IBM z/OS Connect (OpenAPI 2.0)

Developing RESTful APIs for IMS Transactions

IBM Washington Systems Center

Lab Version Date: December 7, 2025

Table of Contents

Overview

Important – You do not need any skills with IMS to perform this exercise. Even if IMS is not relevant to your current plans, performing this exercise will give additional hands-on experience using the Toolkit to develop services and APIs.

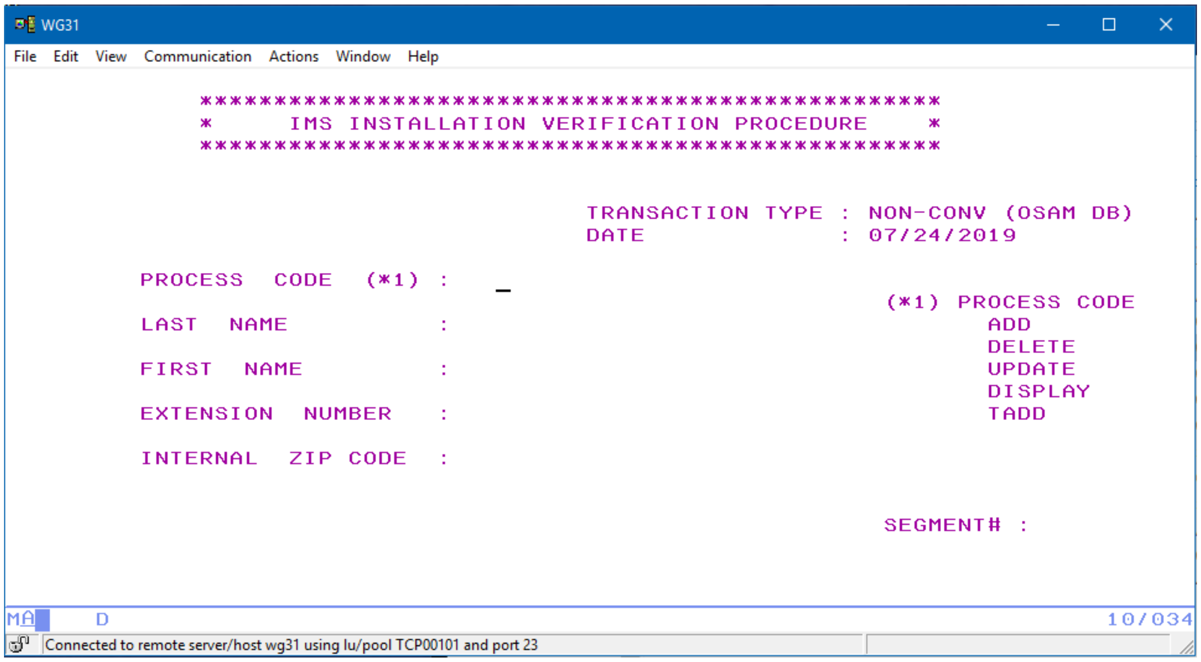
The objective of these exercises is to gain experience with working with z/OS Connect and the API Toolkit. These two products allow the exposure of z/OS resources to JSON clients. For information about scheduling this workshop in your area contact your IBM representative.

General Exercise Information and Guidelines

- ✓ This exercise requires using z/OS user identity *USER1*. The RACF password for this user is *user1* (lower case sensitive).
- ✓ Any time you have any questions about the use of IBM z/OS Explorer, 3270 screens, features or tools do not hesitate to ask the instructor for assistance.
- ✓ Please note that there may be minor differences between the screen shots in this exercise versus what you see on your desktop. These differences should not impact the completion of this exercise.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *OpenAPI2 developing IMS APIs CopyPaste* file.
- ✓ For information regarding the use of the Personal Communication 3270 emulator, see the *Personal Communications Tips* PDF in the exercise folder.

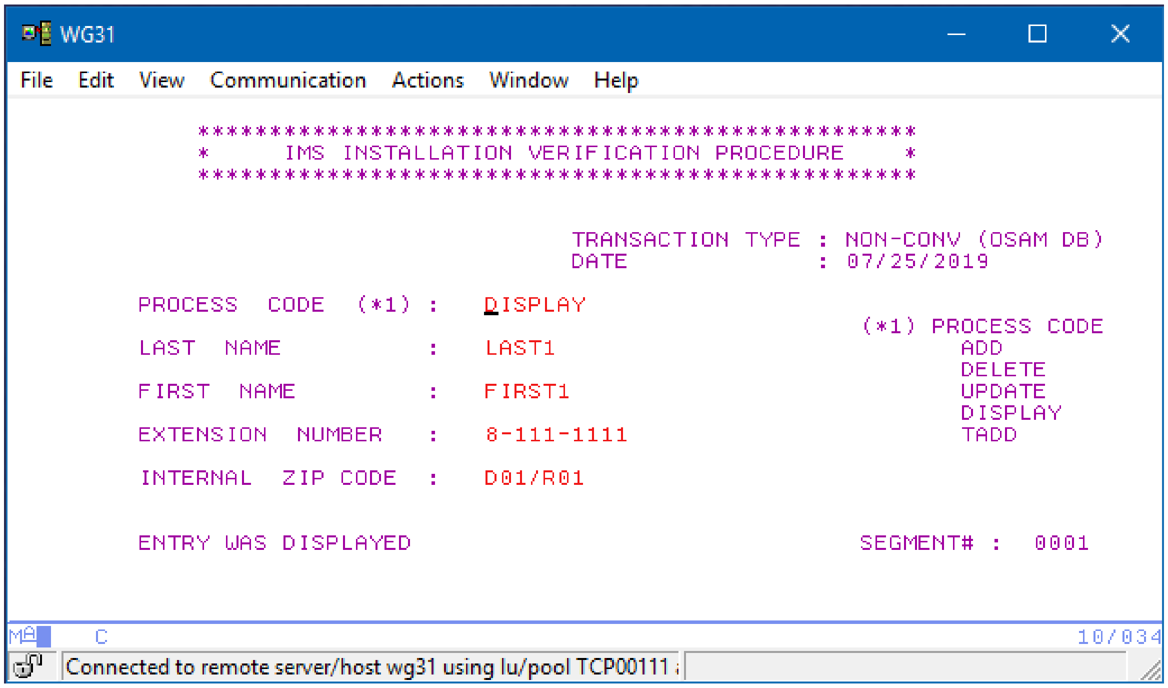
Review the IMS Phonebook IVP

The IMS IVP sample application Phonebook is provided by IMS and is used to verify the installation and customization of an IMS region. It is a simple telephone book application that uses MFS to schedule the *IVTNO* transaction to request and/or update information in the IMS Telephone database.



```
        **************************************************
        *      IMS INSTALLATION VERIFICATION PROCEDURE    *
        **************************************************

                         TRANSACTION TYPE : NON-CONV (OSAM DB)
                         DATE             : 07/24/2019

    PROCESS  CODE  (*1) :   _
                                         (*1) PROCESS CODE
    LAST  NAME          :                       ADD
                                                DELETE
    FIRST  NAME         :                       UPDATE
                                                DISPLAY
    EXTENSION  NUMBER   :                       TADD

    INTERNAL  ZIP CODE  :


                                         SEGMENT# :

MA      D                                                 10/034
Connected to remote server/host wg31 using lu/pool TCP00101 and port 23
```

Which action the IVP application program performs is determined by a process code provided in the input message. The process codes are ADD (POST), DELETE, UPDATE (PUT) and DISPLAY (GET). To display or delete a record, only the process code and the last name field are required. To add or update a record, the last name, first name, extension and zip code fields are required, see example output screen below.



```
        **************************************************
        *      IMS INSTALLATION VERIFICATION PROCEDURE    *
        **************************************************

                         TRANSACTION TYPE : NON-CONV (OSAM DB)
                         DATE             : 07/25/2019

    PROCESS  CODE  (*1) :  DISPLAY
                                         (*1) PROCESS CODE
    LAST  NAME          :  LAST1                ADD
                                                DELETE
    FIRST  NAME         :  FIRST1               UPDATE
                                                DISPLAY
    EXTENSION  NUMBER   :  8-111-1111           TADD

    INTERNAL  ZIP CODE  :  D01/R01

    ENTRY WAS DISPLAYED                  SEGMENT# :  0001

MA      C                                                 10/034
Connected to remote server/host wg31 using lu/pool TCP00111 ;
```

In this exercise unique service will be developed for each process code. All of the services will set the transaction code in the IMS message and one of the 4 process codes. Every field with the exception of the last name will be omitted from the request message interface for the DELETE and DISPLAY process code services All phone book required fields will be included in the request message for the ADD and UPDATE process codes services. The response message for the ADD, DELETE and UPDATE services will simply be the message indicting the status of the requested action. Only the DISPLAY service will return all of the phone book fields.

The input message and output messages are represented by the COBOL code below.

```
      * DATA AREA FOR TERMINAL INPUT

       01  INPUT-MSG.
           02  IN-LL          PICTURE S9(3) COMP.
           02  IN-ZZ          PICTURE S9(3) COMP.
           02  IN-TRANCDE     PICTURE X(10).
           02  IN-COMMAND     PICTURE X(8).
           02  IN-LAST-NAME   PICTURE X(10).
           02  IN-FIRST-NAME  PICTURE X(10).
           02  IN-EXTENSION   PICTURE X(10).
           02  IN-ZIP-CODE    PICTURE X(7).

      * DATA AREA OUTPUT
       01  OUTPUT-AREA.
           02  OUT-LL       PICTURE S9(3) COMP VALUE +95.
           02  OUT-ZZ       PICTURE S9(3) COMP VALUE +0.
           02  OUT-MESSAGE   PIC X(40).
           02  OUT-COMMAND   PIC X(8).
           02  OUT-LAST-NAME   PIC X(10).
           02  OUT-FIRST-NAME  PIC X(10).
           02  OUT-EXTENSION   PIC X(10).
           02  OUT-ZIP-CODE    PIC X(7).
           02  OUT-SEGMENT-NO  PICTURE X(4) VALUE '0001'.
```

In the input message the IN-TRANCDE field must be set to the IMS transaction *IVTNO*. The IN-COMMAND field identifies the process code.

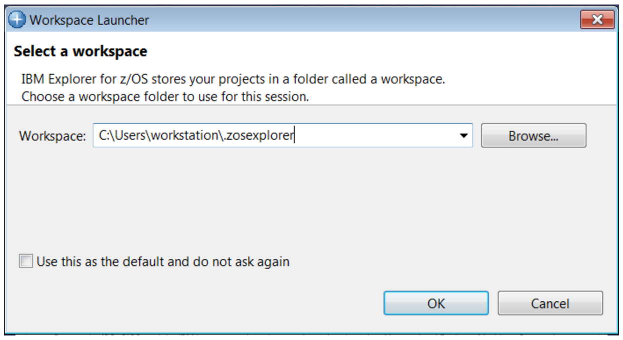
Connect the IBM z/OS Explorer to the z/OS Connect Server

Begin by establishing a connection to your z/OS Connect server from IBM z/OS Explorer. If you have performed one of the other exercises in this series of exercises this step may not be required.

___1. On the workstation desktop, locate the *z/OS Explorer* icon and double click on it to open the tool.

Tech-Tip: Windows desktop tools can be opened either by double clicking the icon or by selecting the icon and right mouse button clicking and then selecting the *Open* option.

___2. You will be prompted for a workspace:


Workspace Launcher

Select a workspace

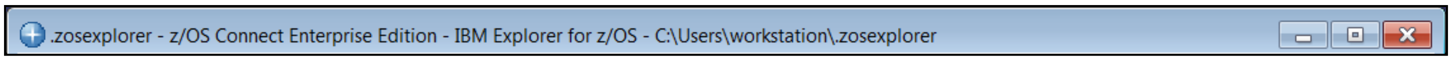
IBM Explorer for z/OS stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\workstation\.zosexplorer

Use this as the default and do not ask again

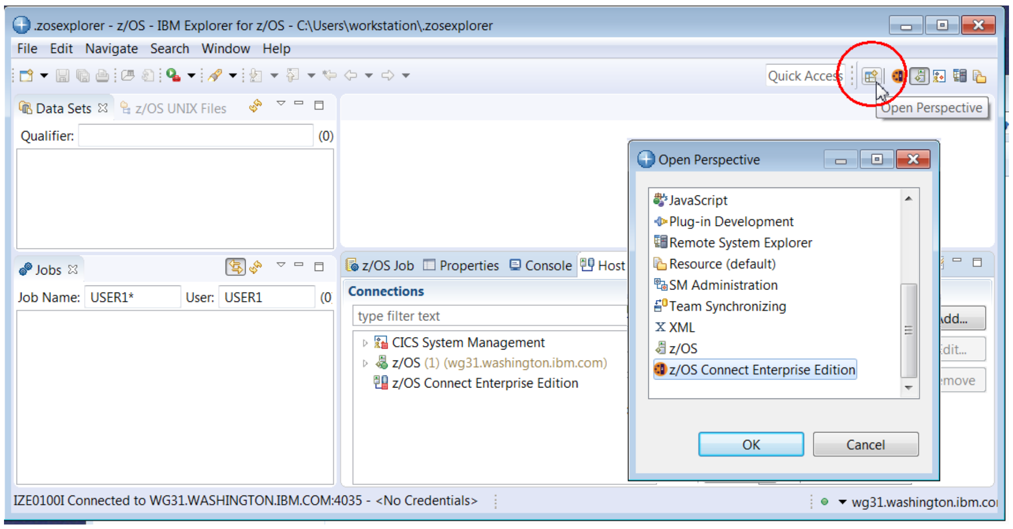
Take the default value by clicking **OK**.

___3. The Explorer should open in the *z/OS Connect Enterprise Edition* perspective. Verify this by looking in the upper left corner. You should see:


.zosexplorer - z/OS Connect Enterprise Edition - IBM Explorer for z/OS - C:\Users\workstation\.zosexplorer

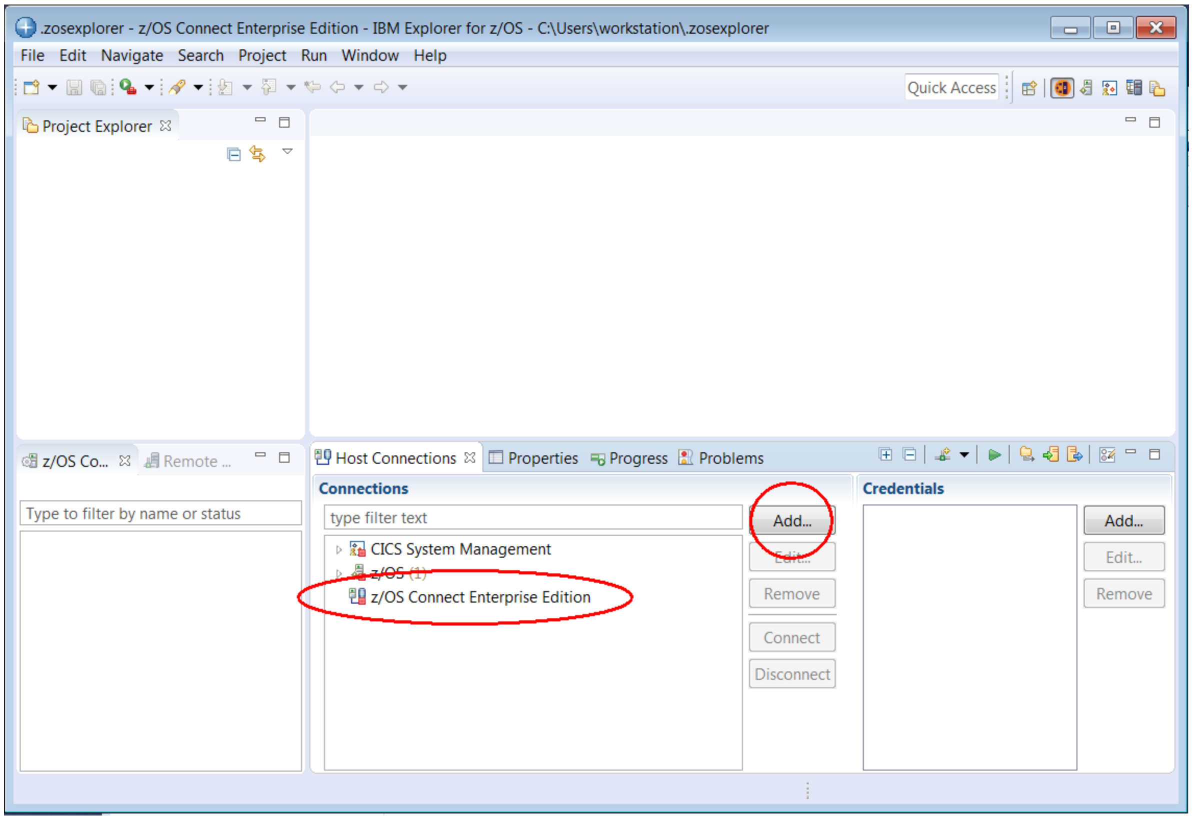
N.B. If a *Welcome* screen is displayed then click the white X beside *Welcome* to close this view.

___4. If the current perspective is not *z/OS Connect Enterprise Edition*, select the *Open Perspective* icon on the top right side to display the list of available perspectives, see below. Select **z/OS Connect Enterprise Edition** and click the **OK** button to switch to this perspective.



IZE0100I Connected to WG31.WASHINGTON.IBM.COM:4035 - <No Credentials>

___5. To add a connection to the z/OS Connect Server select *z/OS Connect Enterprise Edition* connection in the *Host connections* tab in the lower view and then click the **Add** button.


.zosexplorer - z/OS Connect Enterprise Edition - IBM Explorer for z/OS - C:\Users\workstation\.zosexplorer
File Edit Navigate Search Project Run Window Help
Quick Access
Project Explorer
z/OS Co...
Remote ...
Type to filter by name or status
Host Connections
Properties
Progress
Problems
Connections
type filter text
CICS System Management
z/OS (1)
z/OS Connect Enterprise Edition
Add...
Edit...
Remove
Connect
Disconnect
Credentials
Add...
Edit...
Remove

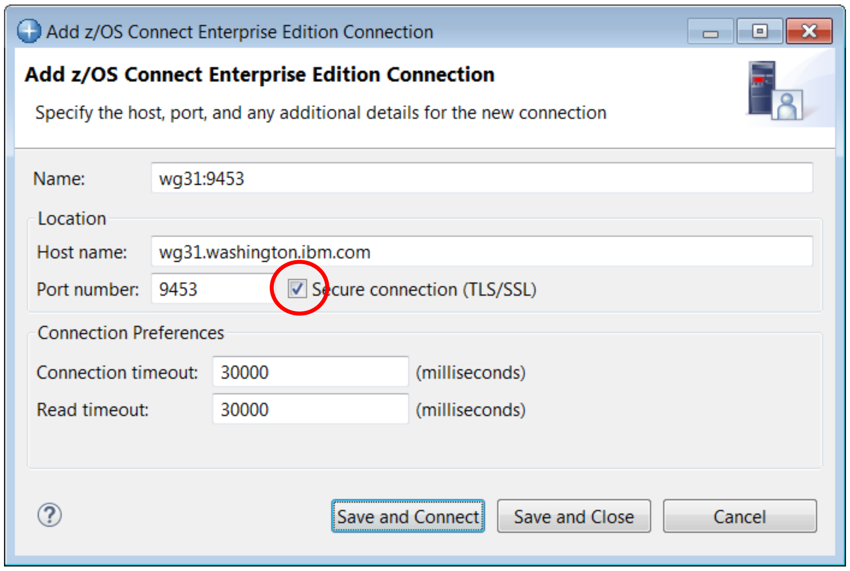
Tech-Tip: Eclipse based development tools like z/OS Explorer provide a graphical interface consisting of multiple views within a single window.

A view is an area in the window dedicated to providing a specific tool or function. For example, in the window above, *Host Connections* and *Project Explorer* are views that use different areas of the window for displaying information. At bottom on the right there is a single area for displaying the contents of four views stacked together (commonly called a *stacked views*), *z/OS Host Connections, Properties, Progress* and *Problems*. In a stacked view, the contents of each view can be displayed by clicking on the view tab (the name of the view).

At any time, a specific view can be enlarged to fill the entire window by double clicking in the view's title bar. Double clicking in the view's title bar will be restored the original arrangement. If a z/OS Explorer view is closed or otherwise disappears, the original arrangement can be restored by selecting Windows → Reset Perspective in the window's tool bar.

Eclipse based tools also can display multiple views based on the current role of the user. In this context, a window is known as a perspective. The contents (or views) of a perspective are based on the role the user, i.e., developer or administrator.

___6. In the pop-up list displayed select *z/OS Connect Enterprise Edition* and on the *Add z/OS Connect Enterprise Edition Connection* screen enter ***wg31.washington.ibm.com*** for the *Host name*, ***9453*** for the *Port Number*, check the box for *Secure connection (TLS/SSL)* and then click the **Save and Connect** button.


Add z/OS Connect Enterprise Edition Connection

Add z/OS Connect Enterprise Edition Connection

Specify the host, port, and any additional details for the new connection

Name: wg31:9453

Location

Host name: wg31.washington.ibm.com

Port number: 9453 [✓ Secure connection (TLS/SSL)]

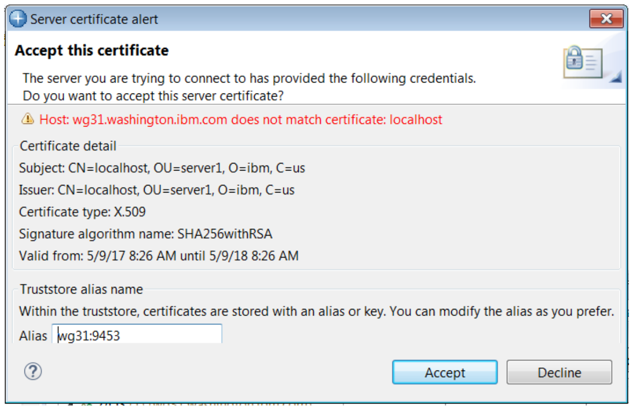
Connection Preferences

Connection timeout: 30000 (milliseconds)

Read timeout: 30000 (milliseconds)

___7. On the *z/OS Connect Enterprise Edition – User ID* required screen create new credentials for a *User ID* of ***Fred*** and a *Password or Passphrase* of ***fredpwd*** (case matters). Remember the server is configured to use basic security. If SAF security had been enabled, then a valid RACF User ID and password will have to be used instead. Click **OK** to continue.

___8. Click the **Accept** button on the *Server certificate alert – Accept this certificate* screen. You may be presented with another prompt for a userid and password, enter ***Fred*** and ***fredpwd*** again.


Server certificate alert

Accept this certificate

The server you are trying to connect to has provided the following credentials.
Do you want to accept this server certificate?

⚠ Host: wg31.washington.ibm.com does not match certificate: localhost

Certificate detail

Subject: CN=localhost, OU=server1, O=ibm, C=us
Issuer: CN=localhost, OU=server1, O=ibm, C=us
Certificate type: X.509
Signature algorithm name: SHA256withRSA
Valid from: 5/9/17 8:26 AM until 5/9/18 8:26 AM

Truststore alias name

Within the truststore, certificates are stored with an alias or key. You can modify the alias as you prefer.

Alias wg31:9453

___9. The status icon beside *wg31:9453* should now be a green circle with a lock. This shows that a secure connection has been established between the z/OS Explorer and the z/OS Connect server. A red box indicates that no connection exists.

___10. A connection to the remote z/OS system was previously added. In the *Host Connection* view expand *z/OS Remote System* under *z/OS* and select *wg31.washington.ibm.com*. If the connection is not active the **Connect** button will be enabled. Click the **Connect** button and this will establish a session to the z/OS system. This step is required when submitting job for execution and viewing the output of these jobs later in this exercise.

