

IBM z/OS Connect EE V3.0

Developing RESTful APIs for IMS Transactions



WSC Wildfire Team
IBM z Systems

Lab Version Date: January 1, 2019

Table of Contents

Overview	3
<i>Connect the IBM z/OS Explorer to the z/OS Connect EE Server.....</i>	4
<i>z/OS Connect EE and IMS.....</i>	7
<i>Create a service for the PhoneBook API.....</i>	7
<i>Export and deploy the Service Archive files</i>	14
<i>Compose API using z/OS Connect EE V3.0 API Toolkit</i>	16
<i>Deployment of the APIs into z/OS Connect EE V3.0.....</i>	32
<i>Test the IMS API.....</i>	33

Important: On the desktop there is a file named *Developing APIs CopyPaste.txt*. This file contains commands and other text used in this workshop. Locate that file and open it. Use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors. As a reminder text that appears in this file will be highlighted in yellow.

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 EE and the API Toolkit. These two products allow the exposure of z/OS resources to JSON clients. More in-depth information about the customization of z/OS Connect EE, z/OS Connect EE security, the use of the API Toolkit and other topics is provided by the 1 day *ZCONEE – z/OS Connect Workshop*. 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 password for this user will be provided by the lab instructor.
- ✓ 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.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *Development APIs CopyPaste* file on the desktop.

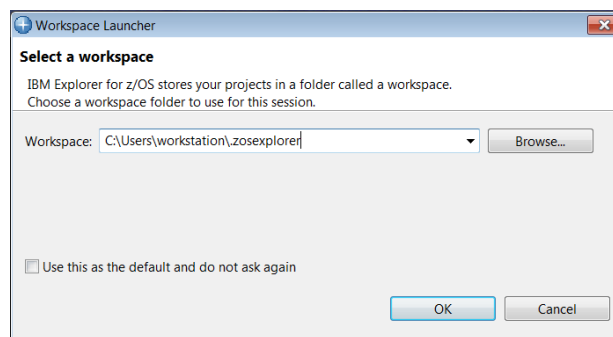
Connect the IBM z/OS Explorer to the z/OS Connect EE Server

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.

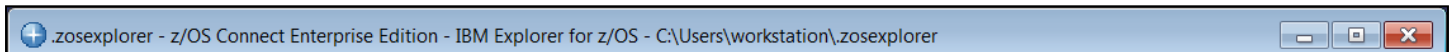
Tech-Tip: Windows desktop tools can be opened either by double clicking the icon or by selecting the icon and right mouse button clicking and then selecting the *Open* option.

2. You will be prompted for a workspace:



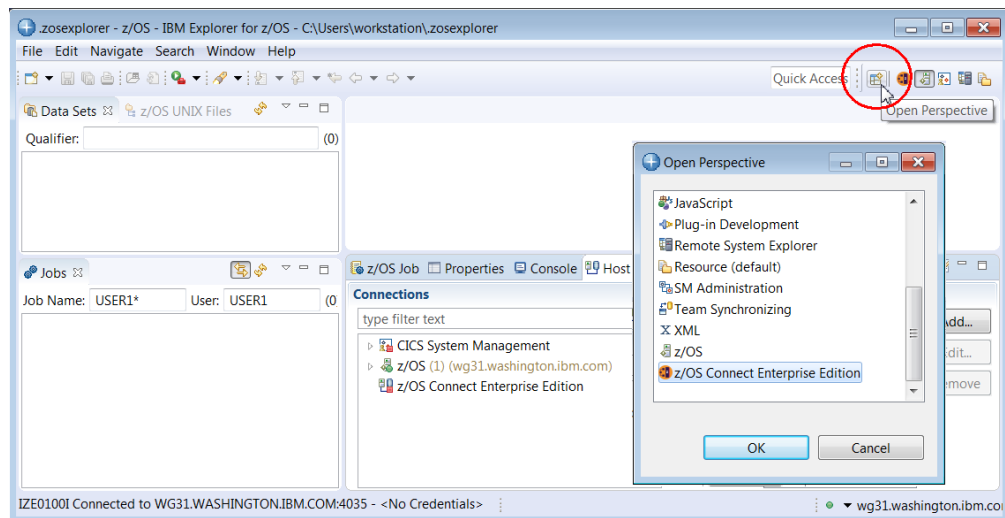
Take the default value by clicking **OK**.

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:



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.



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.



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.

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.

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.

z/OS Connect EE and IMS

For this exercise, a z/OS Connect EE server has been pre-configured with the IMS mobile feature. The initial configuration was done by executing command

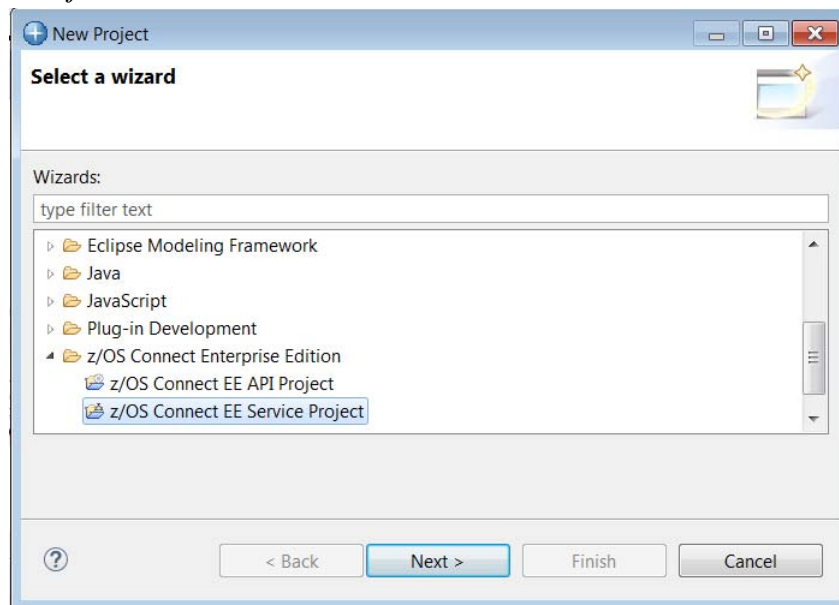
```
zosconnect create zceeims --template=imsmobile:imsDefault
```

This command created the initial directory structure, *server.xml* file, etc. but did it not fully augment the directory structure or the *server.xml* file with the all the artifacts required for IMS support in a z/OS Connect EE server. Starting the server at least once is required to fully augment the server for IMS support. Starting the server at least once creates and populates the configuration directory structure, e.g. the *resources/imsmobile-config* directory and updates the *server.xml* with additional information required for accessing IMS with RESTful APIs.

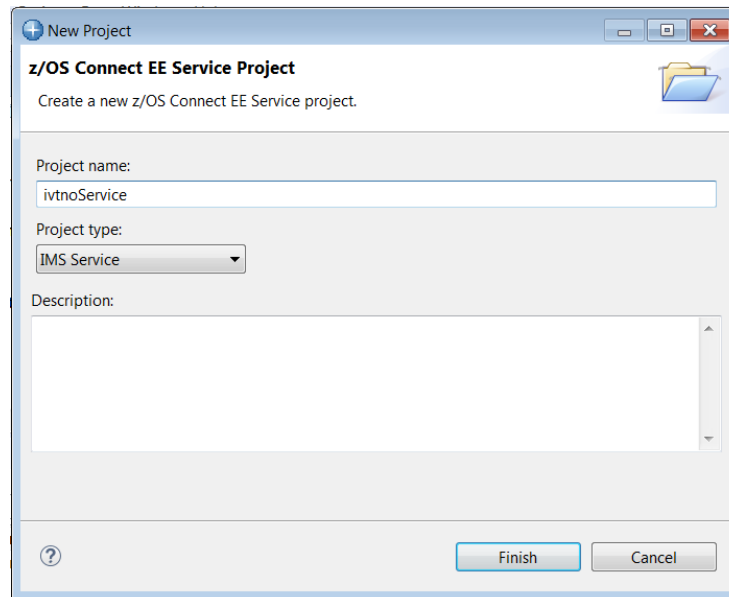
Create a service for the PhoneBook API

The next step is to create the service which correspond to IMS transaction IVTNO.

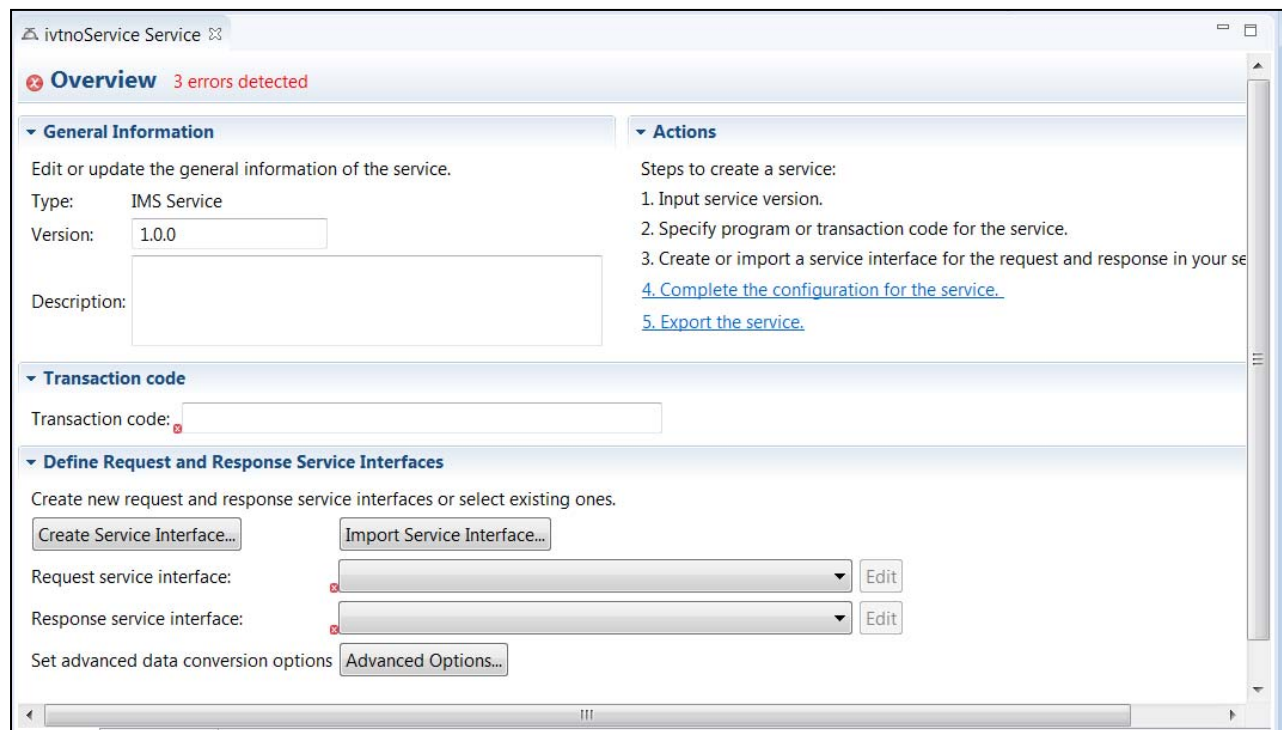
- ___1. In the upper left, position your mouse anywhere in the *Project Explorer* view and right-mouse click, then select *New → Project*:
- ___2. In the *New Project* window, scroll down and open the *z/OS Connect Enterprise Edition* folder and select *z/OS Connect EE Service Project* and then click the **Next** button.



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



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

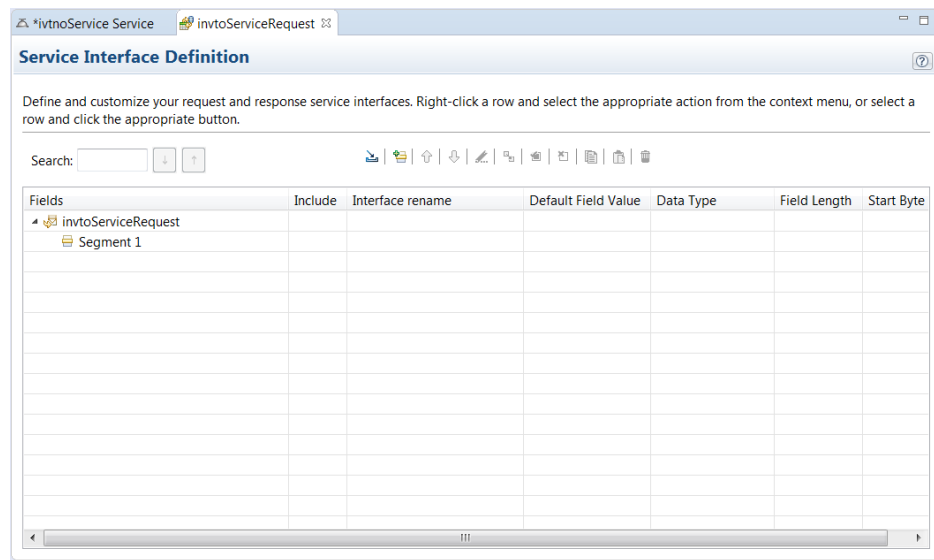


5. Next enter the IMS transaction **IVTNO** in the area beside *Transaction code*.

IVTNO is an IMS installation verification transaction which is used to access and maintain a simple phone book. The inbound request message identifies a function such as add an entry, update an entry, display an entry or delete an entry. Which function is performed by a field in the input message which is set to ADD, DISPLAY, UPDATE or DELETE. The input and output message are different, so two service interfaces will be required, one for input messages and another for output messages.

