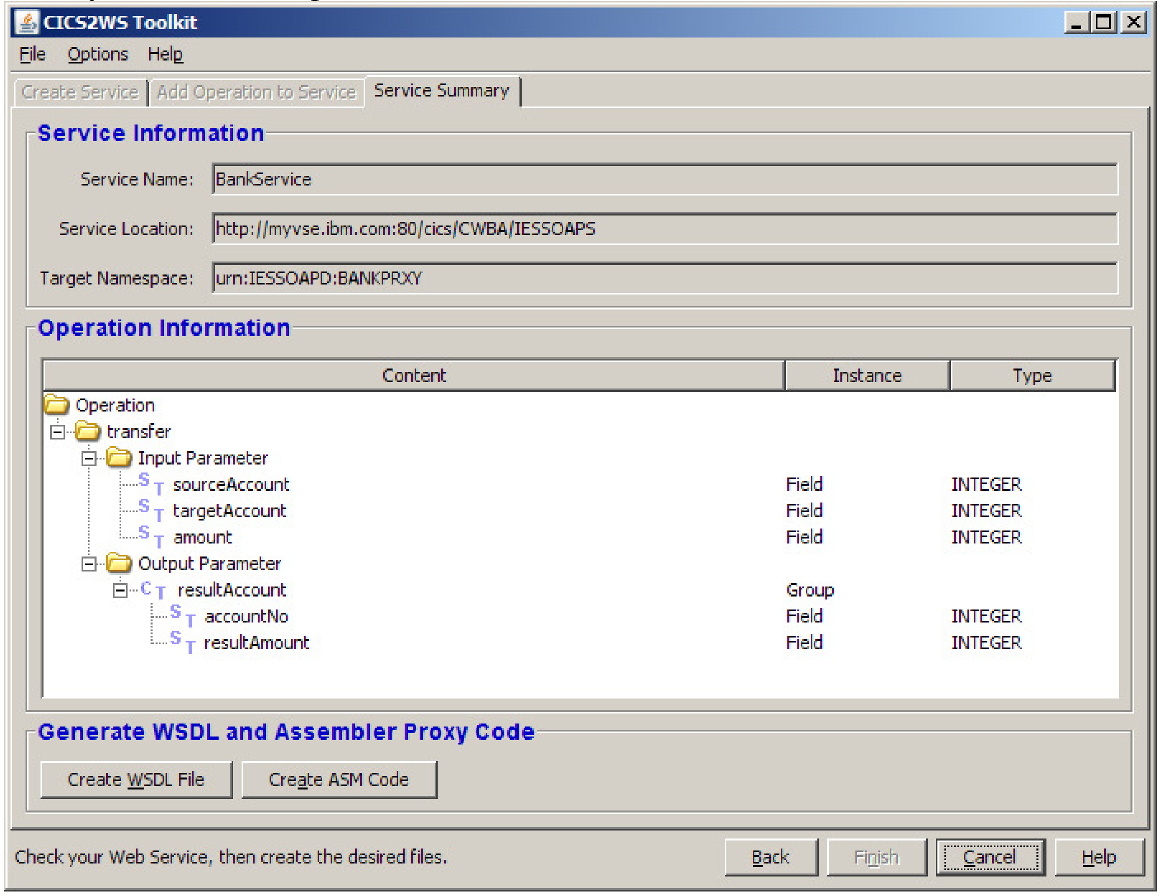
You specify the name of the operation you want to provide and a description (optional).

In the upper table you see the contents of your copybook that has been imported on the previous panel. You need to tell CICS2WS which fields are to be used as input parameters for the service, and which fields are output parameters (or both). First, select a field or group of fields, then select either the "Input Parameter" or "Output Parameter" node in the table below and press the "Add" button. You can now change the name of the parameter or leave it as it is (Note that a parameter name can only have up to 16 characters, unless you are using long-name to short-name mapping as described in Appendix E). Repeat this for every field that you want to use with this service.

Some of the fields may be marked in red or orange. Orange indicates that some parameter within the group is marked red or that the group is a redefine. Read means that you need to perform an action before you can continue to use this operation. In most cases it indicates that the fields name is longer than 16 characters. Right click the field and choose the desired action (e.g. rename the field).

The table on the very bottom of this panel shows the external view of the service you are providing. All names used there will be visible in the WSDL and to the caller of this Web Services.

When you are finished, press the "Next" button.



Service Information

Service Name: BankService

Service Location: http://myvse.ibm.com:80/cics/CWBA/IESSOAPS

Target Namespace: urn:IESSOAPD:BANKPRXY

Operation Information

Content	Instance	Type
Operation		
transfer		
Input Parameter		
sourceAccount	Field	INTEGER
targetAccount	Field	INTEGER
amount	Field	INTEGER
Output Parameter		
resultAccount	Group	
accountNo	Field	INTEGER
resultAmount	Field	INTEGER

Generate WSDL and Assembler Proxy Code

Check your Web Service, then create the desired files.

On the last panel, you see a summary of all the settings you have done so far. Please double check it. You can go back and correct them if needed.

Press the “Create WSDL File” button to create the WSDL file for this service. The WSDL will describe the Web Service, so that an external user has all the information needed to invoke this Web Service.

Press the “Create ASM Code” button to generate the proxy code used for this service. The name of the proxy code is what you have specified on the first panel in field “Proxy name”.

Remember the location of the generated files (WSDL and proxy code); you will need them later on.

Assemble and define the proxy code to CICS

The CICS2WS Toolkit has now generated the proxy code for you. The proxy code is a CICS/assembler program. You need to assemble it and define it to CICS.

Together with the proxy code source, a compile job has been generated that can be used to assemble the proxy code. You need to upload the proxy code source to your z/VSE system. You can do this using standard file transfer functions like FTP. Adapt the compile job and use it to assemble the proxy code. You need to change the libraries used for cataloging the phase and the LIBDEFs. Check for a clean compile (i.e. return code zero). Make sure you place the phase of the proxy code into a library that is in the CICS LIBDEF.

You need to define the proxy program to CICS using CEDA DEFINE PROGRAM. The language is Assembler. The “DATALOCATION” parameter can be set to ANY; the “EXECKey” parameter should be set to User:

```
CEDA  DEFine PROGram( MYPROXY  )
 PROGram        : MYPROXY
 Group          : VSESPG
 DEscription  ==>
 Language     ==> Assembler          CObol | Assembler | C | Pli
 RELoad       ==> No                 No | Yes
 RESident     ==> No                 No | Yes
 USAge        ==> Normal             Normal | Transient
 USEsvacopy   ==> No                 No | Yes
 Status       ==> Enabled            Enabled | Disabled
 RSl            : 00                 0-24 | Public
 Cedf         ==> Yes                Yes | No
 DAtalocation ==> Any                Below | Any
 EXECKey      ==> User               User | Cics
REMOTE ATTRIBUTES
 REMOTESystem ==>
 REMOTEName   ==>
 Transid      ==>
```

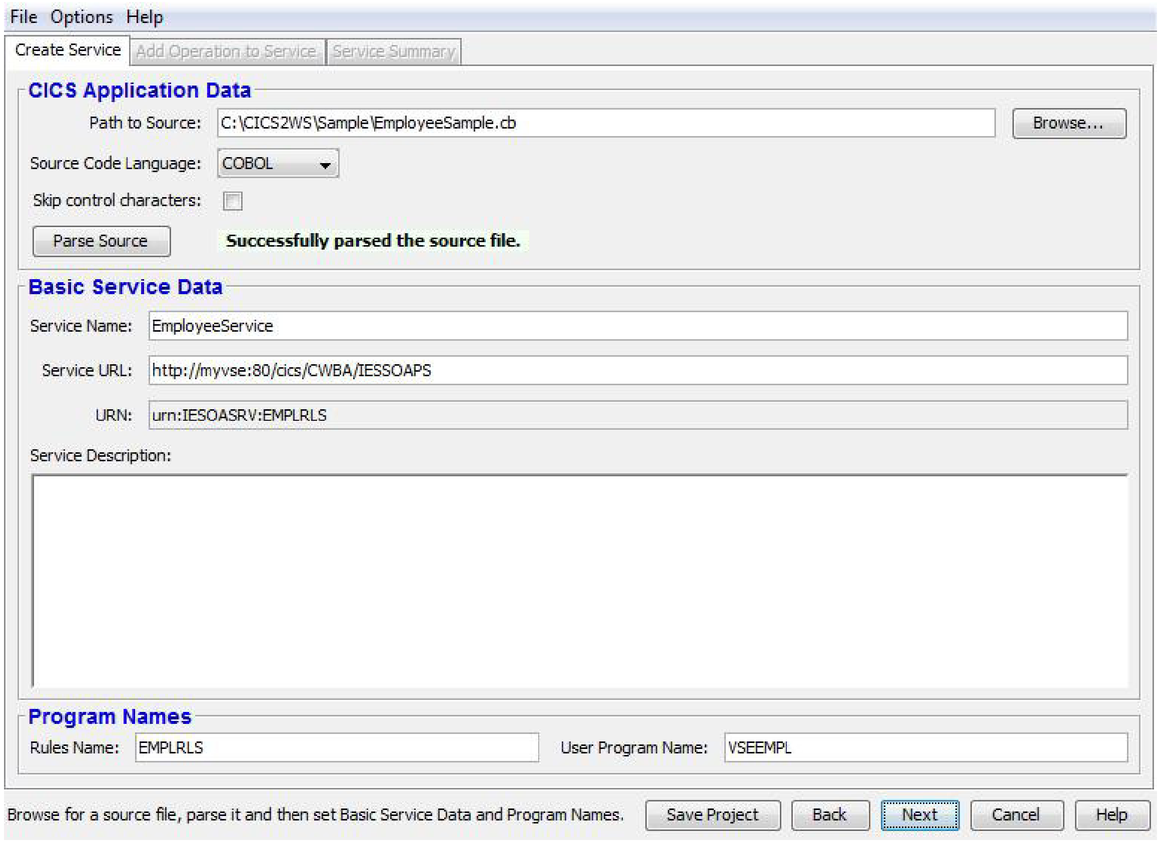
After defining the program, you need to install it using CEDA INSTALL, or restart CICS.

2.5. *Create a Web Service from a CICS Application with literal support (z/VSE 5.2 or higher)*

By choosing upper button "… with literal support" CICS2WS Tool will provide with mapping rules and wsdl file. In this case copybook can have arrays with fix or variable length. At the end Tool will give possibility to chose type of wsdl: encoded or literal (see Appendix A for more information about encoding styles).

Import Copybook and create the Web Service to provide

In this step, we use the Copybook2 as described in chapter 2.3 and import it into the CICS2WS Toolkit. CICS2WS can read copybooks in COBOL, PL/1 and Assembler language. You may need to download the copybook from z/VSE in order to import it into the CICS2WS Toolkit. You can download the copybook using standard file transfer functions, like FTP.



Path to Source: C:\CICS2WS\Sample\EmployeeSample.cb

Source Code Language: COBOL

Skip control characters: ☐

Successfully parsed the source file.

Service Name: EmployeeService

Service URL: http://myvse:80/cics/CWBA/IESSOAPS

URN: urn:IESOASRV:EMPLRLS

Service Description:

Rules Name: EMPLRLS

User Program Name: VSEEMPL

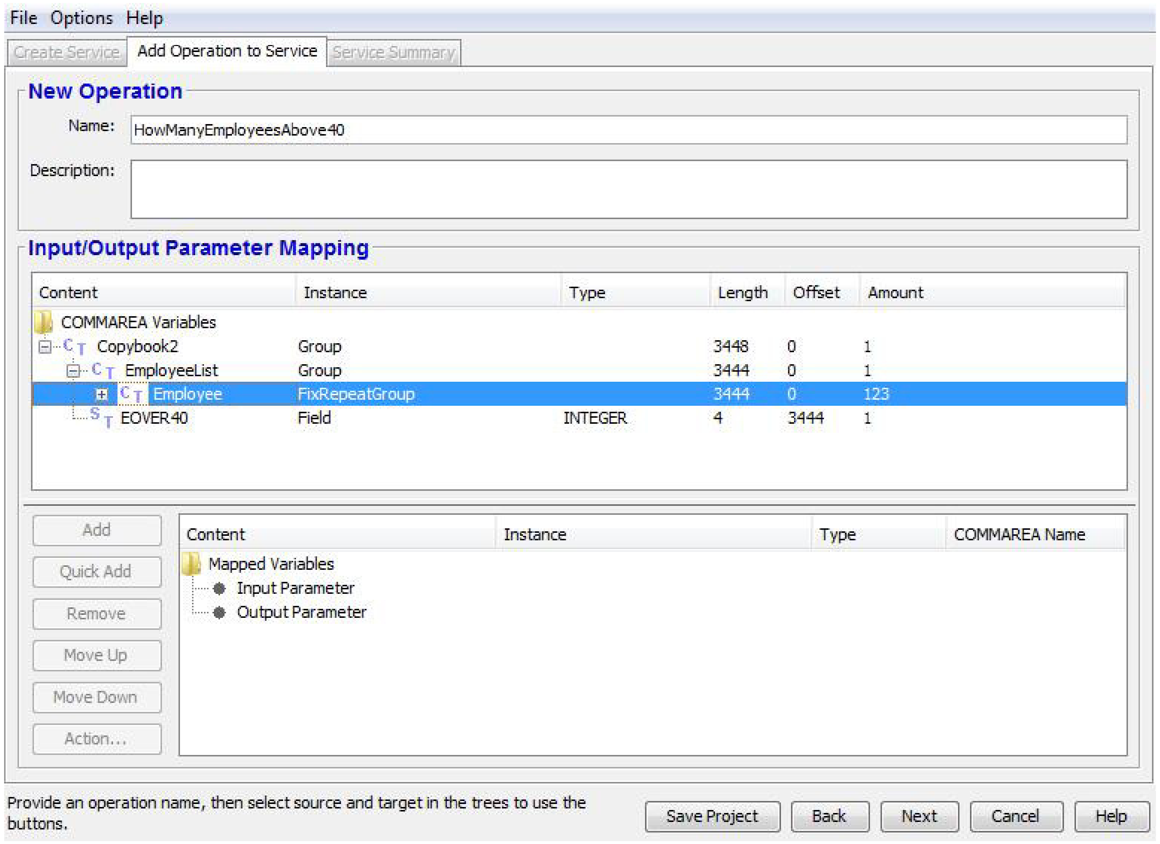
Browse for a source file, parse it and then set Basic Service Data and Program Names.

In CICS2WS, press the "Browse…" button and choose the file that contains the copybook. The source code language is automatically detected based on the file extension. If it did not detect it correctly, choose the language manually. Then press the "Parse" button. In case you receive any errors, check the copybook source and correct the errors.

Once the copybook was imported successfully, you need to provide some basic information about the Web Service you want to provide. This includes the name of the service and the URL, where it will be hosted later on. The URL has a fixed format; you should only change the "server-name" and "port". The server-name is the hostname or IP address of your VSE system that will host the service. The port is the TCP/IP port you defined in the TCPIPSERVICE in chapter 2.2

Last, you specify the name of the mapping rules that is to be generated, and the name of the program that implements the service and is to be web service enabled.

Press the "Next" button to continue to the next page.



Name: HowManyEmployeesAbove40

Description:

Content	Instance	Type	Length	Offset	Amount
COMMAREA Variables					
Copybook2	Group		3448	0	1
EmployeeList	Group		3444	0	1
Employee	FixRepeatGroup		3444	0	123
EOVER40	Field	INTEGER	4	3444	1

Content	Instance	Type	COMMAREA Name
Mapped Variables			
Input Parameter			
Output Parameter			

Provide an operation name, then select source and target in the trees to use the buttons.

You now define the Web Service operation (also called method) that you want to provide and define what input and output parameters it has.

Note: In general a Web Service can provide multiple operations. However, CICS2WS only supports one operation per Web Service. If you want to provide multiple operations, you need to create multiple Web Services.