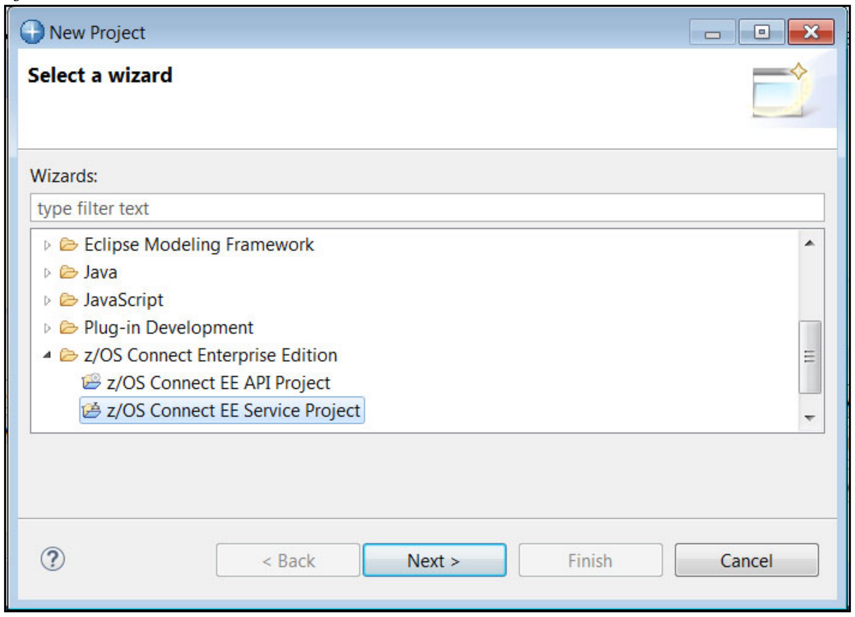
Create services for the Phonebook API

The first step is to create the four services which correspond to the four process codes. The response messages will be the same format for three of the process codes, ADD, UPDATE and DELETE. So, there will be only two response messages created. One with only the *message* field for the ADD, UPDATE and DELETE process codes and one response message with all of the fields for the DISPLAY process code.

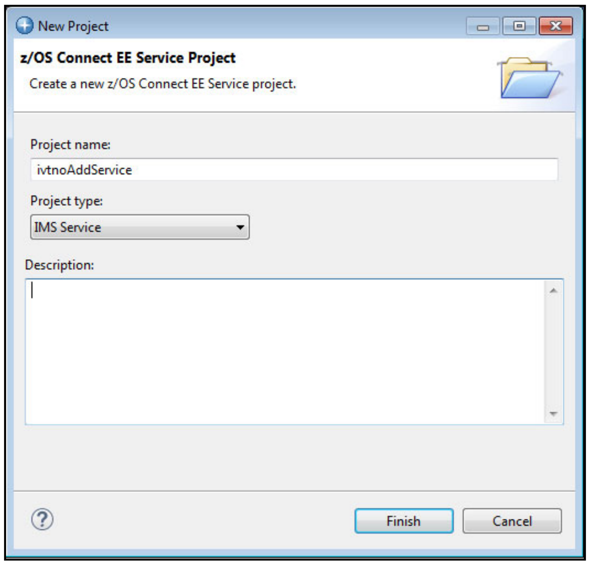
Create the "Add" Service

___1. In the upper left, position your mouse anywhere in the *Project Explorer* view and right-mouse click, then select *New →Project*:

___2. In the *New Project* window, scroll down and open the *z/OS Connect Enterprise Edition* folder and select *z/OS Connect Service Project* and then click the **Next** button.


New Project
Select a wizard
Wizards:
type filter text
Eclipse Modeling Framework
Java
JavaScript
Plug-in Development
z/OS Connect Enterprise Edition
z/OS Connect EE API Project
z/OS Connect EE Service Project
< Back
Next >
Finish
Cancel

___3. On the new *New Project* window enter ***ivtnoAddService*** as the *Project name* and use the pull-down arrow to select *IMS Service* as the *Project type*. Click **Finish** to continue


New Project

z/OS Connect EE Service Project

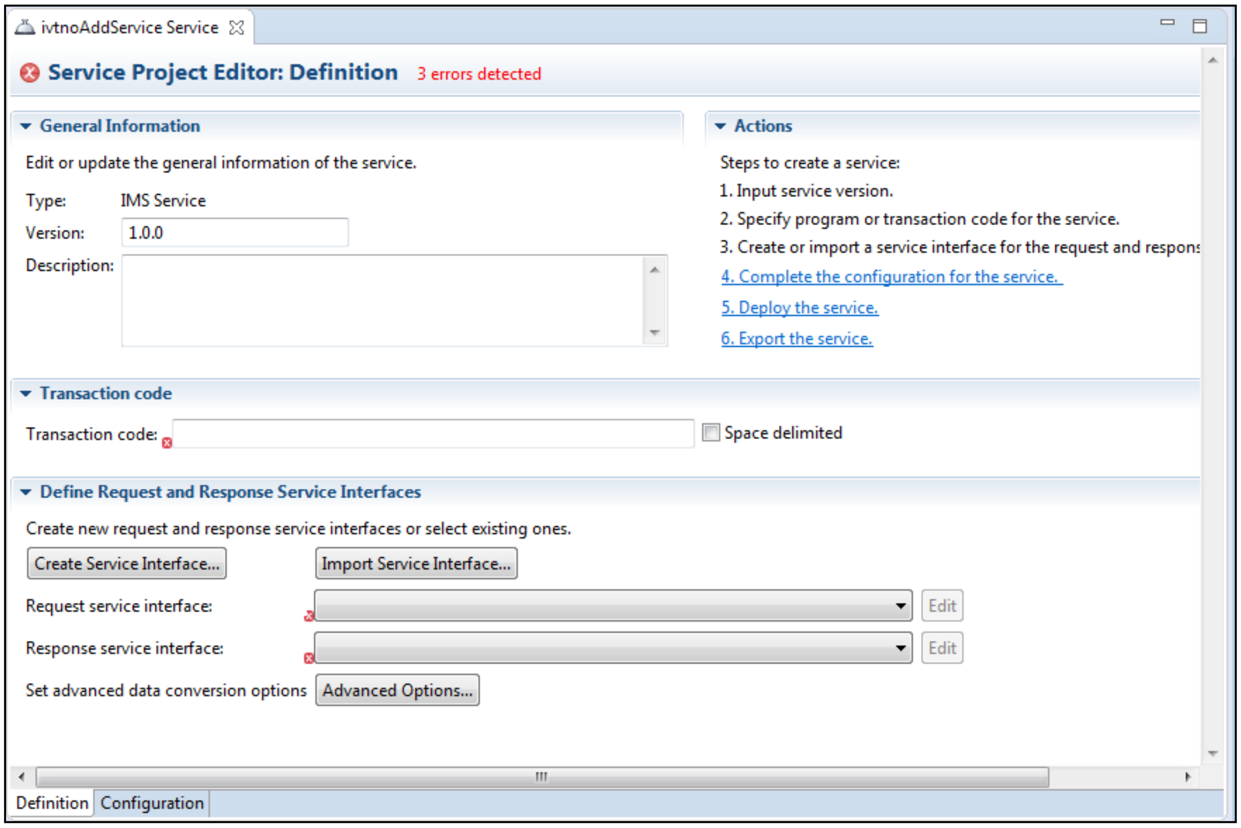
Create a new z/OS Connect EE Service project.

Project name: ivtnoAddService

Project type: IMS Service

Description:

___4. This will open the *Overview* window for the *ivtnoAddService*. For now, disregard the message about the 3 errors detected, they will be addressed shortly.


ivtnoAddService Service

Service Project Editor: Definition 3 errors detected

General Information

Edit or update the general information of the service.

Type: IMS Service

Version: 1.0.0

Description:

Actions

Steps to create a service:

1. Input service version.
2. Specify program or transaction code for the service.
3. Create or import a service interface for the request and respons
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Transaction code

Transaction code:

Space delimited

Define Request and Response Service Interfaces

Create new request and response service interfaces or select existing ones.

Request service interface:

Response service interface:

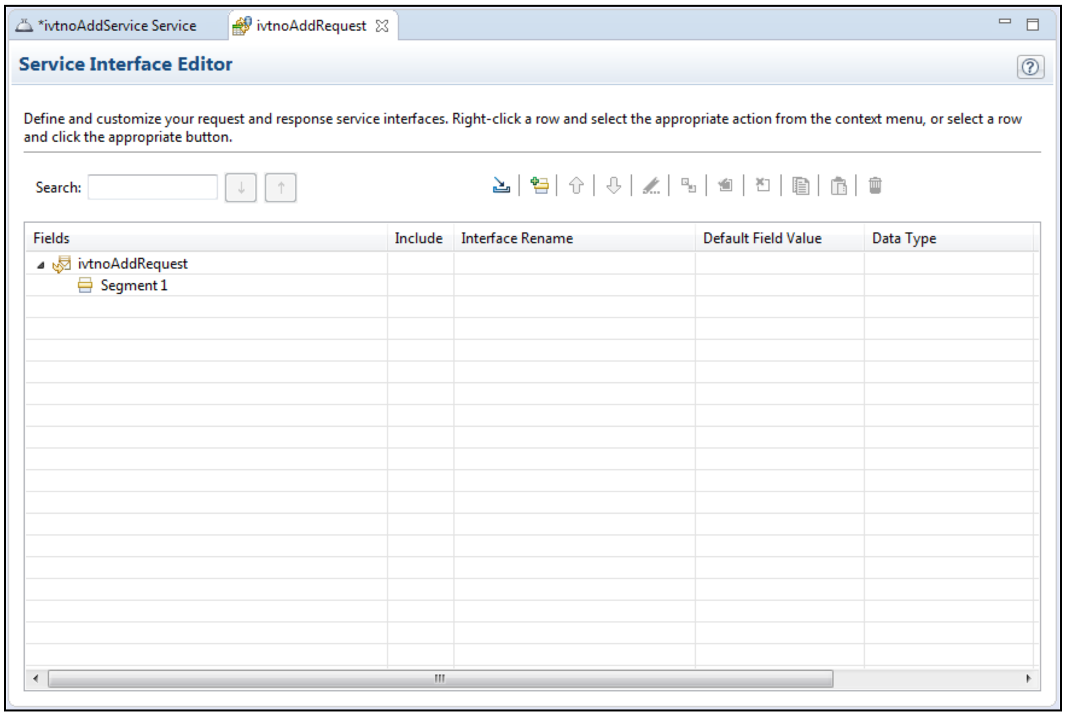
Set advanced data conversion options

Tech-Tip: If this view is closed it can be reopened by double clicking the *service.properties* file in the service project.

___5. Next enter the IMS transaction ***IVTNO*** in the area beside *Transaction code*.

___6. Click the **Create Service Interface** button to create the first service required by this API and enter a *Service interface name* of ***ivtnoAddRequest***. Click **OK** to continue.

___7. This will open a *Service Interface Definition* window.



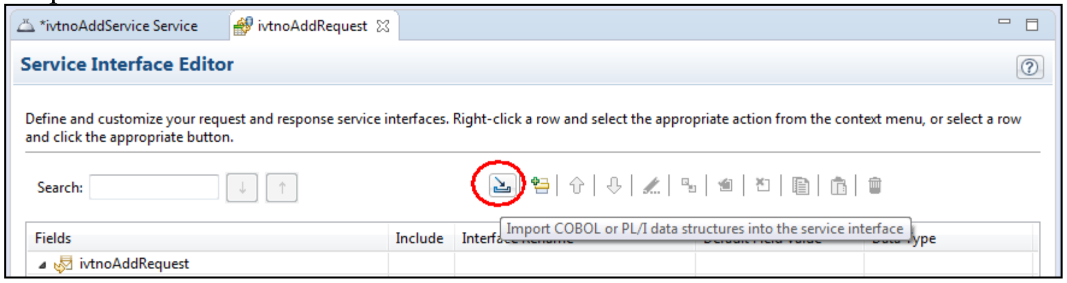
Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type
ivtnoAddRequest				
Segment 1				

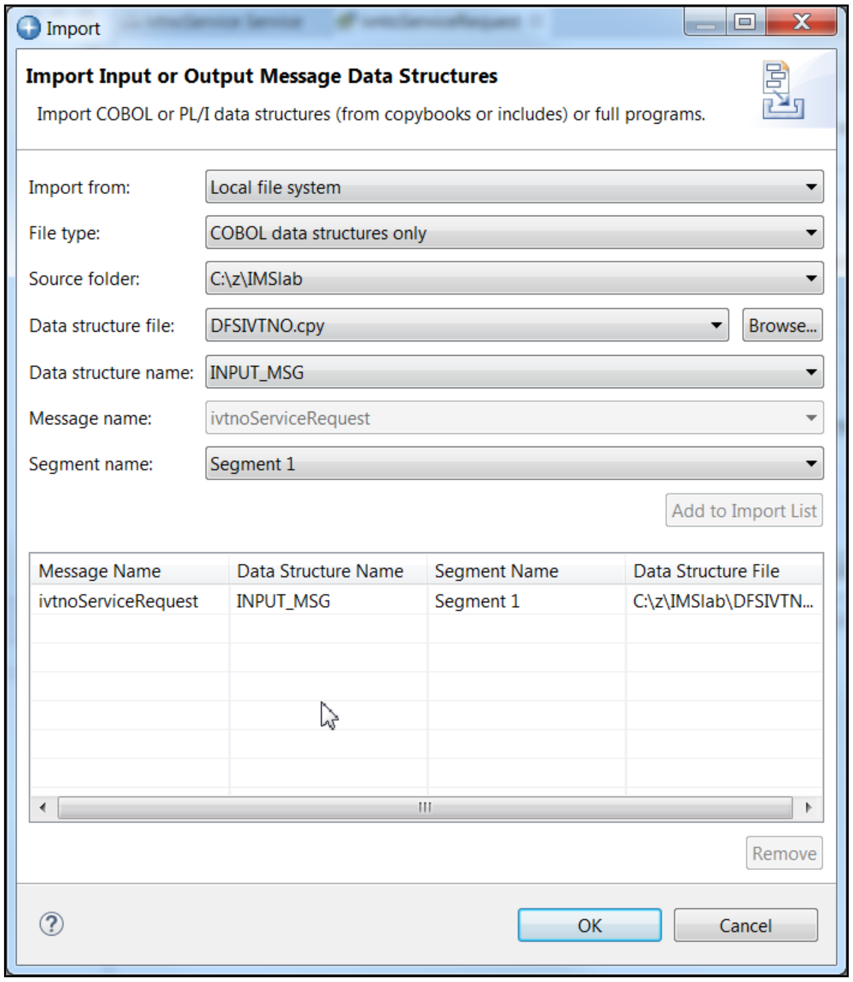
The first step is to import the COBOL copy book that provides the COBOL lay out of the input and output messages. As a reminder the COBOL source is shown below.

```
       * DATA AREA FOR TERMINAL INPUT

        01  INPUT-MSG.
            02  IN-LL           PICTURE S9(3) COMP.
            02  IN-ZZ           PICTURE S9(3) COMP.
            02  IN-TRANCDE      PICTURE X(10).
            02  IN-COMMAND      PICTURE X(8).
            02  IN-LAST-NAME    PICTURE X(10).
            02  IN-FIRST-NAME   PICTURE X(10).
            02  IN-EXTENSION    PICTURE X(10).
            02  IN-ZIP-CODE     PICTURE X(7).

       * DATA AREA OUTPUT
        01  OUTPUT-AREA.
            02  OUT-LL        PICTURE S9(3) COMP VALUE +95.
            02  OUT-ZZ        PICTURE S9(3) COMP VALUE +0.
            02  OUT-MESSAGE    PIC X(40).
            02  OUT-COMMAND    PIC X(8).
            02  OUT-LAST-NAME    PIC X(10).
            02  OUT-FIRST-NAME   PIC X(10).
            02  OUT-EXTENSION    PIC X(10).
            02  OUT-ZIP-CODE     PIC X(7).
            02  OUT-SEGMENT-NO   PICTURE X(4) VALUE '0001'.
```

___8. On the *Service Interface Definition* window, there is a tool bar near the top. If you hover over an icon its function will be display as below. Click the *Import COBOL or PL/I data structure into the service interface* icon to start the import process.


*ivtnoAddService Service
ivtnoAddRequest
Service Interface Editor
Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.
Search:
Import COBOL or PL/I data structures into the service interface
Fields
Include
Interface
Type
ivtnoAddRequest

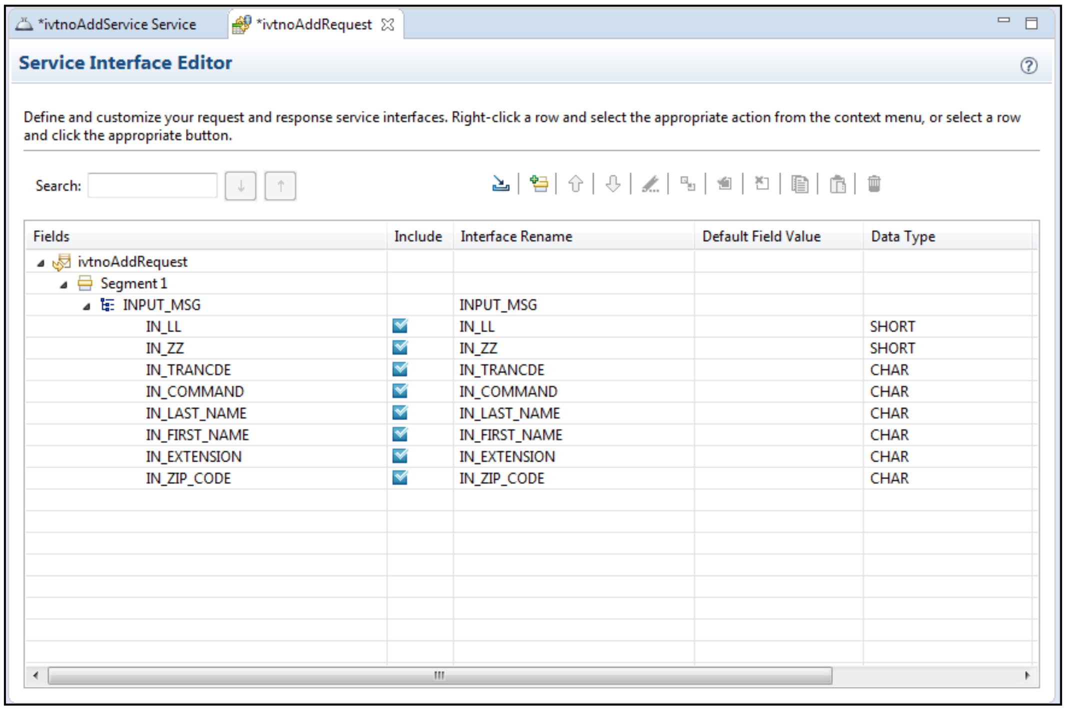
___9. This will open the *Import* window. On this window select *Local file system* as source of the import and *COBOL data structure only* as the *File type*. Press the **Browse** button and **Open** directory *C:\z\IMSlab* and then select file *DFSIVTNO*.cpy and click **Open** to import this file into this project. Use the pull down allow to select *INPT_MSG* as the *Data Structure name* and then click the **Add to Import List** button to continue.


Import
Import Input or Output Message Data Structures
Import COBOL or PL/I data structures (from copybooks or includes) or full programs.
Browse...
Add to Import List
Remove
OK
Cancel

Import from: Local file system
File type: COBOL data structures only
Source folder: C:\z\IMSlab
Data structure file: DFSIVTNO.cpy
Data structure name: INPUT_MSG
Message name: ivtnoServiceRequest
Segment name: Segment 1

Message Name	Data Structure Name	Segment Name	Data Structure File
ivtnoServiceRequest	INPUT_MSG	Segment 1	C:\z\IMSlab\DFSIVTN...

___10. Click **OK** and when you expand *INPUT_MSG* you will see the COBOL 'variables' that have been imported into the service project as interface fields.



Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type
ivtnoAddRequest				
Segment 1				
INPUT_MSG		INPUT_MSG		
IN_LL	✓	IN_LL		SHORT
IN_ZZ	✓	IN_ZZ		SHORT
IN_TRANCDE	✓	IN_TRANCDE		CHAR
IN_COMMAND	✓	IN_COMMAND		CHAR
IN_LAST_NAME	✓	IN_LAST_NAME		CHAR
IN_FIRST_NAME	✓	IN_FIRST_NAME		CHAR
IN_EXTENSION	✓	IN_EXTENSION		CHAR
IN_ZIP_CODE	✓	IN_ZIP_CODE		CHAR

N.B. the interface names were derived from the COBOL source shown earlier.

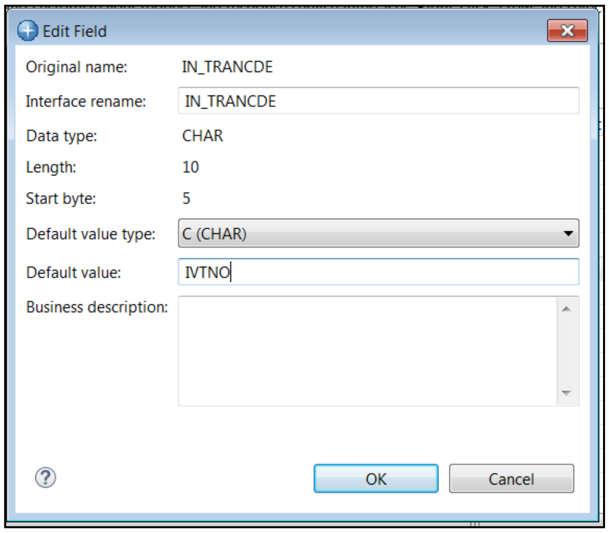
___11. In this window, you can edit and change the property name (e.g. *Interface name*) or exclude specific fields entirely from the interface. Either can be done by selecting a field and right mouse button clicking or by selecting a field and using the desired tool icon in the Service Interface toolbar. Let's try both techniques to remove the FILLER fields.

___12. Select field *IN_LL* and right mouse button click and select the *Exclude field from interface* option on the list of options.

___13. Next select field *IN_ZZ* and use the *Exclude selected fields(s) from the interface* tool icon.

___14. Notice that the check boxes besides these two fields are now unchecked. (You could have simply unchecked the box to accomplish the same results.)

___15. Next select field *IN_TRANCDE*. Exclude this field from interface and set a default value for this field to ***IVTNO*** using the right mouse button technique or the *Edit select field* icon in the tool bar.

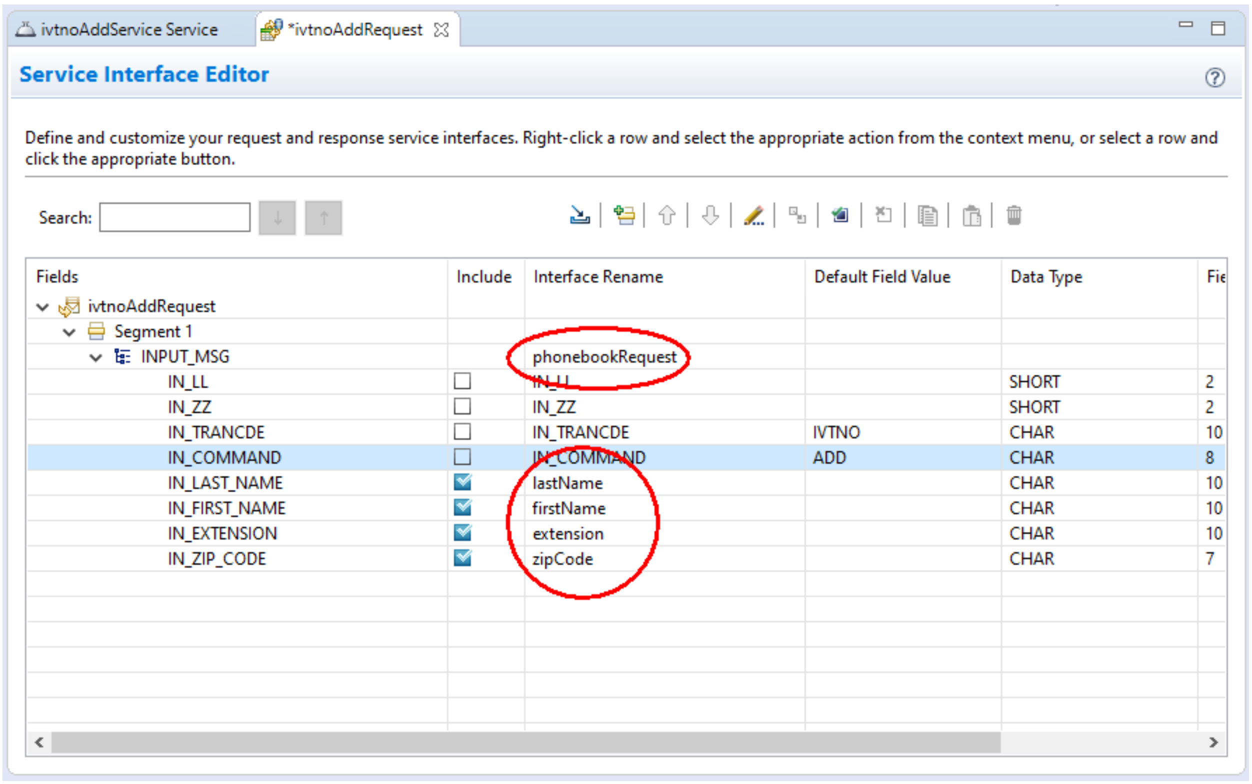


Original name:	IN_TRANCDE
Interface rename:	IN_TRANCDE
Data type:	CHAR
Length:	10
Start byte:	5
Default value type:	C (CHAR)
Default value:	IVTNO
Business description:	

___16. Select field IN_COMMAND and exclude this field from the interface and set a default value for this filed to ***ADD*** using the same technique as above.