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

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



8. The first step is to import the COBOL copy book that provides the COBOL layout of the input and output messages.

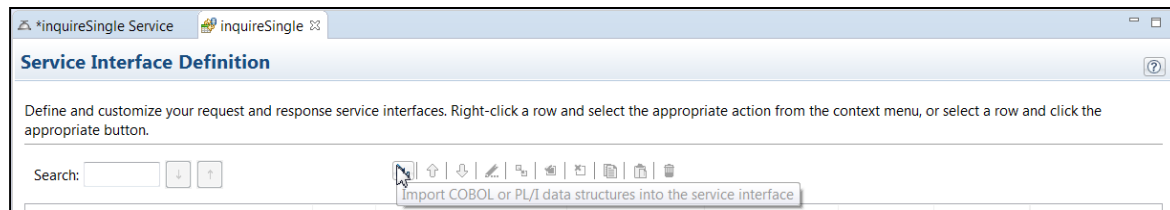
```
* DATA AREA FOR TERMINAL INPUT

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).

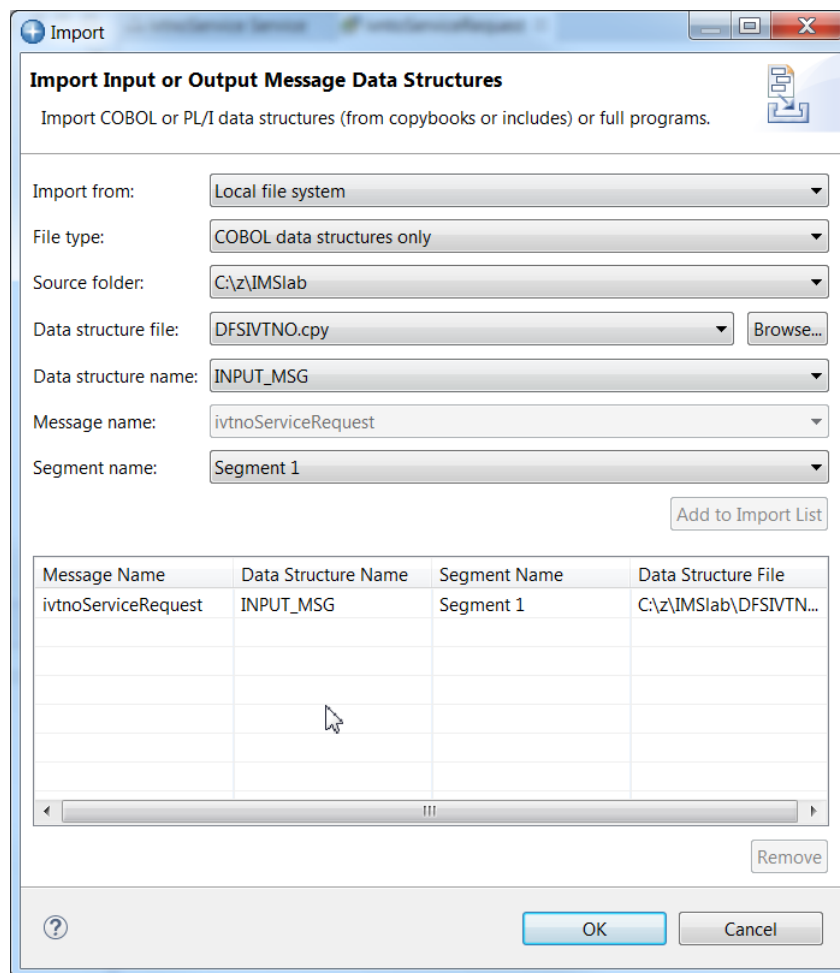
* DATA AREA OUTPUT

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'.
```

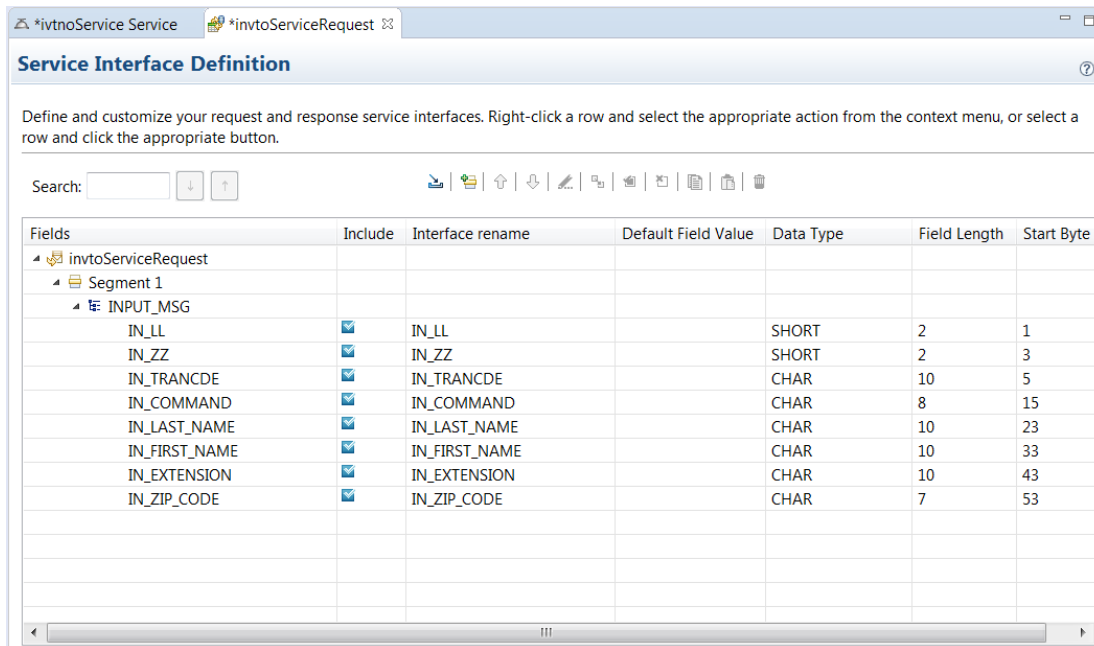
9. On the *Service Interface Definition* window, there is a tool bar near the top. If you hover over an icon its function will be display as below. Click the *Import COBOL or PL/I data structure into the service interface* icon to start the import process.



10. 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.



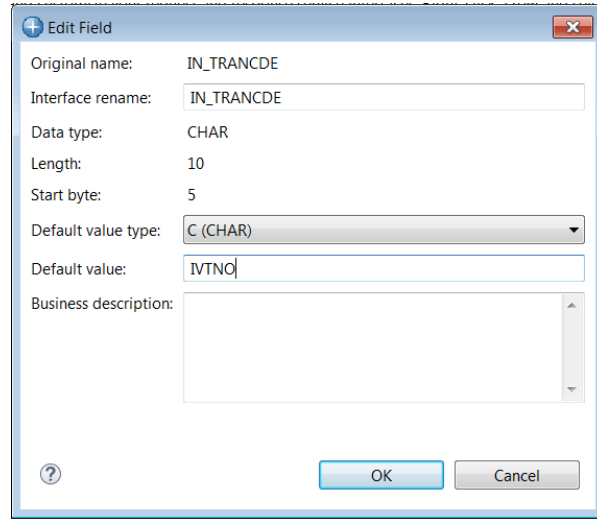
11. Click **OK** and when you expand *INPUT_MSG* you will see the COBOL ‘variables’ that have been imported into the service project as interface fields.



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

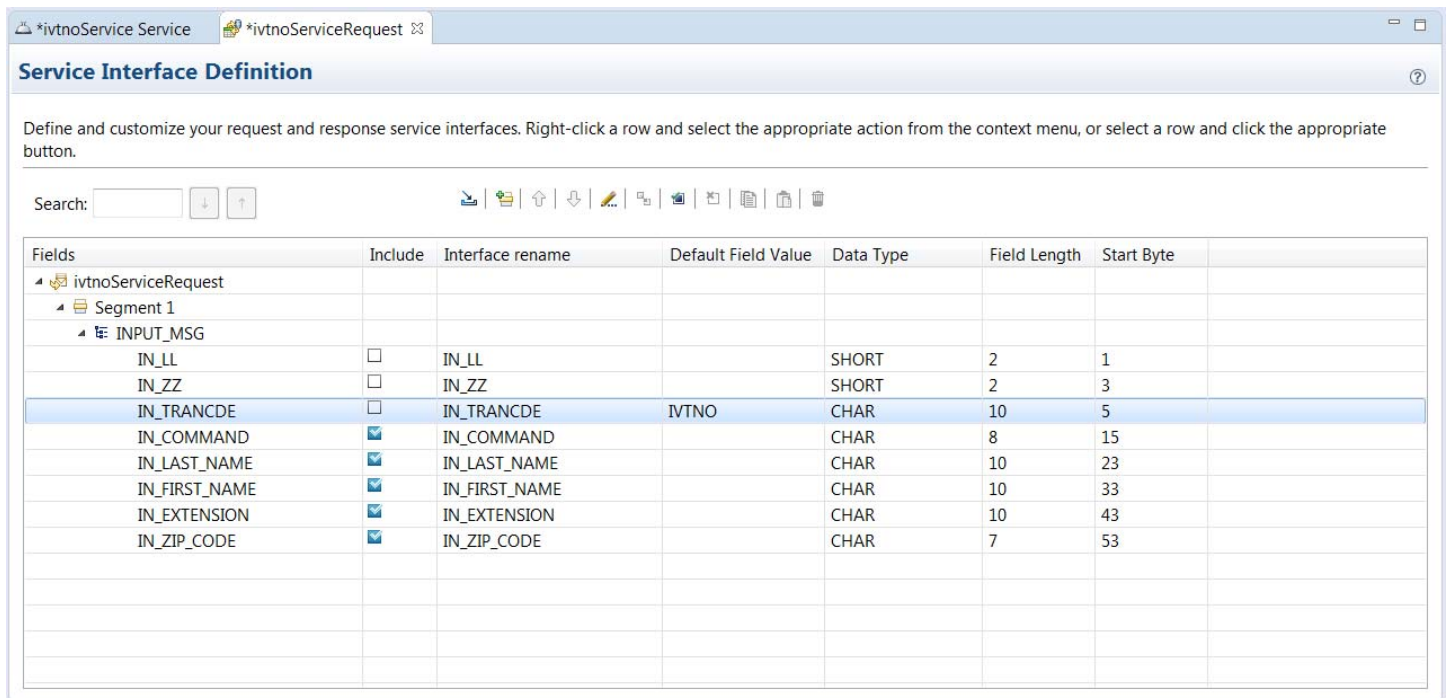
12. In this window, you can edit and change the property name (e.g. *Interface name*) or exclude specific fields entirely from the interface. Either can be done by selecting a field and right mouse button clicking or by selecting a field and using the desired tool icon in the Service Interface toolbar. Let's try both techniques to remove the FILLER fields.
13. Select field *IN_LL* and right mouse button click and select the *Exclude field from interface* option on the list of options.
14. Next select field *IN_ZZ* and use the *Exclude selected fields(s) from the interface* tool icon.
15. Notice that the check boxes besides these two fields are now unchecked. (You could have simply unchecked the box to accomplish the same results.)

16. 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.



The 'Edit Field' dialog box shows the configuration for the field *IN_TRANCDE*. The 'Original name' is *IN_TRANCDE*, and the 'Interface rename' is also *IN_TRANCDE*. The 'Data type' is *CHAR*, 'Length' is 10, and 'Start byte' is 5. The 'Default value type' is set to *C (CHAR)*, and the 'Default value' is *IVTNO*. The 'Business description' field is empty. The dialog has 'OK' and 'Cancel' buttons at the bottom right.

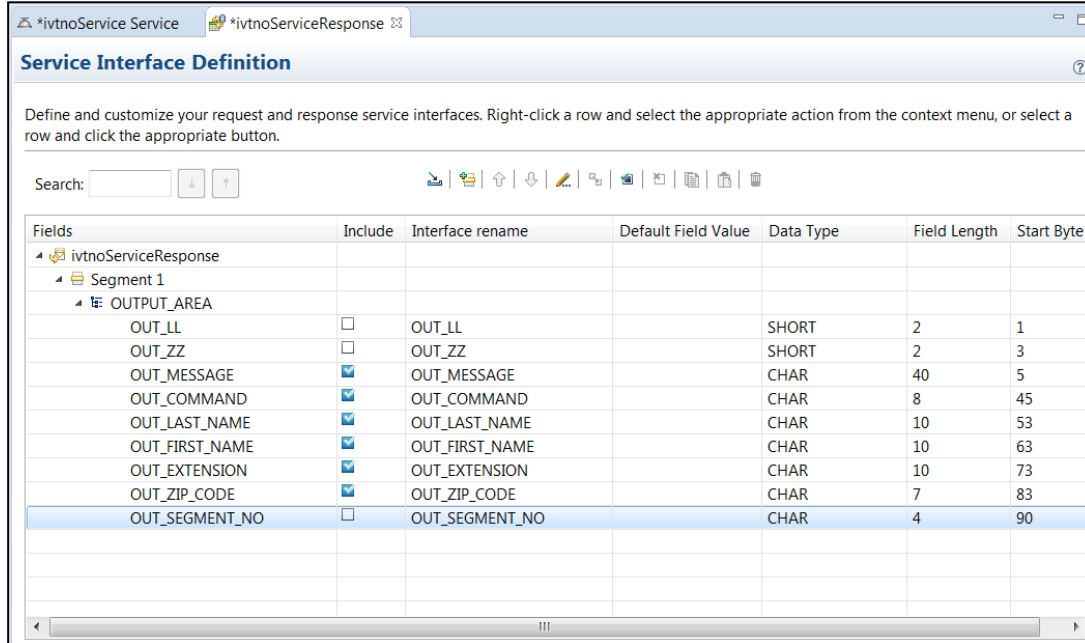
17. When finished your service definition interface should look like this.