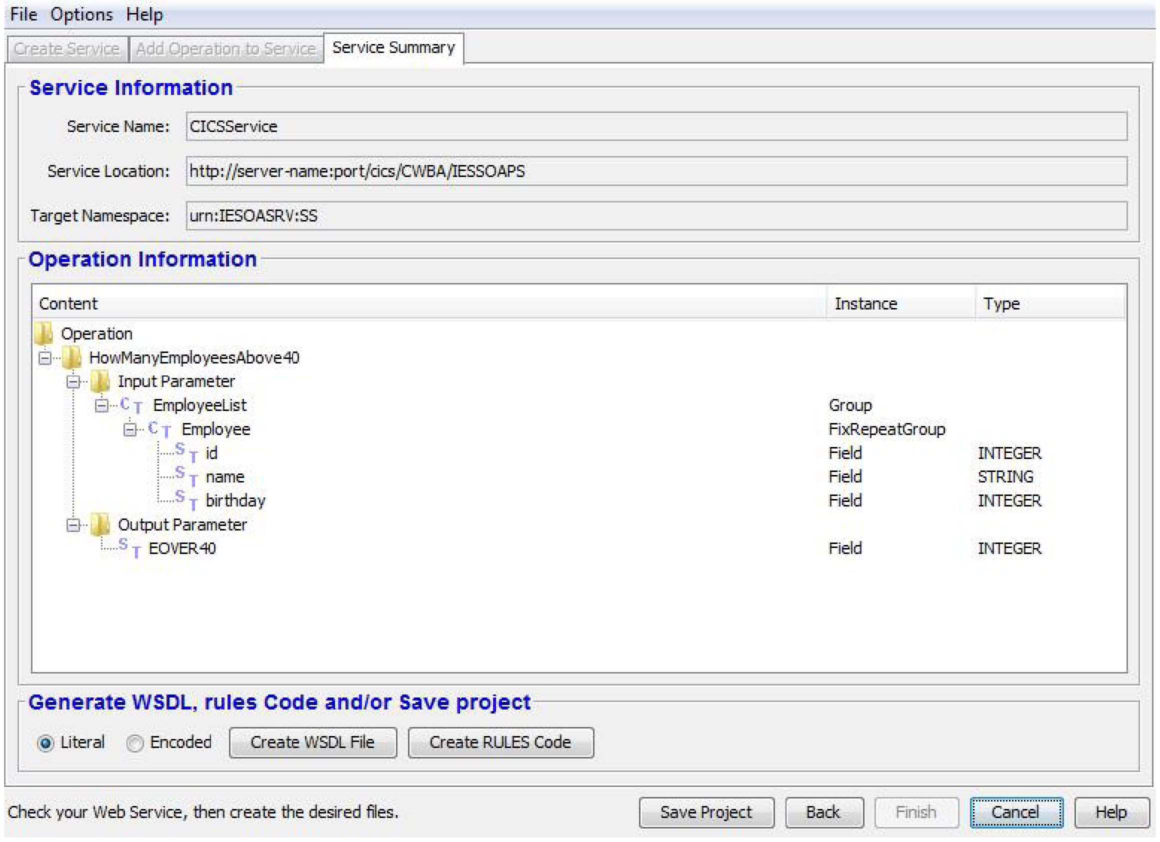
You specify the name of the operation you want to provide and a description (optional).

In the upper table you see the contents of your copybook that has been imported on the previous panel. You need to tell CICS2WS which fields are to be used as input parameters for the service, and which fields are output parameters (or both). First, select a field or group of fields, then select either the "Input Parameter" or "Output Parameter" node in the table below and press the "Add" button. You can now change the name of the parameter or leave it as it is (Note that a parameter name can only have up to 16 characters, unless you are using long-name to short-name mapping as described in Appendix E). Repeat this for every field that you want to use with this service.

Some of the fields may be marked in red or orange. Orange indicates that some parameter within the group is marked red or that the group is a redefine. Read means that you need to perform an action before you can continue to use this operation. In most cases it indicates that the fields name is longer than 16 characters. Right click the field and choose the desired action (e.g. rename the field).

The table on the very bottom of this panel shows the external view of the service you are providing. All names used there will be visible in the WSDL and to the caller of this Web Services.

When you are finished, press the "Next" button.



Service Name: CICSService

Service Location: http://server-name:port/cics/CWBA/IESSOAPS

Target Namespace: urn:IESOASRV:SS

Content	Instance	Type
Operation		
HowManyEmployeesAbove40		
Input Parameter		
EmployeeList	Group	
Employee	FixRepeatGroup	
id	Field	INTEGER
name	Field	STRING
birthday	Field	INTEGER
Output Parameter		
EOVER40	Field	INTEGER

[✓ Literal] Encoded

Check your Web Service, then create the desired files.

On the last panel, you see a summary of all the settings you have done so far. Please double check it. You can go back and correct them if needed.

Press the “Create WSDL File” button to create the WSDL file for this service. The WSDL will describe the Web Service, so that an external user has all the information needed to invoke this Web Service.

Press the “Create RULES Code” button to generate the rules code used for this service. The name of the rules code is what you have specified on the first panel in field “Rules name”.

Note. Choice literal/encoded should be the same for wsdl and mapping rules.

Remember the location of the generated files (WSDL and rules code); you will need them later on.

Assemble and define the rules code to CICS

The CICS2WS Toolkit has now generated the rules code for you. The rules code is a CICS/assembler program containing just static data, but no executable code. You need to assemble it and define it to CICS.

Together with the rules code source, a compile job has been generated that can be used to assemble the rules code. You need to upload the rules code source to your z/VSE system. You can do this using standard file transfer functions like FTP. Adapt the compile job and use it to assemble the rules code. You need to change the libraries used for cataloging the phase and the LIBDEFs. Check for a clean compile (i.e. return code zero). Make sure you place the phase of the rules code into a library that is in the CICS LIBDEF.

You need to define the rules program to CICS using CEDA DEFINE PROGRAM. The language is Assembler. The “DATALOCATION” parameter can be set to ANY; the “EXECKey” parameter should be set to User:

```
CEDA  DEFine PROGram( MYPROXY  )
 PROGram         : MYRULES
 Group           : VSESPG
 DEscription  ==>
 Language     ==> Assembler           CObol | Assembler | C | Pli
 RELoad       ==> No                  No | Yes
 RESident     ==> No                  No | Yes
 USAge        ==> Normal              Normal | Transient
 USEsvacopy   ==> No                  No | Yes
 Status       ==> Enabled             Enabled | Disabled
 RSl             : 00                 0-24 | Public
 Cedf         ==> Yes                 Yes | No
 DAtalocation ==> Any                 Below | Any
 EXECKey      ==> User                User | Cics
REMOTE ATTRIBUTES
 REMOTESystem ==>
 REMOTEName   ==>
 Transid      ==>
```

After defining the program, you need to install it using CEDA INSTALL, or restart CICS.

2.6. Test the Web Service

When the proxy code has been created and define to CICS, you should test the newly created Web Service. The test uses the generated WSDL to invoke the Web Service.

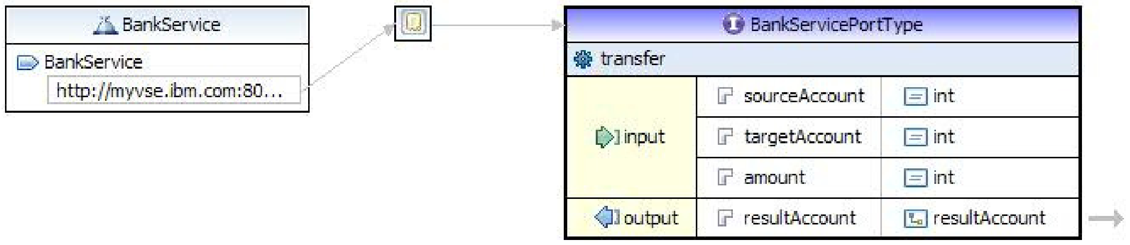
There are different Web Service test tools available. This article discusses the use of IBM Rational Application Developer (RAD) and the SoapUI Tool (www.soapui.org).

IBM Rational Application Developer

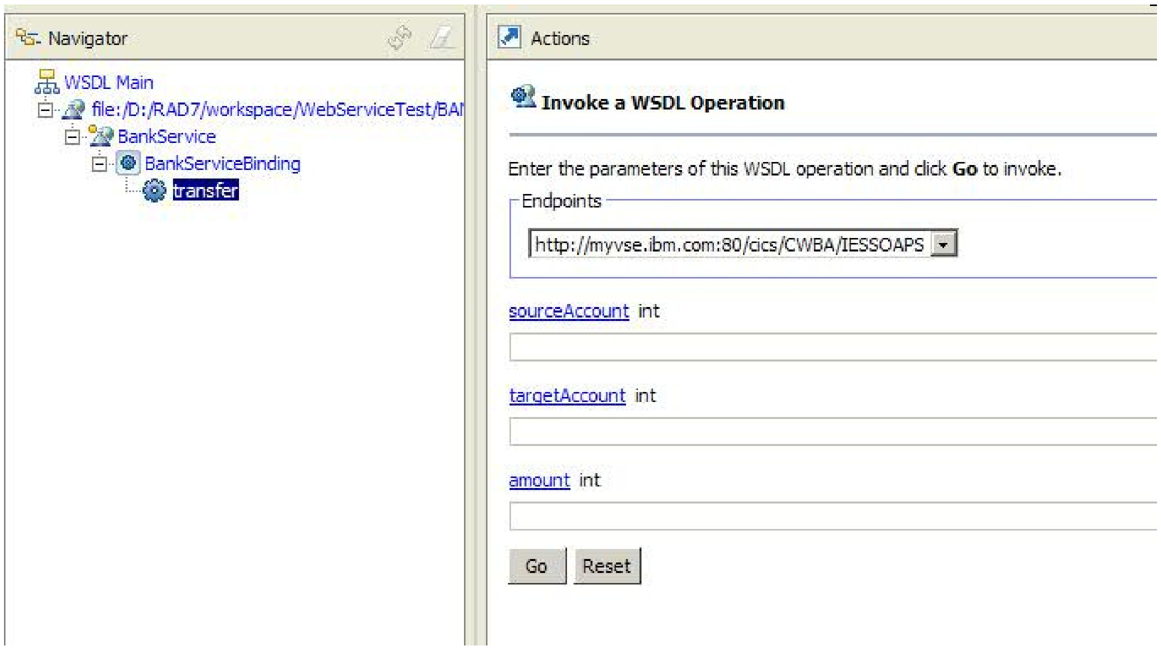
The IBM Rational Application Developer (RAD) is a priced product. It is based on the Eclipse platform and allows you to perform various development tasks. For more information about RAD, please see here:
http://www.ibm.com/software/products/en/application

It also provides a Web Service test function, called the "Web Service Explorer" which can also be used to test Web Service provided by z/VSE.

Create a new project in RAD and import the generated WSDL into the project. With a double click on the WSDL file, you can open the WSDL editor. It shows a graphical view of the WSDL definitions, as well as a source view.



To test the Web Service, right click the WSDL file in the project, choose “Web Services” → “Test with Web Service Explorer”. Choose the operation to execute in the Navigator tree. You can then specify the values for the input parameters in the right part of the window and press the “Go” button to invoke the Web Service.



Enter the parameters of this WSDL operation and click **Go** to invoke.

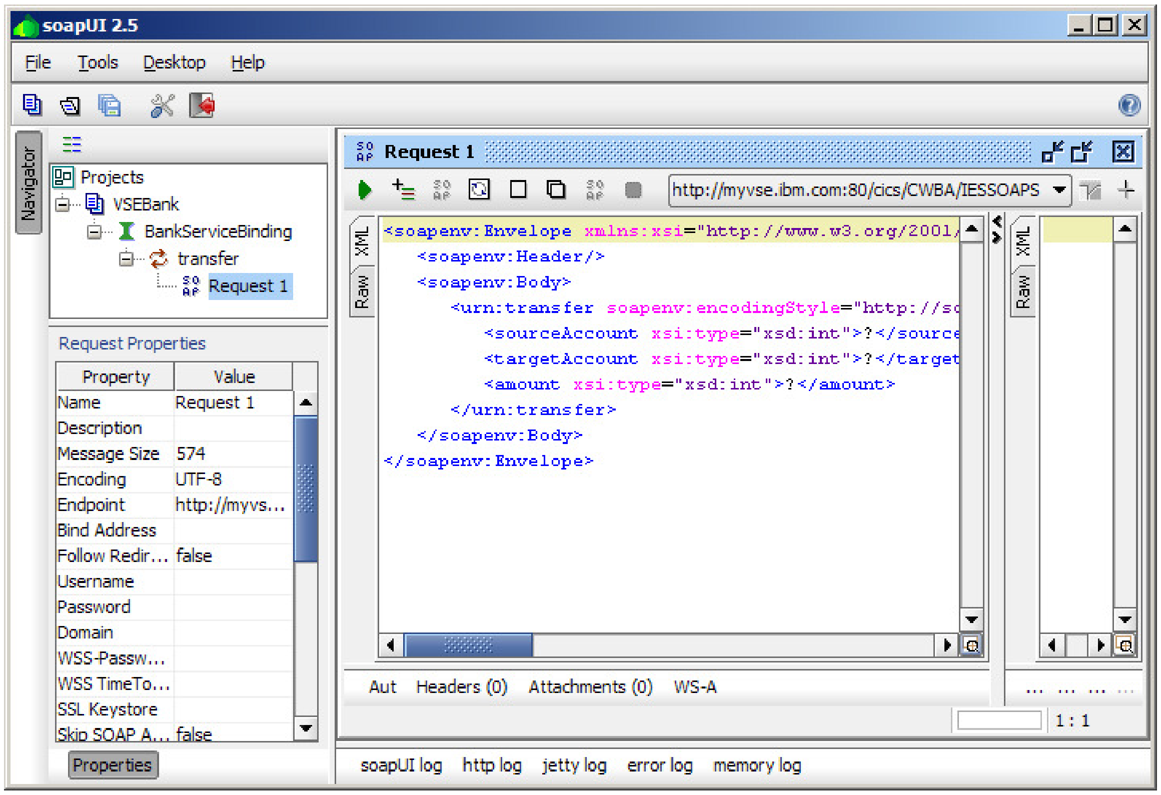
http://myvse.ibm.com:80/cics/CWBA/IESSOAPS

It will then connect to z/VSE and send a SOAP message through HTTP. The response is also displayed in the right part of the window. You can also switch to the “Source” view, where you see the plain SOAP Message. This can be helpful when problems occur and you want to check the exact SOAP message.

SoapUI

The SopaUI Tool is an open source tool; you can download a free version at http://www.soapui.org/. Its primary focus is on testing Web Services, but it also can generate Web Service clients for various environments.

To test the generated WSDL, you need to create a new project in SoapUI, and load the WSDL file into it. You can then select the method to invoke in the Navigator tree, Double click on the operation to create a Request view.



```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/
   <soapenv:Header/>
   <soapenv:Body>
      <urn:transfer soapenv:encodingStyle="http://sc
         <sourceAccount xsi:type="xsd:int">?</source
         <targetAccount xsi:type="xsd:int">?</target
         <amount xsi:type="xsd:int">?</amount>
      </urn:transfer>
   </soapenv:Body>
</soapenv:Envelope>
```

Property	Value
Name	Request 1
Description	
Message Size	574
Encoding	UTF-8
Endpoint	http://myvs...
Bind Address	
Follow Redir...	false
Username	
Password	
Domain	
WSS-Passw...	
WSS TimeTo...	
SSL Keystore	
Skip SOAP A...	false

You will see the SOAP Request on the right side. It contains question marks for each input parameter. You can overtype the question marks with the input values. Then press the green arrow icon to start the request. You will see the response SOAP Message in the right side of the Request window.

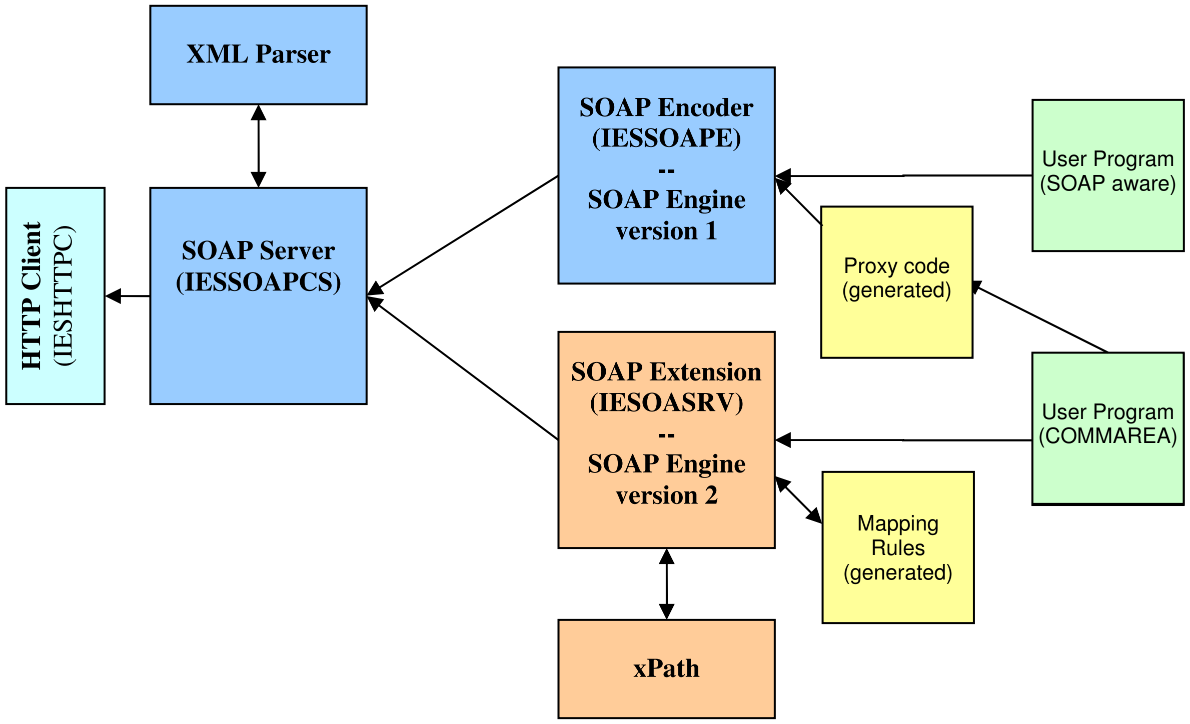
3. Using Web Services with z/VSE – z/VSE acts as SOAP client

This chapter describes step by step how to setup CICS TS and the z/VSE SOAP Engine and how to call an external Web Service from a CICS program.