___17. Next change the interface names of the remaining fields from *IN_LAST_NAME* to ***lastName,*** from *IN_FIRST_NAME* to ***firstName,*** from *IN_EXTENSION* to ***extension*** and from *IN_ZIP_CODE* to ***zipCode.*** Finally change the interface name of *INPUT_MSG* to ***phonebookRequest***.

___18. When finished your service definition interface should look like this.

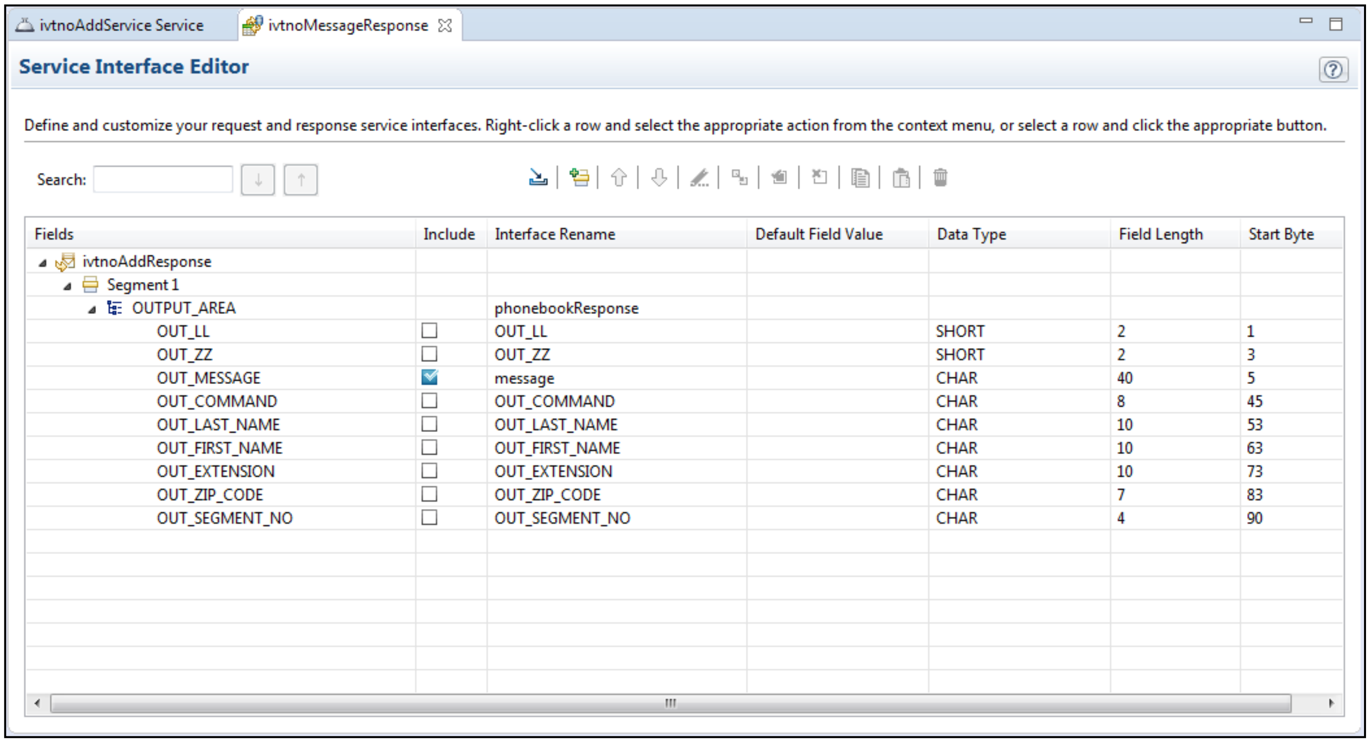


Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type	Fie
ivtnoAddRequest					
Segment 1					
INPUT_MSG		phonebookRequest			
IN_LL	☐	IN_LL		SHORT	2
IN_ZZ	☐	IN_ZZ		SHORT	2
IN_TRANCDE	☐	IN_TRANCDE	IVTNO	CHAR	10
IN_COMMAND	☐	IN_COMMAND	ADD	CHAR	8
IN_LAST_NAME	☑	lastName		CHAR	10
IN_FIRST_NAME	☑	firstName		CHAR	10
IN_EXTENSION	☑	extension		CHAR	10
IN_ZIP_CODE	☑	zipCode		CHAR	7

___19. Close the *Service Interface Definition* window by clicking on the white X in the tab being sure to save the changes. Note now that the *Request service interface* and the *Response service interface* areas have now been populated with *ivtnoAddRequest.si*. Also note that you can use their respective **Edit** buttons to return to the *Service Interface Definition* window for each interface.

___20. The default response service interface is not we want so we need to create another service interface named *ivtnoMessageResponse* for the output message. Start at step 6 above and repeat the steps but this time add the *OUTPUT_AREA* data structure to the service interface and remove all fields except for *OUT_MESSAGE.* Finally, change the interface name for field OUT_MESSAGE to ***message*** and the interface name for field *OUTPUT_AREA* to ***phonebookResponse***. When finished your service interface definition should look like this.

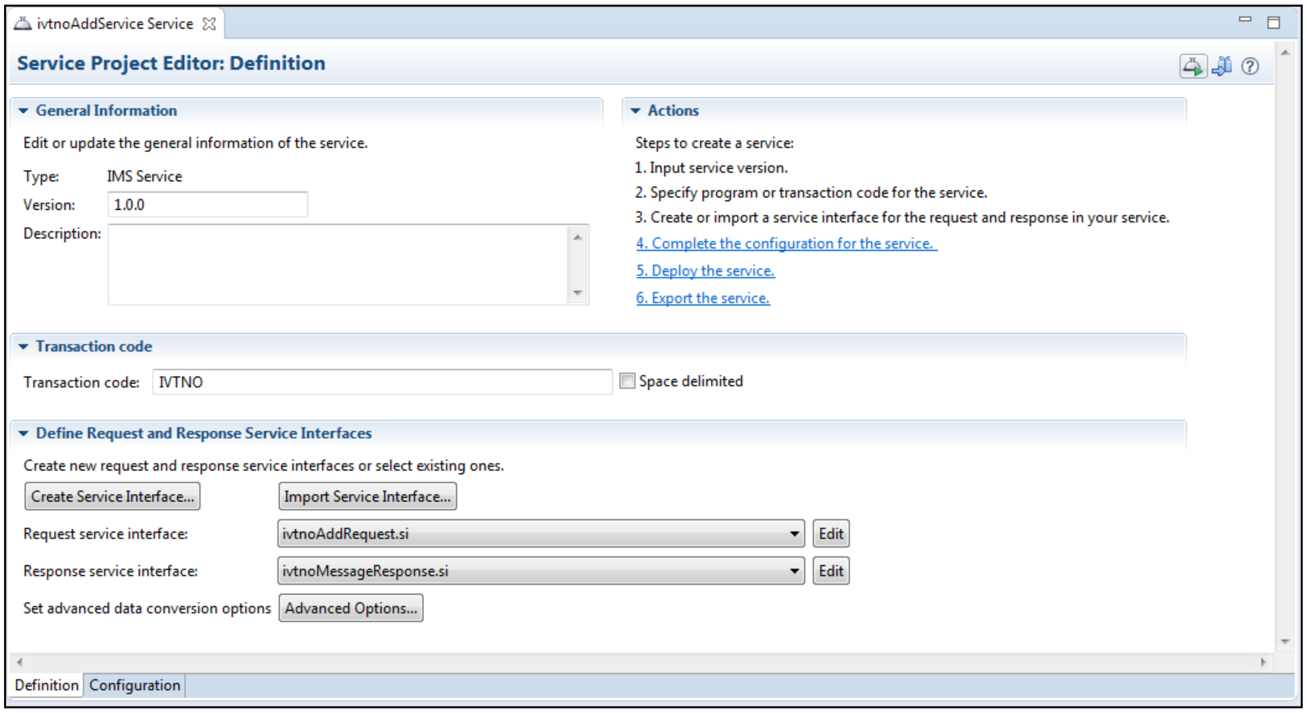


Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length	Start Byte
ivtnoAddResponse						
Segment 1						
OUTPUT_AREA		phonebookResponse				
OUT_LL	☐	OUT_LL		SHORT	2	1
OUT_ZZ	☐	OUT_ZZ		SHORT	2	3
OUT_MESSAGE	☑	message		CHAR	40	5
OUT_COMMAND	☐	OUT_COMMAND		CHAR	8	45
OUT_LAST_NAME	☐	OUT_LAST_NAME		CHAR	10	53
OUT_FIRST_NAME	☐	OUT_FIRST_NAME		CHAR	10	63
OUT_EXTENSION	☐	OUT_EXTENSION		CHAR	10	73
OUT_ZIP_CODE	☐	OUT_ZIP_CODE		CHAR	7	83
OUT_SEGMENT_NO	☐	OUT_SEGMENT_NO		CHAR	4	90

___21. Close the *Service Interface Definition* window.

___22. Set the *Response service interface* to *ivtnoMessageResponse.si* as shown below.



ivtnoAddService Service

Service Project Editor: Definition

General Information

Edit or update the general information of the service.

Type: IMS Service

Version: 1.0.0

Description:

Actions

Steps to create a service:

1. Input service version.
2. Specify program or transaction code for the service.
3. Create or import a service interface for the request and response in your service.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Transaction code

Transaction code: IVTNO ☐ Space delimited

Define Request and Response Service Interfaces

Create new request and response service interfaces or select existing ones.

Create Service Interface... Import Service Interface...

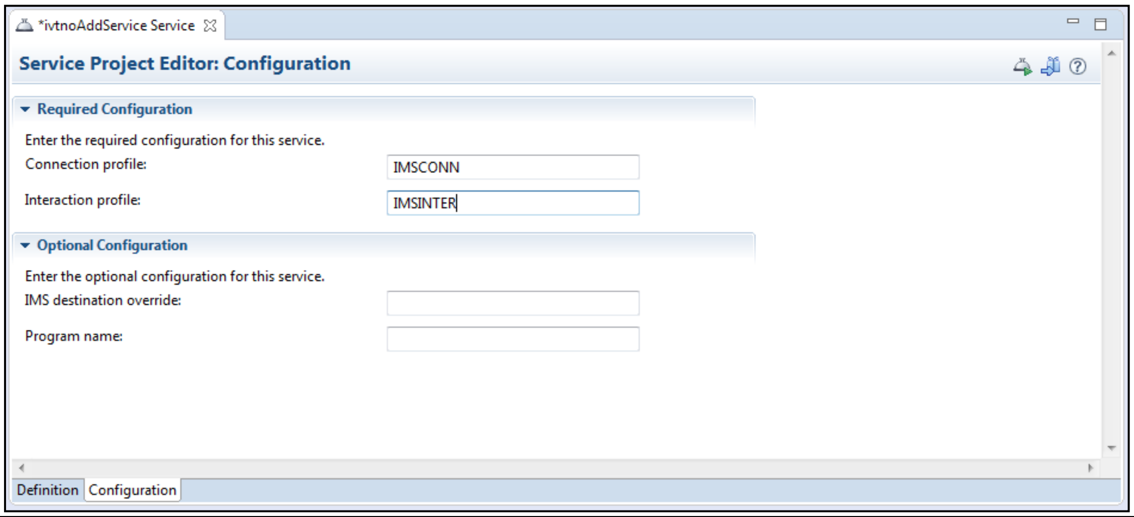
Request service interface: ivtnoAddRequest.si Edit

Response service interface: ivtnoMessageResponse.si Edit

Set advanced data conversion options Advanced Options...

Definition Configuration

___23. Next, we need to identify a connection profile and interaction properties profile that will be used. Click on the Configuration tab at the bottom of the *Overview* window to display the *Configuration* window. Enter ***IMSCONN*** in the area beside *Connection profile* and ***IMSINTER*** in the area beside *Interaction properties profile*.



*ivtnoAddService Service

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection profile: IMSCONN

Interaction profile: IMSINTER

Optional Configuration

Enter the optional configuration for this service.

IMS destination override:

Program name:

Definition Configuration

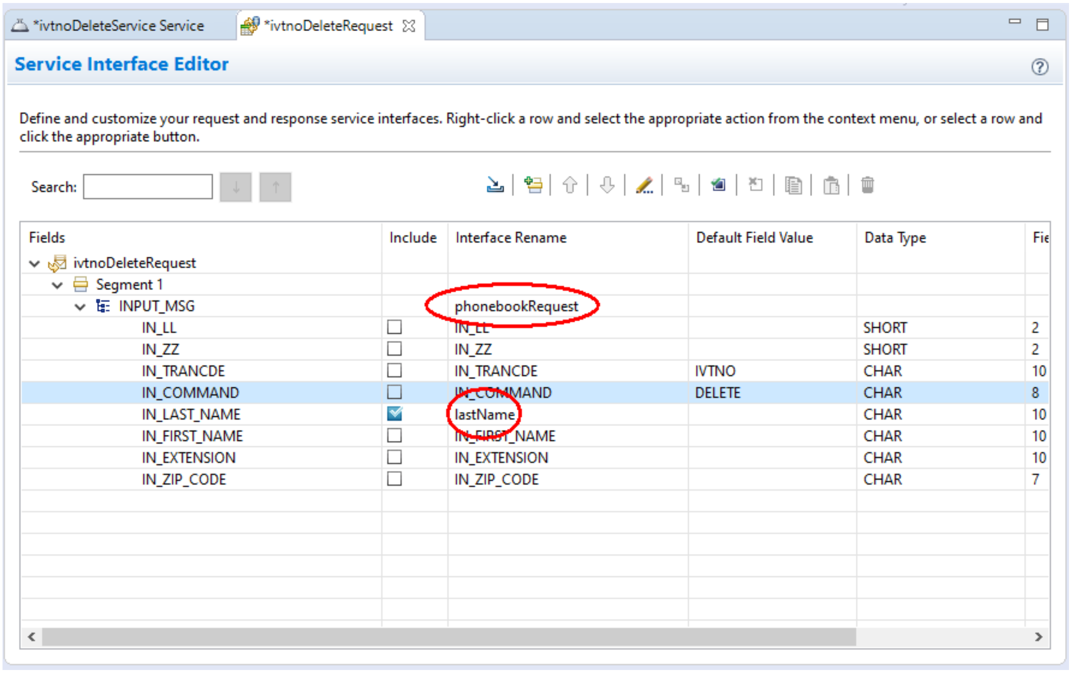
Tech-Tip: These values corresponds to the name provided for the connection and interaction earlier in the exercise.

___24. Save the *ivtnoAddService* service either by closing the tab or using the **Ctrl-S** key sequence.

Create the "Delete" Service

___25. Use Steps 1 through 19 to create service *ivtnoDeleteService.* In the case the last name is the only field exposed in the request message as ***lastName***. The command is set to ***DELETE*** and the transaction code is set to ***IVTNO*** with both fields omitted from the interface. Finally change the interface name of *INPUT_MSG* to ***phonebookRequest***.

- When finished the *ivntoDeleteRequest* service interface should look like:



*ivtnoDeleteService Service | *ivtnoDeleteRequest

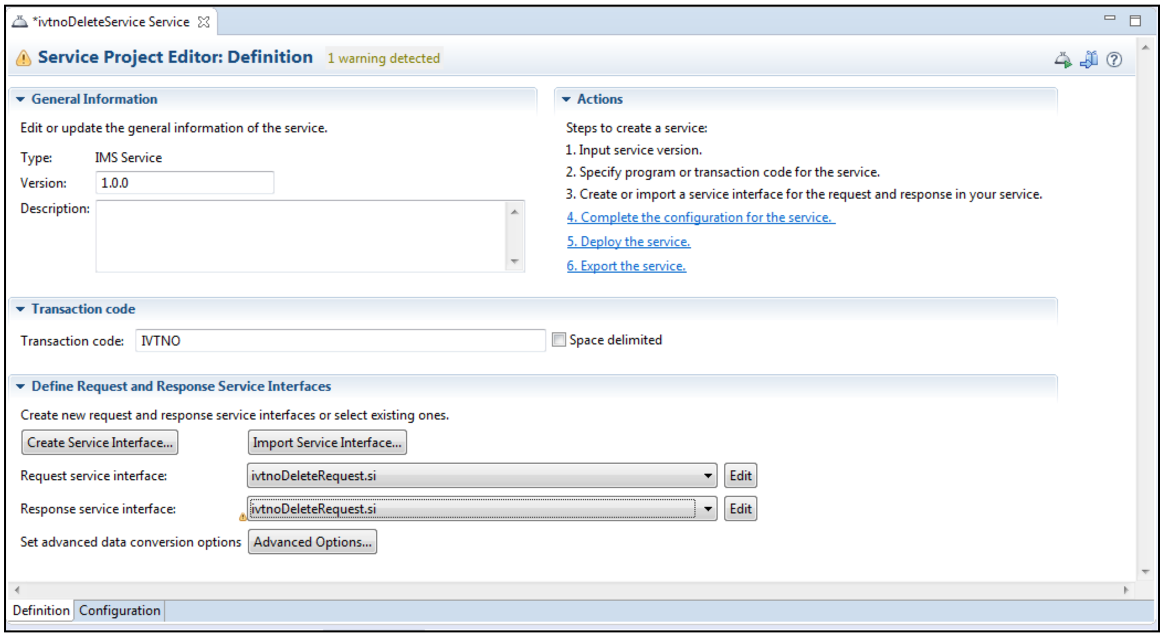
Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Search:

Fields	Include	Interface Rename	Default Field Value	Data Type	Fie
ivtnoDeleteRequest					
Segment 1					
INPUT_MSG		phonebookRequest			
IN_LL	☐	IN_LL		SHORT	2
IN_ZZ	☐	IN_ZZ		SHORT	2
IN_TRANCDE	☐	IN_TRANCDE	IVTNO	CHAR	10
IN_COMMAND	☐	IN_COMMAND	DELETE	CHAR	8
IN_LAST_NAME	☑	lastName		CHAR	10
IN_FIRST_NAME	☐	IN_FIRST_NAME		CHAR	10
IN_EXTENSION	☐	IN_EXTENSION		CHAR	10
IN_ZIP_CODE	☐	IN_ZIP_CODE		CHAR	7

- And the service definition for the *ivtnoDeleteService* should look like:



*ivtnoDeleteService Service

Service Project Editor: Definition 1 warning detected

General Information

Edit or update the general information of the service.

Type: IMS Service
Version: 1.0.0
Description:

Actions

Steps to create a service:
1. Input service version.
2. Specify program or transaction code for the service.
3. Create or import a service interface for the request and response in your service.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Transaction code

Transaction code: IVTNO ☐ Space delimited

Define Request and Response Service Interfaces

Create new request and response service interfaces or select existing ones.

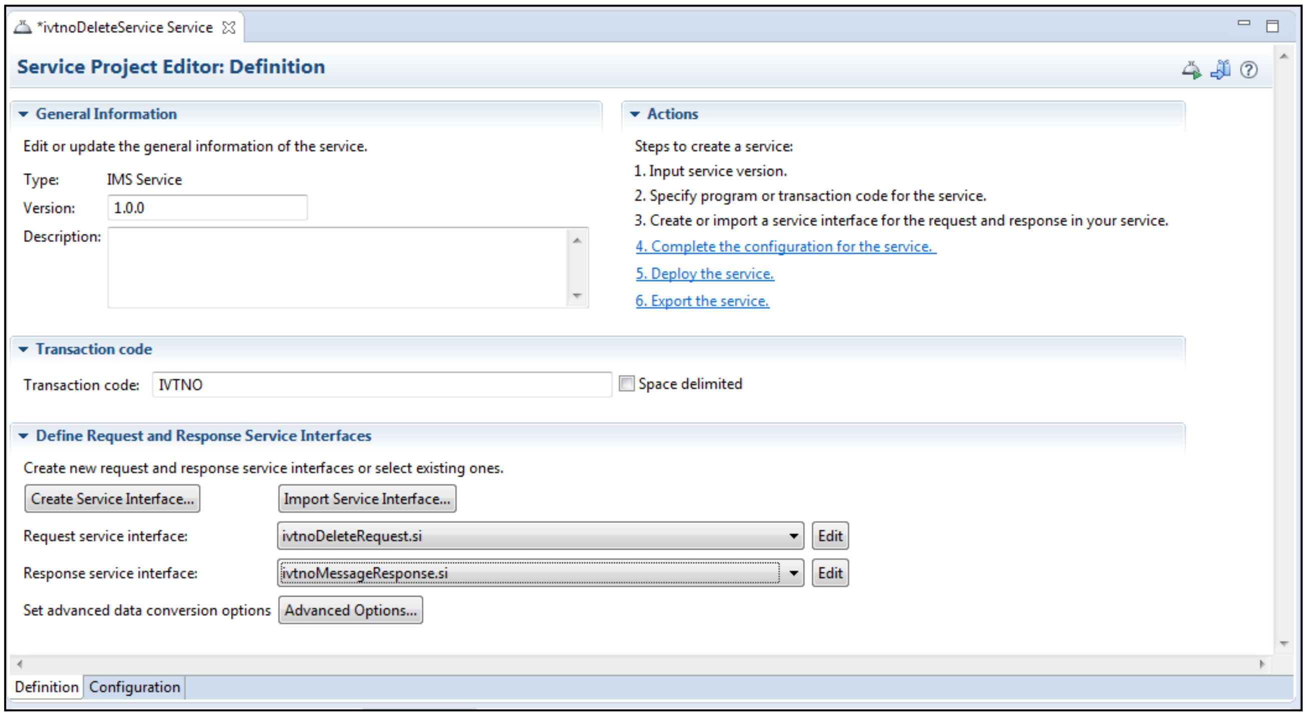
Create Service Interface... Import Service Interface...

Request service interface: ivtnoDeleteRequest.si Edit
Response service interface: ivtnoDeleteRequest.si Edit
Set advanced data conversion options Advanced Options...

Definition | Configuration

___26. On the *Service Project Editor: Definition* view click the **Import Service Interface** button. Click the **Workspace** button on the *Import a service interface* screen. On the *Import Service Interface – Import Service Interface* window expand the *ivtnoAddService* project and then expand *service-interfaces* and then select *ivtnoMessageResponse.si*. Click **OK** to import this service interface into this service project.

___27. On the *Service Project Editor: Definition* view use the pull-down arrow beside the area for *Response service interface* and select service interface *ivtnoMessageResponse.si*, see below:


*ivtnoDeleteService Service

Service Project Editor: Definition

General Information

Edit or update the general information of the service.

Type: IMS Service

Version: 1.0.0

Description:

Actions

Steps to create a service:

1. Input service version.
2. Specify program or transaction code for the service.
3. Create or import a service interface for the request and response in your service.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Transaction code

Transaction code: IVTNO ☐ Space delimited

Define Request and Response Service Interfaces

Create new request and response service interfaces or select existing ones.

Create Service Interface... Import Service Interface...