The 'Service Interface Definition' window shows a table of fields for the *ivtnoServiceRequest* service. The table has columns: Fields, Include, Interface rename, Default Field Value, Data Type, Field Length, and Start Byte. The *IN_TRANCDE* field is highlighted in blue, indicating it is selected. The 'Include' column has checkboxes, and the 'Default Field Value' column shows *IVTNO* for the selected field.

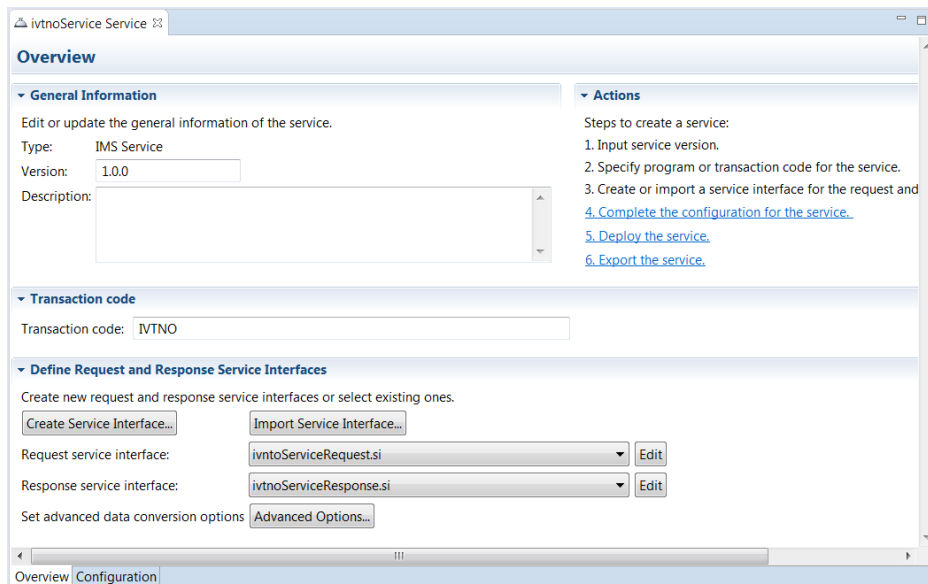
Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
ivtnoServiceRequest						
Segment 1						
INPUT_MSG						
IN_LL	<input type="checkbox"/>	IN_LL		SHORT	2	1
IN_ZZ	<input type="checkbox"/>	IN_ZZ		SHORT	2	3
IN_TRANCDE	<input type="checkbox"/>	IN_TRANCDE	IVTNO	CHAR	10	5
IN_COMMAND	<input checked="" type="checkbox"/>	IN_COMMAND		CHAR	8	15
IN_LAST_NAME	<input checked="" type="checkbox"/>	IN_LAST_NAME		CHAR	10	23
IN_FIRST_NAME	<input checked="" type="checkbox"/>	IN_FIRST_NAME		CHAR	10	33
IN_EXTENSION	<input checked="" type="checkbox"/>	IN_EXTENSION		CHAR	10	43
IN_ZIP_CODE	<input checked="" type="checkbox"/>	IN_ZIP_CODE		CHAR	7	53

18. 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 *ivtnoServiceRequest.si*. Also note that you can use their respective **Edit** buttons to return to the *Service Interface Definition* window for each interface.
19. The Response service interface is not we want so we need to create another service interface named *ivtnoServiceResponse*. Start at step 6 above and repeat these steps but this time add the *OUTPUT_AREA* data structure to the service interface and remove fields *OUT_LL*, *OUT_ZZ* and *OUT_SEGMENT_NO*. When finished your service interface definition should look like this.



Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
ivtnoServiceResponse						
Segment 1						
OUTPUT_AREA						
OUT_LL	<input type="checkbox"/>	OUT_LL		SHORT	2	1
OUT_ZZ	<input type="checkbox"/>	OUT_ZZ		SHORT	2	3
OUT_MESSAGE	<input checked="" type="checkbox"/>	OUT_MESSAGE		CHAR	40	5
OUT_COMMAND	<input checked="" type="checkbox"/>	OUT_COMMAND		CHAR	8	45
OUT_LAST_NAME	<input checked="" type="checkbox"/>	OUT_LAST_NAME		CHAR	10	53
OUT_FIRST_NAME	<input checked="" type="checkbox"/>	OUT_FIRST_NAME		CHAR	10	63
OUT_EXTENSION	<input checked="" type="checkbox"/>	OUT_EXTENSION		CHAR	10	73
OUT_ZIP_CODE	<input checked="" type="checkbox"/>	OUT_ZIP_CODE		CHAR	7	83
OUT_SEGMENT_NO	<input type="checkbox"/>	OUT_SEGMENT_NO		CHAR	4	90

20. Close the *Service Interface Definition* window.
21. Set the *Response service interface* to *ivtnoServiceResponse.si* as shown below.



Overview

General Information

Edit or update the general information of the service.

Type: IMS Service

Version: 1.0.0

Description:

Transaction code

Transaction code: IVTNO

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: ivtnoServiceRequest.si Edit

Response service interface: ivtnoServiceResponse.si Edit

Set advanced data conversion options Advanced Options...

Overview Configuration

22. 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*.

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

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

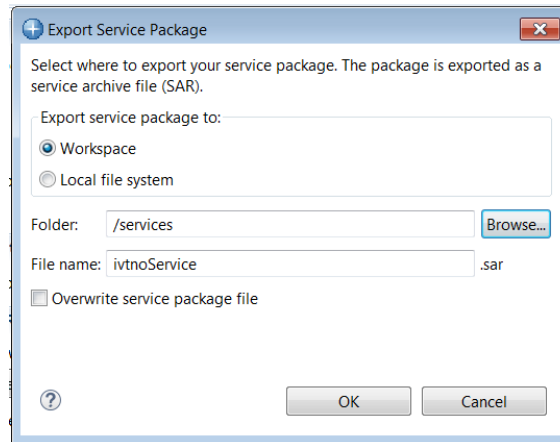
This service now need to be make available for developing the API and for deployment to the z/OS Connect EE server.

Export and deploy the Service Archive files

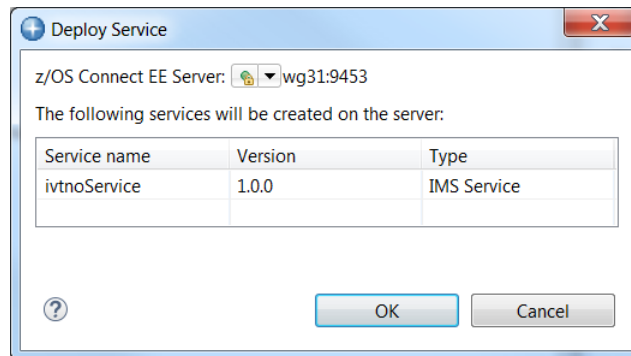
Before a Service Interface can be used it must be exported to create Service Archive (SAR) file). There are two uses for a SAR file. The first is for use in developing an API in the z/OS Connect EE Toolkit and the second is for deploying to a z/OS Connect EE server. This section describes the process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

1. First 'export' them into another project in the z/OS Connect EE 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 *ivtnoService* service project and right mouse button click. On the pop-up selection select **z/OS Connect EE → Export z/OS Connect EE 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.



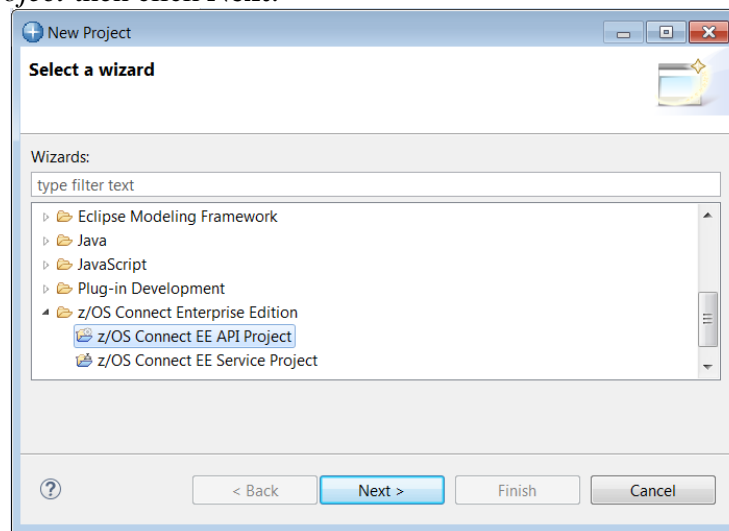
1. Next select *ivtnoService* service project and right mouse button click again and on the pop-up selection select **z/OS Connect EE → Deploy Service to z/OS Connect EE Server**. On the *Deploy Service* window select the target server (*wg31:9453*) and click **OK** twice to continue.



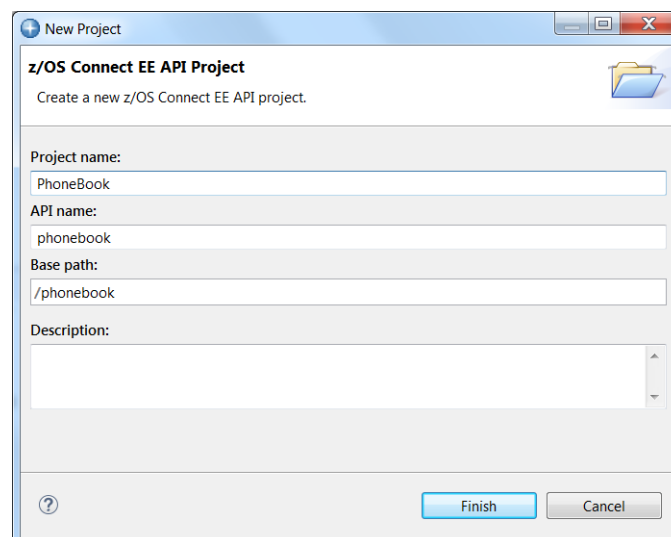
Compose API using z/OS Connect EE V3.0 API Toolkit

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

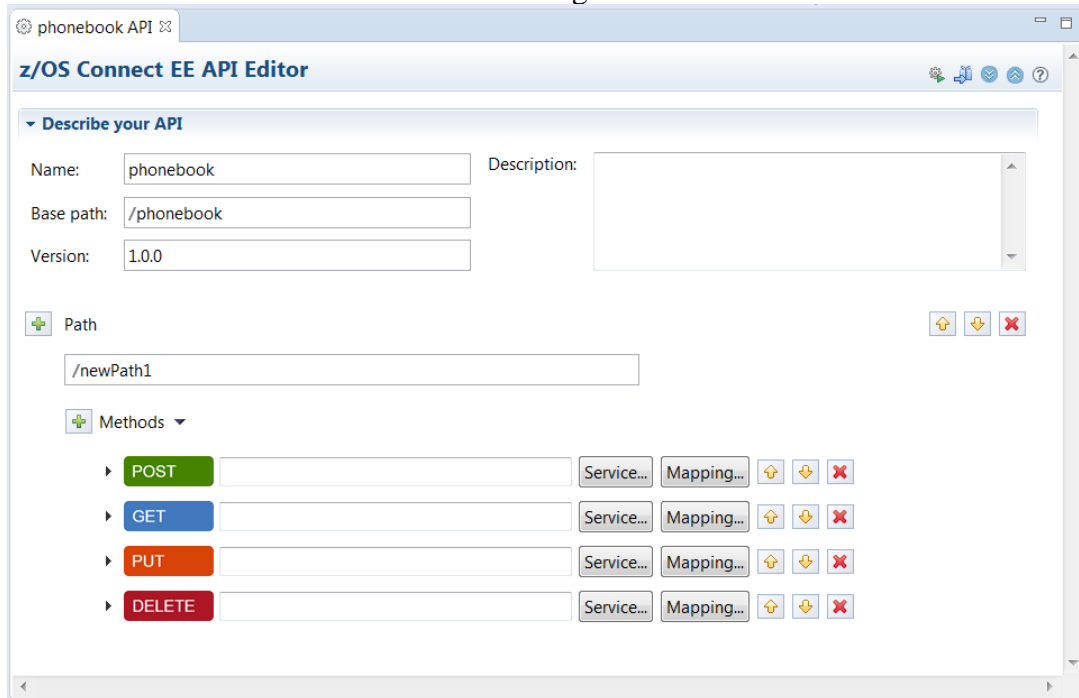
- ___1. Switch back to the IBM z/OS Explorer eclipse tool and switch to the perspective to *z/OS Connect Enterprise Edition*.
- ___2. Now from the menu bar select *File* → *New* and then *Other*.
- ___3. Then scroll down and locate the *z/OS Connect Enterprise Edition* folder, open that and select *z/OS Connect EE API Project* then click **Next**.