Some of the steps below are setup tasks that only have to be done once, when you call your very first web service enable. Other steps are tasks that you must repeat for every Web Service that you want to call.

3.1. Overview

The figure below shows the modules that are involved when a Web Service request is processed.


XML Parser
HTTP Client (IESHTTPC)
SOAP Server (IESSOAPCS)
SOAP Encoder (IESSOAPE) -- SOAP Engine version 1
SOAP Extension (IESOASRV) -- SOAP Engine version 2
xPath
Proxy code (generated)
Mapping Rules (generated)
User Program (SOAP aware)
User Program (COMMAREA)

The SOAP Client part of the z/VSE SOAP Engine does not require CICS Web Support. The z/VSE SOAP Engine comes with all required parts, including a HTTP Client.

When a user program wants to call an external web service, it calls the client part of the z/VSE SOAP Engine. The user program:

- can be either a program that is SOAP aware (i.e. it uses the z/VSE SOAP Engines programming interface directly),
- can call a proxy program, that has been generated by the CICS2WS Toolkit for use with z/VSE SOAP Engine version 1,
- can call SOAP Engine version 2 with providing a mapping rules, that has been generated by the CICS2WS Toolkit (starting z/VSE 5.2).

The Proxy program or mapping Rules then translates the input parameters from a COMMAREA into a SOAP specific format and calls the z/VSE SOAP Engine.

The SOAP processing starts with the SOAP Encoder program (IESSOAPE/IESOACLN). The SOAP encoder serializes the Web Service's input parameters from its z/VSE specific representation into a XML representation and calls the SOAP Client program (IESSOAPC).

The SOAP Client program generates a SOAP message containing the web service's input parameters, the method to call, as well as a SOAP Body and Envelope. It then calls the HTTP Client program with the Web Service's URL.

The HTTP Client opens a TCP connection to the server denoted in the URL and sends a HTTP request together with the XML data containing the SOAP message. It receives the HTTP response, and passes it back to the SOAP client program.

The SOAP client program parses the XML data, analyses the SOAP response message and passes the Web Service's output parameters to the SOAP encoder program. The output parameters are then deserialized and passed back to the user program or proxy/rules code.

3.2. Setup CICS and TCP/IP

There is no special CICS setup required to call an external Web Service. However, you need to setup a few things to allow the z/VSE SOAP Engine to use TCP/IP sockets to connect external servers.

1. URLs usually use hostnames rather than IP addresses. Therefore, the HTTP Client needs to resolve the hostname before it can connect to the server. To do so, you need to configure TCP/IP to use a domain name server. For TCP/IP for VSE you do that by specifying SET DNS1=n.n.n.n in the TCP/IP configuration. If you forget this step, you usually get a return code 1005 when calling IESSOAPE.
2. You need to adjust the CICS startup job, to specify the ID of the TCP/IP stack with which CICS should work. You specify the ID using the // OPTION SYSPARM statement (e.g. // OPTION SYSPARM=nn).
3. Make sure that your z/VSE system is able and allowed to connect to the server denoted in the Web Service's URL. You may need to adapt your networks firewall rules.

3.3. *Requirements for the Web Service to be used with CICS2WS*

To call an external Web Service, you need to know some information about the Web Service:

- Where is it located (Web Service's URL)
- Name of the service and method
- What input parameters is it expecting
- What output parameters it is returning

All of above information is usually contained in a WSDL file (WSDL: Web Service Description Language). A WSDL file is an XML file. It contains all information that a Web Service consumer (i.e. user or client) needs to know to be able to call the Web Services. Using the definitions in the WSDL file, a development environment that supports Web Services can generate all required programs. For z/VSE, we use the CICS2WS Toolkit to generate the proxy/rules code.

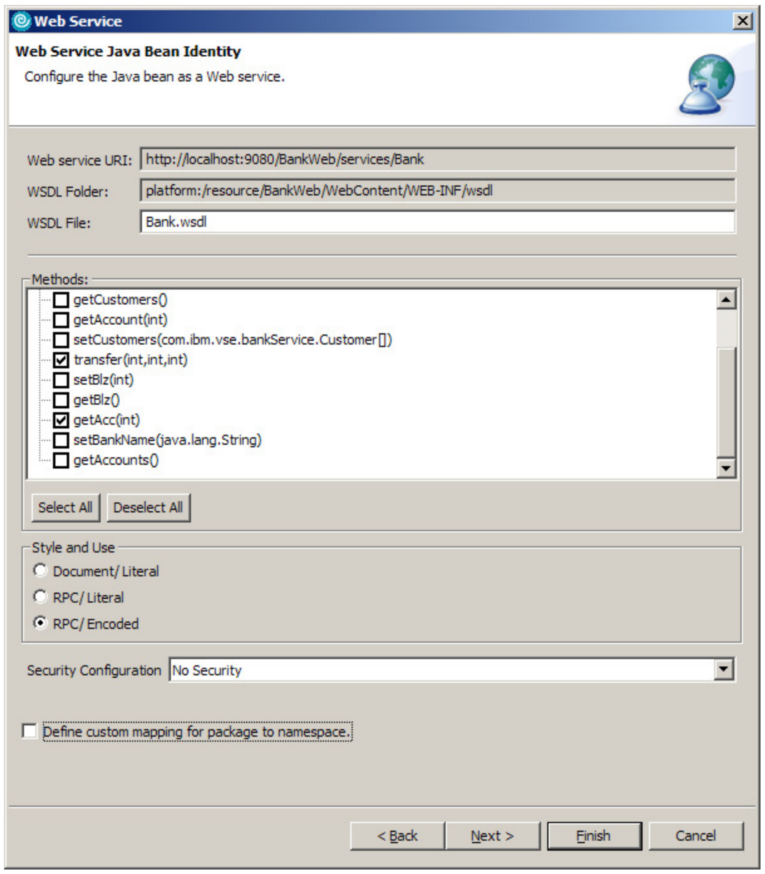
The CICS2WS Toolkit only supports Web Services that use RPC style with SOAP encoded messages (also see Appendix A:), or literal style messages (starting with z/VSE 5.2). CICS2WS will check the definitions in the WSDL file during import and will show error messages if the requirements are not fulfilled. The external Web Service must meet the following requirements to work with CICS2WS:

- It must use the SOAP 1.1 protocol.
- It must use SOAP encoding (use="encoded"), in case of using proxy program
- It can use SOAP literal style (use="literal") only starting z/VSE 5.2 and by using rules program.
- It must use RPC style. Document style is not supported.
- In case of using z/VSE SOAP Engine version 1 parameter names must not be larger than 16 characters (you may use the name mapping as described in Appendix E, but this requires some manual changes in the WSDL and the generated proxy code)

Please also see Appendix A for more information about Literal vs. Encoded and RPC- vs. document-style and how the z/VSE SOAP Engine supports it.

Using IBM Rational Application Developer to create a Web Service

If you are using IBM Rational Application Developer to create a Web Service that is to be called by z/VSE, you need to select "RPC/encoded" for the "Style and Use" when generating the WSDL file:



Web Service

Web Service Java Bean Identity

Configure the Java bean as a Web service.

Web service URI: http://localhost:9080/BankWeb/services/Bank

WSDL Folder: platform:/resource/BankWeb/WebContent/WEB-INF/wsdl

WSDL File: Bank.wsdl

Methods:
- [ ] getCustomers()
- [ ] getAccount(int)
- [ ] setCustomers(com.ibm.vse.bankService.Customer[])
- [x] transfer(int,int,int)
- [ ] setBlz(int)
- [ ] getBlz()
- [x] getAcc(int)
- [ ] setBankName(java.lang.String)
- [ ] getAccounts()

Style and Use: Document/ Literal RPC/ Literal [✓ RPC/ Encoded]

Security Configuration: No Security

- [ ] Define custom mapping for package to namespace.

Using Microsoft .Net with WCF to create a Web Service (encoded only)

If you are using Microsoft .Net with WCF (Windows Communication Foundation) to create a Web Service that is to be called by z/VSE, the following information should be helpful:

When creating the ServiceContract for a WCF service, you can specify whether to use RPC/encoded or document/literal by including the following in the ServiceContract attribute:

```
[ServiceContract,
 XmlSerializerFormat(Style = OperationFormatStyle.Rpc,
 Use = OperationFormatUse.Encoded)]
```

To use SOAP 1.1 instead of SOAP 1.2, a configuration change must be made in the web.config file. By default, all endpoint bindings are set to 'wsHttpBinding'. Changing the binding to 'basicHttpBinding' and specifying an address of 'soap11' allows you to use SOAP 1.1.

When generating the WSDL file, these changes are taken into consideration and automatically added into the file.

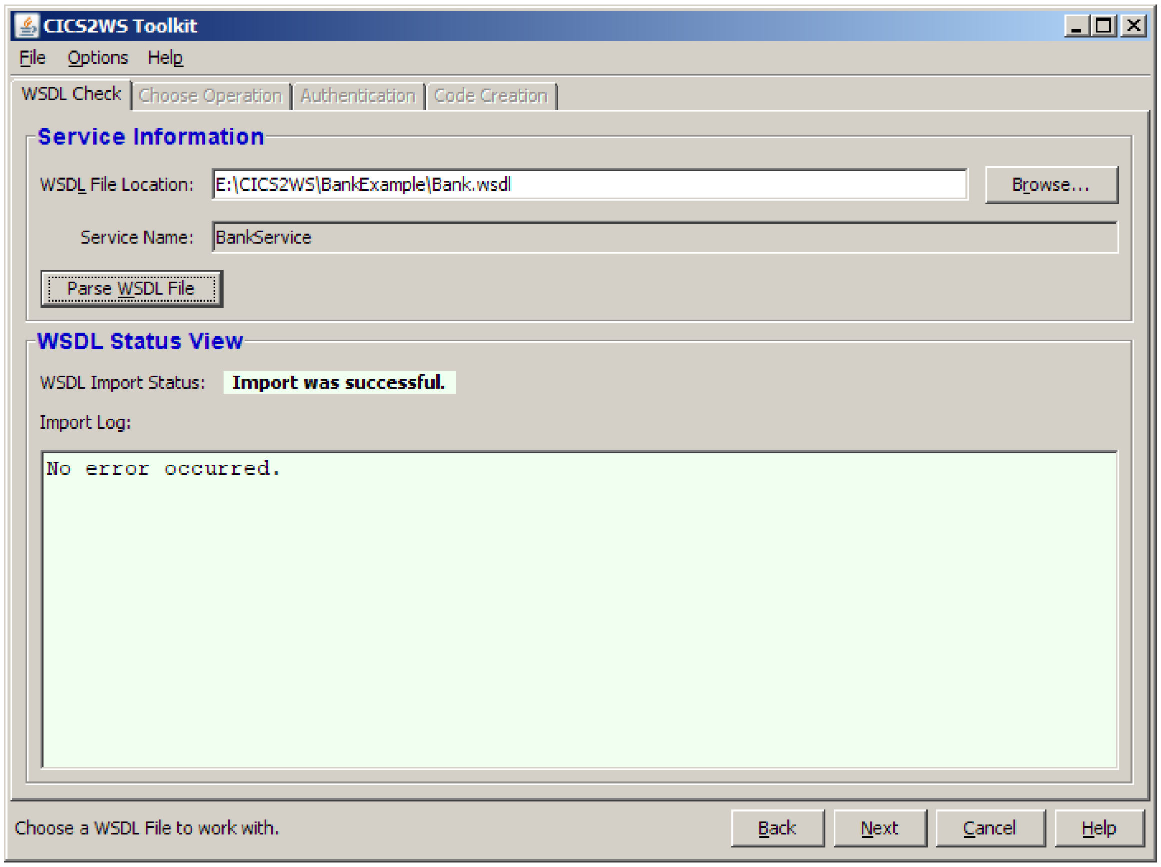
For z/VSE as a SOAP Client, we will use buttons “Use a Web Service with a CICS Application” and “… with literal support (z/VSE 5.2 or higher)” in the next steps.

3.4. Use a Web Service with a CICS Application

To work with SOAP engine version 1 button “Use a Web Service with a CICS Application” should be chosen. In this case if wsdl file has an array or literal encoding CICS2WS will identify it as an error.

Import WSDL and generate Proxy Code & Copybook

As first step, you import the WSDL file into CICS2WS. Press the “Browse” button to locate the WSDL file. Then press the “Parse WSDL file” button to parse the WSDL file. Look for errors in the WSDL Status View below.



Service Information

WSDL File Location: E:\CICS2WS\BankExample\Bank.wsdl

Service Name: BankService

WSDL Status View

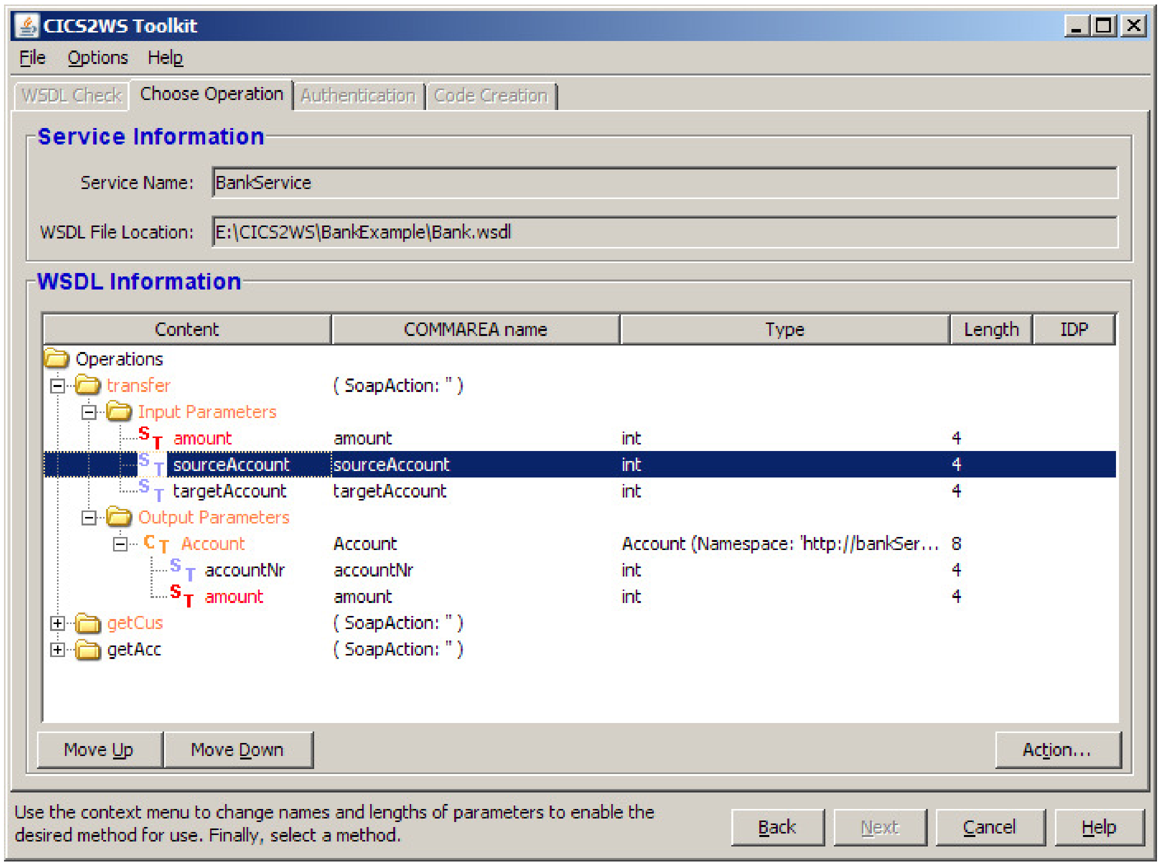
WSDL Import Status: **Import was successful.**

Import Log:

```
No error occurred.
```

Choose a WSDL File to work with.