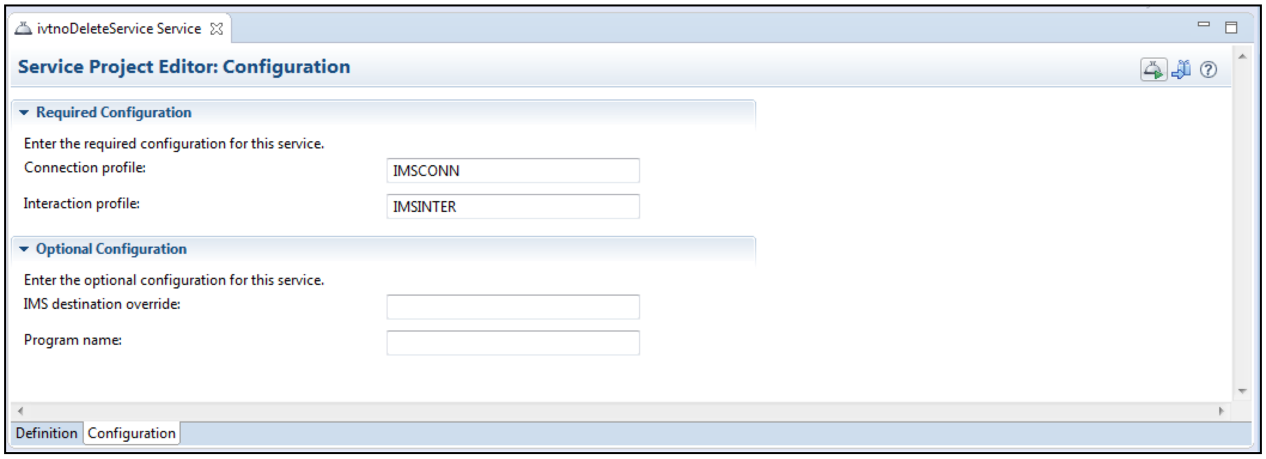
Request service interface: ivtnoDeleteRequest.si Edit

Response service interface: ivtnoMessageResponse.si Edit

Set advanced data conversion options Advanced Options...

Definition Configuration

___28. Next, we need to identify a connection profile and interaction properties profile that will be used. Click on the Configuration tab at the bottom of the *Overview* window to display the *Configuration* window. Enter ***IMSCONN*** in the area beside *Connection profile* and ***IMSINTER*** in the area beside *Interaction properties profile*.


ivtnoDeleteService Service

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection profile: IMSCONN

Interaction profile: IMSINTER

Optional Configuration

Enter the optional configuration for this service.

IMS destination override:

Program name:

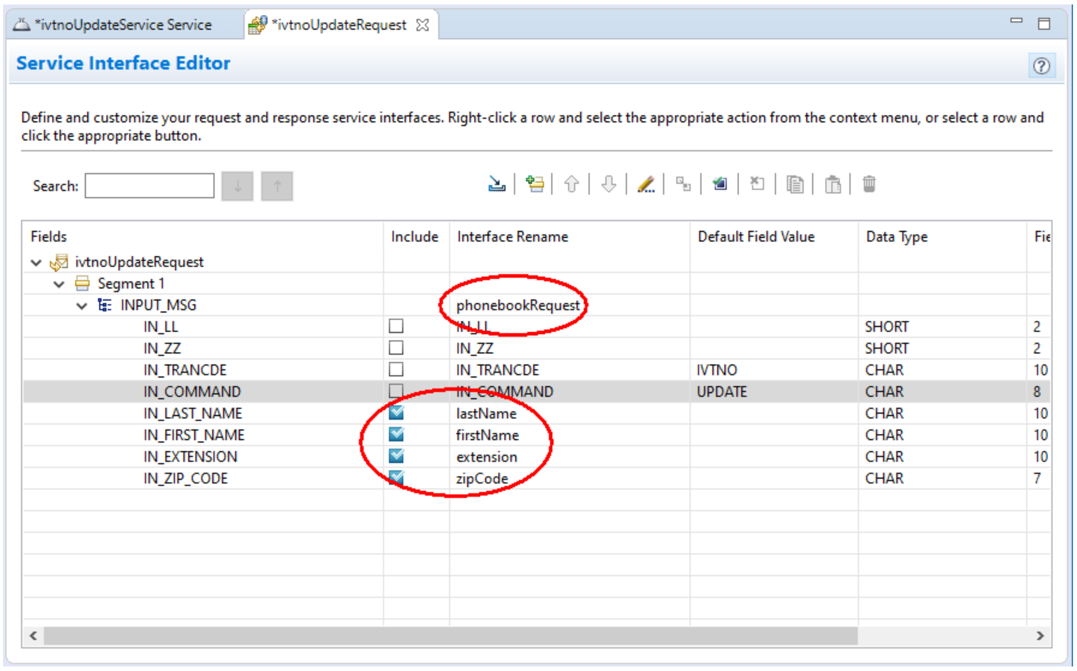
Definition Configuration

___29. Save the *ivtnoDeleteService* service either by closing the tab or using the **Ctrl-S** key sequence.

Create the "Update" Service

___30. Use Steps 1 through 19 to create service *ivtnoUpdateService.* The command is set to ***UPDATE*** and the transaction code is set to ***IVTNO*** with both fields omitted from the interface. Next change the interface names of the remaining fields from *IN_LAST_NAME* to ***lastName,*** from *IN_FIRST_NAME* to ***firstName,*** from *IN_EXTENSION* to ***extension*** and from *IN_ZIP_CODE* to ***zipCode.*** Finally change the interface name of *INPUT_MSG* to ***phonebookRequest***.

- The request message for the *ivtnoUpdateRequest* should look like this.


*ivtnoUpdateService Service | *ivtnoUpdateRequest

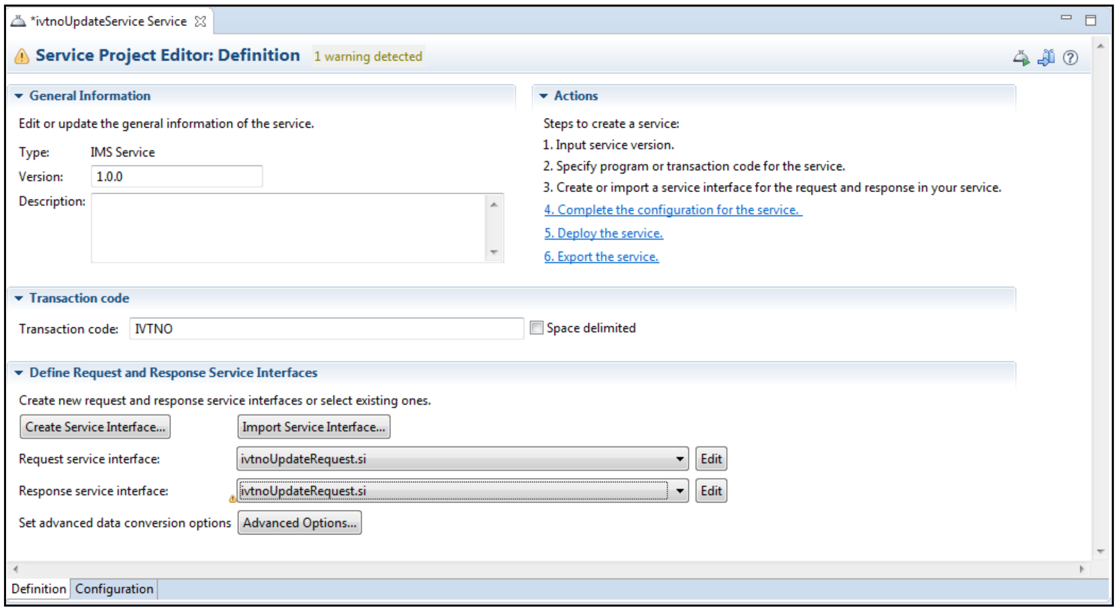
Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Search:

Fields	Include	Interface Rename	Default Field Value	Data Type	Fie
ivtnoUpdateRequest					
Segment 1					
INPUT_MSG		phonebookRequest			
IN_LL	☐	IN_LL		SHORT	2
IN_ZZ	☐	IN_ZZ		SHORT	2
IN_TRANCDE	☐	IN_TRANCDE	IVTNO	CHAR	10
IN_COMMAND	☐	IN_COMMAND	UPDATE	CHAR	8
IN_LAST_NAME	☑	lastName		CHAR	10
IN_FIRST_NAME	☑	firstName		CHAR	10
IN_EXTENSION	☑	extension		CHAR	10
IN_ZIP_CODE	☑	zipCode		CHAR	7

- When finished the service definition for the *ivtnoUpdateService* should look like this:


*ivtnoUpdateService Service

Service Project Editor: Definition 1 warning detected

General Information

Edit or update the general information of the service.

Type: IMS Service

Version: 1.0.0

Description:

Actions

Steps to create a service:
1. Input service version.
2. Specify program or transaction code for the service.
3. Create or import a service interface for the request and response in your service.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Transaction code

Transaction code: IVTNO ☐ Space delimited

Define Request and Response Service Interfaces

Create new request and response service interfaces or select existing ones.

Create Service Interface... Import Service Interface...

Request service interface: ivtnoUpdateRequest.si Edit

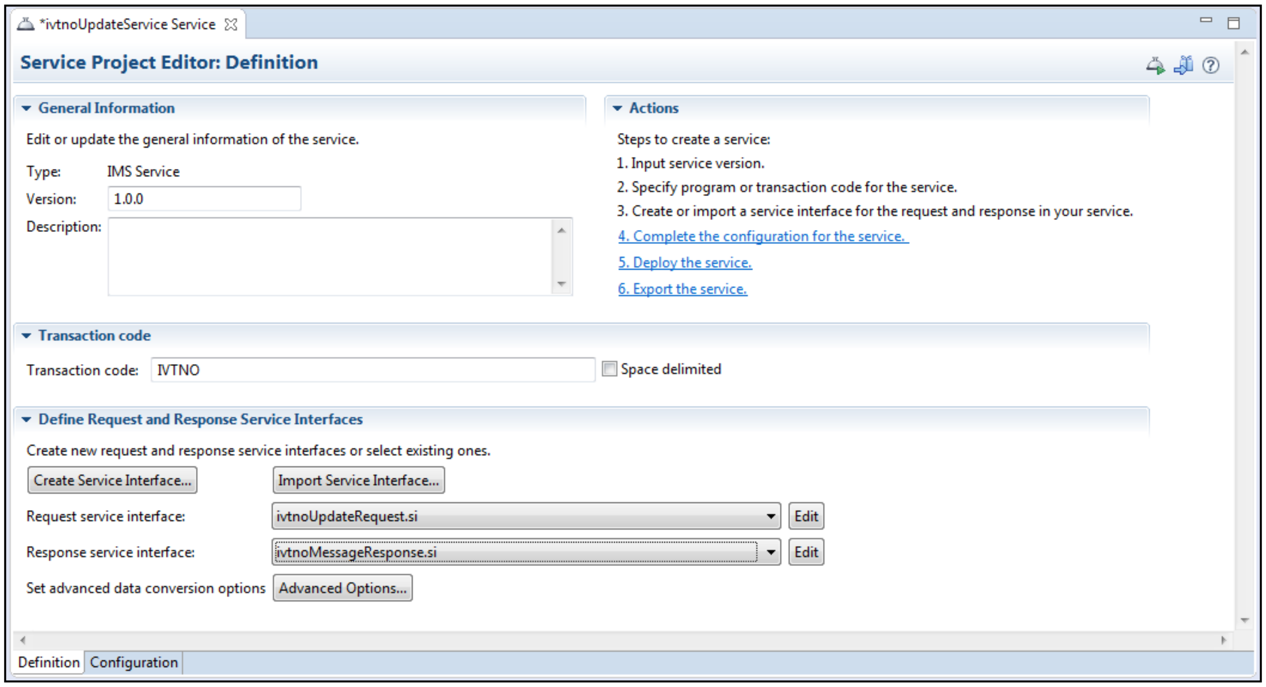
Response service interface: ivtnoUpdateRequest.si Edit

Set advanced data conversion options Advanced Options...

Definition | Configuration

___31. On the *Service Project Editor: Definition* view click the **Import Service Interface** button. Click the **Workspace** button on the *Import a service interface* screen. On the *Import Service Interface – Import Service Interface* window expand the *ivtnoAddService* project and then expand *service-interfaces* and then select *ivtnoMessageResponse.si*. Click **OK** to import this service interface into this service project.

___32. On the *Service Project Editor: Definition* view use the pull-down arrow beside the area for *Response service interface* and select service interface *ivtnoMessageResponse.si*, see below:


*ivtnoUpdateService Service

Service Project Editor: Definition

General Information
Edit or update the general information of the service.
Type: IMS Service
Version: 1.0.0
Description:

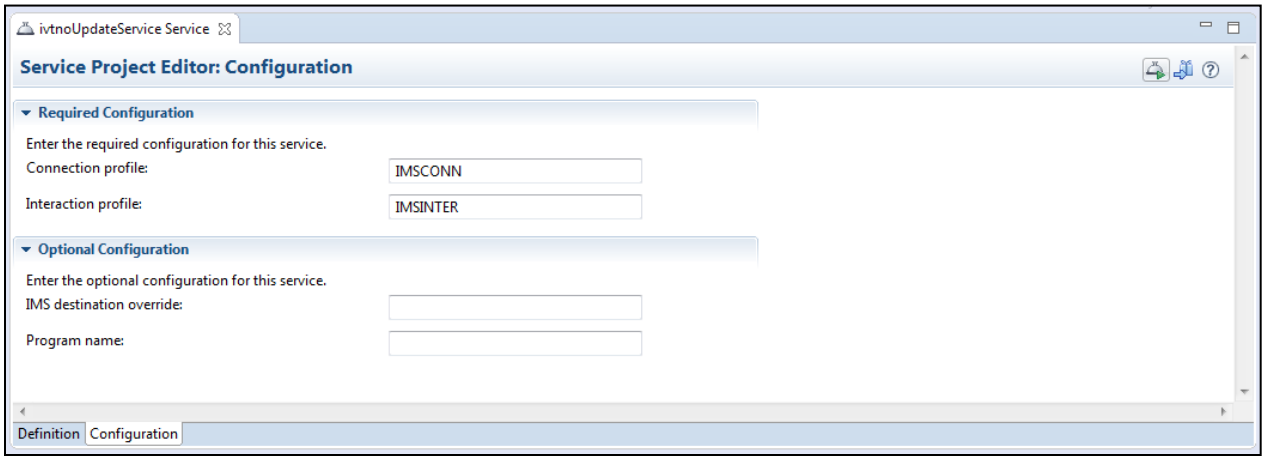
Actions
Steps to create a service:
1. Input service version.
2. Specify program or transaction code for the service.
3. Create or import a service interface for the request and response in your service.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Transaction code
Transaction code: IVTNO
Space delimited

Define Request and Response Service Interfaces
Create new request and response service interfaces or select existing ones.
Create Service Interface... Import Service Interface...
Request service interface: ivtnoUpdateRequest.si Edit
Response service interface: ivtnoMessageResponse.si Edit
Set advanced data conversion options Advanced Options...

Definition Configuration

___33. Next, we need to identify a connection profile and interaction properties profile that will be used. Click on the Configuration tab at the bottom of the *Overview* window to display the *Configuration* window. Enter ***IMSCONN*** in the area beside *Connection profile* and ***IMSINTER*** in the area beside *Interaction properties profile.*


ivtnoUpdateService Service

Service Project Editor: Configuration

Required Configuration
Enter the required configuration for this service.
Connection profile: IMSCONN
Interaction profile: IMSINTER

Optional Configuration
Enter the optional configuration for this service.
IMS destination override:
Program name:

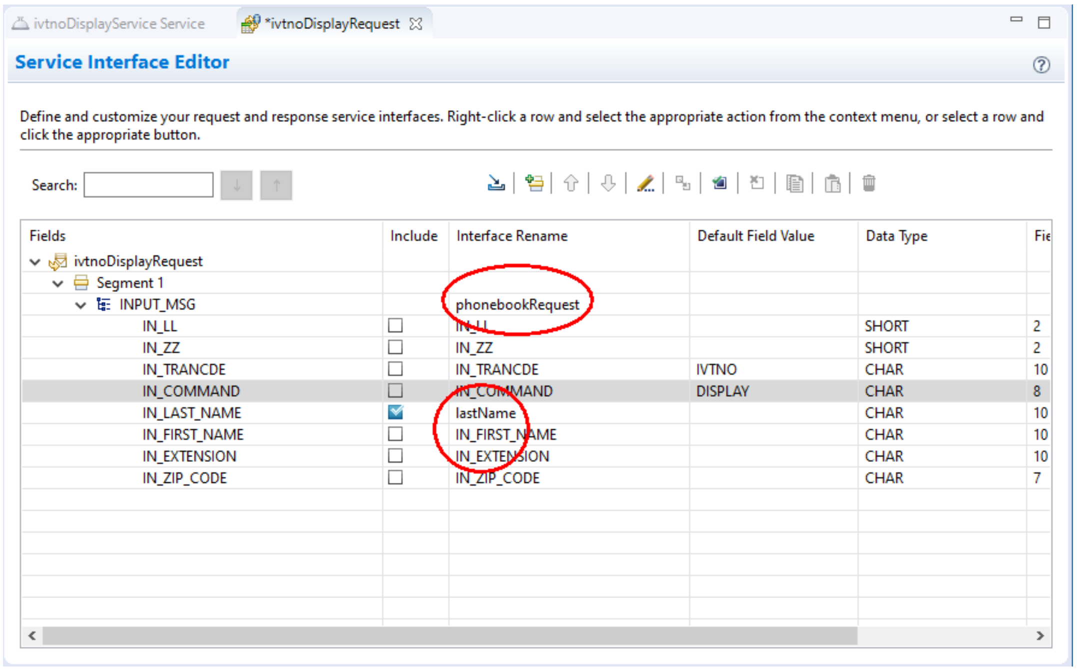
Definition Configuration

___34. Save the *ivtnoUpdateService* service either by closing the tab or using the **Ctrl-S** key sequence.

Create the "Display" Service

___35. Use Steps 1 through 19 to create service *ivtnoDisplayService.* In the case the last name is the only field exposed in the request message as ***lastName***. The command is set to ***DISPLAY*** and the transaction code is set to ***IVTNO*** with both fields omitted from the interface. Finally change the interface name of *INPUT_MSG* to ***phonebookRequest***.

- When finished the *ivntoDisplayRequest* service interface should look like this:



ivtnoDisplayService Service | *ivtnoDisplayRequest

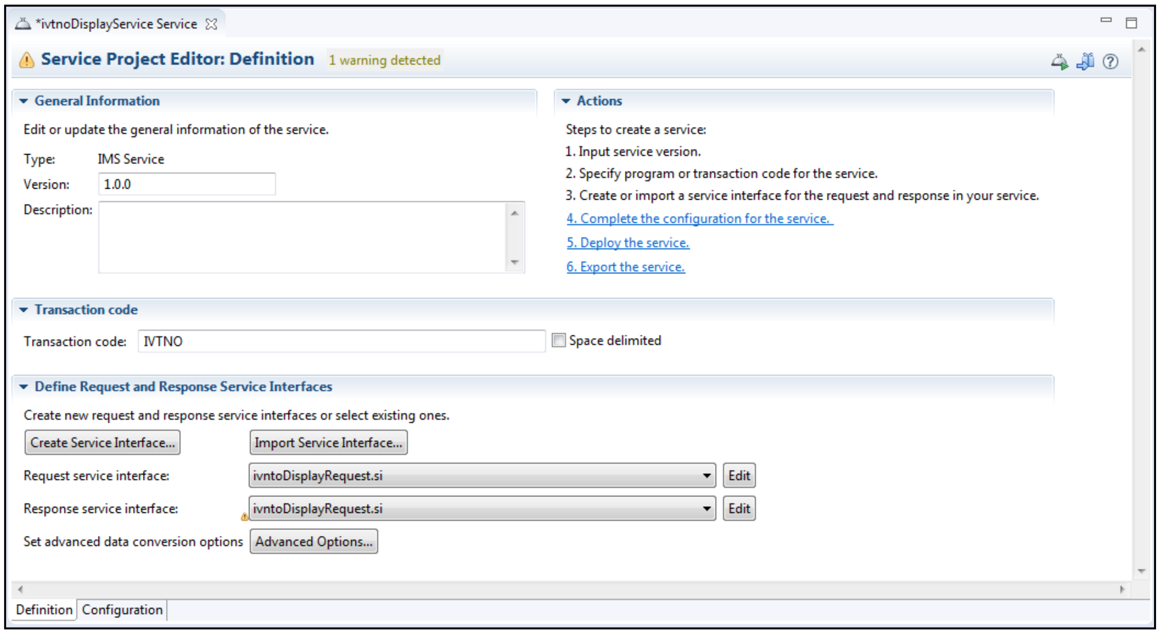
Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Search:

Fields	Include	Interface Rename	Default Field Value	Data Type	Fie
ivtnoDisplayRequest					
Segment 1					
INPUT_MSG		phonebookRequest			
IN_LL	☐	IN_LL		SHORT	2
IN_ZZ	☐	IN_ZZ		SHORT	2
IN_TRANCDE	☐	IN_TRANCDE	IVTNO	CHAR	10
IN_COMMAND	☐	IN_COMMAND	DISPLAY	CHAR	8
IN_LAST_NAME	☑	lastName		CHAR	10
IN_FIRST_NAME	☐	IN_FIRST_NAME		CHAR	10
IN_EXTENSION	☐	IN_EXTENSION		CHAR	10
IN_ZIP_CODE	☐	IN_ZIP_CODE		CHAR	7

- When finished the service definition for the *ivtnoDisplayService* should look like this:



*ivtnoDisplayService Service

Service Project Editor: Definition 1 warning detected

General Information

Edit or update the general information of the service.

Type: IMS Service

Version: 1.0.0

Description:

Actions

Steps to create a service:

1. Input service version.
2. Specify program or transaction code for the service.
3. Create or import a service interface for the request and response in your service.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Transaction code

Transaction code: IVTNO ☐ Space delimited

Define Request and Response Service Interfaces

Create new request and response service interfaces or select existing ones.

Create Service Interface... Import Service Interface...

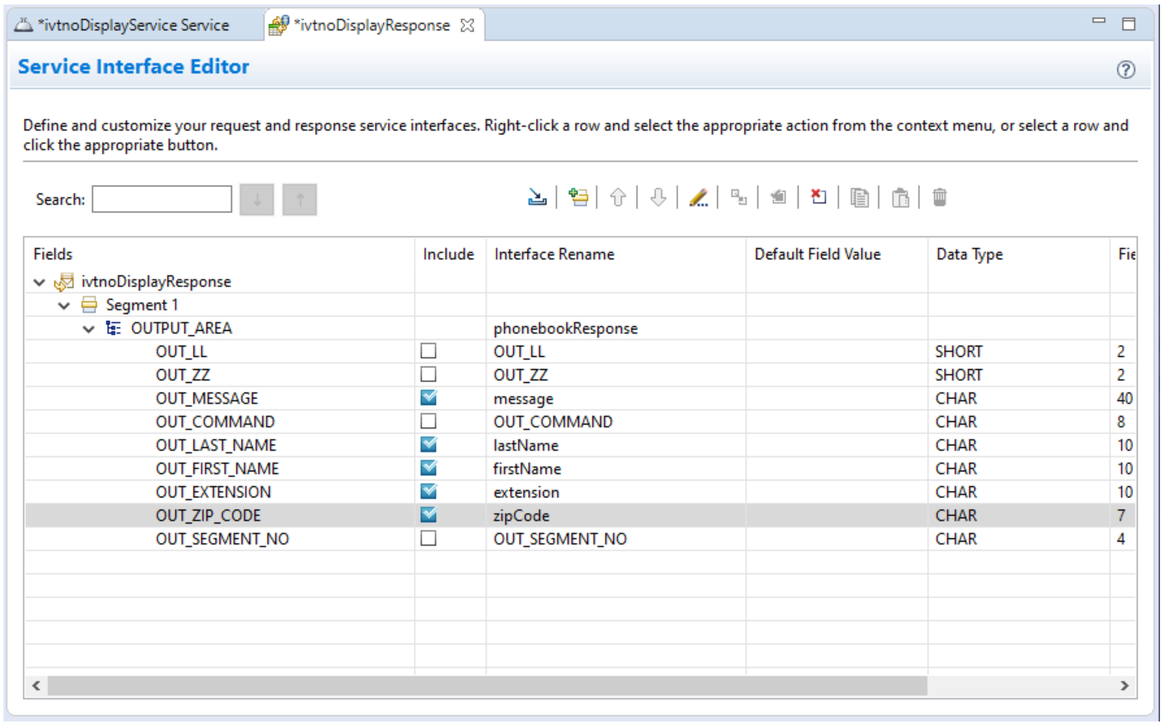
Request service interface: ivntoDisplayRequest.si Edit

Response service interface: ivntoDisplayRequest.si Edit

Set advanced data conversion options Advanced Options...