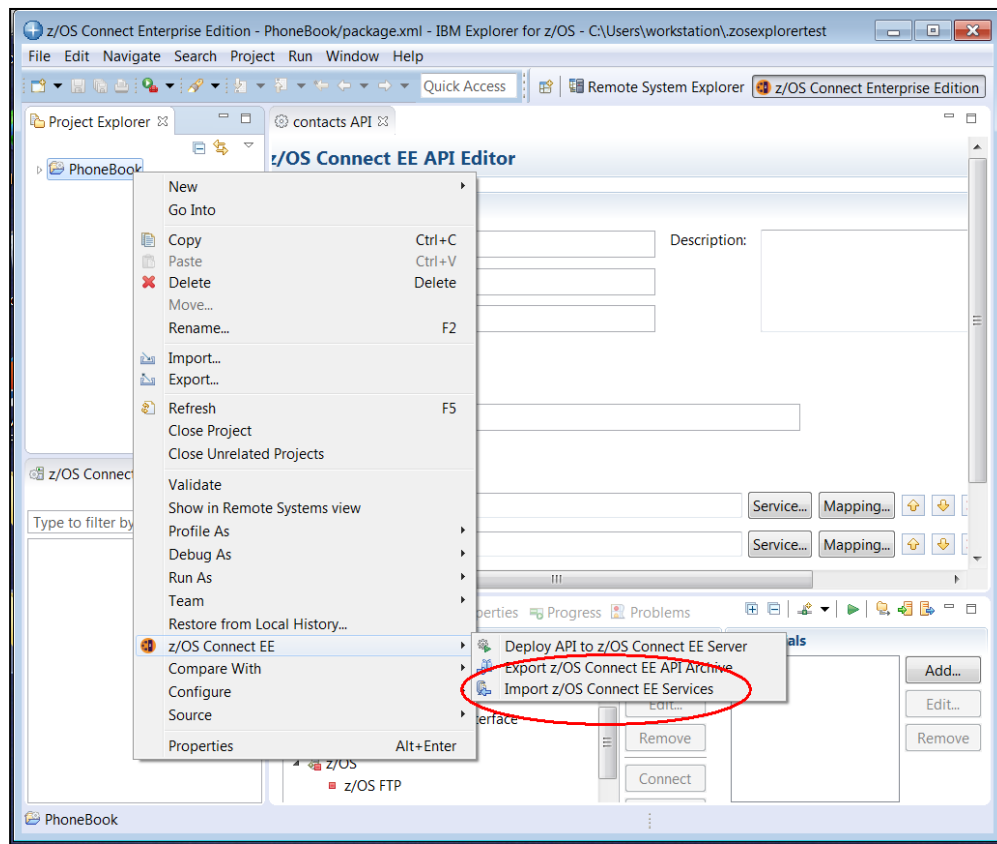
- ___4. For the project name values, enter **PhoneBook** for the *Project name*, **phonebook** for the *API name* and **/phonebook** for the *Base path*, then click **Finish**.



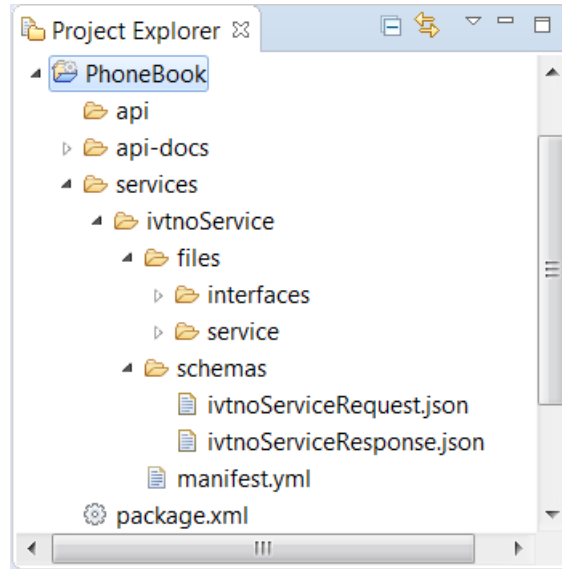
5. You should now have a screen that looks something like this:



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



- ___7. Then, use the **Workspace** button and expand the *services* folder and select *ivtnoService.sar* and click **Open** and then **OK** three times to import the SAR file in the API Toolkit.
- ___8. In the *Project Explorer* window (upper left), expand the folders and you'll see something like the following:



Those artifacts were brought in with the SAR file.

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

Action	Verb	URI (base path + API path)
Add	POST	/phoneBook/contacts
Update	PUT	/phoneBook/contacts/{lastName}
Display	GET	/phoneBook/contacts/{lastName}
Delete	DELETE	/phoneBook/contacts/{lastName}

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

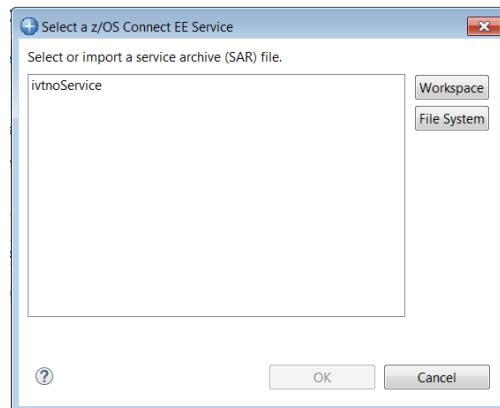
___9. Do the *POST* verb definition first. Set the *Path* value to **/contacts** as shown here:



Next, click on the **Service...** button that's on the *POST* row:



And then select the *ivtnoService* (it should be the only one present), then click **OK**:

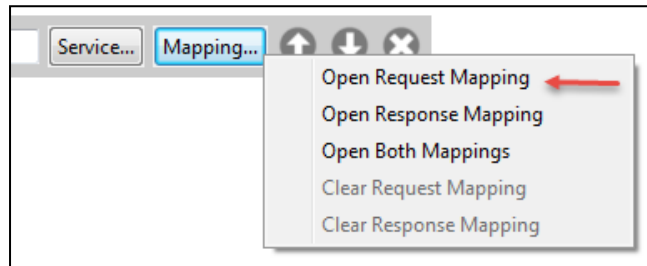


___10. For *this* API path (just /contacts with no path parameter) we will use only *POST*. Delete the *GET*, *PUT* and *DELETE* verbs. Click the "x" symbols to the right of each of those to remove them:

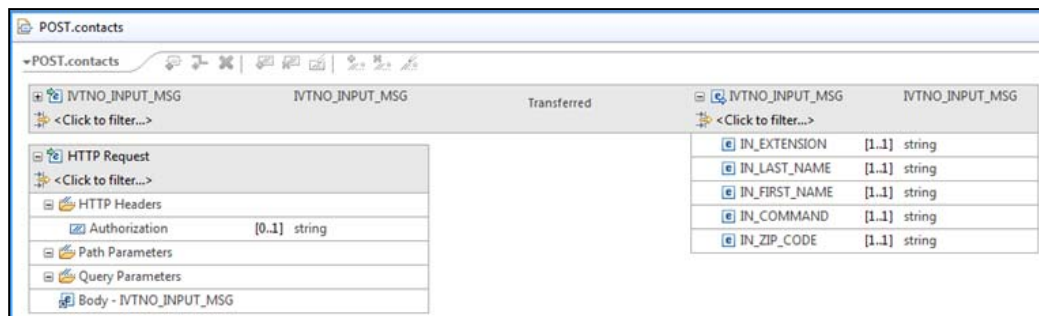
You should be left with:



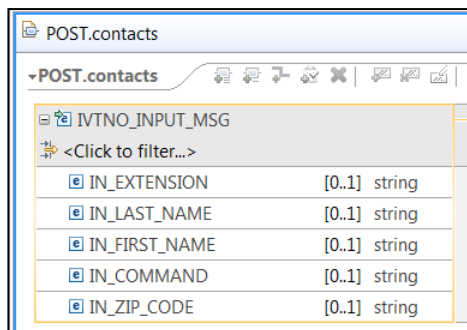
- ___11. Before you can do the field mappings you have to save your changes. From the menu bar, select *File→Save* or key sequence **Ctrl-S**.
- ___12. Now, click on the **Mapping...** button and select *Open Request Mapping*:



You should then see:



- ___13. In the upper-left, next to *INPUT_MSG*, click the little "+" sign symbol to expand the field:



You should now see:

The screenshot shows the 'POST.contacts' configuration window. It is divided into two main sections: 'Transferred' (left) and 'Service Definition' (right). Both sections show a table of fields for 'IVTNO_INPUT_MSG'.

Transferred (Left)		Service Definition (Right)	
IN_EXTENSION	[0..1] string	IN_EXTENSION	[0..1] string
IN_LAST_NAME	[0..1] string	IN_LAST_NAME	[0..1] string
IN_FIRST_NAME	[0..1] string	IN_FIRST_NAME	[0..1] string
IN_COMMAND	[0..1] string	IN_COMMAND	[0..1] string
IN_ZIP_CODE	[0..1] string	IN_ZIP_CODE	[0..1] string

Below the 'Transferred' table is the 'HTTP Request' section, which includes 'HTTP Headers' (Authorization: [0..1] string), 'Path Parameters', 'Query Parameters', and 'Body - IVTNO_INPUT_MSG'.

Red boxes with numbers 1, 2, and 3 are overlaid on the image to highlight specific areas:

- Box 1: Points to the 'Service Definition' table.
- Box 2: Points to the 'Transferred' table.
- Box 3: Points to the 'HTTP Request' section.

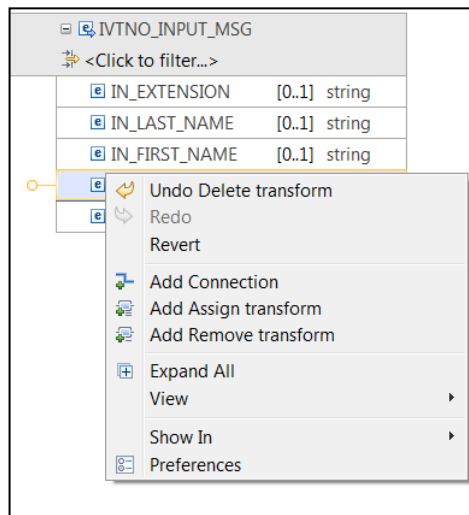
Notes:

1. The right side represents the fields exposed by the service definition for the request.
2. The left side represents the fields that will be exposed to the REST client. Initially it's a one-for-one mapping of fields from right-to-left. You will soon change that by assigning a fixed value to IN_COMMAND, leaving *four* fields exposed to the REST client.

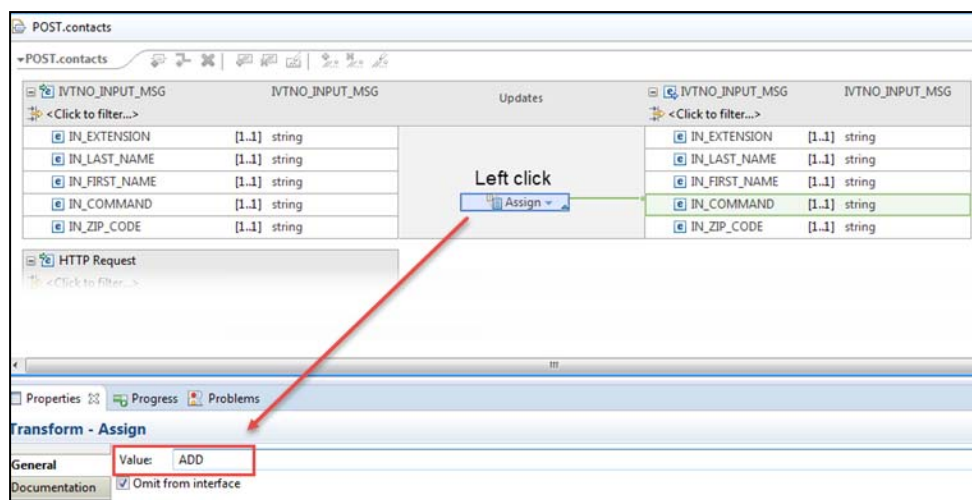
Note: The IN_COMMAND field is one field we do not wish to expose to the REST client. We know to add a contact the IMS transaction calls for a command of ADD. The POST verb is what we are using to "add a contact." Therefore, we can have z/OS Connect EE V3.0 assign the value ADD when this API's POST action is invoked.

3. 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 /contacts, so there is no path or query parameters.