If the WSDL import worked without errors, press the “Next” button to continue.



Service Name: BankService

WSDL File Location: E:\CICS2WS\BankExample\Bank.wsdl

Content	COMMAREA name	Type	Length	IDP
Operations				
transfer	(SoapAction: '')			
Input Parameters				
amount	amount	int	4	
sourceAccount	sourceAccount	int	4	
targetAccount	targetAccount	int	4	
Output Parameters				
Account	Account	Account (Namespace: 'http://bankSer...	8	
accountNr	accountNr	int	4	
amount	amount	int	4	
getCus	(SoapAction: '')			
getAcc	(SoapAction: '')			

Use the context menu to change names and lengths of parameters to enable the desired method for use. Finally, select a method.

On this panel, you must choose which operation of the Web Service (also called method) you want to call. In our banking example, the Web Service provides 3 operations:

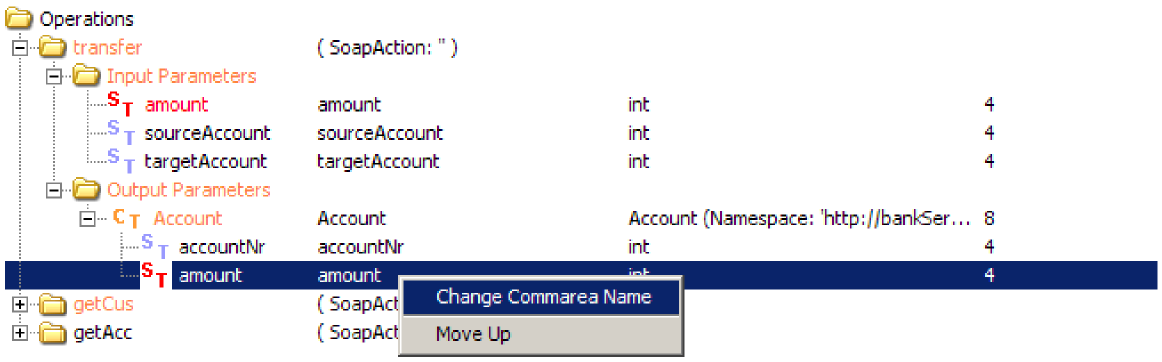
- **transfer**: transfer money from one bank account to another account
- **getCus**: get customer information
- **getAcc**: get account information

Underneath the operations, you see which input parameter and which output parameters the operation uses. Some of the parameters may be marked in red or orange. Orange indicates that some parameter within the group is marked red. Read means that you need to perform an action before you can continue to use this operation.

In the example above, the parameter “amount” appears as input parameter as well as output parameter. That’s why it is marked in red.

Every Web Service parameter will have its corresponding field in the generated COMMAREA. Per default, the COMMAREA field gets the same name as the parameter in the Web Service operation. You can change the name of the COMMAREA field. Right click the parameter and choose “Change Commarea Name”, then type in the new name.

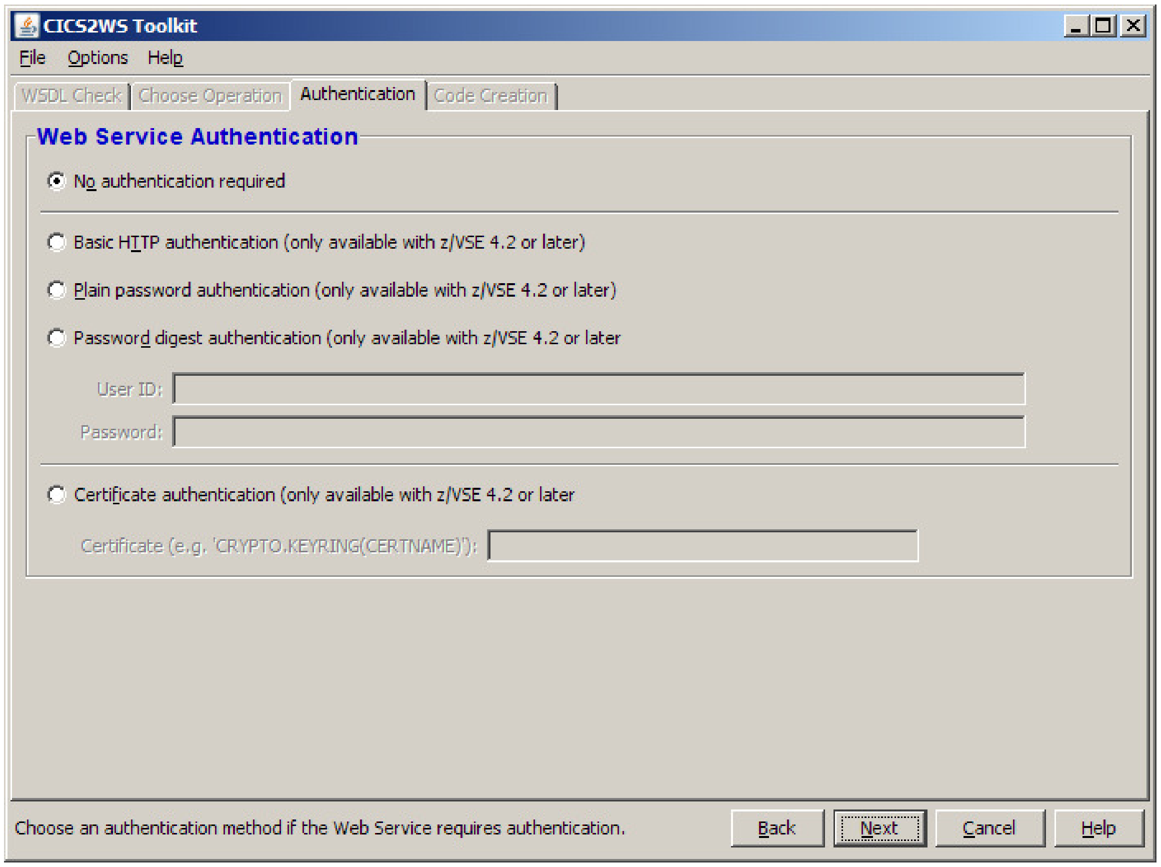
The names appearing on the left side represent the external view; the COMMREA name represents the name under which the parameter will be known in the generated proxy code and the copybook you use to call the proxy code later on.



```
Operations
  transfer                    ( SoapAction: '' )
    Input Parameters
      amount                  amount          int                                   4
      sourceAccount           sourceAccount   int                                   4
      targetAccount           targetAccount   int                                   4
    Output Parameters
      Account                 Account         Account (Namespace: 'http://bankSer…  8
        accountNr             accountNr       int                                   4
        amount                amount          int                                   4
  getCus                      ( SoapAct
  getAcc                      ( SoapAct
```

In our case, we need to change the COMMAREA name of one of the duplicate “amount” parameters. Here, we call it “resultAmount” in the output parameters. Once all problems have been resolved, the operation is no longer marked in red or orange.

Select the operation you want to generate a proxy code for (i.e. the “transfer” operation) and press the “Next” button. You can only press the “Next” button if all problems of the chosen operation have been resolved.



Web Service Authentication

[● No authentication required]

○ Basic HTTP authentication (only available with z/VSE 4.2 or later)

○ Plain password authentication (only available with z/VSE 4.2 or later)

○ Password digest authentication (only available with z/VSE 4.2 or later

User ID:

Password:

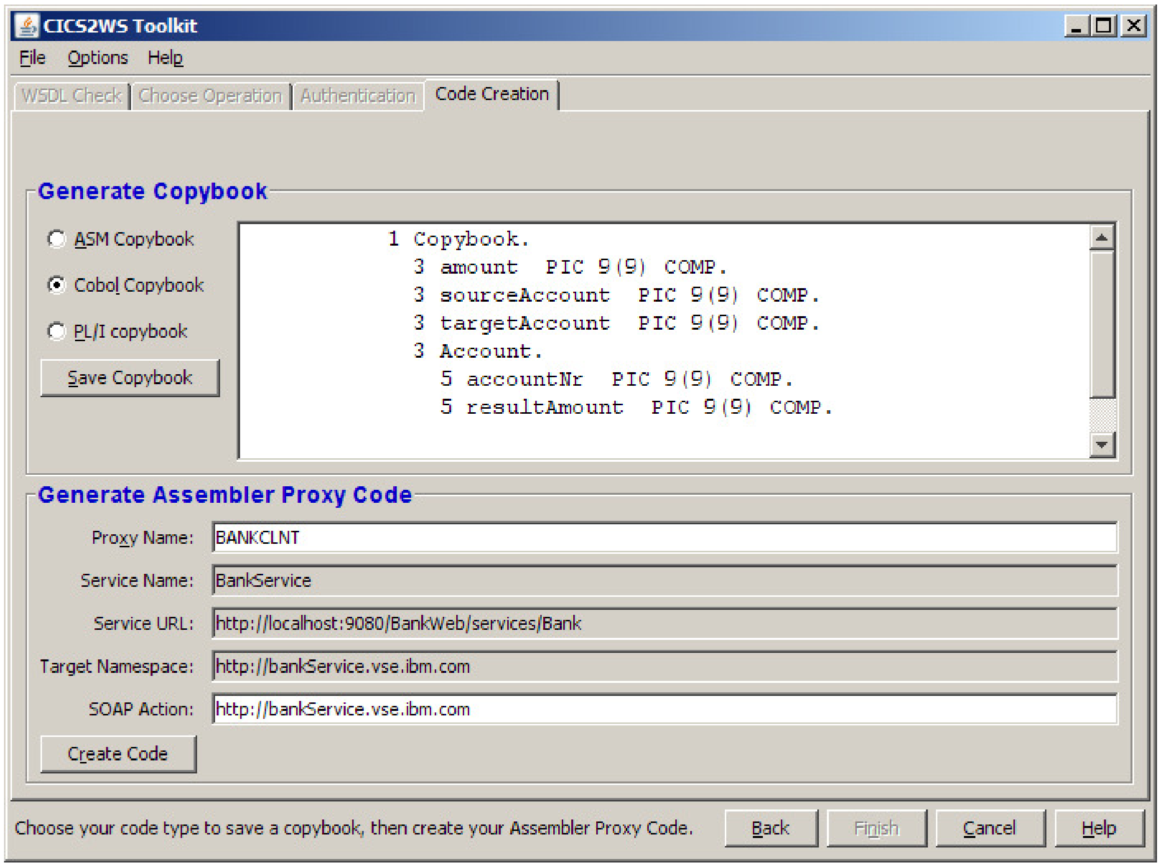
○ Certificate authentication (only available with z/VSE 4.2 or later

Certificate (e.g. 'CRYPTO.KEYRING(CERTNAME)'):

Choose an authentication method if the Web Service requires authentication.

On this panel, you can choose if you want to use Web Service authentication. Authentication is only supported on z/VSE 4.2 and onwards.

Press the “Next” button to continue.



( ) ASM Copybook
(●) Cobol Copybook
( ) PL/I copybook

```
1 Copybook.
  3 amount  PIC 9(9) COMP.
  3 sourceAccount  PIC 9(9) COMP.
  3 targetAccount  PIC 9(9) COMP.
  3 Account.
    5 accountNr  PIC 9(9) COMP.
    5 resultAmount  PIC 9(9) COMP.
```

Proxy Name:	BANKCLNT
Service Name:	BankService
Service URL:	http://localhost:9080/BankWeb/services/Bank
Target Namespace:	http://bankService.vse.ibm.com
SOAP Action:	http://bankService.vse.ibm.com

Choose your code type to save a copybook, then create your Assembler Proxy Code.

Now you are ready to generate the proxy code and the copybook for the chosen Web Service operation.

The copybook can be generated in Assembler, COBOL or PL/1 language. Choose the desired language and press the “Save Copybook” button to save the generated copybook. You will see a preview of the copybook in the text area on the right.

As you see, the copybook will use the COMMAREA names as specified on the previous panel (see “resultAmount”). The data type of each field has been mapped to the closest data type available in the selected programming language. In our example, all fields are using “int” (4 byte binary integer number), which corresponds to a PIC 9(9) in COBOL.

You will later on use the generated copybook to call (EXEC CICS LINK) the generated proxy code.

To generate the proxy code, enter the name of the proxy code ("Proxy Name") and press the "Create Code" button. This will generate the proxy program source code. Remember the location of the generated files (copybook and proxy code); you will need them later on.

Assemble and define the proxy code to CICS

The CICS2WS Toolkit has now generated the proxy code for you. The proxy code is a CICS/assembler program. You need to assemble it and define it to CICS.

Together with the proxy code source, a compile job has been generated that can be used to assemble the proxy code. You need to upload the proxy code source to your z/VSE system. You can do this using standard file transfer functions like FTP. Adapt the compile job and use it to assemble the proxy code. You need to change the libraries used for cataloging the phase and the LIBDEFs. Check for a clean compile (i.e. return code zero). Make sure you place the phase of the proxy code into a library that is in the CICS LIBDEF.

You need to define the proxy program to CICS using CEDA DEFINE PROGRAM. The language is Assembler. The "DATALOCATION" parameter can be set to ANY; the "EXECKey" parameter should be set to User:

```
CEDA  DEFine PROGram( MYPROXY  )
 PROGram        : MYPROXY
 Group          : VSESPG
 DEscription  ==>
 Language     ==> Assembler          CObol | Assembler | C | Pli
 RELoad       ==> No                 No | Yes
 RESident     ==> No                 No | Yes
 USAge        ==> Normal             Normal | Transient
 USEsvacopy   ==> No                 No | Yes
 Status       ==> Enabled            Enabled | Disabled
 RSl            : 00                 0-24 | Public
 Cedf         ==> Yes                Yes | No
 DAtalocation ==> Any                Below | Any
 EXECKey      ==> User               User | Cics
REMOTE ATTRIBUTES
 REMOTESystem ==>
 REMOTEName   ==>
 Transid      ==>
```

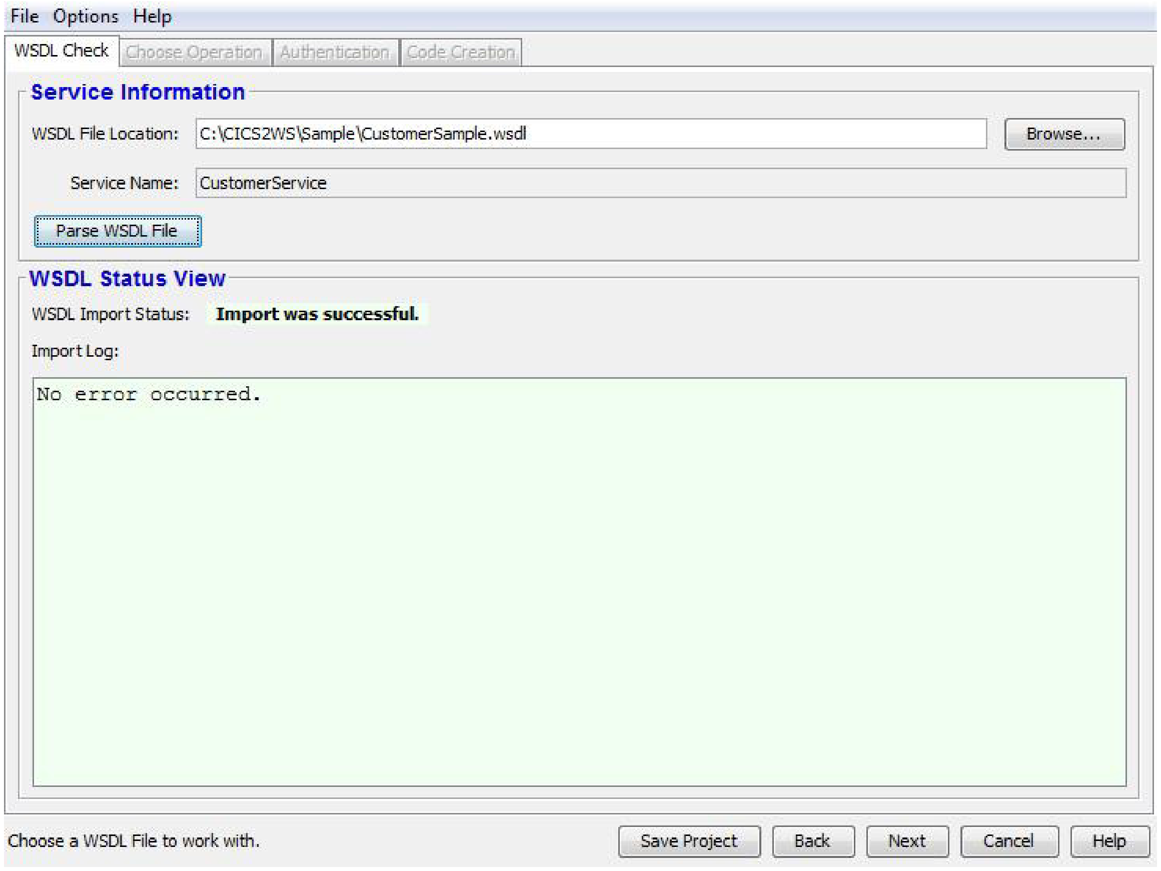
After defining the program, you need to install it using CEDA INSTALL, or restart CICS.