Definition | Configuration

___36. Again, the default response service interface is not we want so we need to create another service interface named *ivtnoDisplayResponse* that will include the contact information in the response message. Start at step 6 above and repeat the steps but this time add the *OUTPUT_AREA* data structure to the service interface and remove all fields from the interface except for *OUT_MESSAGE, OUT_LAST_NAME, OUT_FIRST_NAME, OUT_EXTENSION* and *OUT_ZIP_CODE.* Finally change the interface names as shown below. When finished your service interface definition should look like this.

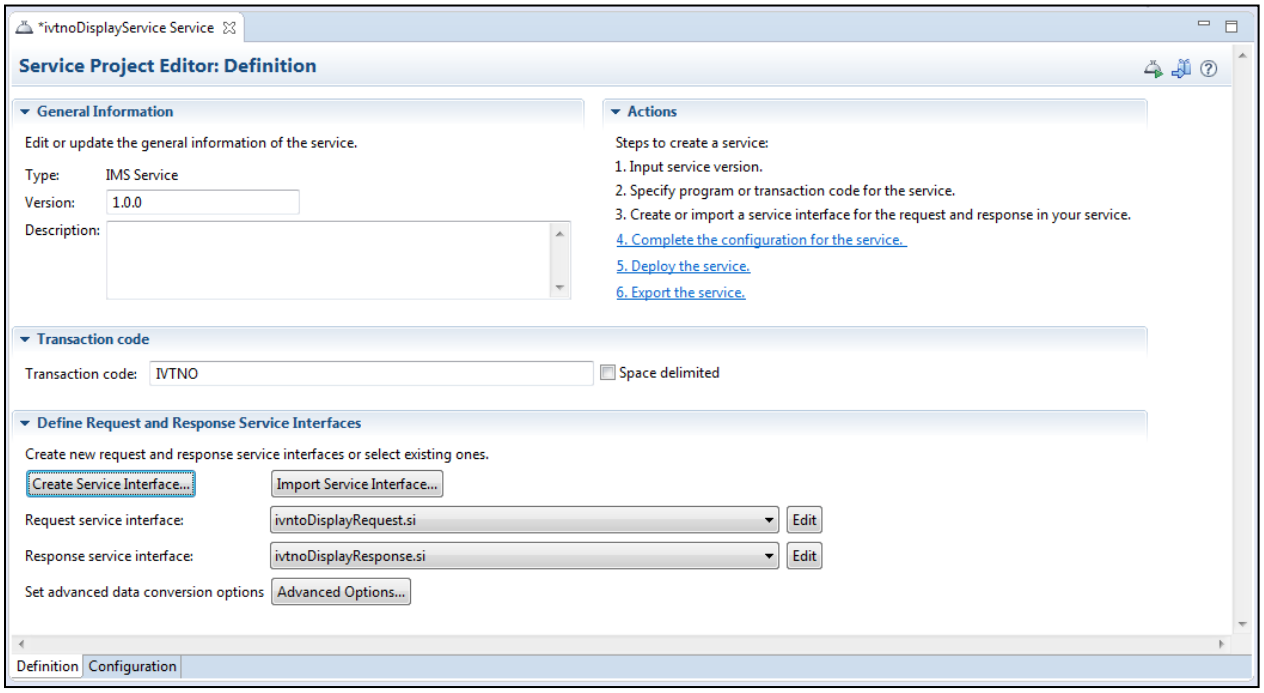


Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type	Fie
ivtnoDisplayResponse					
Segment 1					
OUTPUT_AREA		phonebookResponse			
OUT_LL	☐	OUT_LL		SHORT	2
OUT_ZZ	☐	OUT_ZZ		SHORT	2
OUT_MESSAGE	✓	message		CHAR	40
OUT_COMMAND	☐	OUT_COMMAND		CHAR	8
OUT_LAST_NAME	✓	lastName		CHAR	10
OUT_FIRST_NAME	✓	firstName		CHAR	10
OUT_EXTENSION	✓	extension		CHAR	10
OUT_ZIP_CODE	✓	zipCode		CHAR	7
OUT_SEGMENT_NO	☐	OUT_SEGMENT_NO		CHAR	4

___37. Close the *Service Interface Definition* window.

___38. Set the *Response service interface* to *inquireDisplayResponse.si* as shown below.


*ivtnoDisplayService Service

Service Project Editor: Definition

General Information

Edit or update the general information of the service.

Type: IMS Service

Version: 1.0.0

Description:

Actions

Steps to create a service:
1. Input service version.
2. Specify program or transaction code for the service.
3. Create or import a service interface for the request and response in your service.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Transaction code

Transaction code: IVTNO ☐ Space delimited

Define Request and Response Service Interfaces

Create new request and response service interfaces or select existing ones.

Create Service Interface... Import Service Interface...

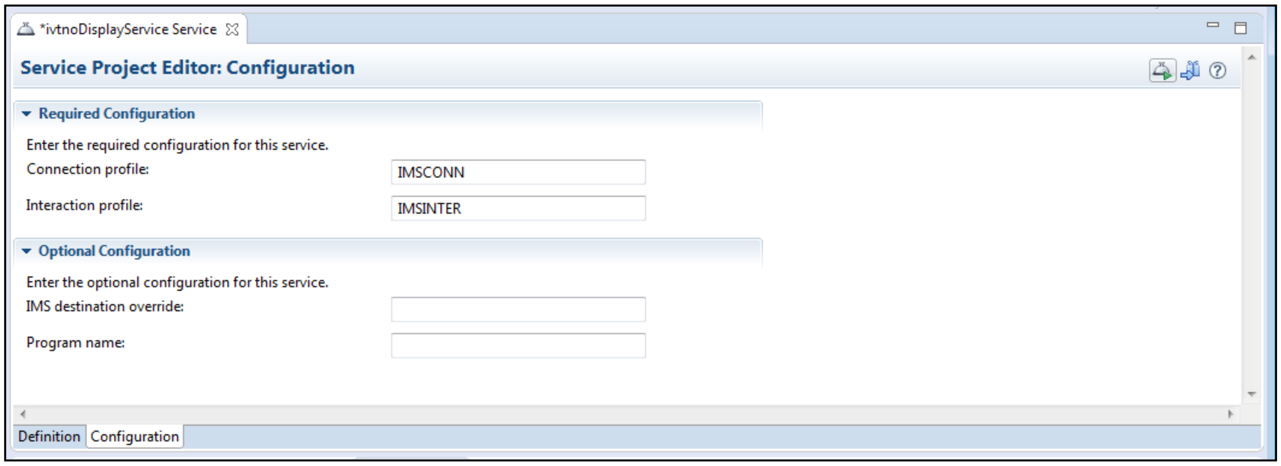
Request service interface: ivntoDisplayRequest.si Edit

Response service interface: ivtnoDisplayResponse.si Edit

Set advanced data conversion options Advanced Options...

Definition Configuration

___39. Next, we need to identify a connection profile and interaction properties profile that will be used. Click on the Configuration tab at the bottom of the *Overview* window to display the *Configuration* window. Enter ***IMSCONN*** in the area beside *Connection profile* and ***IMSINTER*** in the area beside *Interaction properties profile.*


*ivtnoDisplayService Service

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection profile: IMSCONN

Interaction profile: IMSINTER

Optional Configuration

Enter the optional configuration for this service.

IMS destination override:

Program name:

Definition Configuration

___40. Save the *ivtnoDisplayService* service either by closing the tab or using the **Ctrl-S** key sequence.

This services now needs to be made available for developing the API and for deployment to the z/OS Connect server.

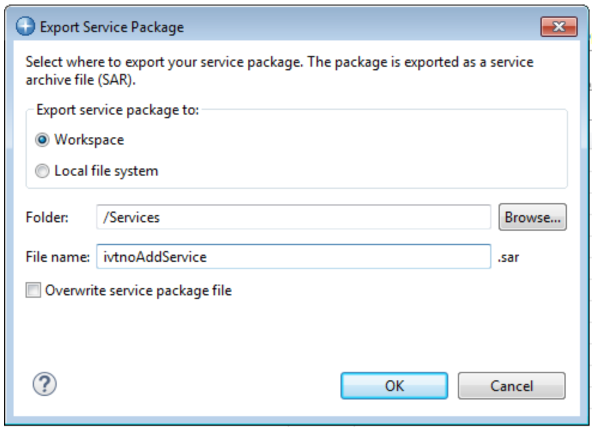
Export and deploy the Service Archive files

Before a Service Interface can be used it must be exported to create Service Archive (SAR) file). There are two uses for a SAR file. The first is for use in developing an API in the z/OS Connect Toolkit and the second is for deploying to a z/OS Connect server. This section describes the process for creating and exporting SAR files.

___1. First 'export' them into another project in the z/OS Connect Toolkit. Select ***File*** on the tool bar and then on the pop up select **New → Project**. Expand the *General* folder and select *Project* to create a target project for exporting the Service Archive (SAR) files. Click **Next** to continue.

___2. On the *New Project* window enter ***Services*** as the *Project name*. Click **Finish** to continue. This action will add a new project in the *Project Explorer* named *Services*. If this project already exists continue with Step 3.

___3. Select the *ivtnoAddService* service project and right mouse button click. On the pop-up selection select **z/OS Connect → Export z/OS Connect Service Archive**. On the *Export Services Package* window select the radio button beside *Workspace* and use the **Browse** button to select the *Services* folder. Click **OK** to continue.


Export Service Package

Select where to export your service package. The package is exported as a service archive file (SAR).

Export service package to:

[✓ Workspace] Local file system

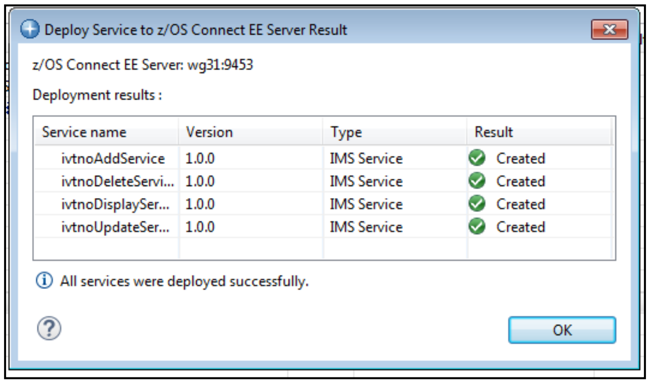
Folder: /Services

File name: ivtnoAddService .sar

Overwrite service package file

___4. Repeat this step to export the other 3 services, *ivtnoDeleteService, ivtnoDisplayService* and *ivtnoUpdateService*.

___5. Use **Shift** key to select all 4 services and right mouse button click again and, on the pop-up, select **z/OS Connect → Deploy Service to z/OS Connect Server**. On the *Deploy Service* window select the target server (*wg31:9453*) and click **OK** twice to continue.


Deploy Service to z/OS Connect EE Server Result

z/OS Connect EE Server: wg31:9453

Deployment results :

Service name	Version	Type	Result
ivtnoAddService	1.0.0	IMS Service	✓ Created
ivtnoDeleteServi...	1.0.0	IMS Service	✓ Created
ivtnoDisplaySer...	1.0.0	IMS Service	✓ Created
ivtnoUpdateSer...	1.0.0	IMS Service	✓ Created

All services were deployed successfully.

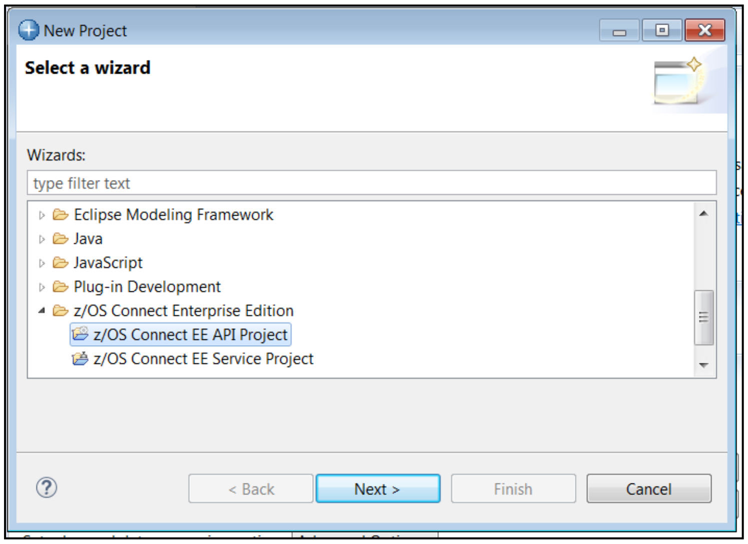
Compose API using z/OS Connect API Toolkit

The next step is to import the Service Archive (SAR) files into the API Project. But first the API project needs to be created.

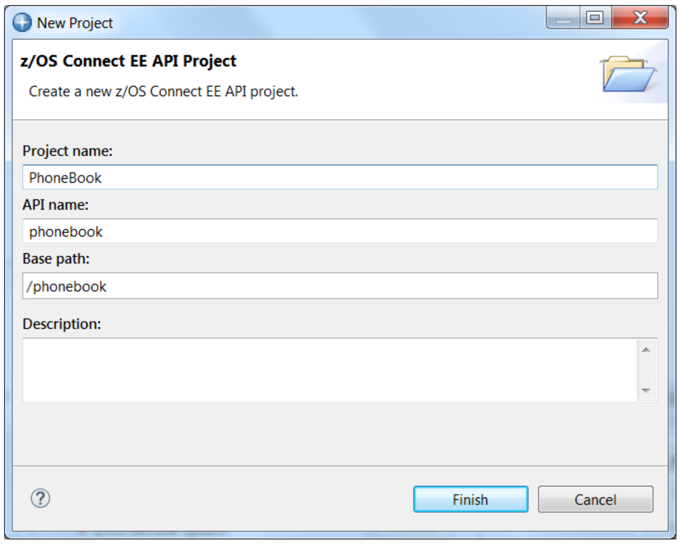
___1. From the menu bar select *File* → *New* and then *Other*.

___2. Then scroll down and locate the *z/OS Connect Enterprise Edition* folder, open that and select *z/OS Connect API Project* then click **Next**.

.


New Project
Select a wizard
Wizards:
type filter text
Eclipse Modeling Framework
Java
JavaScript
Plug-in Development
z/OS Connect Enterprise Edition
z/OS Connect EE API Project
z/OS Connect EE Service Project
< Back
Next >
Finish
Cancel

___3. For the project name values, enter ***PhoneBook*** for the *Project name*, ***phonebook*** for the *API name* and ***/phonebook*** for the *Base path*, then click **Finish**.



Create a new z/OS Connect EE API project.

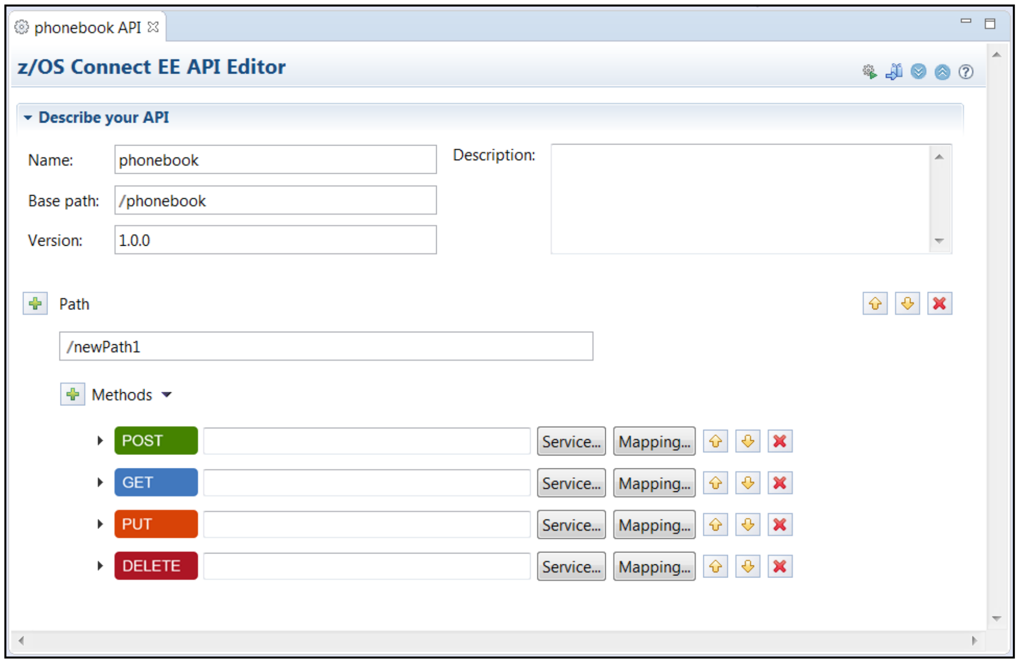
Project name: PhoneBook

API name: phonebook

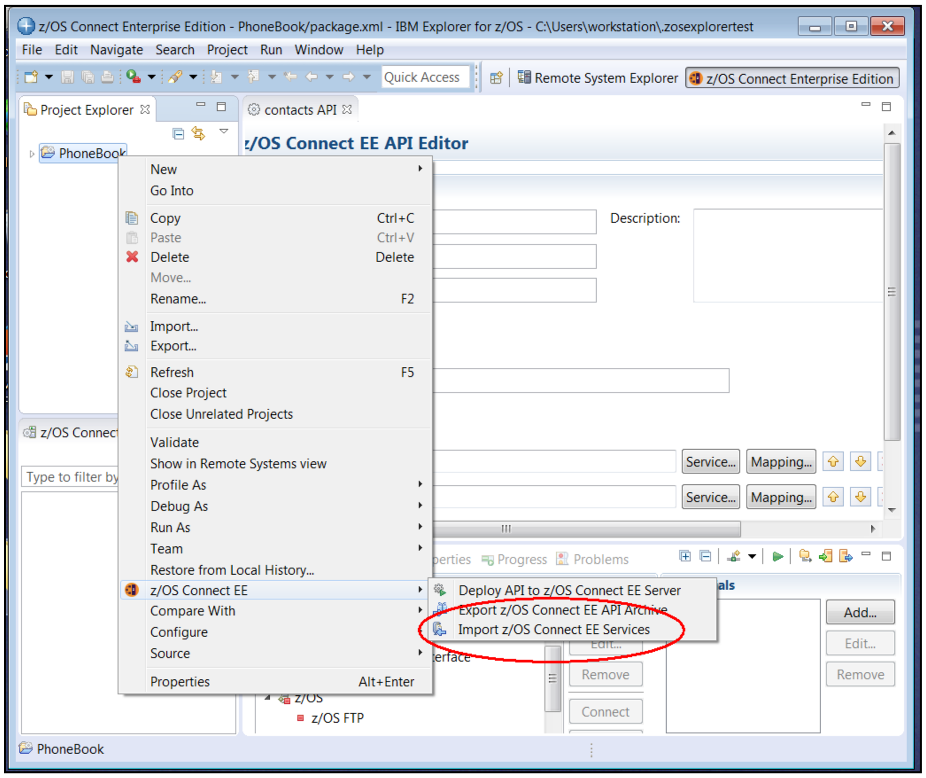
Base path: /phonebook

Description:

___4. You should now have a screen that looks something like this:



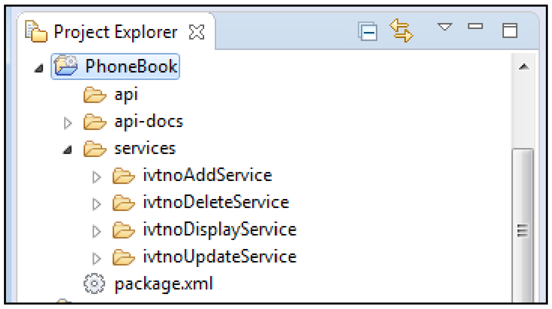
___5. In the *Project Explorer* tab, right click on the *PhoneBook* folder and then select *z/OS Connect* and then *Import z/OS Connect Services*:



Tech-Tip: If the API Editor view is closed, it can be reopened by double clicking the *package.xml* file in the API project.

___6. Then, use the **Workspace** button and expand the *Services* folder and select all of the files and click **Open** and then **OK** three times to import the SAR file in the API Toolkit.

___7. In the *Project Explorer* window (upper left), expand the folders and you'll see something like the following:

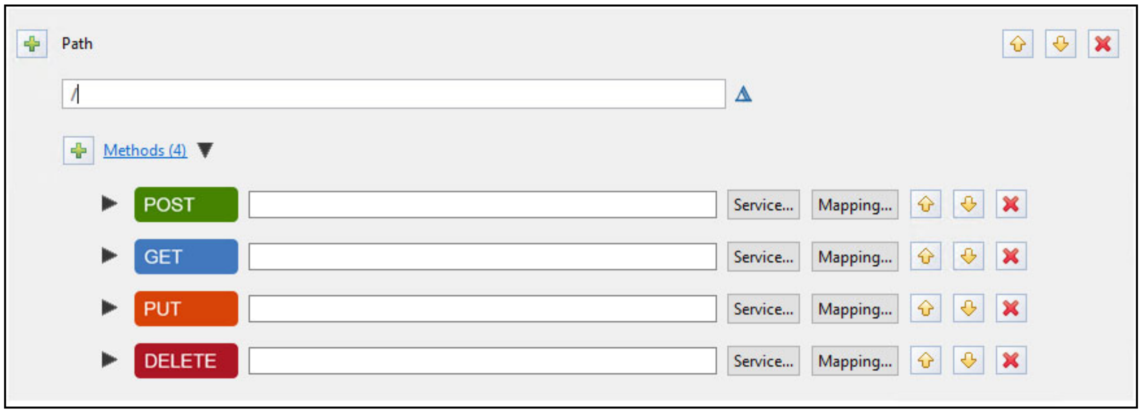


We want to create an API that support the following URIs and actions.

Action	*Verb*	*URI (base path + API path)*	*Service*
Add	POST	/phoneBook/phonebook	*ivtnoAddService*
Update	PUT	/phoneBook/phonebook/{lastName}	*ivtnoUpdateService*
Display	GET	/phoneBook/phonebook/{lastName}	*ivtnoDisplayService*
Delete	DELETE	/phoneBook/phonebook/{lastName}	*ivtnoDeleteService*

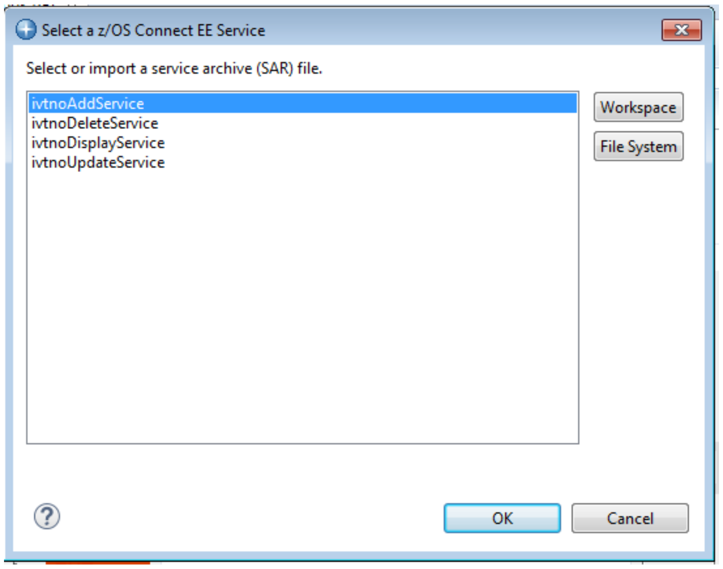
There will be two URI paths: one as just */phoneBook/phonebook* and the other as */phoneBook/phonebook/{lastName}*, where {lastName} is a path parameter. That will be the basis for the work in the API Editor, which comes next.