- ___14. On the right side (block "1") in the picture above, right-click on the *IN_COMMAND* field and select *Add Assign transform*:



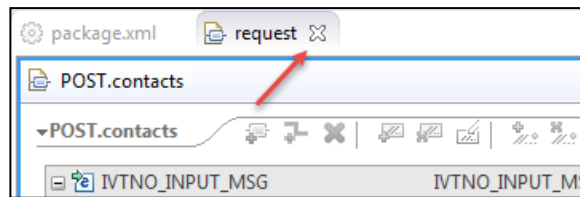
- ___15. A little *Assign* box will appear to the left of the *IN_COMMAND* field. Left click on that and then type **ADD** in the properties tab field at the bottom of the screen:



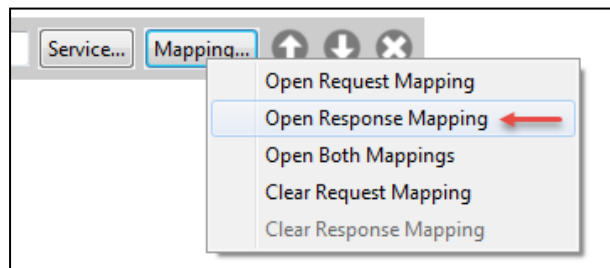
Note: Omit from interface" checkbox under the ADD value tells z/OS Connect EE V3.0 to hide the *IN_COMMAND* field from the REST client. Instead, a static value of **ADD** will be assigned to that field on each request for this API when the **POST** verb is used. Four fields remain exposed to the REST client. Those fields represent the four data elements of the contact record: last name, first name, phone extension and zip code. Those fields will be carried in the JSON body for the request. For the "add a contact" request mapping that is all that's needed

___16. Save the mapping: *File* → *Save* (or key sequence **Ctrl+S**).

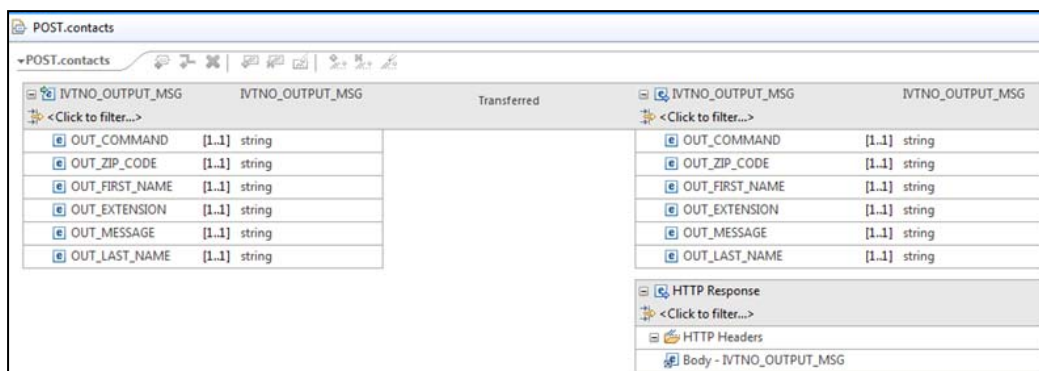
___17. Close the request mapping tab:



___18. Now click on the **Mapping...** button again, but this time select *Open Response Mapping*:

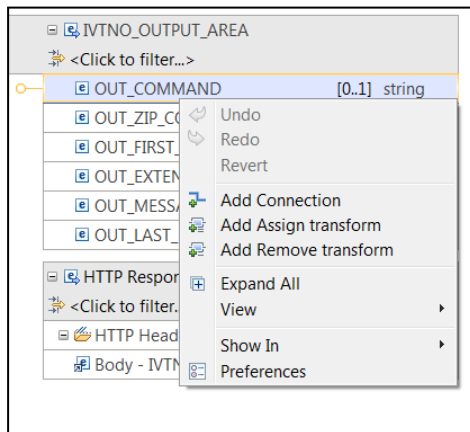


___19. You will see the fields that will be sent back to the REST client on the response:

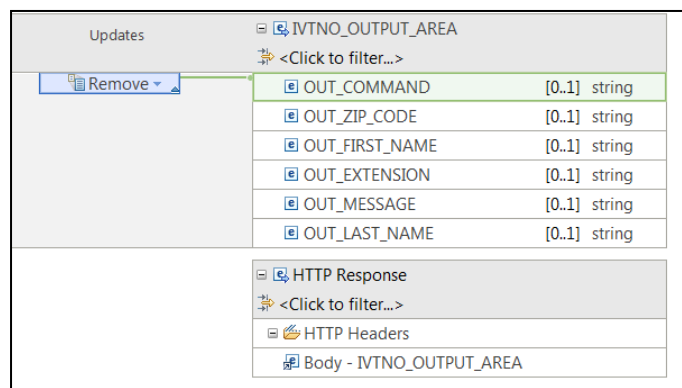


Six fields appear on the right side, but we don't want to send *OUT_COMMAND* back to the REST client. So we will "remove" that field.

- ___20. On the right side of the response mapping display, right click on *OUT_COMMAND* and select *Add Remove Transform*. You will then see:



- ___21. Click You should now see:



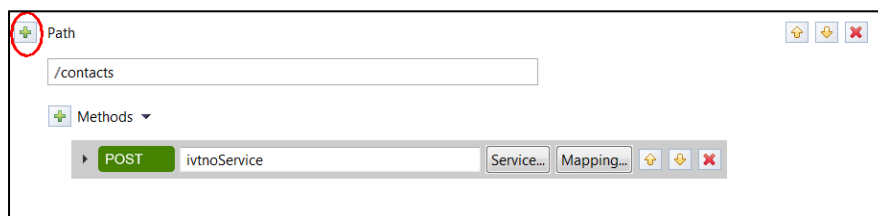
This means z/OS Connect EE V3.0 will hide *OUT_COMMAND* for the response, but will pass through (in a JSON object) the other five fields. *OUT_MESSAGE* will carry the message produced by the transaction (indicating success or failure of the "add" action). The other fields will provide confirmation of the data for the added contact record.

- ___22. Save the mapping: *File* → *Save* (or key sequence **Ctrl+S**).

- ___23. Close the *response* mapping tab.

We have completed the POST verb "add a contact" portion of the API. Now let's create the other three. Do the following:

- ___24. Click on the "+" symbol next to *Path* as shown here:



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

The screenshot shows the API Toolkit interface with two Path sections. The top section is titled 'Path' and contains a text field with '/contacts'. Below it, under 'Methods', there is a single entry for 'POST' with a service field containing 'ivtnoService' and a 'Mapping...' button. The bottom section is also titled 'Path' and contains a text field with '/newPath1'. Below it, under 'Methods', there are four entries: 'POST', 'GET', 'PUT', and 'DELETE'. Each entry has empty service and mapping fields, and a set of control buttons (up, down, and delete 'x').

___25. For the *Path*, provide the string */contacts/{lastName}* as shown here:

A close-up of the Path field in the API Toolkit. It shows a plus icon and the label 'Path' above a text input field containing the string '/contacts/{lastName}'.

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.

___26. 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:

The screenshot shows the API Toolkit interface with the Path field set to '/contacts/{lastName}'. Below it, under 'Methods', there are four entries: 'POST', 'GET', 'PUT', and 'DELETE'. Each entry has empty service and mapping fields, and a set of control buttons (up, down, and delete 'x'). A red arrow points to the delete 'x' button for the 'POST' method, which is also highlighted with a red box.

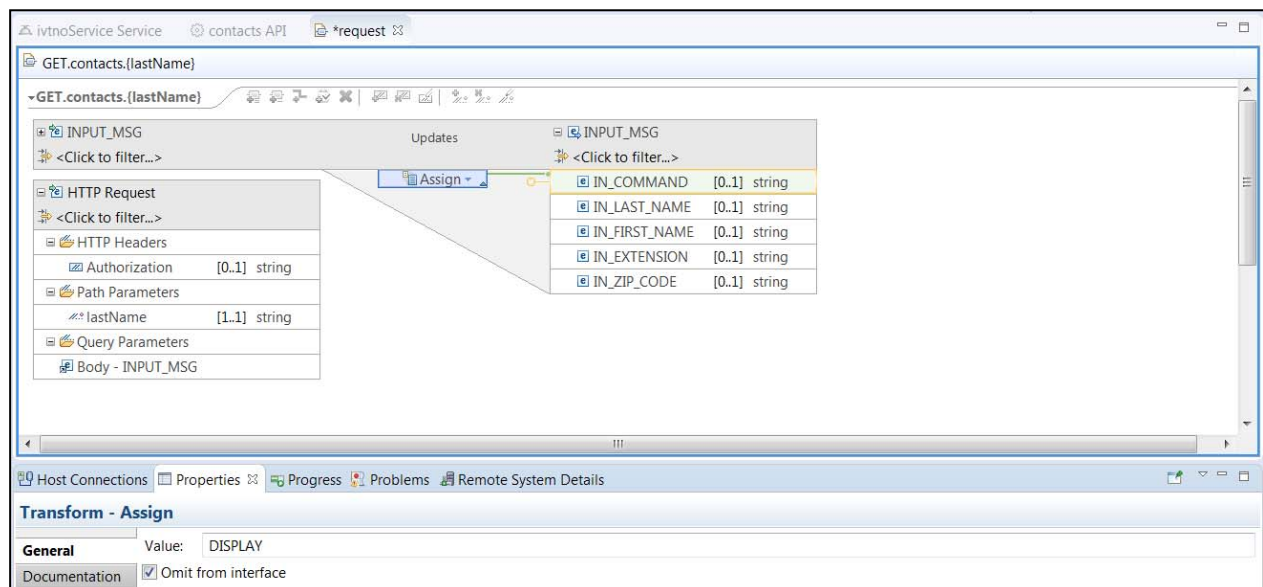
That leaves just *GET*, *PUT*, and *DELETE*:



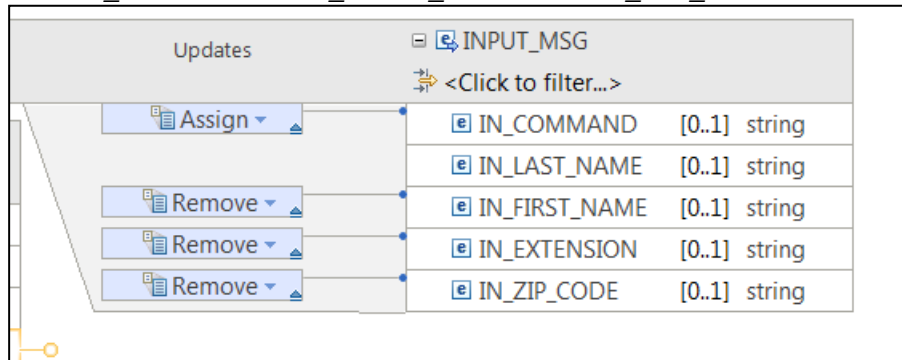
- ___27. For each (*GET*, *PUT*, and *DELETE*), click on the **Service...** button and select the *ivtnoService* service. The result should look like this:



- ___28. Save the changes: *File* → *Save* (or key sequence **Ctrl+S**).
- ___29. For the *GET* method, click on **Mapping...** and select *Open Request Mapping*.
- ___30. Start by assigning the value **DISPLAY** to the *IN_COMMAND* field:



31. Next, *Remove* the *IN_EXTENSION* *IN_FIRST_NAME* and *IN_ZIP_CODE* fields:

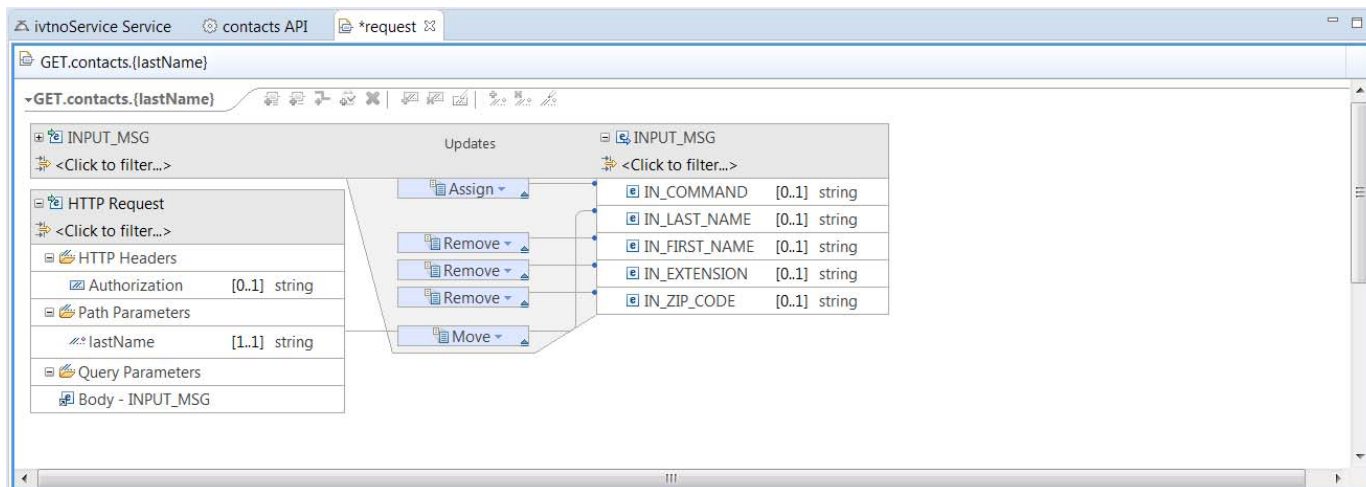


Note: To display a contact record, all we need on the request is the last name value (which is the unique key for the Phone Book sample program database). We do not need to send in the first name, phone extension or zip code. Therefore, we remove those fields and they are hidden from the client for this request.