3.5. Use a Web Service with a CICS Application with literal support

If wsdl contains arrays or uses literal encoding style button “with literal support (z/VSE 5.2 or higher)” should be used. In this case will be used z/VSE SOAP Engine version 2.

Import WSDL and generate Rules Code & Copybook

As first step, you import the WSDL file into CICS2WS. Press the “Browse” button to locate the WSDL file. Then press the “Parse WSDL file” button to parse the WSDL file. Look for errors in the WSDL Status View below.



Service Information

WSDL File Location: C:\CICS2WS\Sample\CustomerSample.wsdl

Service Name: CustomerService

WSDL Status View

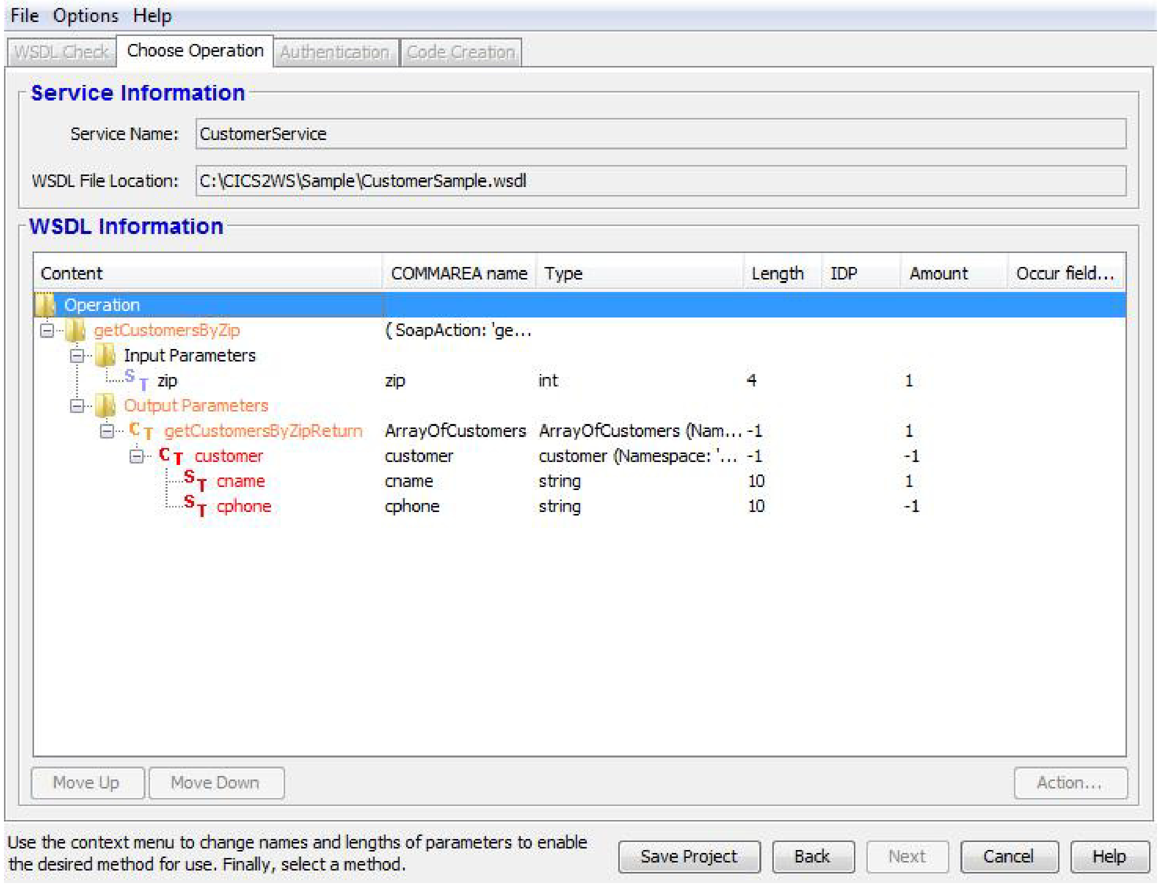
WSDL Import Status: **Import was successful.**

Import Log:

```
No error occurred.
```

Choose a WSDL File to work with.

If the WSDL import worked without errors, press the “Next” button to continue.



Service Information

Service Name: CustomerService

WSDL File Location: C:\CICS2WS\Sample\CustomerSample.wsdl

WSDL Information

Content	COMMAREA name	Type	Length	IDP	Amount	Occur field...
Operation						
getCustomersByZip	(SoapAction: 'ge...					
Input Parameters						
zip	zip	int	4		1	
Output Parameters						
getCustomersByZipReturn	ArrayOfCustomers	ArrayOfCustomers (Nam...	-1		1	
customer	customer	customer (Namespace: '...	-1		-1	
cname	cname	string	10		1	
cphone	cphone	string	10		-1	

Use the context menu to change names and lengths of parameters to enable the desired method for use. Finally, select a method.

On this panel, you must choose which operation of the Web Service (also called method) you want to call. In our customer example, the Web Service provides 1 operation:

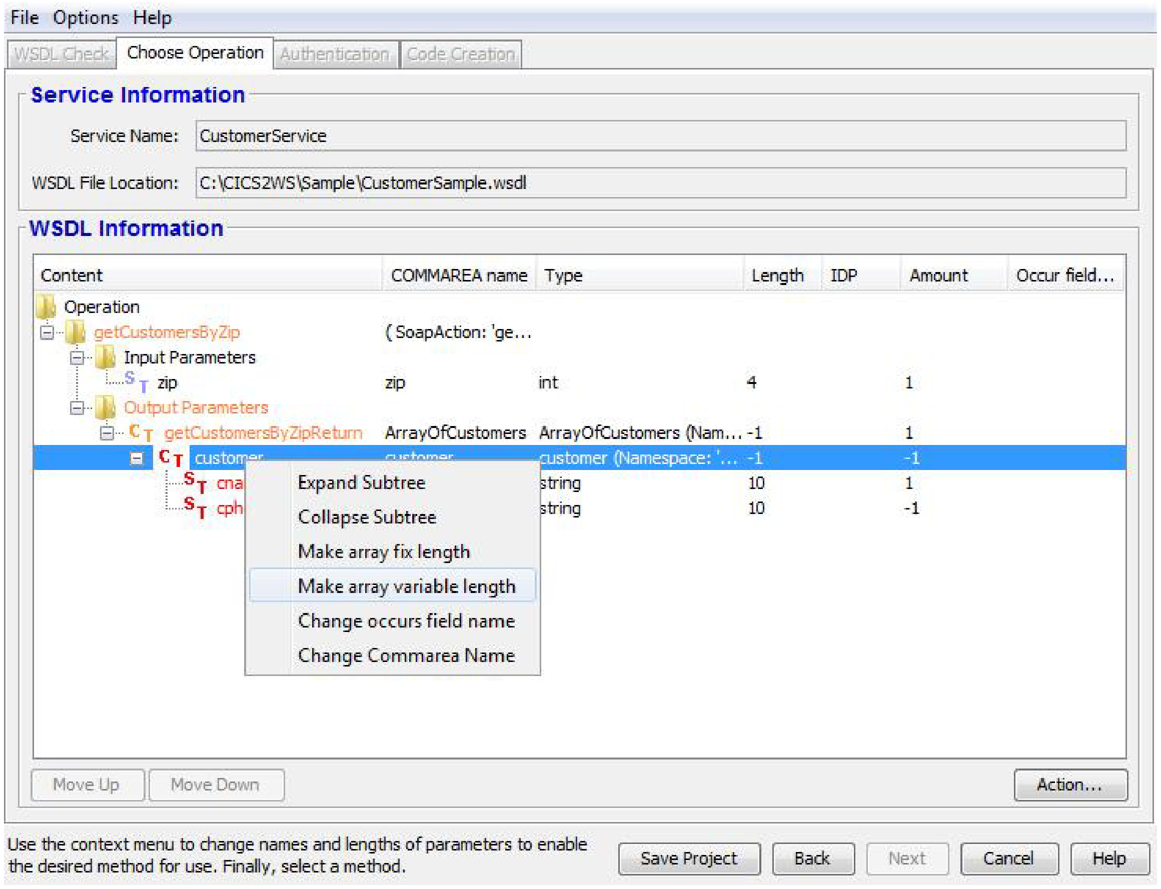
- getCustomersByZip get all customers in the area

Underneath the operation, you see which input parameter and which output parameters the operation uses. Some of the parameters may be marked in red or orange. Orange indicates that some parameter within the group is marked red. Read means that you need to perform an action before you can continue to use this operation.

In the example above, the parameters "*customer*" and "*cphone*" are arrays without a defined size (number of array elements). Parameters "*cname*" and "*cphone*" have data type "string", that data type requires to specify a length. That's why these parameters are marked in red. For more information about red/orange marks please see Appendix G.

Right mouse click allows setting up size of array. There are two possibilities for arrays:

- arrays with fix length
- arrays with variable length

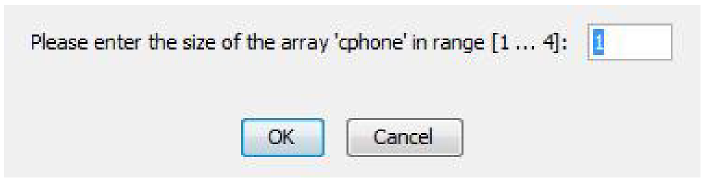


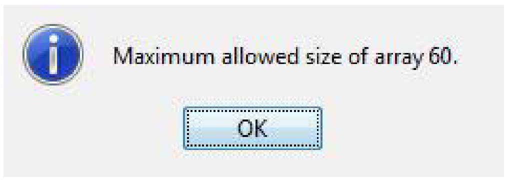
Content	COMMAREA name	Type	Length	IDP	Amount	Occur field...
Operation						
getCustomersByZip	(SoapAction: 'ge...					
Input Parameters						
zip	zip	int	4		1	
Output Parameters						
getCustomersByZipReturn	ArrayOfCustomers	ArrayOfCustomers (Nam...	-1		1	
customer	customer	customer (Namespace: '...	-1		-1	
cna		string	10		1	
cph		string	10		-1	

Use the context menu to change names and lengths of parameters to enable the desired method for use. Finally, select a method.

Fix length array

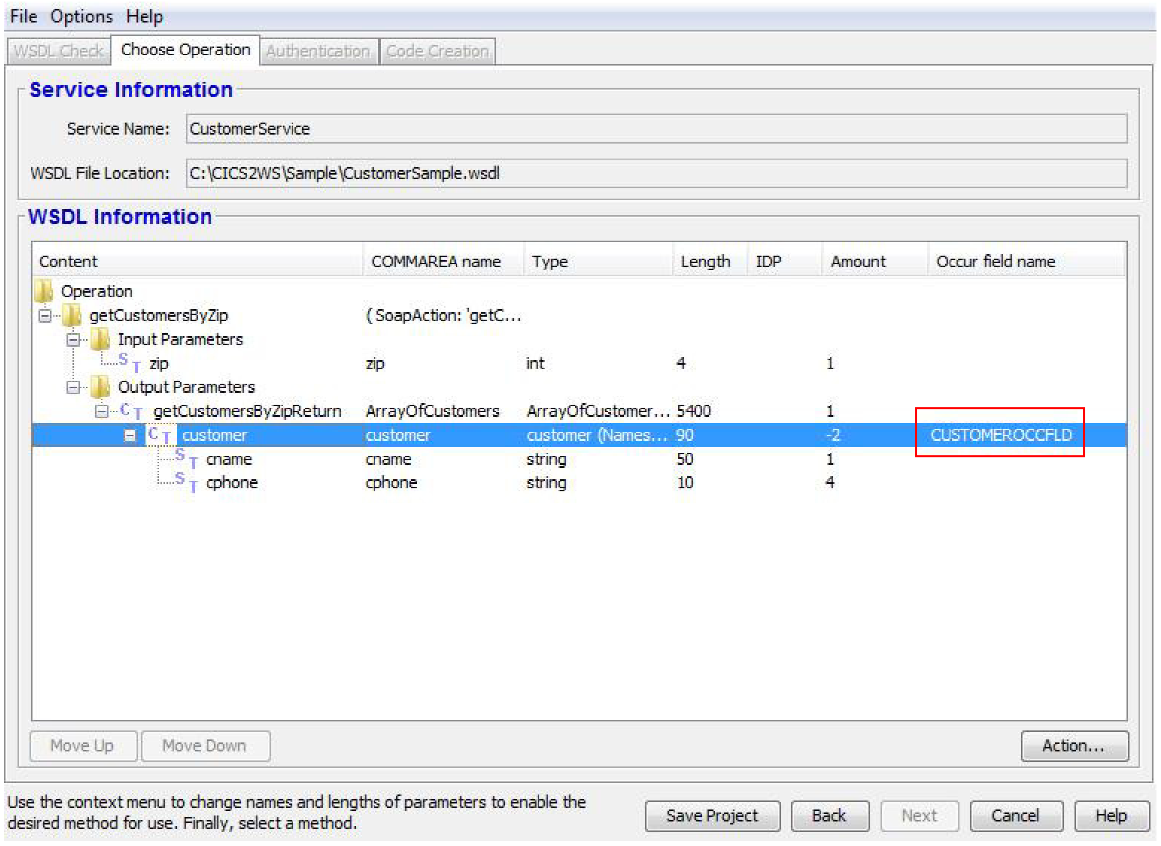
In case of defined maximum array size in wsdl CICS2WS Tool will ask to provide user defined maximum size in range 1 to the maximum size as defined by the WSDL.


Please enter the size of the array 'cphone' in range [1 ... 4]: 1


Maximum allowed size of array 60.

Variable length array

- Variable length array can be defined only for COBOL copybook (Assembler & PL/I do not support).
- Maximum available size is necessary, it will be taken from wsdl (if it’s applied) or user will be asked to provide it.
- It will be created addition field (Occur field) in copybook, where will be stored information about size in real time.



Service Information

Service Name: CustomerService

WSDL File Location: C:\CICS2WS\Sample\CustomerSample.wsdl

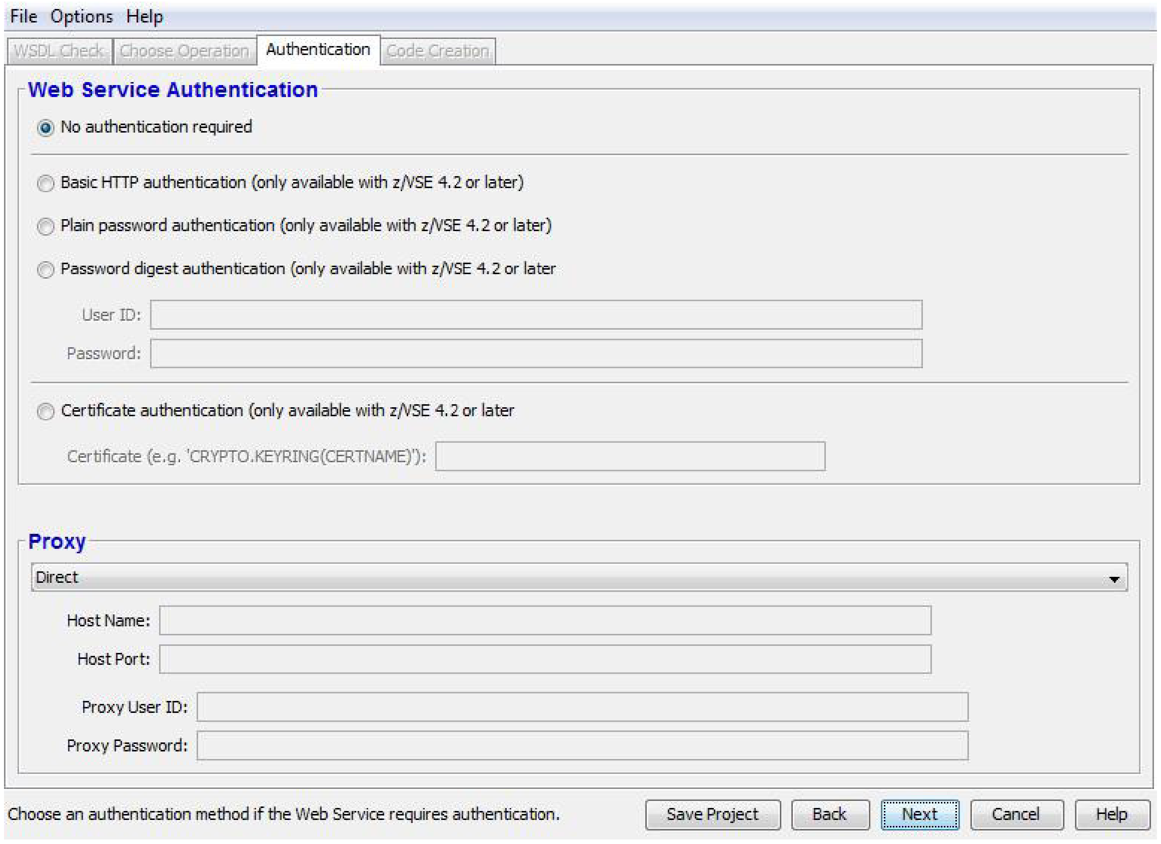
WSDL Information

Content	COMMAREA name	Type	Length	IDP	Amount	Occur field name
Operation						
getCustomersByZip	(SoapAction: 'getC…					
Input Parameters						
zip	zip	int	4		1	
Output Parameters						
getCustomersByZipReturn	ArrayOfCustomers	ArrayOfCustomer…	5400		1	
customer	customer	customer (Names…	90		-2	CUSTOMEROCCFLD
cname	cname	string	50		1	
cphone	cphone	string	10		4	

Use the context menu to change names and lengths of parameters to enable the desired method for use. Finally, select a method.