___8. Do the *POST* verb definition first. Set the *Path* value to / as shown here:

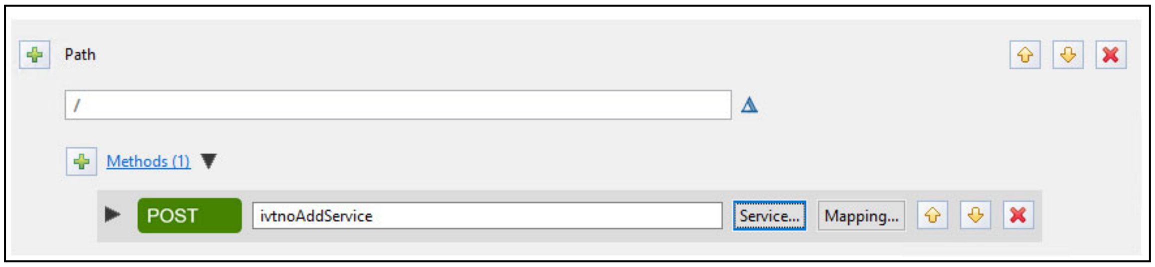


___9. For *this* API path (just /phonebook with no path parameter) we will use only *POST*. Delete the *GET, PUT* and *DELETE* verbs. Click the "x" symbols to the right of each of those to remove them:

Next, click on the **Service..**. button that's on the *POST* row and then select the *ivtnoAddService*), then click **OK**:

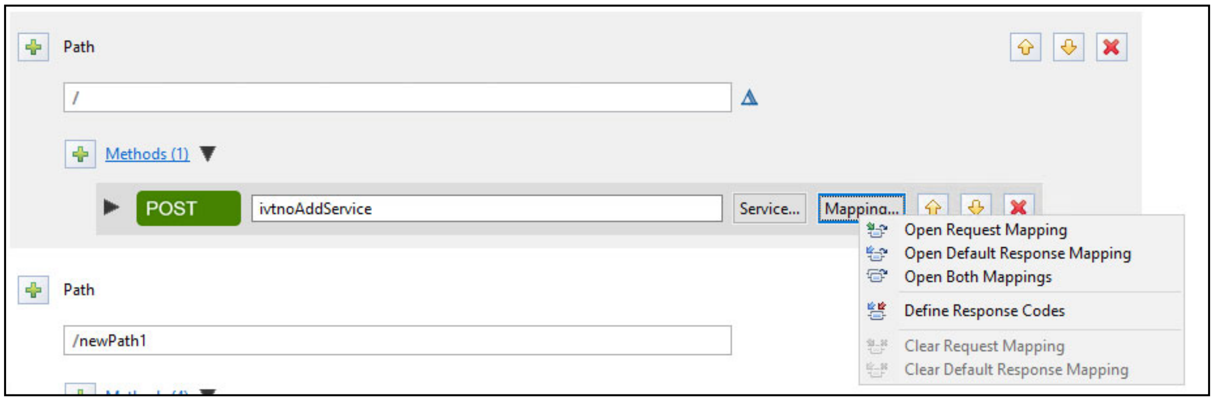


You should be left with:

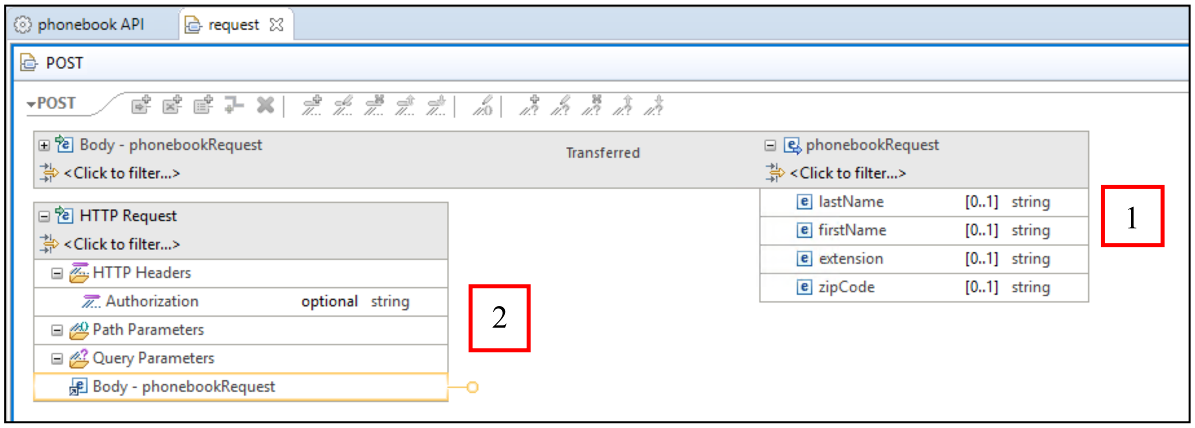


___10. Before you can do the field mappings you have to save your changes. From the menu bar, select *File→Save* or key sequence **Ctrl-S**.

___11. Now, click on the **Mapping...** button and select *Open Request Mapping*:



You should then see:



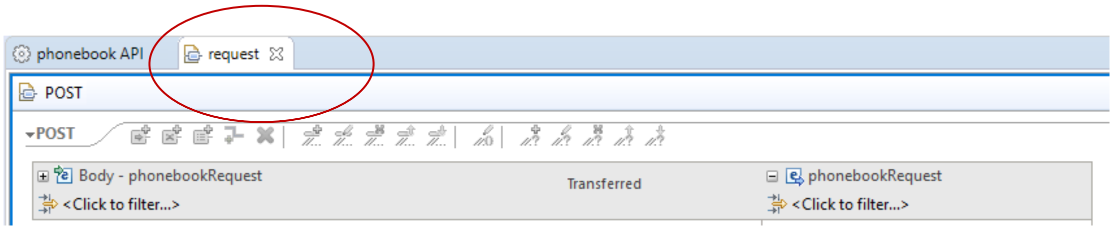
If necessary, in the upper-left, next to *phonebookRequest*, click the little "+" sign symbol to expand the field:

Notes:

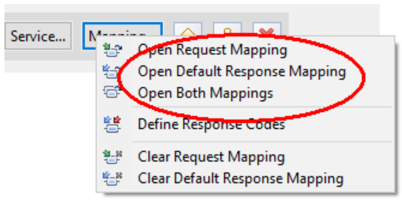
1. The right side represents the fields exposed by the service definition for the request.

2. The "HTTP Request" section represents values from the HTTP request (such as path and query parameters) that can be mapped to the fields on the right side. For this *POST* action our path was just /, so there is no path or query parameters.

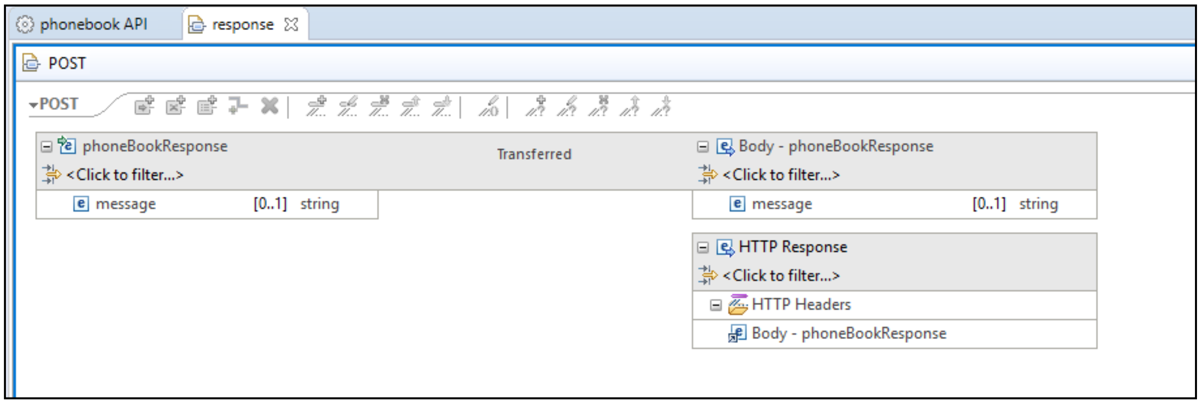
___12. Close the request mapping tab by clicking on the white X.



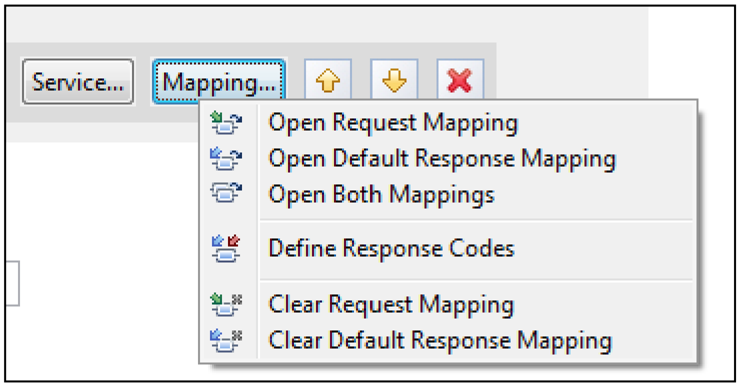
___13. Now click on the **Mapping...** button again, but this time select *Open Default Response Mapping*:



___14. You will see the fields that will be sent back to the REST client on the response:

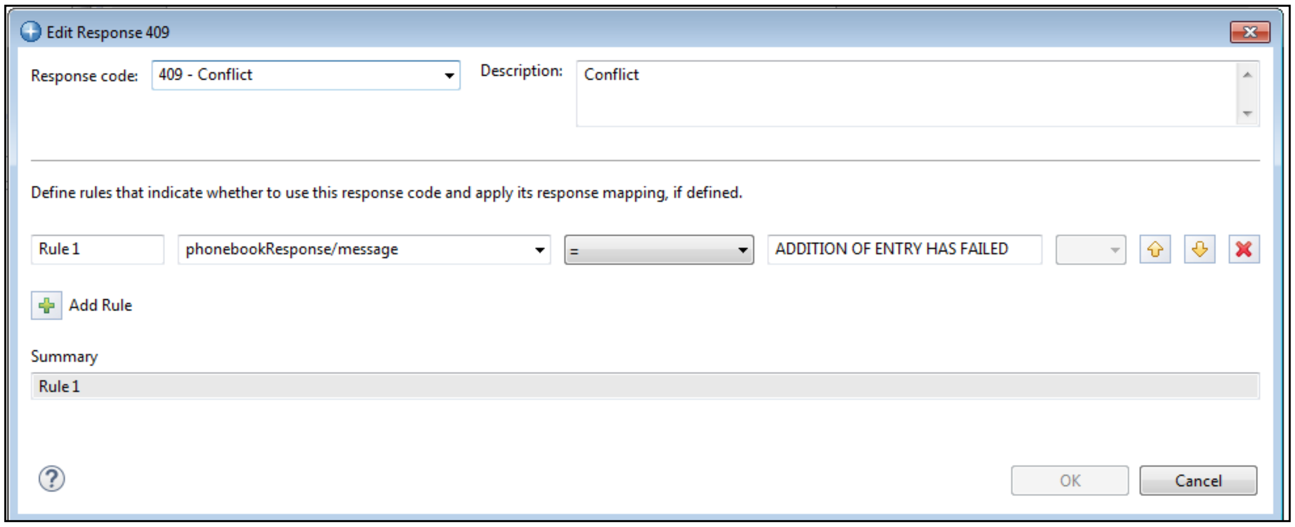


___15. Click the **Mapping** button again but this time select *Define Response Codes*.



This will allow the setting of the HTTP response code based on the contents of the response message.

___16. Click the plus sign beside *Add Response* and use the pull-down arrows to select a *Response code* of *409 – Conflict*, field *phonebookResponse/Message* for the first operand, an equal sign for the operation and enter ***ADDITION OF ENTRY HAS FAILED*** for the second operand.


Edit Response 409

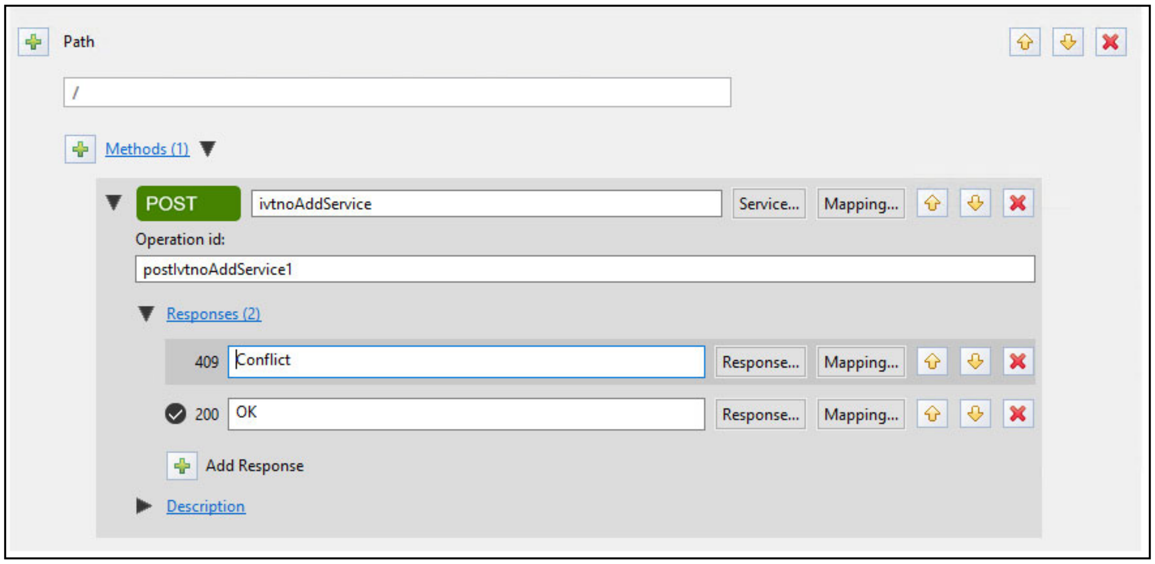
Response code: 409 - Conflict

Description: Conflict

Define rules that indicate whether to use this response code and apply its response mapping, if defined.

Rule 1 | phonebookResponse/message | = | ADDITION OF ENTRY HAS FAILED

___17. Click **OK** and now you should see



POST ivtnoAddService

Operation id: postIvtnoAddService1

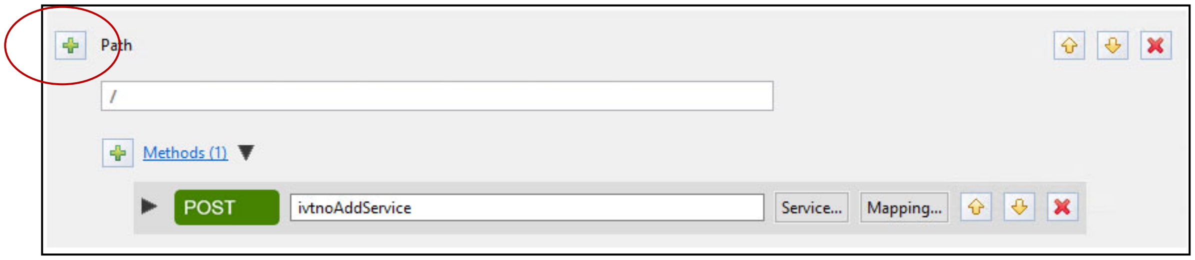
409 Conflict

200 OK

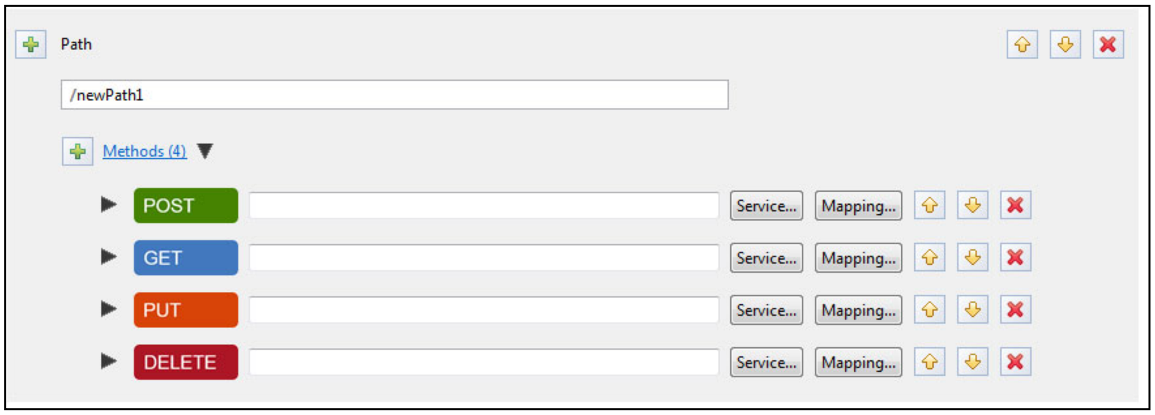
We have completed the POST verb "add a contact" portion of the API. Now let's create a path for the other three methods.

Do the following:

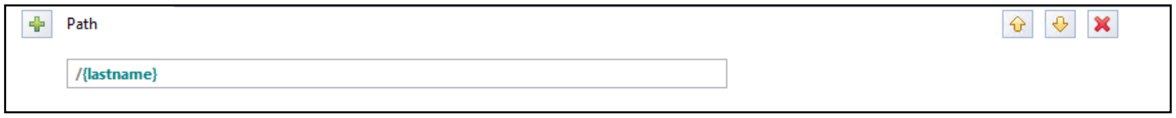
___18. Click on the "+" symbol next to *Path* as shown here:



You should see a new *Path* section with a new set of HTTP verbs:



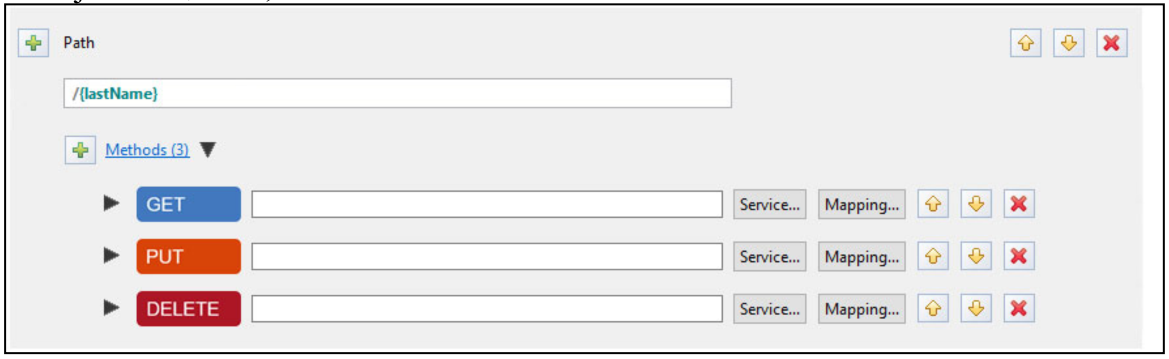
___19. For the *Path*, provide the string ***/phonebook/{lastName}*** as shown here:



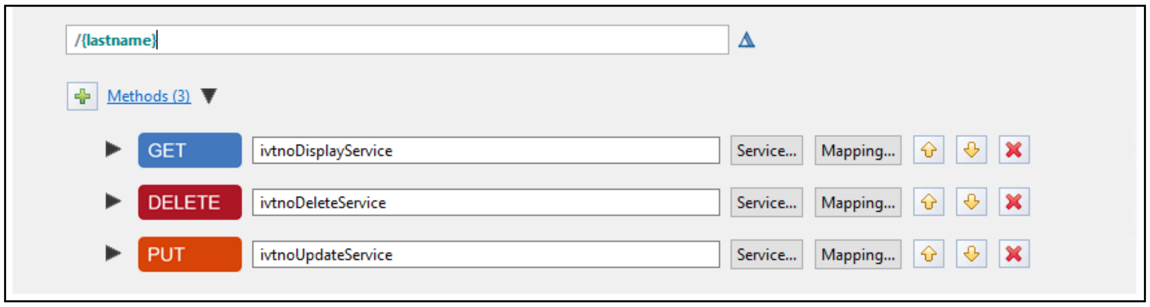
The string /{lastName} represents a path parameter. The REST client will send the last name in on the *GET, PUT,* or *DELETE* action. In the request mapping editors for each of those actions you will map the path parameter to the *IN_LAST_NAME* field of the transaction.

___20. We already defined the POST action, so we don't need it for this API Path. Remove it by clicking on the "x" symbol to the right:

That leaves just *GET, PUT*, and *DELETE*:



___21. For each (*GET, PUT*, and *DELETE*), click on the **Service...** button and select the corresponding service. The result should look like this:

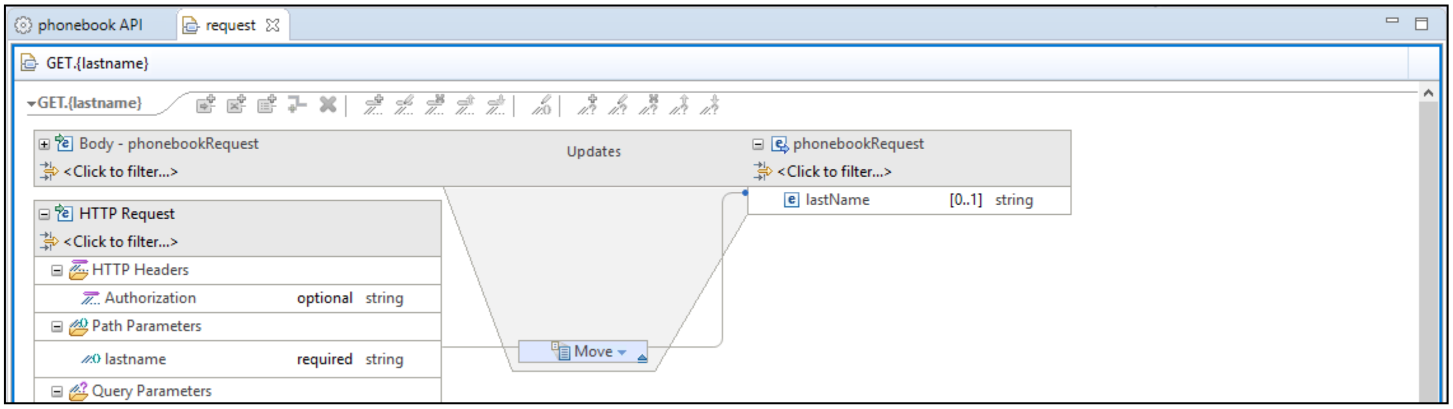


GET	ivtnoDisplayService	Service...	Mapping...
DELETE	ivtnoDeleteService	Service...	Mapping...
PUT	ivtnoUpdateService	Service...	Mapping...

___22. Save the changes: *File* →*Save* (or key sequence ***Ctrl+S***.

___23. For the *GET* method, click on **Mapping...** and select *Open Request Mapping*.

___24. We need to map the path parameter *lastName* from the HTTP Request section to the *IN_LAST_NAME* field on the right side. This is done by left-clicking on the path parameter field, then moving the cursor over to the field and dropping the line there. The result is this:



No JSON body is sent in with this request. All the information needed to display a contact record is carried in on the URI with the path parameter.