You may see the editor move the boxes so they're not exactly lined up with the field row. That's okay. Follow the connector lines and you will see the mapping applies to the fields you selected.

Tech Tip: Another way to do would have been to create an “DISPLAY” service, e.g. *ivtnoServiceDISPLAY* which would have set the default value of the COMMAND field to DISPLAY and omitted this field as well as fields IN_FIRST_NAME, IN_EXTENSION and IN_ZIP_CODE from the service interface. There would be other services for ADD, UPDATE and DELETE COMMANDS and the API developer would use the appropriate HTTP method with the corresponding service. The services developer then could tailor the request and response messages based on the requirements of the command function without including every field for each command. For the API developer the significance of this would be that the request for GET would only contain the IN_LAST_NAME field and the path parameter would be mapped to this field and there would be no need to remove the other fields since they were previously omitted by the service developer.

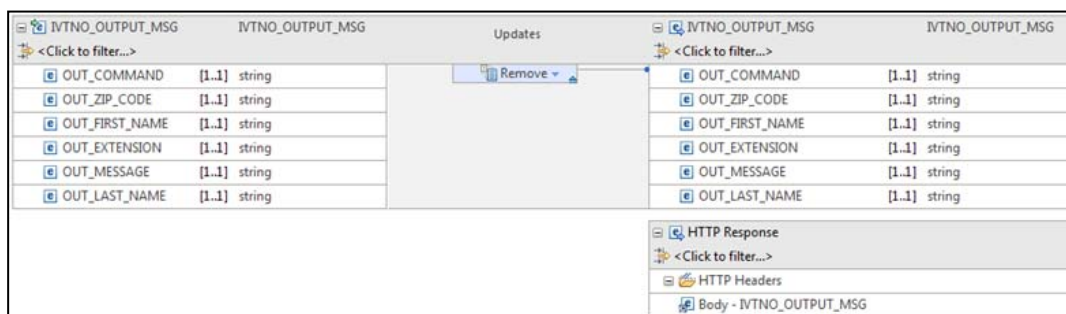
Finally, we need to map the path parameter *lastName* from the HTTP Request section to the *IN_LAST_NAME* field on the right side. This is done by left-clicking on the path parameter field, then moving the cursor over to the field and dropping the line there. The result is this:



And with that the request mapping for the "display a contact" action is complete: the *IN_COMMAND* field has the value *DISPLAY* assigned to it; the path parameter is "moved" to the *IN_LAST_NAME* field; and the first name, phone extension and zip code fields are removed.

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.

- ___33. Save the changes: *File* → *Save* (or key sequence **Ctrl+S**).
- ___34. Close the request mapping tab.
- ___35. Open the *response* mapping tab for the GET action.
- ___36. The mapping for the response is relatively simple. We want to *remove* the *OUT_COMMAND* field¹, but return to the REST client all the other fields. The mapping should look like this:



The *OUT_MESSAGE* field is important to return to the client. If a "lastName" value is on the request and that last name is not found in the database, the *OUT_MESSAGE* will indicate this with a "contact not found" message. The other fields (last name, first name, extension and zip) would be empty.

¹ We could hide this field on the service mapping and we would eliminate the need to "remove" it here. We left it exposed on the service mapping "just in case" an API designer wanted that field. In our case we don't want that field, so we remove it.

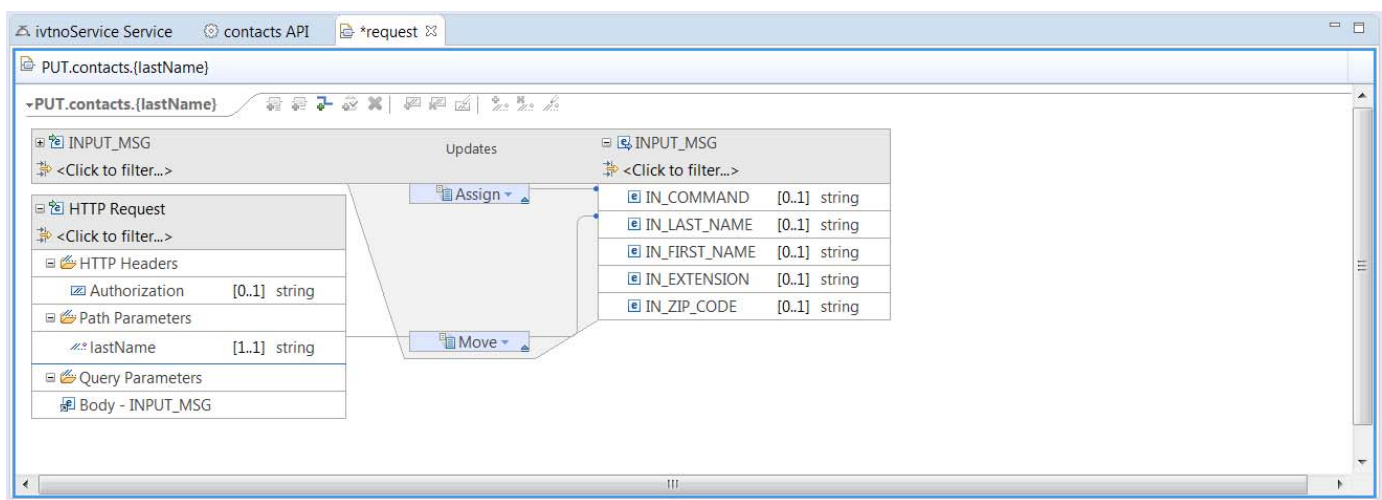
If the last name is found, then the contact information is provided for last name, first name, phone extension and zip code.

___37. Save the changes: *File* → *Save* (or key sequence **Ctrl+S**).

___38. Close the response mapping tab.

You've complete the *GET* action (display a contact) mapping. Now do the final two. Our instructions here will get less explicit. We will supply the desired mappings and commentary about why things are mapped the way they are.

___39. Open the *request* mapping for *PUT* (update a contact). Using the techniques seen in the previous mappings, map this request as follows:



Notes:

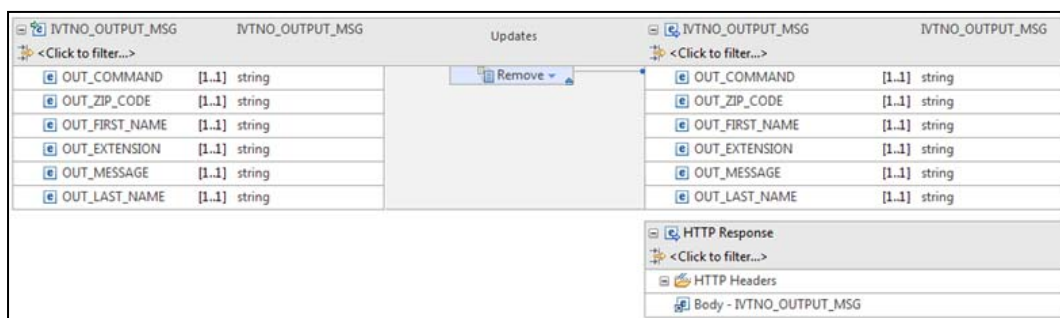
- The command to update a record is **UPDATE**, so we assign that static value to the IN_COMMAND field.
- The lastName path parameter is the key used by the transaction to understand which record is being updated². We move that value from the path parameter to the IN_LAST_NAME field.
- The leaves just IN_FIRST_NAME, IN_EXTENSION and IN_ZIP_CODE to be supplied in a JSON body for this request.
- The Phone Book sample program will update with the information it receives. If you sent in just the update for extension field, but left the first name and zip code fields blank, it would update the extension and make the first name and zip code fields blank.

___40. Save the changes: *File* → *Save* (or **Ctrl+S**).

___41. Close the request mapping tab.

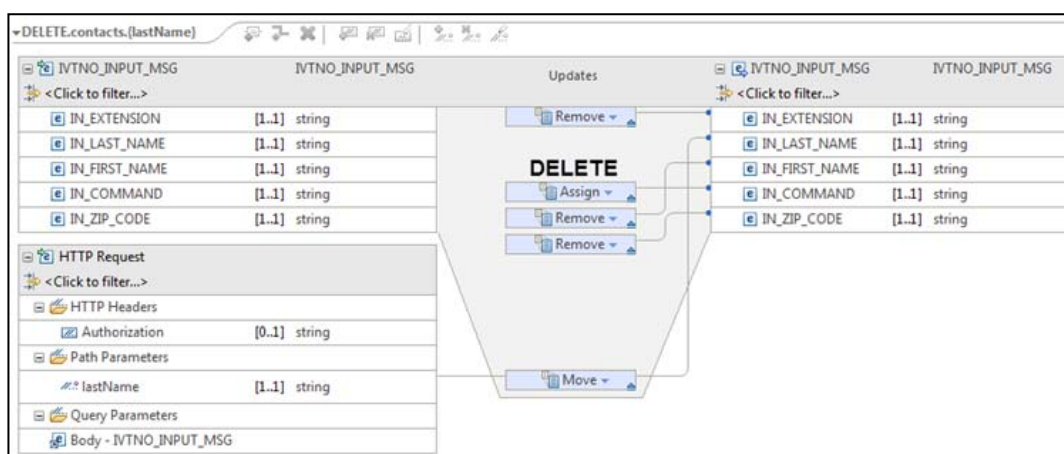
² This program does not allow the last name to be updated.

- ___42. Open the *response* mapping for PUT (update a contact). Remove the *OUT_COMMAND* field, then save and close the tab:



We return all the contact record fields along with the output message. This allows the client to see the updated values. The output message is important to provide because if the lastName path parameter value was not found, then the update action would not succeed. That would be reflected in the OUT_MESSAGE.

- ___43. Open the *request* mapping for *DELETE* (delete a contact). Using the techniques seen in the previous mappings, map this request as follows:

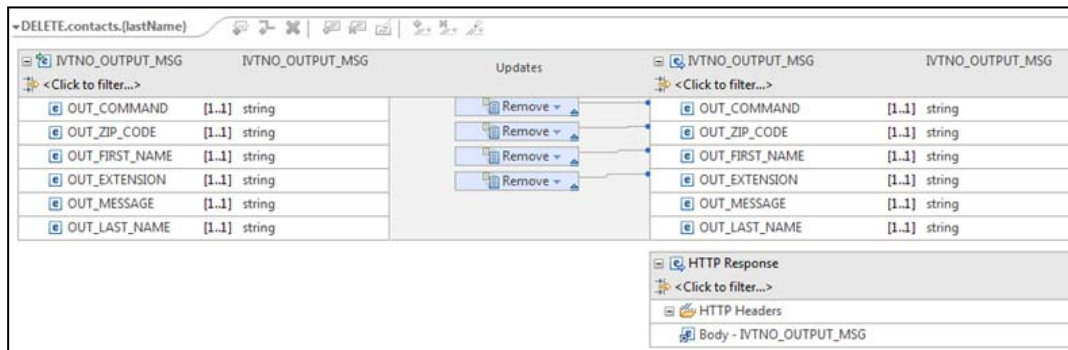


Notes:

- The command to update a record is **DELETE**, so we assign that static value to the *IN_COMMAND* field.
- The *lastName* path parameter is the key used by the transaction to understand which record is being deleted. We move that value from the path parameter to the *IN_LAST_NAME* field.

- ___44. We don't need to supply any other value for delete, so we remove *IN_FIRST_NAME*, *IN_EXTENSION* and *IN_ZIP_CODE* from the request.
- ___45. Save the changes: *File* → *Save* (or key sequence **Ctrl+S**).
- ___46. Close the request mapping tab.

___47. Open the *response* mapping for *DELETE* (delete a contact). Using the techniques seen in the previous mappings, map this request as follows:



Notes:

- When we delete a record, all we are interested in is whether the action succeeded (which will be indicated in OUT_MESSAGE) and the record that was deleted (as indicated by OUT_LAST_NAME).
- If the last name supplied on the path parameter is not found in the contact database, then that result will be indicated in OUT_MESSAGE.
- The other fields (OUT_COMMAND, OUT_ZIP_CODE, OUT_FIRST_NAME and OUT_EXTENSION) are not of interest on a delete action. So they are removed from the response.