Once all problems have been resolved, the operation is no longer marked in red or orange.

Select the operation you want to work with (i.e. the "getCustomerByZip" operation) and press the "Next" button. You can only press the "Next" button if all problems of the chosen operation have been resolved.



Web Service Authentication

(●) No authentication required

( ) Basic HTTP authentication (only available with z/VSE 4.2 or later)

( ) Plain password authentication (only available with z/VSE 4.2 or later)

( ) Password digest authentication (only available with z/VSE 4.2 or later

User ID:

Password:

( ) Certificate authentication (only available with z/VSE 4.2 or later

Certificate (e.g. 'CRYPTO.KEYRING(CERTNAME)'):

Proxy

Direct

Host Name:

Host Port:

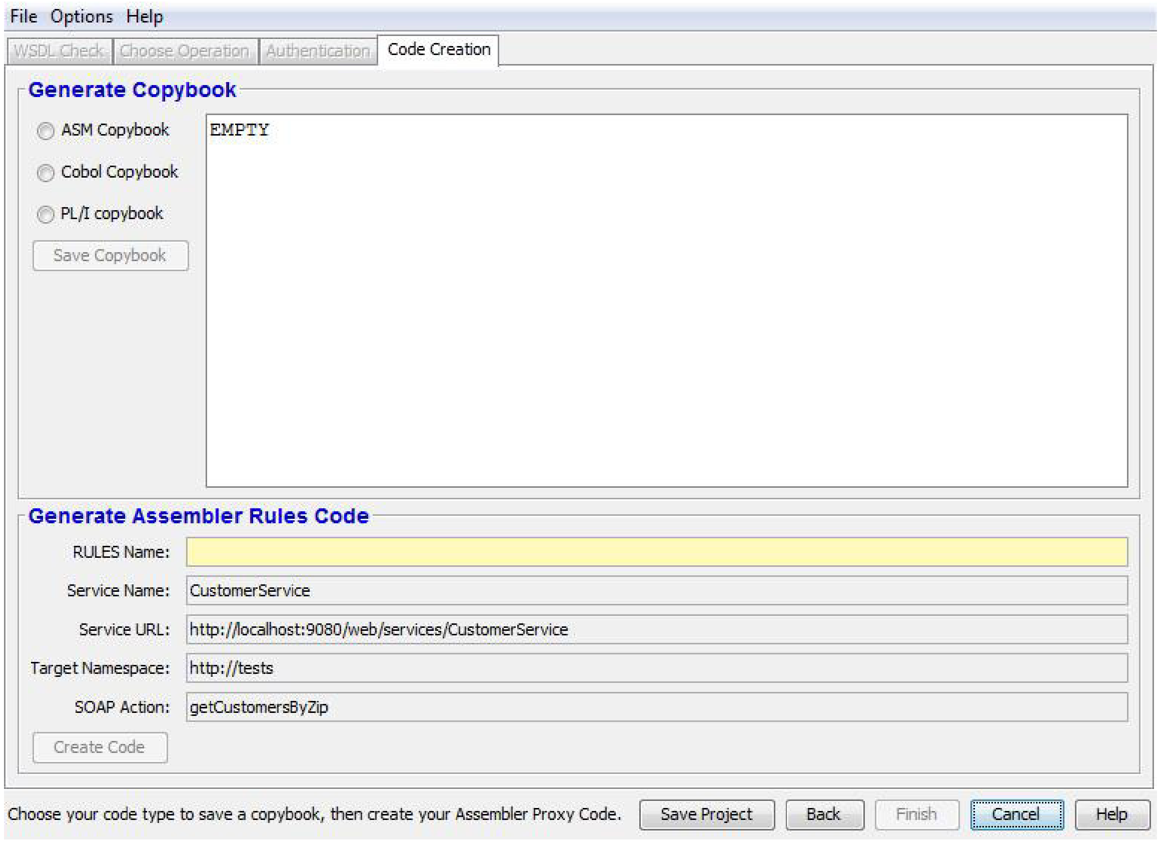
Proxy User ID:

Proxy Password:

Choose an authentication method if the Web Service requires authentication.

On this panel, you can choose if you want to use Web Service authentication. Authentication is only supported on z/VSE 4.2 and onwards. See Appendix D for more information.

Press the “Next” button to continue.



```
EMPTY
```

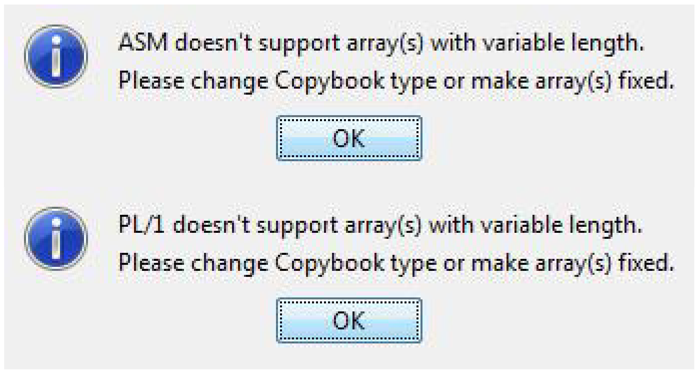
RULES Name:
Service Name: CustomerService
Service URL: http://localhost:9080/web/services/CustomerService
Target Namespace: http://tests
SOAP Action: getCustomersByZip

Choose your code type to save a copybook, then create your Assembler Proxy Code.

Now you are ready to generate the rules code and the copybook for the chosen Web Service operation.

The copybook can be generated in Assembler, COBOL or PL/1 language. Choose the desired language and press the “Save Copybook” button to save the generated copybook. You will see a preview of the copybook in the text area on the right.

Note. Arrays with variable size are supported only in Cobol.


ASM doesn't support array(s) with variable length.
Please change Copybook type or make array(s) fixed.

PL/1 doesn't support array(s) with variable length.
Please change Copybook type or make array(s) fixed.

```
      1 Copybook.
              3 zip  PIC 9(9) COMP.
              3 ArrayOfCustomers.
```

```
                5 CUSTOMEROCCFLD  PIC 9(9) COMP.
                5 customer OCCURS 1 TO 60 DEPENDING ON CUSTOMEROCCFLD.
                  7 cname  PIC X(50).
                  7 cphoneFGR OCCURS 4.
                    9 cphone  PIC X(10).
```

As you see, arrays described in COBOL with word “OCCURS” and range. In case of variable length appears additional field just before the array.

You will later on use the generated copybook to call (EXEC CICS LINK) the generated rules code.

To generate the rules code, enter the name of the rules code (“RULES Name”) and press the “Create Code” button. This will generate the rules source code and compile job. Remember the location of the generated files (copybook and rules code); you will need them later on.

Assemble and define the rules code to CICS

The CICS2WS Toolkit has now generated the rules code for you. The rules code is a CICS/assembler program containing just static data, but no executable code. You need to assemble it and define it to CICS.

Together with the rules code source, a compile job has been generated that can be used to assemble the rules code. You need to upload the rules code source to your z/VSE system. You can do this using standard file transfer functions like FTP. Adapt the compile job and use it to assemble the rules code. You need to change the libraries used for cataloging the phase and the LIBDEFs. Check for a clean compile (i.e. return code zero). Make sure you place the phase of the rules code into a library that is in the CICS LIBDEF.

You need to define the rules program to CICS using CEDA DEFINE PROGRAM. The language is Assembler. The “DATALOCATION” parameter can be set to ANY; the “EXECKey” parameter should be set to User:

```
CEDA  DEFine PROGram( MYRULES  )
 PROGram        : MYRULES
 Group          : VSESPG
 DEscription  ==>
 Language     ==> Assembler          COBol | Assembler | C | Pli
 RELoad       ==> No                 No | Yes
 RESident     ==> No                 No | Yes
 USAge        ==> Normal             Normal | Transient
 USEsvacopy   ==> No                 No | Yes
 Status       ==> Enabled            Enabled | Disabled
 RSl            : 00                 0-24 | Public
 Cedf         ==> Yes                Yes | No
 DAtalocation ==> Any                Below | Any
 EXECKey      ==> User               User | Cics
REMOTE ATTRIBUTES
 REMOTESystem ==>
 REMOTEName   ==>
 Transid      ==>
```

After defining the program, you need to install it using CEDA INSTALL, or restart CICS.

3.6. Implement a CICS program that calls a Web Service

When you want to call an external Web Service from you own CICS program, you have 2 options how to do that:

- Write the program to interface with the z/VSE SOAP Engine directly.
- Use the CICS2WS Toolkit to generate a so called Proxy code that acts as a wrapper between the z/VSE SOAP Engine and your program.

While interfacing directly with the z/VSE SOAP Engine gives you the greatest flexibility, it requires good knowledge about the programming interfaces of the z/VSE SOAP Engine. Please see Appendix C for some details of the programming interfaces. Choosing this approach is recommended when very complex data structures are being used in the Web Service parameters.

z/VSE SOAP Engine version 1
This article will focus on the use of the CICS2WS Toolkit to generate the so called proxy code that acts as a wrapper between the z/VSE SOAP Engine and your program. This approach assumes that your CICS program can call the generated proxy code via EXEC CICS LINK and a generated COMMAREA.

Your program must include the generated copybook, set the input COMMAREA fields and then call the proxy code via EXEC CICS LINK and pass the COMMAREA. On return from the proxy code, the output COMMAREA fields will contain the output values.

In case an error occurs during Web Service call processing, the generated proxy code will issue an abend E100 to E107. You can use EXEC CICS HANDLE ABEND in your program to catch the abends and do proper error handling.

z/VSE SOAP Engine version 2

This article will focus on the use of the CICS2WS Toolkit to generate the so called rules code that acts as a transport layer between the z/VSE SOAP Engine version 2 and your program. This approach assumes that your CICS program can call z/VSE SOAP Engine version 2 via EXEC CICS LINK and a generated COMMAREA.

Your program must include the generated copybook, set the input COMMAREA fields and then call the proxy code via EXEC CICS LINK and pass the COMMAREA. On return from the proxy code, the output COMMAREA fields will contain the output values.

```
EXEC CICS LINK PROGRAM("IESOACLN")
        COMMAREA(carea) LENGTH(sizeof(carea))
        RESP(rc1) RESP2(rc2);
```

where

IESOACLN - z/VSE SOAP Engine version 2
carea – generated COMMAREA
rc1 – return code for CICS LINK
rc2 – return code from SOAP Engine

Note. CICS2WS generates a copybook, there first 8 bytes are reserved for rules file name. User program must provide SOAP Engine with file name of created and stored on z/VSE rules code.

Requirements for programs that call a Web Service

A CICS program in that you want to call an external Web Service has to fulfill the following requirements:

- It can be written in any CICS supported programming language (e.g. Assembler, COBOL, PL/1, C, CICS/REXX)
- It must be able to call the proxy code via EXEC CICS LINK and a COMMAREA. The proxy code returns via EXEC CICS RETURN.

4. Debugging and trouble shooting, typical errors and pitfalls

This chapter describes what to do in case the created Web Service does not work as expected.

In case you encounter a problem, make sure you have applied all existing PTFs related to SOAP, CICS Web Support and TCP/IP. You will find a list of available PTFs here:

http://www.ibm.com/systems/z/os/zvse/support/connectors.html
http://www.ibm.com/systems/z/os/zvse/support/tcpip.html
http://www.ibm.com/systems/z/os/zvse/support/ipv6vse.html
http://www.ibm.com/systems/z/os/zvse/support/cics.html

Errors can occur at various places in the whole call path.

TCP/IP communication

Sometimes firewalls prohibit communication between the SOAP client and server.

- Make sure that you adapt your firewall rules so that the TCP/IP port you have chosen is permitted.
- A TCP/IP packet trace taken on the client side (e.g. using Wireshark/Ethereal) and/or on the z/VSE side can help to find out what goes wrong.
- Make sure you have configured a domain name server (SET DNS1) or a name (DEFINE NAME) for the z/VSE system's IP address.
- Check if you have specified the right system ID (// OPTION SYSPARM=nn) in the CICS startup job.

CICS Web Support

- Check if the TCPIP support and the TCPIPSERVICE used for Web Services is in status OPEN.
- Check the CICS log for error messages related to CWS or TCP/IP.
- Make sure you have configured a domain name server (SET DNS1) or a name (DEFINE NAME) for the z/VSE system's IP address.
- Check if you have specified the right system ID (// OPTION SYSPARM=nn) in the CICS startup job.

SOAP Client side

- Check the logs and error messages produced by the client.
- Check if the client was able to send out the request, or if it fails to receive and process the response.

- If it receives a bad HTTP error code (e.g. 505) or an invalid content type (e.g. text/html instead of text/xml), check the content for error messages.
- If it receives a SOAP Fault message, check the textual error message in the SOAP fault.

z/VSE SOAP Engine

If the client receives an empty or invalid SOAP response:

- Check if you have generated the CICS codepage conversion table DFHCNV. You should also see abend AWBK in transaction CWBA in this case.
- Check the CICS log for messages related to LE programs EDCUCONV and EDCUCNVI. Make sure these programs are defined in group CEE or you have program auto install activated (also see LE APAR PK20013).
- For best interoperability, the ASCII codepage used by the z/VSE SOAP Engine should be UTF-8. You may need to adapt the ASCII codepage in the z/VSE SOAP Option phase IESSOAPO (see Activate the SOAP trace for more details).

Abend during Web Service processing

If you receive an abend during Web Service processing, check the abend code.

- If the abend code is E100 to E107, then the abend is generated by the proxy code due to an error situation. Check the generated proxy code under which situation the abend is produced.
- Abend AWBK: This indicates that you did not generate the CICS codepage conversion table DFHCNV.
- Abend AEZC: This indicates that the user program which is called by the z/VSE SOAP Engine or by the proxy code is AMODE24. The CICS Web Support transaction CWBA runs with TASKDATALOC(ANY). You need to either change your program to AMODE 31 or change transaction CWBA to TASKDATALOC(BELOW) (better use another alias transaction name).
- If the abend is an AKEA, it most likely shows “0C4/AKEA at offset x"FFFFFFFF" in program IESSOAPE”:
    - If you are using the BSI TCP/IP stack, you need to obtain a fix from BSI. The abend happens in module IESHTTPB at function “slib_inet_addr”.
    - If you are running with CICS storage protection, check if the z/VSE SOAP Engine programs (IESHTTPB, IESHTTPC, IESSOAPC, IESSOAPE) are defined with EXECKey=CICS.

z/VSE SOAP Engine version 1 (Proxy code)

- When an error is detected by the proxy code, it either returns a return code (100 to 107) back to the z/VSE SOAP Engine or it abends with an abend code E100 to E107.
- Check the generated proxy code under which situation the error is produced.

- When you make updates to the proxy or rules code, you need to perform a CEMT SET PROG(nnnn) NEWCOPY to let CICS know that you have changed the program.
- When compiling the proxy code, check for return code zero. A non-zero return code most likely indicates that the generated code is too large, so that some fields can not be addressed. You will get assembler error messages like ASMA034E OPERAND xxx BEYOND ACTIVE USING RANGE BY nnn BYTES. In this case you need to reduce the number of fields used by the web service. Alternatively, you may have to write the proxy code in another language to get around this problem.

z/VSE SOAP Engine version 2

When an error is detected by the SOAP Engine, it returns a non-zero return code. This return code is stored in

- RESP2 parameter from "EXEC CICS LINK …" in case of z/VSE acts as a client
- xml-response in case of z/VSE acts as a server

To get more detailed information about problems, turn on trace option (see Activate the SOAP trace below).