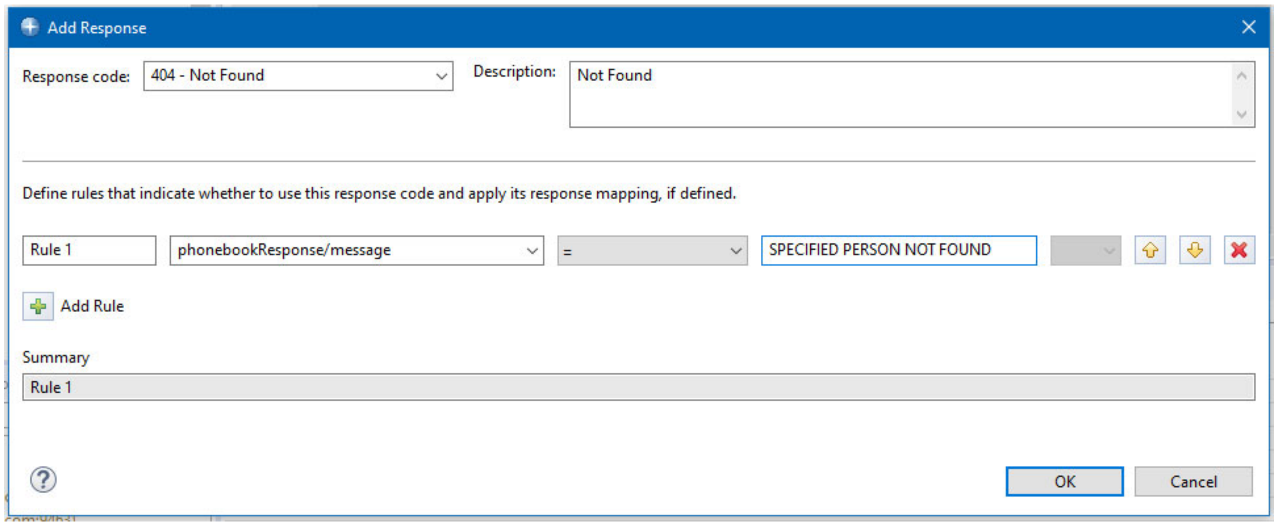
___26. Save the changes: *File* → *Save* (or key sequence **Ctrl+S**).

___27. Close the request mapping tab.

___28. Repeat this mapping for the PUT and DELETE methods.

___29. Now add additional response codes for these methods.

___30. For the GET method add a response code of *404 – Not Found when phonebookResponse/message* is equal to ***SPECIFIED PERSON NOT FOUND***.

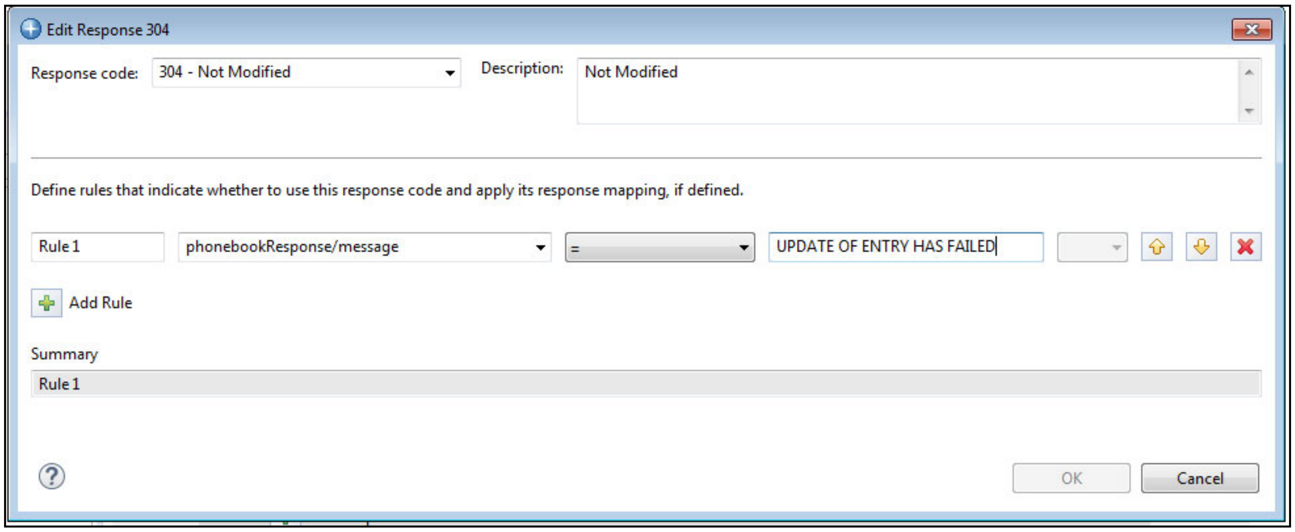

Add Response

Response code: 404 - Not Found
Description: Not Found

Define rules that indicate whether to use this response code and apply its response mapping, if defined.

Rule 1 | phonebookResponse/message | = | SPECIFIED PERSON NOT FOUND

___31. For the PUT method add a response code of *304 – Not Modified* when the *phonebookResponse/message* is equal to ***UPDATE OF ENTRY HAS FAILED***.

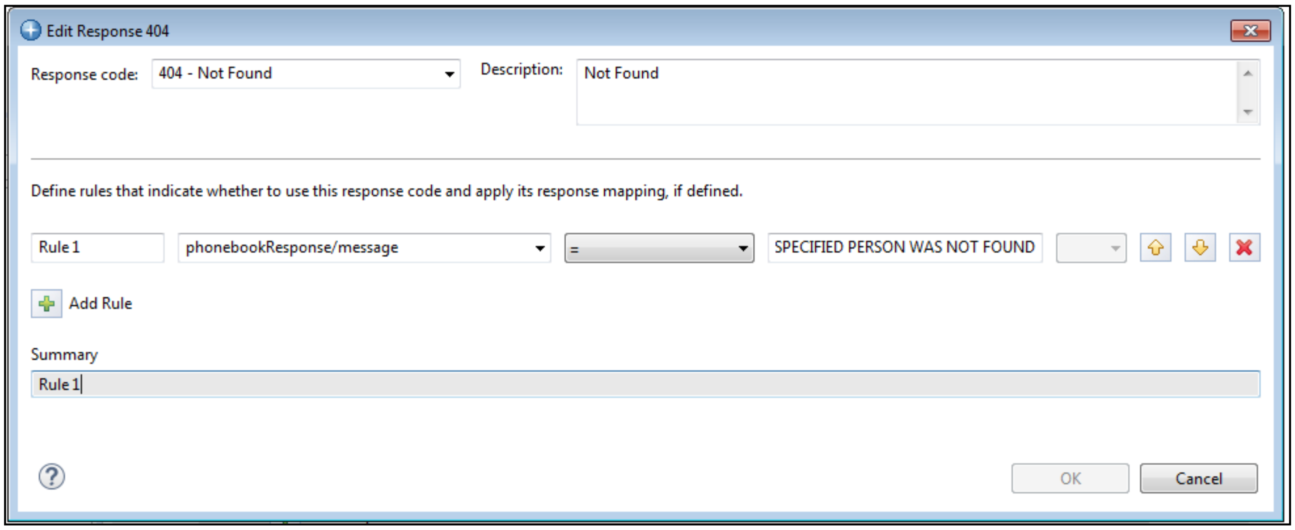

Edit Response 304

Response code: 304 - Not Modified
Description: Not Modified

Define rules that indicate whether to use this response code and apply its response mapping, if defined.

Rule 1 | phonebookResponse/message | = | UPDATE OF ENTRY HAS FAILED

___32. For the DELETE method add a response code of *404 – Not Found* when the *phonebookResponse/message* is equal to ***SPECIFIED PERSON WAS NOT FOUND***


Edit Response 404

Response code: 404 - Not Found

Description: Not Found

Define rules that indicate whether to use this response code and apply its response mapping, if defined.

Rule 1 | phonebookResponse/message | = | SPECIFIED PERSON WAS NOT FOUND

Summary

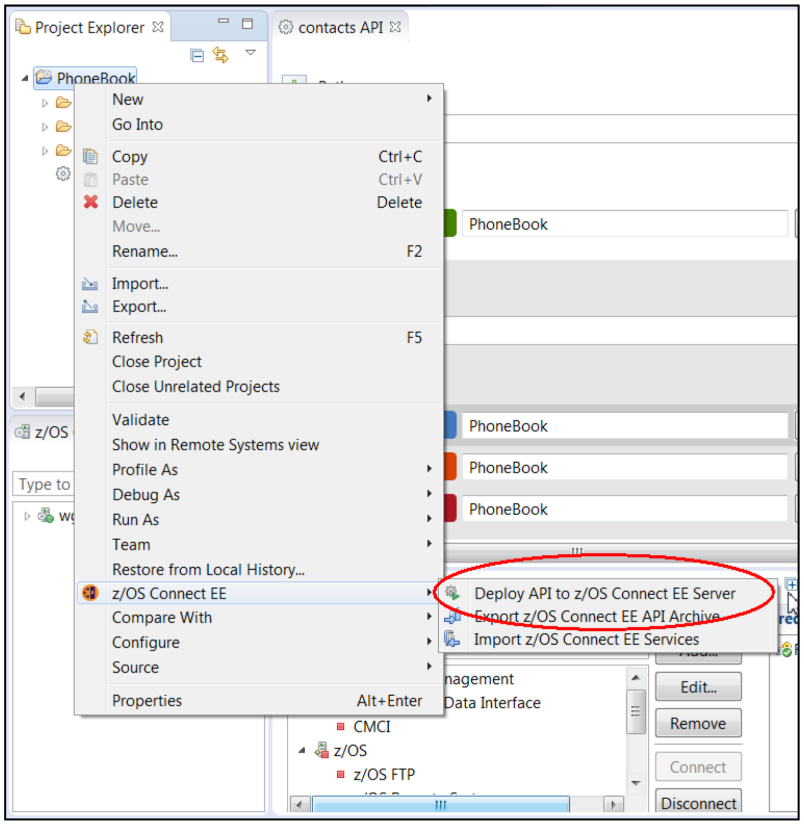
Rule 1

You're ready to deploy the API Archive file to z/OS Connect.

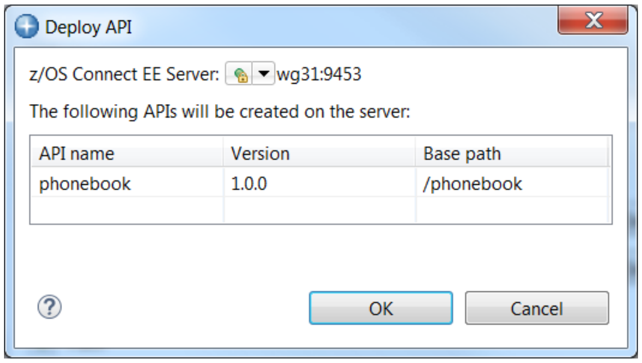
Deployment of the APIs into z/OS Connect

Your API is ready for deployment from the API Editor to z/OS.

___1. Switch back to the *z/OS Connect Enterprise Edition* perspective. In the *Project Explorer* view (upper left), right-mouse click on the *PhoneBook* folder, then select *z/OS Connect EE*→ *Deploy API to z/OS Connect EE Server*.



___2. If z/OS Explorer is connected to multiple z/OS Connect server the pull down arrow may have to be use to select the correct server (*wg31:9453*). If z/OS Explorer had multiple host connections to z/OS Connect servers then the pull down arrow would allow a selection to which server to deploy. Click **OK** on this screen to continue.

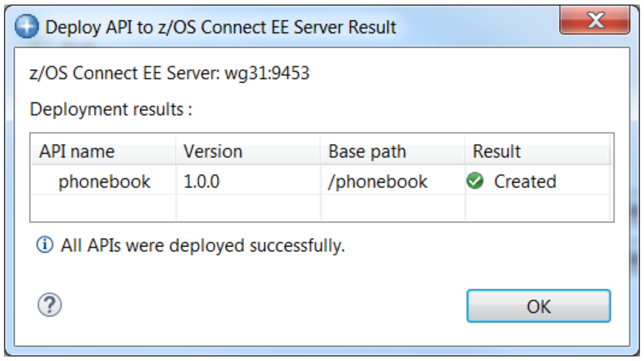


z/OS Connect EE Server: wg31:9453

The following APIs will be created on the server:

API name	Version	Base path
phonebook	1.0.0	/phonebook

The API artifacts will be transferred to z/OS and copied into the *var/zosconnect/servers/zceeims/resources/zosconnect/apis* directory.



Deploy API to z/OS Connect EE Server Result

z/OS Connect EE Server: wg31:9453

Deployment results :

API name	Version	Base path	Result
phonebook	1.0.0	/phonebook	✓ Created

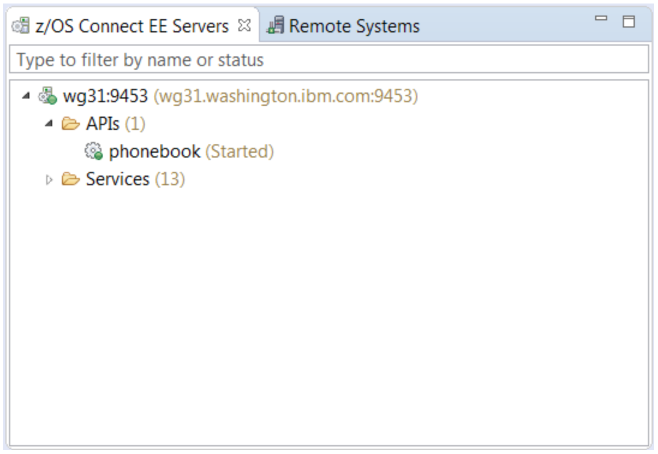
All APIs were deployed successfully.

Test the IMS API

The API is deployed and may now be accessed using any REST client.

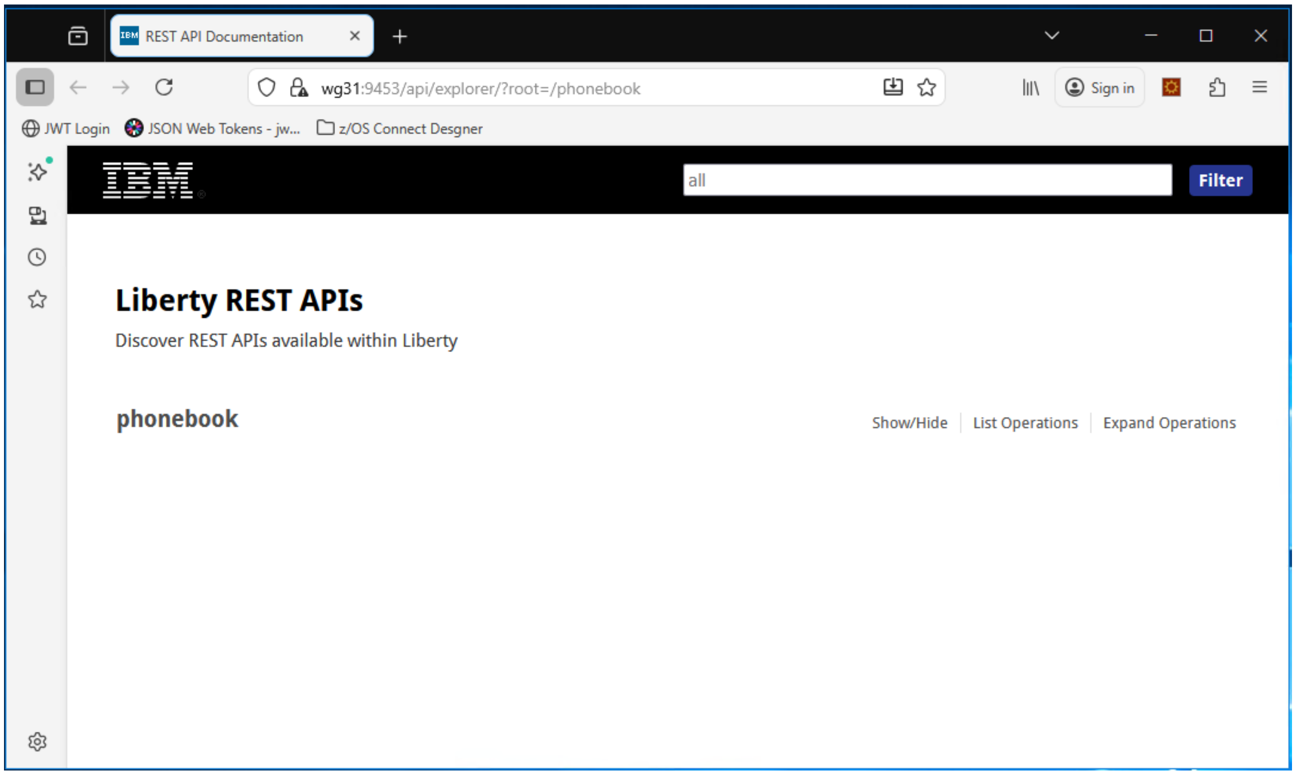
Tech Tip: It is very important to access the z/OS Connect server from a browser prior to any testing using the Swagger UI. Accessing a z/OS Connect URL from a browser starts an SSL handshake between the browser and the server. If this handshake has not performed prior to performing any test the test will fail with no message in the browser and no explanation. Ensuring this handshake has been performed is why you may be directed to access a z/OS Connect URL prior to using the Swagger UI during this exercise.

___1. In the lower left-hand side of the *z/OS Connect Explorer* perspective there is view entitled *z/OS Connect Servers*. Expand *wg31:9453* and the expand the *APIs* folder. You should see a list of the APIs installed in the server.



wg31:9453 (wg31.washington.ibm.com:9453)
APIs (1)
phonebook (Started)
Services (13)

___5. Hover over *phonebook* and click on the right mouse button and then select *Open in API Explorer.* Click **OK** if an informational prompt appears. This will open a Firefox window showing a *Swagger* test client (see below).

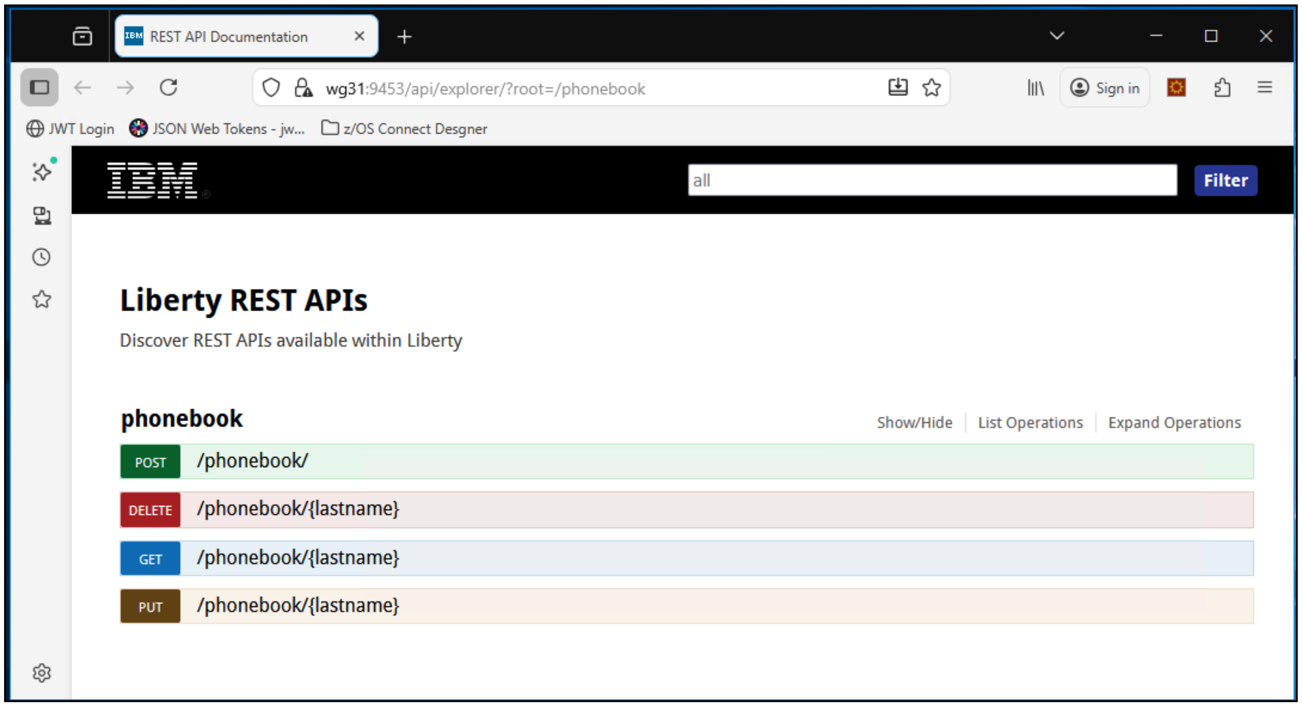


Liberty REST APIs

Discover REST APIs available within Liberty

phonebook

___6. Click the *List Operations* and the browser should show a list of the available HTTP methods like this:



Liberty REST APIs

Discover REST APIs available within Liberty

phonebook

Show/Hide | List Operations | Expand Operations

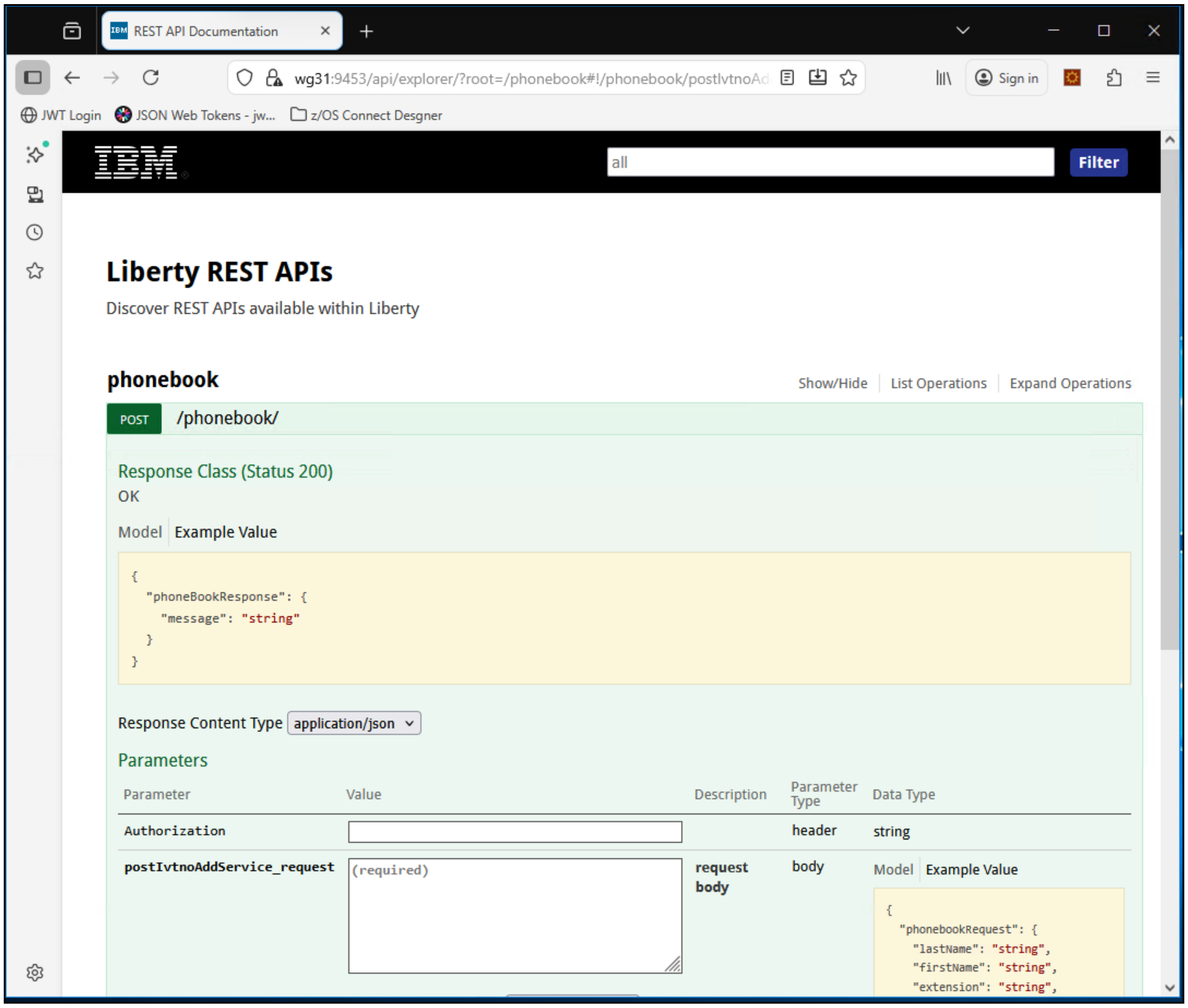
POST /phonebook/

DELETE /phonebook/{lastname}

GET /phonebook/{lastname}

PUT /phonebook/{lastname}

___7. Expand the *Post* method by clicking on the *Post* box and scroll down until the method *Parameters* are displayed as shown below:



REST API Documentation

wg31:9453/api/explorer/?root=/phonebook#!/phonebook/postIvtnoAd

JWT Login | JSON Web Tokens - jw... | z/OS Connect Desgner

all | Filter

Liberty REST APIs

Discover REST APIs available within Liberty

phonebook — Show/Hide | List Operations | Expand Operations

POST /phonebook/

Response Class (Status 200)

OK

Model | Example Value

```
{
  "phoneBookResponse": {
    "message": "string"
  }
}
```

Response Content Type application/json

Parameters

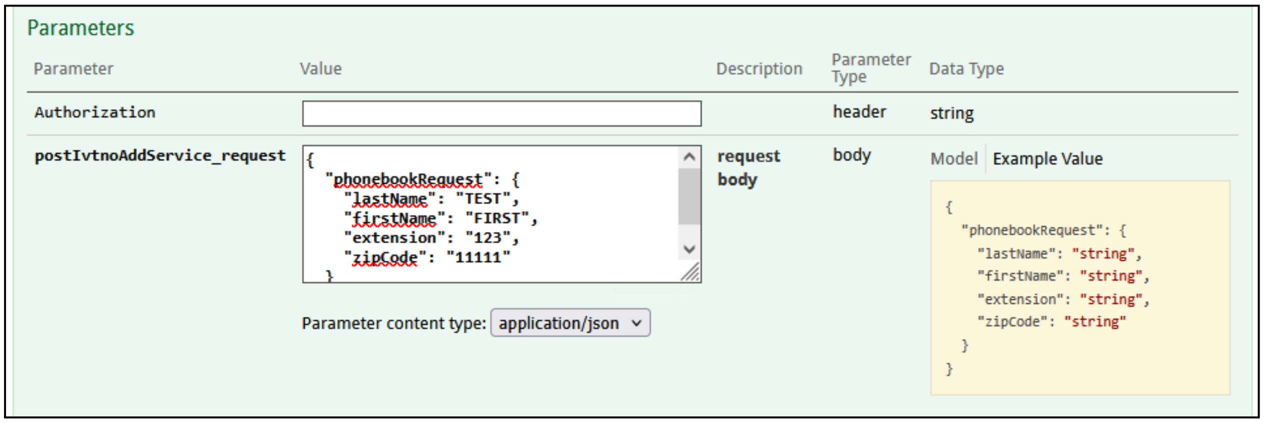
Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
postIvtnoAddService_request	(required)	request body	body	Model | Example Value

```
{
  "phonebookRequest": {
    "lastName": "string",
    "firstName": "string",
    "extension": "string",
```

\

___8. Enter the JSON request message in *postIvtnoAddService_request* area then click the **Try it out!** button.

```
{
 "phonebookRequest": {
   "lastName": "TEST",
   "firstName": "FIRST",
   "extension": "123",
   "zipCode": "11111"
  }
```



Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
postIvtnoAddService_request	(request body below)	request body	body	Model / Example Value

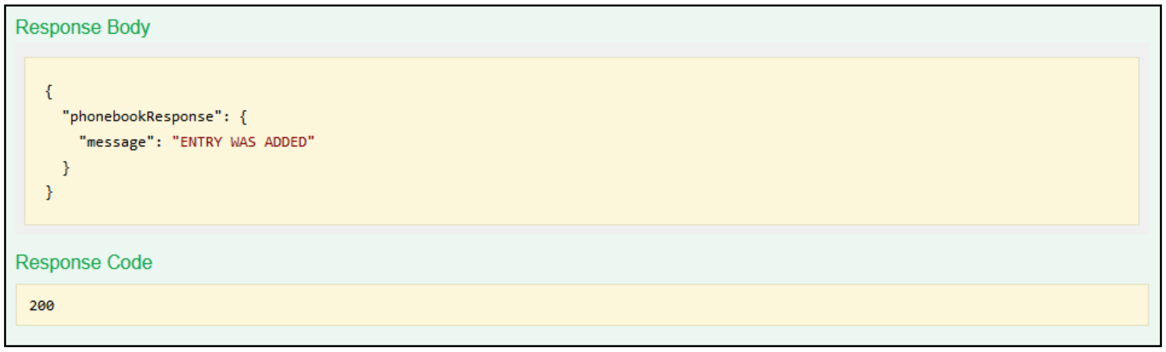
```
{
  "phonebookRequest": {
    "lastName": "TEST",
    "firstName": "FIRST",
    "extension": "123",
    "zipCode": "11111"
  }
```

Parameter content type: application/json

Example Value:

```
{
  "phonebookRequest": {
    "lastName": "string",
    "firstName": "string",
    "extension": "string",
    "zipCode": "string"
  }
}
```

___9. Scroll down to the Response Body and you see a message that the entry was added:



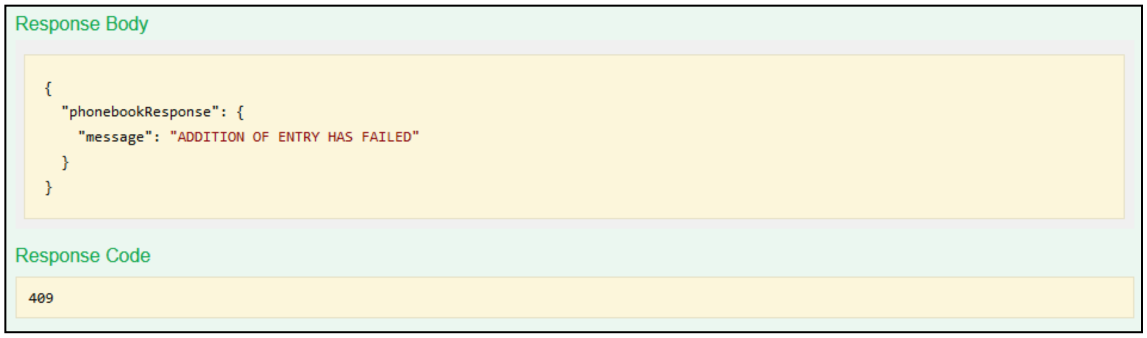
Response Body

```
{
  "phonebookResponse": {
    "message": "ENTRY WAS ADDED"
  }
}
```

Response Code

```
200
```

___10. Click the **Try it Out?** button again. This request should fail since the last name already exists in the database. You should see:

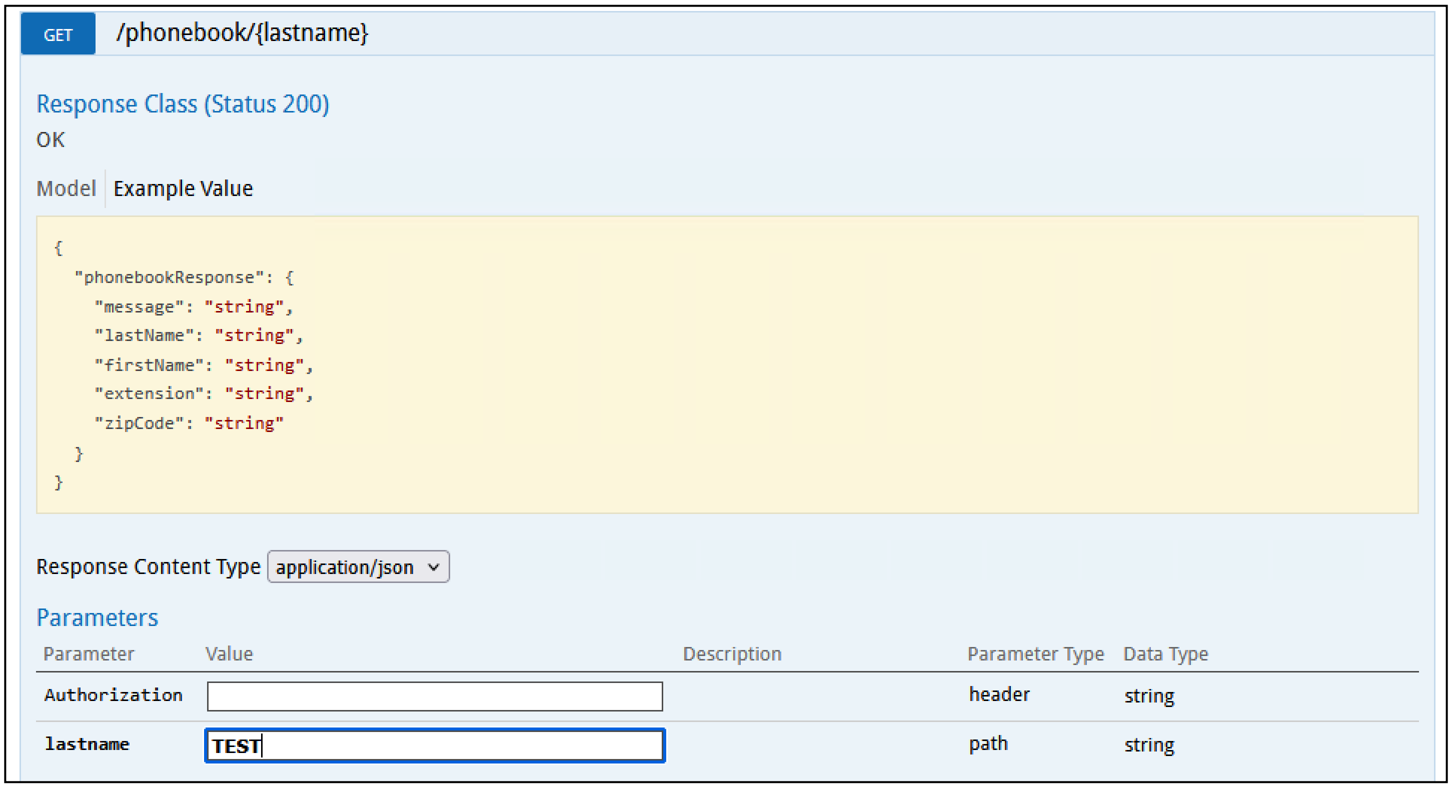


```
{
  "phonebookResponse": {
    "message": "ADDITION OF ENTRY HAS FAILED"
  }
}
```

```
409
```

Note the HTTP response code of 409.

___11. *Display* the contents of the contact you created by expanding the Get method by clicking *Get* box and entering ***TEST*** as the *lastName* and then click the **Try it out!** button.



```
{
  "phonebookResponse": {
    "message": "string",
    "lastName": "string",
    "firstName": "string",
    "extension": "string",
    "zipCode": "string"
  }
}
```

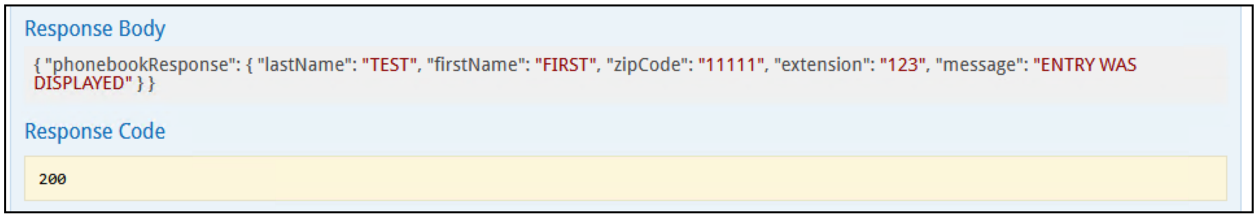
Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
lastname	TEST		path	string

.

___12. You should see the contents of the entry and a message that the entry was displayed in the *Response Body*.



Response Body

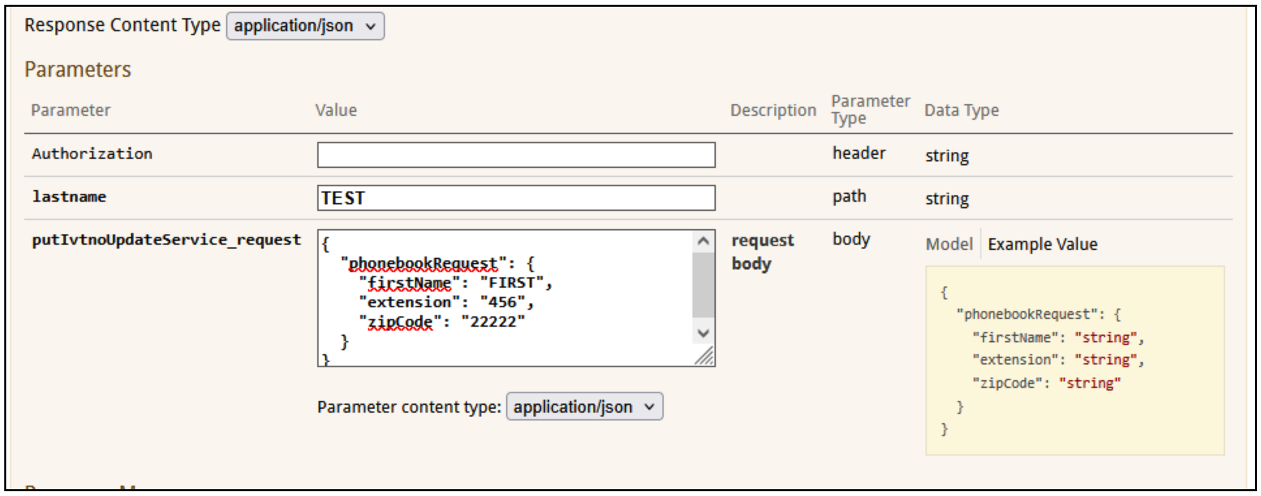
```
{ "phonebookResponse": { "lastName": "TEST", "firstName": "FIRST", "zipCode": "11111", "extension": "123", "message": "ENTRY WAS
DISPLAYED" } }
```

Response Code

```
200
```

___13. Expand the PUT method by clicking on the *Put* box and scrolling down until the method *Parameters* are displayed as shown below: Enter the JSON request message below and press the **Try it Out!** button.

```
{
 "phonebookRequest": {
   "firstName": "FIRST",
   "extension": "456",
   "zipCode": "22222"
  }
}
```



Response Content Type application/json

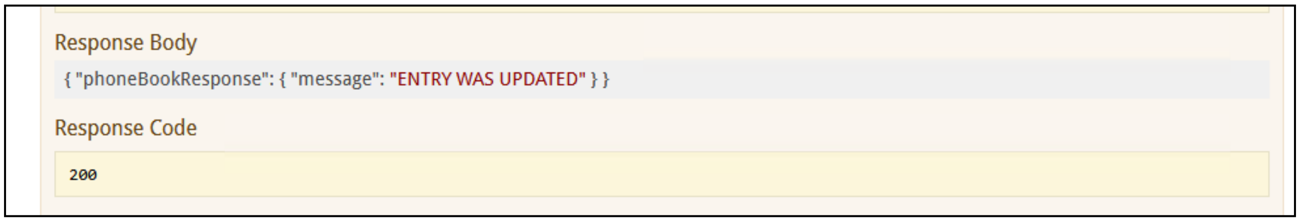
Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
lastname	TEST		path	string
putIvtnoUpdateService_request	{ "phonebookRequest": { "firstName": "FIRST", "extension": "456", "zipCode": "22222" } }	request body	body	Model Example Value

Parameter content type: application/json

```
{
  "phonebookRequest": {
    "firstName": "string",
    "extension": "string",
    "zipCode": "string"
  }
}
```

___14. You should see a *200 OK status code* in the *Response Code* area and a display of the *phonebookResponse* record in the *Response Body (Preview)* area along with *ENTRY WAS UPDATED* in the *message* field.



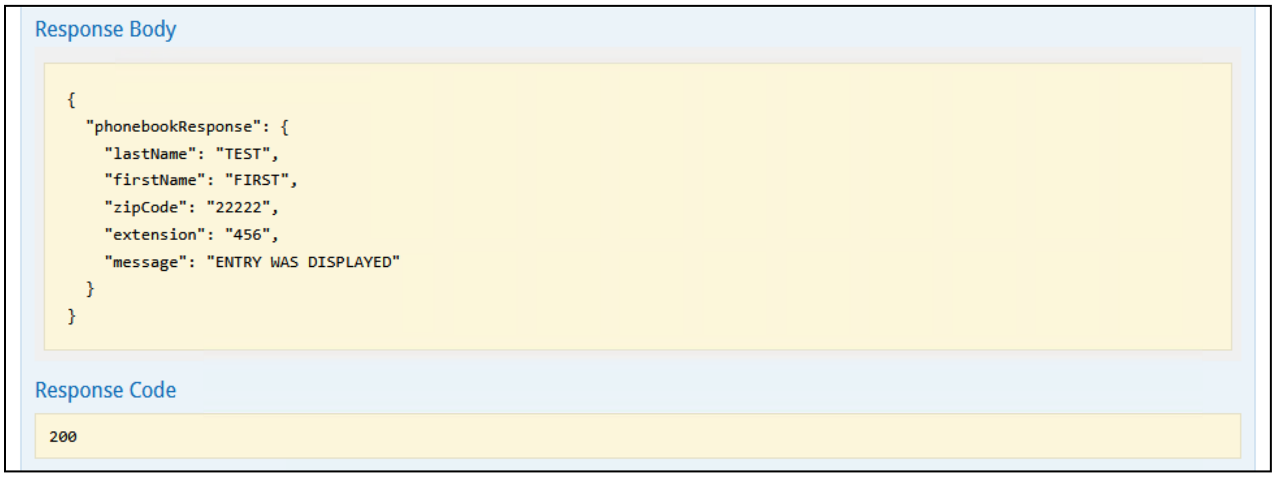
Response Body

```
{ "phoneBookResponse": { "message": "ENTRY WAS UPDATED" } }
```

Response Code

```
200
```

___15. Use the GET method to confirm the changes have been made.



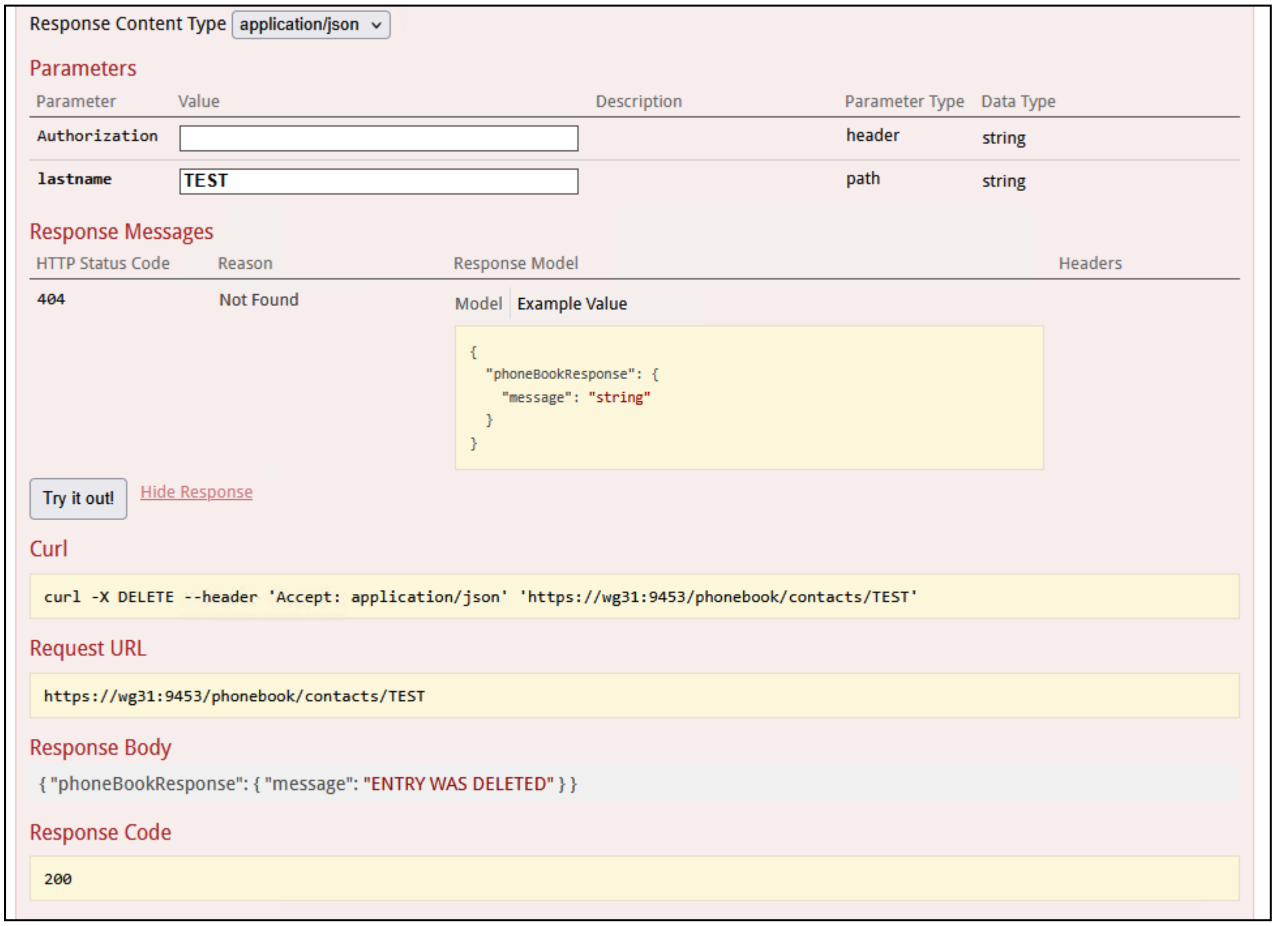
Response Body

```
{
  "phonebookResponse": {
    "lastName": "TEST",
    "firstName": "FIRST",
    "zipCode": "22222",
    "extension": "456",
    "message": "ENTRY WAS DISPLAYED"
  }
}
```

Response Code

```
200
```

___16. Delete the contents of the contact you created by expanding the *DELETE* entering the *lastName* and *Authorization* as below and click the **Try it out!** button.



Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
lastname	TEST		path	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
404	Not Found	Model Example Value `{ "phoneBookResponse": { "message": "string" } }`	

Curl

```
curl -X DELETE --header 'Accept: application/json' 'https://wg31:9453/phonebook/contacts/TEST'
```

Request URL

```
https://wg31:9453/phonebook/contacts/TEST
```

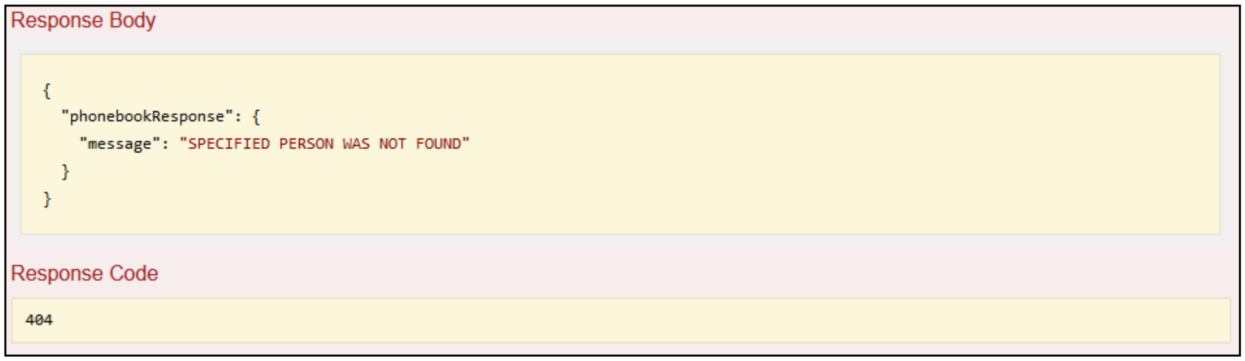
Response Body

```
{ "phoneBookResponse": { "message": "ENTRY WAS DELETED" } }
```

Response Code

```
200
```

___17. Press **Try it out?** again and the response code should 404, which is a not found response.



```
{
  "phonebookResponse": {
    "message": "SPECIFIED PERSON WAS NOT FOUND"
  }
}
```

```
404
```

Summary

You have verified the API. The API layer operates above the service layer you defined and tested earlier. The API layer provides a further level of abstraction and allows a more flexible use of HTTP verbs, and better mapping of data via the API editor function.