___48. Save the changes: *File* → *Save* (or key sequence **Ctrl+S**).

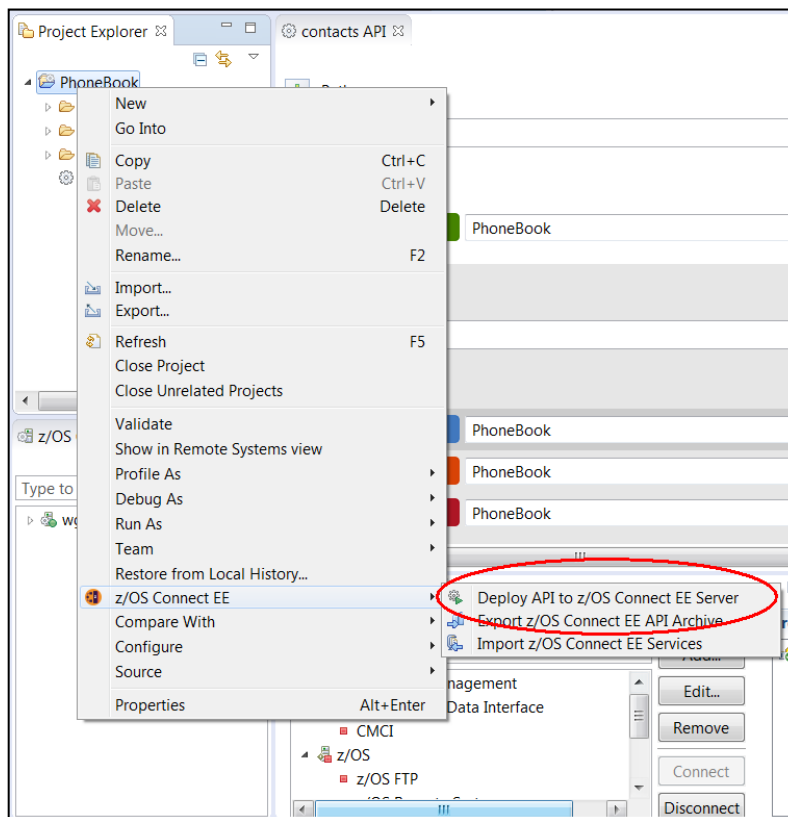
___49. Close the response mapping tab.

You're ready to deploy the API Archive file to z/OS Connect EE V3.0.

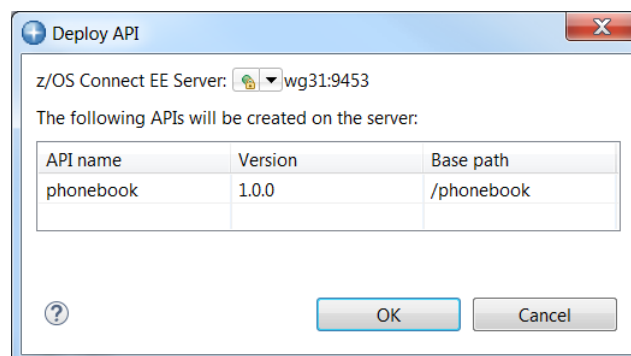
Deployment of the APIs into z/OS Connect EE V3.0

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

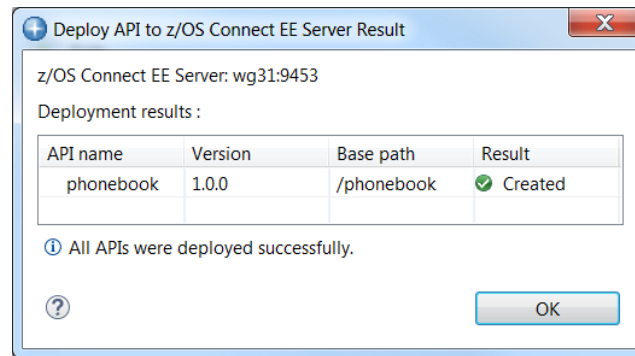
1. Switch back to the *z/OS Connect Enterprise Edition* perspective. In the *Project Explorer* view (upper left), right-mouse click on the *PhoneBook* folder, then select *z/OS Connect EE* → *Deploy API to z/OS Connect EE Server*.



2. If z/OS Explorer is connected to multiple z/OS Connect server the pull down arrow may have to be used 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.



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

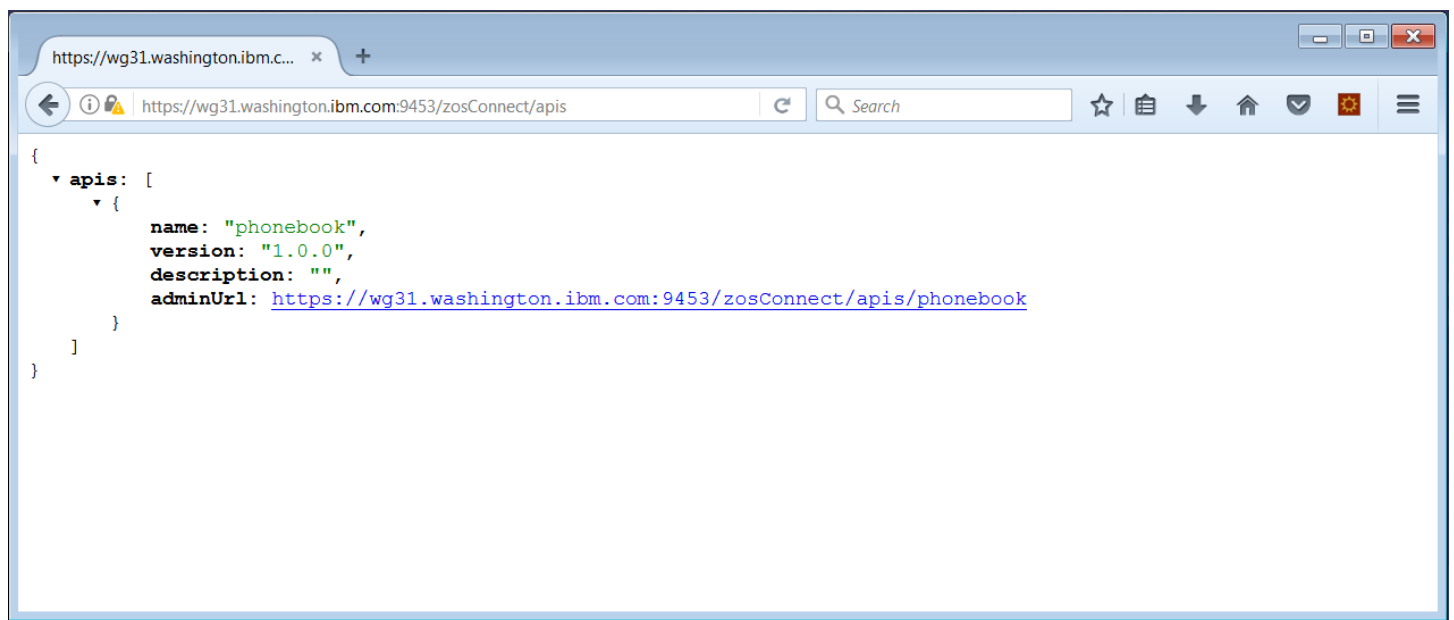


Test the IMS API

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

1. In a Firefox browser enter URL <https://wg31.washington.ibm.com:9453/zosConnect/apis> in the Firefox browser and you should see the window below. The `contacts` API now shows where none were displayed before. This is because this API has now been deployed to this server.

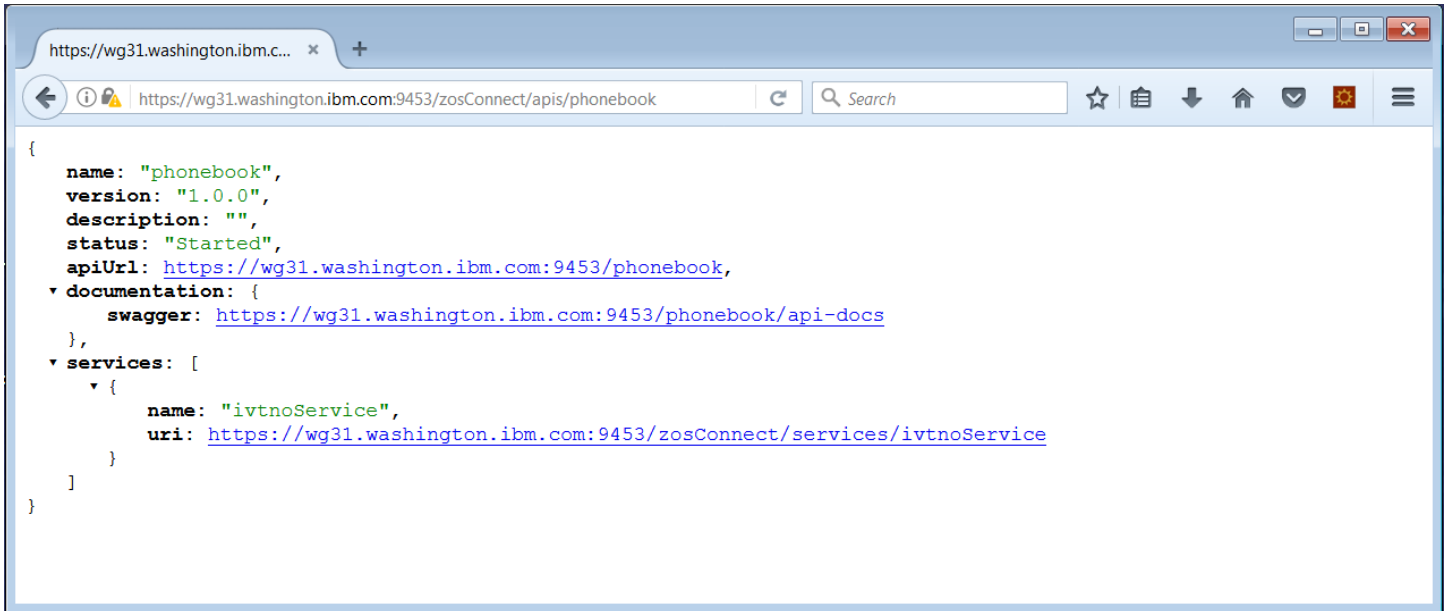
Tech Tip: You may be challenged by Firefox because the digital certificate used by the Liberty z/OS server is self-signed Click the **Add Exception** button to continue. If the **Add Exception** button is not displayed click the **Advanced** button. Then click on the **Confirm Security Exception** button. Next you may see a prompt you for a userid and password. If you do see the prompt, enter the username **Fred** and password **fredpwd** (case matters) and click **OK**. Remember we are using basic security and this is the user identity and password defined in the server.xml file.



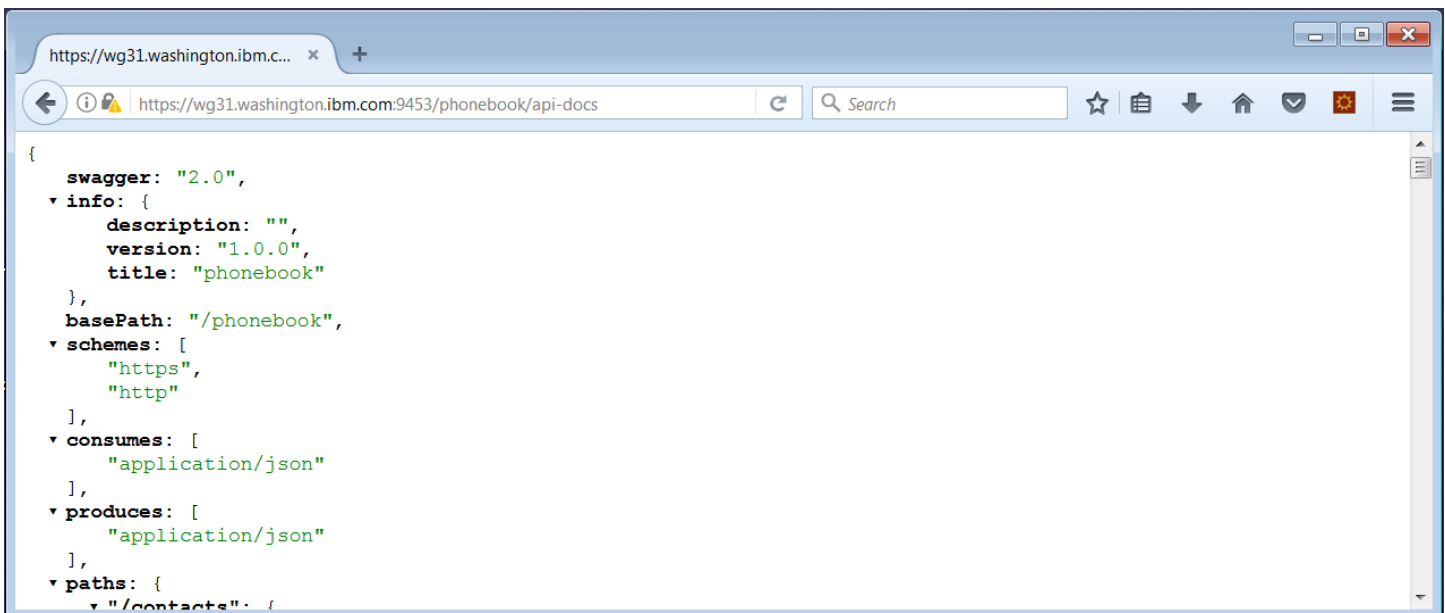
Tech Tip: It is very important to access the z/OS Connect server from a browser prior to any testing using the Swagger UI. Accessing a z/OS Connect URL from a browser starts an SSL handshake between the browser and the server. If this handshake has not performed prior to performing any test the test 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.

Note that other APIs may also be displayed.