User program errors

In case the user program detects an error and abends, either the proxy code or CICS Web Support will handle the abend and produce an appropriate error response.

- Check the original abend code and lookup the meaning of the abend code.
- Make sure that the user program does not use any terminal related functions, since there is no terminal associated to a program running as Web Service.

Client receives a SOAP Fault Message

When the z/VSE SOAP Engine detects an error situation, it sends back a SOAP Fault message. A SOAP Fault message is a special form of a SOAP response and contains a textual description about the error. It may also contain return codes from the converter/proxy code.

Activate the SOAP trace

If you want to change the SOAP options, you need to generate the SOAP, option phase IESSOAPO.

The z/VSE SOAP Engine tries to load the SOAP option phase IESSOAPO during processing. If IESSOAPO is not available (this is the default, unless it has been generated), it uses default options.

The SOAP option phase contains options for:

- Trace flags to activate SOAP tracing
- Codepages (ASCII and EBCDIC codepages)
- Secure Socket Layer specific settings for use with HTTPS (since z/VSE 4.2), such as keyring library, key name, SSL cipher suites, SSL session timeout.
- Long name to short name mapping table (since z/VSE 4.2), see Appendix A for more details.

Activate the SOAP trace if the problem persists. You can do that by generating the SOAP option phase IESSOAPO, either using skeleton SKSOAPOP in ICCF library 59, or by using this job:
ftp://ftp.software.ibm.com/eserver/zseries/zos/vse/download/vsecon/iessoapo.job

Set the desired trace flags to 1:

```
*
* TRACE FLAGS:   USED TO ACTIVATE TRACING
*
TRSYSLOG DC      XL2'001F'   TRACE TO SYSLOG
TRSYSLST DC      XL2'001F'   TRACE TO SYSLST
TRC_SERV EQU     X'0001'
TRC_CLNT EQU     X'0002'
TRC_DEC  EQU     X'0004'
TRC_ENC  EQU     X'0008'
TRC_HTTP EQU     X'0010'
```

After you have generated the SOAP option phase, you need to perform a NEWCOPY in CICS to activate the changes:

```
     CEMT SET PROG(IESSOAPO) NEWCOPY
```

Appendix A: Literal vs. encoded, RPC- vs. document-style

This chapter briefly explains the differences between literal and encoded SOAP, as well as RPC style and document style binding. For more information please also see here: http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

A WSDL document describes a Web Service. A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP messaging protocol. A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use. This gives you four style/use models:

- RPC/encoded
- RPC/literal
- Document/encoded (not used in practice)
- Document/literal

WSDL distinguishes between two message styles: document and RPC. The message style affects the contents of the SOAP Body:

- **Document style**: The SOAP Body contains one or more child elements called parts. There are no SOAP formatting rules for what the body contains; it contains whatever the sender and the receiver agrees upon.

- **RPC style**: RPC implies that SOAP body contains an element with the name of the method or operation being invoked. This element in turn contains an element for each parameter of that method/operation.

For applications that use serialization/deserialization to abstract away the data wire format, there's one more choice to be made: the serialization format. There are two popular serialization formats today:

- **SOAP Encoding**: SOAP encoding is a set of serialization. The rules specify how objects, structures, arrays, and object graphs should be serialized. Generally speaking, an application using SOAP encoding is focused on remote procedure calls and will likely use RPC message style. When SOAP Encoding is used, the SOAP Message contains data type information within the SOAP message. This makes serialization (data translation) easier since the data type of each parameter is denoted with the parameter.

- **Literal**: Data is serialized according to a schema. In practice, this schema is usually expressed using W3C XML Schema. The SOAP message does not directly contain any data type information, just a reference (namespace) to the schema that is used. To perform proper serialization (data translation) both, the sender and the receiver, must know the schema and must use the same rules for translating data.

Example: RPC/encoded SOAP message:

The following SOAP message uses RPC style and SOAP encoding:

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

The SOAP message contains an operation name (myMethod). The method transports 2 parameters, x and y, which both specify its data type (int and float) through the type attributes.

There are 2 namespaces used here: xsi and xsd. Both are defined within the SOAP Envelope (not shown in this example). The xsi namespace is the schema instance (http://www.w3.org/2001/XMLSchema-instance) and defines the type attribute (besides others), xsd is the schema namespace (http://www.w3.org/2001/XMLSchema), and it defines the meaning of the data types like int or float.

Example: RPC/literal SOAP Message:

The following SOAP message uses RPC style and literal:

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

In contrast to SOAP encoding, the parameters x and y do not specify any data type within the SOAP message. The sender and the receiver must 'know' how what data type the parameters are. Usually that information is available in the WSDL.

Example: Document/literal SOAP Message:

The following SOAP message uses document style and literal:

```
<soap:envelope>
    <soap:body>
        <xElement>5</xElement>
        <yElement>5.0</yElement>
    </soap:body>
</soap:envelope>
```

Here, there is no longer a method or operation name part of the body. Instead, the body directly contains the parameters. Also, there is not data type information contained in the SOAP message. Everything that appears within the body is usually defined in a schema.

Example: Document/literal wrapped SOAP Message:

The following SOAP message uses document style and literal wrapped:

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

This SOAP Message looks pretty much the same as the RPC/literal example. Here, the myMethod tag does not specify the method name, but a wrapper element, which the single input message's part refers to.

Recommendations for use with the z/VSE SOAP Engine

The z/VSE SOAP Engine supports both, SOAP encoded and literal SOAP messages. The z/VSE SOAP Engine does not support document style SOAP Messages, since it does not contain a method/operation name. The CICS2WS Toolkit only supports RPC style and SOAP Encoded and Literal WSDLs.

When z/VSE SOAP Engine receives a SOAP message (either a request when z/VSE acts as a SOAP server, or a response, when z/VSE acts as a SOAP client), it

- translates the parameters into TS-Queue entries. The **z/VSE SOAP Engine version 1** tries to deserialize (translate) the data into the appropriate native data type whenever possible.
- translates the parameters directly to the memory according defined rules. The **z/VSE SOAP Engine version 2** translate the data into appropriate data type using mapping rules.

Encoded SOAP messages contain data type information as part of the SOAP message, thus data translation can be done (if the type is known and supported). For literal SOAP messages, no data type is contained in the SOAP message, thus the z/VSE SOAP Engine version 1 can not translate the parameters into a native format. Thus, the z/VSE SOAP Engine version 1 creates TS-Queue entries with nun-translated data, the data will appear in textual format in the TS-queue entry as it appears in the SOAP message.

z/VSE SOAP Engine version 2 works for encoded and for literal SOAP messages and data translation can be done (if the type is known and supported).

For more information about how SOAP Messages are translated into TS-Queue entries, please see Appendix B.

Appendix B: Translating SOAP Messages into TS-Queue entries and vice versa

This chapter explains how the z/VSE SOAP Engine version 1 translates SOAP Messages into TS-Queue entries and vice versa. This chapter assumes that you have a good knowledge about the programming interfaces of the z/VSE SOAP Engine. Please see Appendix C and the z/VSE e-business Connectors, User's Guide, SC34-2629-02 for a detailed description.

Translating SOAP messages into TS-Queue entries:

When z/VSE SOAP Engine receives a SOAP message (either a request when z/VSE acts as a SOAP server, or a response, when z/VSE acts as a SOAP client), it translates the parameters into TS-Queue entries. The z/VSE SOAP Engine tries to deserialize (translate) the data into the appropriate native data type whenever possible.

Data translation can only be done when data type information is present in the SOAP message. Thus only messages that use SOAP encoding are eligible for data translation. For literal messages, which do not contain data type information, the z/VSE SOAP Engine can not perform any data translation. Thus it will pass the data unchanged (i.e. in textual form) into TS-Queue entries. The application will have to translate the textual representation into anything useful. The SOAP_PARAM_HDR field TYPE will contain UNSPECIFIED in this case; the field TYPENAME will contain blanks.

For SOAP encoded messages, data translation is performed. However, not all data types are known or not supported by the z/VSE SOAP Engine. Those parameters with data types that area not handed by the z/VSE SOAP Engine will also be passed un-translated into the TS Queue entries, that is in textual form. The application is then responsible to translate the textual representation into anything useful. The SOAP_PARAM_HDR field TYPE will contain PRIVATE in this case; the field TYPENAME will contain the name of the type as in the SOAP message. You can use the TYPENAME together with the NAMESPACEURL field to determine the application specific data type.

Complex data types with nested parameters (e.g. structures, arrays, complex objects, …) are in any case translated by the z/VSE SOAP Engine. Such a complex parameter will be translated into a single TS-Queue entry, which is flagged as complex or array type. The entry will have inner parameters with its own type information.

Translating TS-Queue entries into SOAP Messages:

When the z/VSE SOAP Engine sends out a SOAP message (either a request when z/VSE acts as a SOAP client, or a response, when z/VSE acts as a SOAP server), it uses the information in the TS-Queue entries to build the SOAP message. The z/VSE SOAP Engine performs serialization (translation) of the data if data type information is available.

If a literal SOAP message is to be sent, you should set the field TYPE in SOAP_PARAM_HDR to UNSPECIFIED and field TYPENAME to blanks. This indicates that no type information is required. The data of the parameter is passed unchanged into the SOAP Message, i.e. it must match the required textual representation.

When sending complex parameters, you set the field TYPE in SOAP_PARAM_HDR to STRUCT or ARRAY and set the field TYPENAME to the name of the application specific type. Usually you will use your own schema to describe the types. You should therefore set the field NAMESPACEURL to the schema's URL. The same applies for simple private types. In this case you set the field TYPE to PRIVATE, the field TYPENAME to the name of the application specific type, and the NAMESPACEURL to the schema's URL.

Appendix C: Programming interface of the z/VSE SOAP Engine version 1

This chapter describes the programming interface between the z/VSE SOAP Engine version 1 and proxy code or SOAP aware user program. Please see the z/VSE e-business Connectors, User's Guide, SC34-2629-02 for a detailed description.

The following describes the control blocks in some detail. You find the C definitions in the C header file IESSOAPH.h in PRD1.BASE. This file also contains an Assembler version of the definitions.

Note: Most of the fields used with the z/VSE SOAP Engine are case sensitive!

SOAP Parameter header (SOAP_PARAM_HDR):
The SOAP Parameter header is used to provide information about each parameter. It is used within the TS-Queue entries.

Offset	Length	Data type	Name	Description
0	16	Text	NAME	The name of the parameter
16	16	Text	TYPENAME	The type name of the parameter
32	4	Integer	LENGTH	The length in bytes including this header
36	4	Integer	TYPE	The type code of this parameter (see below)
36 (overlay)	2	Integer	ARRAY LENGTH	For type ARRAY only: Specifies the number of elements in the array
36 (overlay)	2	Integer	IMPLIED DECIMAL POS	For type DECIMAL only: Implied decimal position (number of decimal digits)

The following type codes are supported by the z/VSE SOAP Engine:

Type	Code	Corresponding type name	Description
UNSPECIFIED	0	none (blank)	No data type is specified. Used for literal SOAP messages.
PRIVATE	1	any private type name	A private data type. Uses a private namespace.
STRUCT	2	any private type name	A complex parameter
ARRAY	3	Array	A complex array parameter. The typename field specifies the typename of the inner array items
STRING	10	string	A text string (variable length).

LONG	16	long	A 8 byte signed integer
INTEGER	11	int	A 4 byte signed integer
SHORT	12	short	A 2 byte signed integer
BYTE	13	byte	A 1 byte signed integer
ULONG	20	unsignedLong	A 8 byte unsigned integer
UINTEGER	17	unsignedInt	A 4 byte unsigned integer
USHORT	18	unsignedShort	A 2 byte unsigned integer
UBYTE	19	unsignedByte	A 1 byte unsigned integer
BOOLEAN	14	boolean	A boolean parameter.
BINARY	15	base64Binary base64	Binary data (variable length)
DECIMAL	21	decimal	A decimal number, may contain decimal places
DECINT	22	integer negativeInteger nonNegativeInteger positiveInteger positiveInteger	A decimal number without decimal places
DATETINE	100	dateTime timeInstant recurringDate	A timestamp in ISO 8601 format
DATE	101	date	A date value (e.g. “2002-10-10”)
TIME	102	time	A time value (e.g. "13:20:00")
GYEAR	103	gYear year	A year value (e.g. "2005")
GMONTH	104	gMonth month	A month value (e.g. "--05")
GDAY	105	gDay recurringDay	A day value (e.g. "---01")
GYEARMONTH	106	gYearMonth	A year and month value (e.g. "1999-05")
GMONTHDAY	107	gMonthDay	A month and day value (e.g. "--05-01")

The corresponding type name column specifies the value of the type attribute in the SOAP message. Most of the types are from one of the following schemas:

- http://www.w3.org/2001/XMLSchema
- http://www.w3.org/2000/10/XMLSchema
- http://www.w3.org/1999/XMLSchema
- http://schemas.xmlsoap.org/soap/encoding/

SOAP Program Call Commarea (SOAP_PROG_PARAM):

This control block is used as COMMAREA when z/VSE acts as a SOAP Server. When the z/VSE SOAP Engine receives in incoming SOAP Request, it processes it, translates the parameters into TS-Queue entries and finally calls the corresponding user program via EXEC CICS LINK with this control block as COMMAREA. All fields are set by the z/VSE SOAP Engine and should not be modified by the user program, except the return code field (RETCODE).

Offset	Length	Data type	Name	Description
0	16	Text	METHOD	The method/operation name (first 16 chars only)
16	8	Text	INQUEUE	The name of the TS-Queue containing input parameters
24	8	Text	OUTQUEUE	The name of the TS-Queue that that user program will use to put output parameters to
32	128	Text	NAMESPACEURL	The namespace URL of the private or complex parameters. If multiple different namespaces are used in a SOAP message, you will only find one of it in this field. The namespace URL can be used in combination with the typename to determine the data type of a parameter.
160	4	Integer	RETCODE	Return code value. This field has to be set by the user program on return. If the ret code is non zero, the z/VSE SOAP Engine will generate a SOAP Fault message.
164	128	Text	METHODLONG	Long version of the method/ operation name with up to 128 characters.
292	4	Integer	AUTHTYPE	Since z/VSE 4.2: Authentication type used with this request (see Appendix H).
296	64	Text	AUTHUSER	Since z/VSE 4.2: User-ID that have been used for authentication
360	64	Text	AUTHPWD	Since z/VSE 4.2: Password that have been used for authentication

SOAP Web Service Call COMMAREA (SOAP_DEC_PARAM):

This control block is used as COMMAREA when z/VSE acts as a SOAP Client. When a user program wants to call an external Web Service, it calls the z/VSE SOAP Engine (IESSOAPD) via EXEC CICS LINK with this control block used as COMMAREA. Before calling the z/VSE SOAP Engine, the user program must set all fields of this control block and it must create the TS-Queues used for input and output parameters. Any parameters that are passed to the Web Service as input must be put onto the input TS-Queue. The z/VSE SOAP Engine will process the request and update the return code field (RETCODE) upon return. The z/VSE SOAP Engine also puts output parameters from the Web Service to the output TS-Queue.

Offset	**Length**	**Data type**	**Name**	**Description**
0	128	Text	URL	The URL under which the Web Service can be reached
128	16	Text	METHOD	The method/operation name (first 16 chars only)
144	128	Text	URN	The URN of the Web Service to call. This is most often the namespace of the Web Service to call.
172	8	Text	INQUEUE	The name of the TS-Queue containing input parameters
280	8	Text	OUTQUEUE	The name of the TS-Queue that that z/VSE SOAP Engine will use to put output parameters to
288	128	Text	NAMESPACEURL	The namespace URL of the private or complex parameters. The namespace URL can be used in combination with the typename to specify the private data type of a parameter.
416	4	Integer	PROXYTYPE	If the you want to use a proxy or socks server to reach the Web Service, specify one of the following values: 0: DIRECT (no proxy/socks) 1: PROXY 2: SOCKS V4 3: SOCKS V5
420	128	Text	PROXY	The hostname or dotted IP address of the proxy/socks server
548	4	Integer	PROXYPORT	The port number under which the proxy server can be reached
552	16	Text	PROXYUSER	The user-Id used for SOCKS authentication
568	16	Text	PROXYPWD	The password used for SOCKS authentication
584	4	Integer	RETCODE	Return code value. The z/VSE SOAP Engine sets this field upon return to the user program.