2. If you click on *adminUrl* URL and the window below should be displayed:

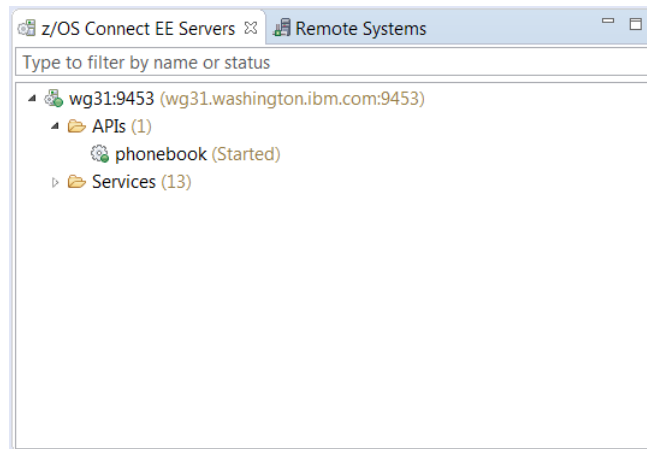


3. Finally click on the *swagger* URL and you should see the Swagger document associated with this API.

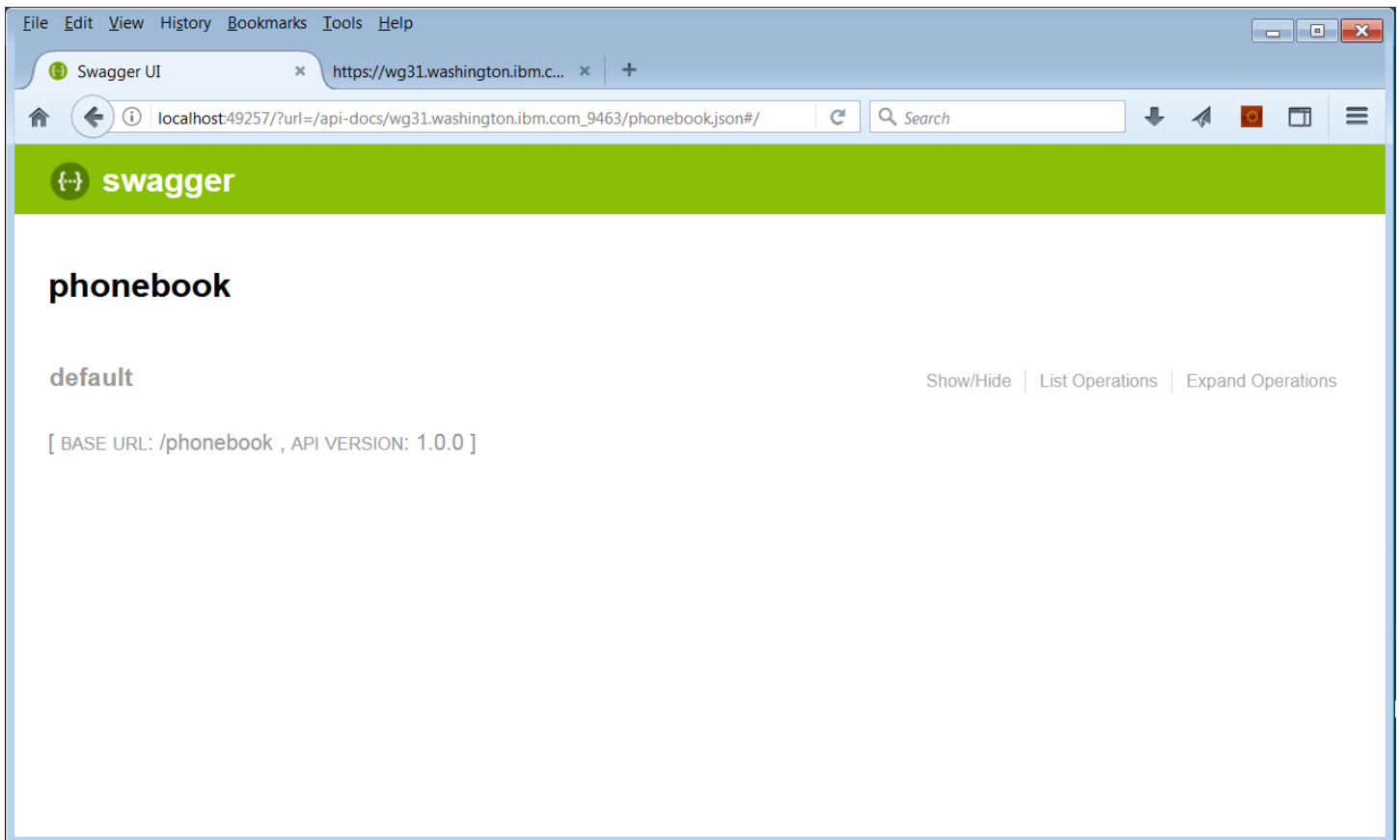


Explore this Swagger document and you will see the results of the request and response mapping performed earlier. This Swagger document can be used by a developer or other tooling to develop REST clients for this specific API.

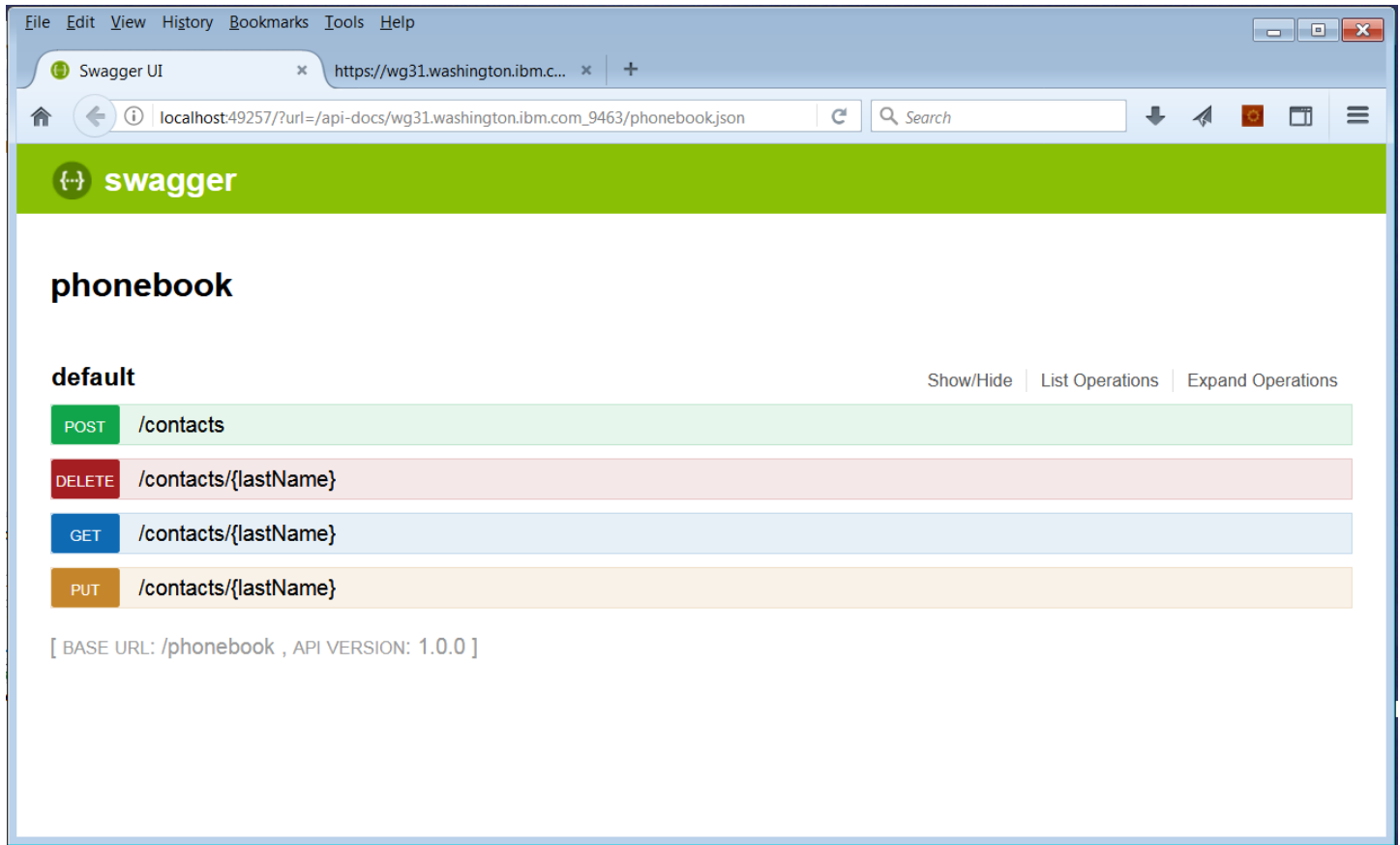
4. In the lower left-hand side of the *z/OS Connect Explorer* perspective there is view entitled *z/OS Connect EE Servers*. Expand *wg31:9453* and then expand the *APIs* folder. You should see a list of the APIs installed in the server.



5. Right mouse button click on *phonebook* and select *Open in Swagger UI*. Click OK if an informational prompt appears. This will open a Firefox window showing a *Swagger* test client (see below).



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



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

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
postIvtNoService_request	<input type="button" value="-"/> <div> <input type="button" value="-"/> <div> IN_LAST_NAME <input type="text"/> IN_FIRST_NAME <input type="text"/> IN_EXTENSION <input type="text"/> IN_ZIP_CODE <input type="text"/> </div> </div>	request body	body	Model Example Value

Parameter content type: application/json

Authorization:

8. Enter **Basic RnJlZDpmcmVkcHdk** (no trailing spaces) in the area beside Authorization and enter the values below for each field then click the **Try it out!** button.

- IN_LAST_NAME enter TEST
- IN_FIRST_NAME enter FIRST
- IN_EXTENSION enter 123
- IN_ZIP_CODE enter 11111

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
postIvtNoService_request	<input type="button" value="-"/> <div> <input type="button" value="-"/> <div> IN_LAST_NAME TEST IN_FIRST_NAME FIRST IN_EXTENSION 123 IN_ZIP_CODE 11111 </div> </div>	request body	body	Model Example Value

Parameter content type: application/json

Authorization: Basic RnJlZDpmcmVkcHdk

[Hide Response](#)

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

Response Body

```
{
  "OUTPUT_AREA": {
    "OUT_ZIP_CODE": "111111",
    "OUT_FIRST_NAME": "FIRST",
    "OUT_EXTENSION": "123",
    "OUT_MESSAGE": "ENTRY WAS ADDED",
    "OUT_LAST_NAME": "TEST"
  }
}
```

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:

Response Body

```
{
  "OUTPUT_AREA": {
    "OUT_ZIP_CODE": "111111",
    "OUT_FIRST_NAME": "FIRST",
    "OUT_EXTENSION": "123",
    "OUT_MESSAGE": "ADDITION OF ENTRY HAS FAILED",
    "OUT_LAST_NAME": "TEST"
  }
}
```

Response Code

200

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 **Basic RnJlZDpmcmVkcHdk** (no trailing spaces) in the area beside *Authorization* and then click the **Try it out!** button.

GET
/contacts/{lastName}

Response Class (Status 200)
normal response

Model | Example Value

```
{
  "OUTPUT_AREA": {
    "OUT_MESSAGE": "string",
    "OUT_LAST_NAME": "string",
    "OUT_FIRST_NAME": "string",
    "OUT_EXTENSION": "string",
    "OUT_ZIP_CODE": "string"
  }
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
lastName	TEST		path	string
Authorization	Basic RnJlZDpmcmVkcHdk		header	string

Try it out!

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

Response Body

```
{
  "OUTPUT_AREA": {
    "OUT_ZIP_CODE": "11111",
    "OUT_FIRST_NAME": "FIRST",
    "OUT_EXTENSION": "123",
    "OUT_MESSAGE": "ENTRY WAS DISPLAYED",
    "OUT_LAST_NAME": "TEST"
  }
}
```

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:

Parameter	Value	Description	Parameter Type	Data Type
putIvtnoService_request	<input type="button" value="-"/>	request body	body	Model Example Value
<div> <div>INPUT_MSG <input type="button" value="-"/></div> <div> <div>IN_FIRST_NAME</div> <div>IN_EXTENSION</div> <div>IN_ZIP_CODE</div> </div> </div> <div>Parameter content type: application/json</div>				
lastName	<input type="text" value="(required)"/>		path	string
Authorization	<input type="text"/>		header	string

14. Enter the values below:

- For IN_FIRST_NAME enter **FIRST**
- For IN_EXTENSION enter **456**
- For IN_ZIP_CODE enter **22222**
- For lastName enter **TEST**
- For Authorization enter **Basic RnJlZDpmcmVkcHdk**

and press the **Try it Out!** button.

15. You should see a *200 OK status code* in the *Response Header* area and a display of the *IVTNO_OUTPUT_AREA* record in the *Response Body (Preview)* area along with *ENTRY WAS UPDATED* in the *OUT_MESSAGE* field.

Response Body

```

{
  "OUTPUT_AREA": {
    "OUT_ZIP_CODE": "22222",
    "OUT_FIRST_NAME": "FIRST",
    "OUT_EXTENSION": "456",
    "OUT_MESSAGE": "ENTRY WAS UPDATED",
    "OUT_LAST_NAME": "TEST"
  }
}

```

Response Code

200

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

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
lastName	TEST		path	string
Authorization	Basic RnJlZDpmcmVkcHdk		header	string

Try it out! [Hide Response](#)

Note how only the last name appears in the response. The response mapping for the delete action was planned to return only the last name since that was the key information needed: the record that was deleted.

Response Body

```
{
  "OUTPUT_AREA": {
    "OUT_MESSAGE": "ENTRY WAS DELETED",
    "OUT_LAST_NAME": "TEST"
  }
}
```

Response Code

200

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.