588	128	Text	SOAPACTION	Some Web Services require to send a SOAP-Action HTTP header along with the SOAP message
716	128	Text	METHODLONG	Long version of the method/ operation name with up to 128 characters.
844	4	Integer	AUTHTYPE	Since z/VSE 4.2: Authentication type that is to be used with this request (see Appendix D:).
912	64	Text	AUTHUSER	Since z/VSE 4.2: User-ID used for authentication
976	64	Text	AUTHPWD	Since z/VSE 4.2: Password used for authentication

Notes:

- The METHODLONG field is used for supporting method names that are greater than 16 characters. If the METHODLONG field contains all zeros or blanks, or if the COMMAREA length that is passed indicates that the new field is not present, the method name is taken from the field METHOD. This provides backward compatibility with existing programs. If the COMMAREA is large enough and METHODLONG is not zero or blanks, the method name is taken from the field METHODLONG.

- It is the user program's responsibility to generate TS-Queue names that are unique across other running programs and TS-Queues. A good practice is to use the CICS task-ID of the current task as part of the TS-Queue names. This makes it unique. It is also the user programs responsibility to delete the TS-Queues when they are no longer needed.

Appendix D: Using Web Service Security

Web Service security can be described under:

- Transport-layer security
- Message-layer security.

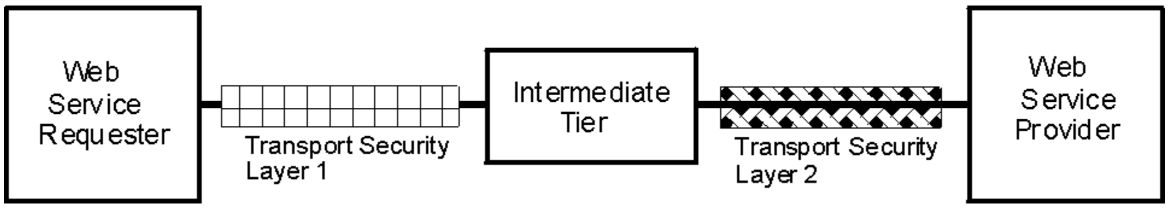
Both transport-layer and message-layer security provides security features for:

- Authentication/authorization
- Data encryption and signatures.

Comparison of Transport-Layer Security and Message-Layer Security

Transport-layer security secures the network communication between the communication partners by encrypting the data that is being transmitted over the network. In addition, data integrity, authentication, and confidentiality can be achieved. Transport-layer security typically uses digital signatures, PKI certificates, and secure hash functions to prevent messages from being "camouflaged," passwords from being hacked, and transactions from being denied.

In situations where an environment consists of several hops, the communication between each hop has to be considered separately in terms of transport-layer security:



As shown in the figure above, the connections between each hop might use different transport-layer security methods (or even no transport security for some connections). Transport-layer security does not "span" multiple hops. This means, an intermediate hop might be able to read the message. To achieve end-to-end security, you must therefore use message-layer security. Using message-layer security, the message itself is secure and does not change when sent over multiple hops.

Transport-layer security can be implemented using any of the industry-wide protocols, such as:

- SSL (Secure Socket Layer), which is denoted by HTTPS.
- VPN/IPSec (which is transparent to applications).

Message-layer security includes security-related information in the SOAP message (or more specifically, within the SOAP header).

Using Authentication with Web Service Security

Using authentication allows a service provider to check who is using the requested service. In addition, the service provider may use this information to execute the service under a specific user-ID, with its associated access rights (authorization).

To fully understand authentication and authorization, it is important to understand the following concepts:

Authentication
The process of identifying an individual using the credentials of that individual.

Authorization
The process of determining whether an authenticated client is allowed to access a resource or perform a task within a security domain. Authorization uses information about a client's identity and/or roles to determine the resources or tasks that a client can perform.

Credentials
A set of claims used to prove the identity of a client. They contain an identifier for the client and a proof of the client's identity such as a password. They may also include information, such as a signature, to indicate that the issuer certifies the claims in the credential.

Identification
The use of an identifier that allows a system to recognize a particular subject and distinguish it from other users of the system.

There are two possible methods of performing authentication:

- Transport-layer authentication. Here, the transport layer carries information about who is requesting the service. The possible implementations are:
    - HTTP Authentication (Basic and Digest Access Authorization, see RFC 2617).
    - The use of SSL Client Authentication with SSL/HTTPS.

- Message-layer authentication. Here, the SOAP message itself carries information about who is requesting the service. The possible implementations are:
    - Direct authentication, using plain text passwords or a password digest.
    - Brokered Authentication, using a X.509 Certificate, Kerberos, Security Token Services, or SAML Assertion. Brokered Authentication using a X.509 Certificate carries the X.509 Certificate as part of the SOAP header. Here is an example:

```
<soap:Header>
  <Security xmlns="...secext-1.0.xsd>
```

```
      <BinarySecurityToken EncodigType="wsse:Base64Binary"
                           ValueType="wsse:X509v3">
        MIICuzCCAiQCBF...
          ...
      </BinarySecurityToken>
    </Security>
    ...
```

Direct authentication defines *two* ways of transporting the password:

- *Plain text password*, in which UsernameToken is used to transport the actual password. If you use plain-text password configuration, you must use a secure transport method (such as HTTPS).
  Here is an example:

```
    <soap:Header>
      <Security xmlns="...secext-1.0.xsd>
        <UsernameToken>
          <Username>John Smith</Username>
          <Password>Pass12wd</Password>
        </UsernameToken>
      </Security>
      ...
```

- *Password digest*, in which the communicating parties (the requester and the service) use an insecure transport channel. Steps must be taken to protect the passwords from being exposed to others. Here, the requester creates a `digest` of the actual password that is concatenated with a set of random bytes (field `nonce`) and another value that is dependent on the creation-time (field `created`). This digest is computed as follows:

```
  digest = Base64_encode(SHA-1(nonce+created+password))
```

  To authenticate the request, the service computes the digest value using the password bound to the received username. It compares the received digest value with the computed digest value. Here is an example:

```
    <soap:Header>
      <Security xmlns="...secext-1.0.xsd>
        <UsernameToken>
          <Username>John Smith</Username>
          <Password
            Type="...#PasswordDigest">AFHHF23wger=</Password>
          <Nonce>ksSDGFljdfD=</Nonce>
          <Created>2010-07-15T07:12:19.573Z>/Created>
        </UsernameToken>
      </Security>
      ...
```

From z/VSE 4.2 onwards, z/VSE supports:

- Transport-layer authentication using:
  - HTTP authentication (Basic and Digest Access Authorization).
  - SSL Client Authentication with HTTPS.
- Message-layer authentication using:

- a `UsernameToken` (plaintext password or password digest).
- an X.509 Certificate (`BinarySecurityToken`).

Using Web Service Security Features when z/VSE Acts As the SOAP Server

These are the areas you should consider:

- **Transport-Layer Encryption**: When transport-layer authentication is used, z/VSE acts as an HTTP server. This is implemented using CICS Web Support (CWS) as the HTTP server. CWS passes the SOAP request to the z/VSE SOAP Engine for further processing. Since CWS implements support for HTTP over SSL (HTTPS), the z/VSE SOAP Engine inherits the security features from CWS. To use HTTPS, you must:
  - Configure `TCPIPSERVICE` in CICS for use with SSL.
  - Create the required keys and certificates.

- **Transport-Layer Authentication**: CWS supports SSL client authentication (HTTPS), as well as HTTP Basic Authentication, so the z/VSE SOAP Engine inherits the security features from CWS. To force a client to use HTTP basic authentication, you need to configure the TCPIPSERVICE to use the CICS provided converter program DFH$WBSB (specify `URM=DFH$WBSB`). In addition, the z/VSE SOAP Engine extracts authentication information (user-ID and password for HTTP basic authentication, or the mapped user-ID for SSL client authentication). This information can be used by the converter code to check if transport layer authentication was used. If authentication was not used, the converter code might reject the request.

- **Message-Layer Authentication:** To support message layer authentication, the z/VSE SOAP Engine (that is, the z/VSE SOAP Server) extracts the authentication token from the SOAP header after parsing the XML data stream. In case of `UsernameToken`, the user-ID and password have to be verified against a local identity store. To do this, the identity store must be able to compare the plain text password of password digest against its stored password. If user authorization is to be additionally performed, a user mapping must be performed to map the received username to a z/VSE user-ID. Also, a CICS SIGNON has to be performed using the mapped user, to allow the transaction to run under that user. In addition to `UsernameToken`, the use of certificates for authentication is possible. In this case, the converter code maps the received certificate to a z/VSE user. z/VSE supports this functionality as part of its support for SSL client authentication.

You should be aware that the z/VSE SOAP Engine does not itself perform the authentication. Instead, it simply extracts the security information and passes it to the converter code or user application. It is the user application's responsibility to perform authorization checking or signon processing.

The SOAP control block SOAP_PROG_PARAM specifies what authentication method have been used. User programs may use this information to perform authentication checking. In SOAP_PROG_PARAM, the fields AUTHTYPE, AUTHUSER and AUTHPWD are used to pass security related information to the user program:

AUTHTYPE	**AUTHUSER**	**AUTHPWD**
NONE (0)	Not used	Not used
HTTP_BASIC (1)	User-ID as in HTTP request	Password as in HTTP request
WS_PLAIN (2)	User-ID as in SOAP Header	Password as in SOAP Header
WS_DIGEST (3)	User-ID as in SOAP Header	Name of a 3rd TS-Queue that holds 3 entries: - The password digest in base64 - The created timestamp as text - The nonce value in base64
WS_CERT (4)	User-ID that was mapped from the certificate in the SOAP Header or blanks if no mapping found	Name of a 3rd TS-Queue that holds 1 entry: - the binary certificate data
SSL_CERT (5)	User-ID that was mapped from the SSL Client certificate or blanks if no mapping found	Name of a 3rd TS-Queue that holds 1 entry: - the binary certificate data

Using Web Service Security Features when z/VSE Acts As the SOAP Client

These are the areas you should consider:

- **Transport-Layer Encryption**: When transport-layer authentication is used, z/VSE acts as an HTTP client. The HTTP client that is implemented in z/VSE then supports HTTPS. To use HTTPS:
    - The URL needs to specify `https://`
    - You must provide a public/private key pair, together with certificates. For details of how to specify the keys, refer to the skeleton SKSOAPOP in VSE/ICCF Library 59.

- **Transport-Layer Authentication**: From z/VSE 4.2 onwards, the HTTP Client supports SSL/HTTPS. Therefore, you can use SSL client authentication using certificates. If requested, the SSL protocol can send the client's certificate to the server (service provider). If required, the server can use the client's certificate to perform authentication and authorization. In addition, HTTP basic authentication is supported by the z/VSE HTTP Client.

- **Message-Layer Authentication**: From z/VSE 4.2 onwards, the z/VSE SOAP Engine (that is, the z/VSE SOAP Client) supports the authentication token in the SOAP header. For the `UsernameToken`, the user application (or converter code) requesting the service must pass the `username` and `password` to the z/VSE SOAP Engine. If authentication is done using a certificate, the certificate name must be provided. Code for passing this information can either be part of the user application, or part of the converter code.

The SOAP control block SOAP_DEC_PARAM specifies what authentication method is to be used. In SOAP_DEC_PARAM, the fields AUTHTYPE, AUTHUSER and AUTHPWD are used to pass security related information to the z/VSE SOAP Engine:

AUTHTYPE	**AUTHUSER**	**AUTHPWD**
NONE (0)	Not used	Not used
HTTP_BASIC (1)	User-ID to be used in HTTP request	Password to be used in HTTP request
WS_PLAIN (2)	User-ID to be used in SOAP Header	Password to be used in SOAP Header
WS_DIGEST (3)	User-ID to be used in SOAP Header	Password to be used in SOAP Header
WS_CERT (4)	Specifies the name of the certificate member that contains the certificate to send in the SOAP header: CRYPTO.KEYRING(CERTNAME)	Not used
SSL_CERT (5)	Not supported for VSE as SOAP Client. The Web Service provider may request the SSL client certificate to perform SSL client authentication.	

Appendix E: Mapping long names to short names

In case of SOAP Engine version 1 should be additional attention for the length of SOAP parameters. Due to the size restriction for TS Queue entries, SOAP parameters can only have names up to 16 characters (as shown in the SOAP_PROG_PARAM control block).

If you wish to use SOAP parameters that are greater than 16 characters, you can supply your own mapping to map long names (greater than 16 chars) to short names (less than or equal to 16 characters). This support is available since z/VSE 4.2.

The z/VSE SOAP Engine will translate:

- Long names to their corresponding short names when it receives SOAP messages that contain parameters with long names.
- Short names to their corresponding long names when sending out SOAP messages containing parameters with long names.

Short names that belong to a long name must start with a "#" character, so that the z/VSE SOAP Engine can recognize this as a name that needs to be translated.

You can supply the mapping in the SOAP Option phase IESSOAPO. Please use skeleton SKSOAPOP in ICCF library 59 to generate the option phase. Below you see an example of the mapping table:

```
* *******************************************************************
* Start of the mapping table. Entries are specified as follows:
*      ENTRY SHORT='short name',LONG='long name'
* *******************************************************************
*
TABSTART DS     0F              START OF TABLE
*
         ENTRY SHORT='#shortin',                                      X
               LONG='ThisIsAVeryLongInputWithMoreThan16Chars'
         ENTRY SHORT='#shortout',                                     X
               LONG='ThisIsAVeryLongOutputWithMoreThan16Chars'
*
TABEND   DC     H'0'            END OF TABLE
```

Appendix F: Differences between SOAP Web Services and REST services

SOAP Web Services often use HTTP as the underlying transport protocol, and the XML based Simple Object Access Protocol (SOAP) as container to transport data (i.e. parameters, objects) within a SOAP message.

REST stands for "Representational State Transfer". It describes a simple interface which transmits domain-specific data over HTTP without an additional messaging layer such as SOAP. REST often uses XML within the HTTP content, but it does not require a specific encoding such as SOAP.

An important concept in REST is the existence of resources (sources of specific information), each of which is referenced with a global identifier (e.g., a URI in HTTP). In order to manipulate these resources, components of the network (user agents and origin servers) communicate via a standardized interface (e.g., HTTP) and exchange representations of these resources (the actual documents conveying the information).

With HTTP, REST uses the following HTTP requests to create, update and delete resources:

POST	Create the resource
GET	Read the resource
PUT	Update or Create the resource
DELETE	Delete the resource

Some REST services also use additional attributes e.g. as part of the URL to specify the action to perform.

To provide a REST service in z/VSE, you need to implement your own web-aware program that runs inside CICS Web Support (CWS). It gets called when CWS receives an HTTP request and uses EXEC CICS WEB nnn commands to retrieve information about the HTTP request as well as the HTTP content. For more information about writing a web-aware program in CICS please see the CICS Internet Guide (SC34-5765) chapter "Writing CICS programs to process HTTP requests":
http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/DFHWTL06/2.9

With REST, the content is most likely XML data. You can use the z/VSE XML Parser to parse XML data. For more information about the z/VSE XML Parser, see here:
http://www.ibm.com/systems/z/os/zvse/documentation/ebusiness.html#xmlparser

To call an REST service from a z/VSE program, you also must implement your own program that communicates via HTTP with the REST service. You can use the z/VSE

HTTP Client to connect to the REST service, send the HTTP request and receive the HTTP response. For more information about the z/VSE HTTP Client, see here:
http://www.ibm.com/systems/z/os/zvse/documentation/ebusiness.html#httpclient

Appendix G: Red and orange marks

On the CICS2WS panel “Add Operation to Service” and panel “Choose Operation” some of the fields may be marked in red or orange.
Orange indicates that some parameter within the group is marked red or that the group is a redefine. Red means that you need to perform an action before you can continue to use this operation.

There are the following reasons for red field:

- fields name is longer than 16 characters. Right click the field and choose the desired action (e.g. rename the field).
- the parameter appears as input parameter as well as output parameter. Every Web Service parameter will have its corresponding field in the generated COMMAREA. Per default, the COMMAREA field gets the same name as the parameter in the Web Service operation. You can change the name of the COMMAREA field. Right click the parameter and choose “Change Commarea Name”, then type in the new name.
- array size should be specified. Right click the field and choose variable of fix size of array.

Appendix H: More information

z/VSE V4R2 e-business Connectors User's Guide, SC34-2629-02
http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IESCUE50/CCONTENTS

CICS2WS Toolkit download
http://www.ibm.com/systems/z/os/zvse/downloads/#cics2ws

Web Services Documentation
http://www.ibm.com/systems/z/os/zvse/documentation/ebusiness.html#soap

Presentation: SOA Roadmap and Application integration for z/VSE
ftp://ftp.software.ibm.com/eserver/zseries/zos/vse/pdf3/techconf2008/lasvegas/zEO02.pdf

Presentation: Using SOA Web Services with z/VSE
ftp://ftp.software.ibm.com/eserver/zseries/zos/vse/pdf3/techconf2008/lasvegas/zEO03.pdf

Presentation: How to Exploit SOA using z/VSE Web Services
ftp://ftp.software.ibm.com/eserver/zseries/zos/vse/pdf3/techconf2008/lasvegas/zEO52.pdf