**IBM z/OS Connect (OpenAPI 2.0)**

# Developing RESTful APIs for MQ Services



IBM Washington Systems Center

# Table of Contents

> **Important:** There is a folder on the Windows desktop named *CopyPaste Files.* This folder contains file with the commands and other text used in this workshop. Locate the file identified in the General Exercise Information and Guidelines section of this exercise and copy it to the desktop. Open the file and 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 MQ to perform this exercise. Even if MQ is not relevant to your current plans performing this exercise will give additional experience using the API Toolkit with developing 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.

If you have completed either the developing APIs exercise for Db2 or MVS Batch you can start with section *The MQ for z/OS Service Provider for z/OS Connect* on page 7.

## *General Exercise Information and Guidelines*

&#10003;  This exercise is based on the MQ Provider supplied by z/OS Connect.  There is another exercise which is based on the MQ Provider supplied by the IBM MQ product.

&#10003;  This exercise requires using z/OS user identity *USER1*.  The RACF password for this user is USER1.

&#10003;  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.

&#10003;  Please note that there may be minor differences between the screen shots in this exercise versus what you see when performing this exercise. These differences should not impact the completion of this exercise.

&#10003;  Text in **bold** and highlighted in <mark>yellow</mark> in this document should be available for copying and pasting in a file named *OpenAPI2 developing APIs CopyPaste* file.

&#10003;  For information regarding the use of the Personal Communication 3270 emulator, see the *Personal Communications Tips* PDF in the exercise folder.

# Connect 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 Explorer.

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

___2. You will be prompted for a workspace:
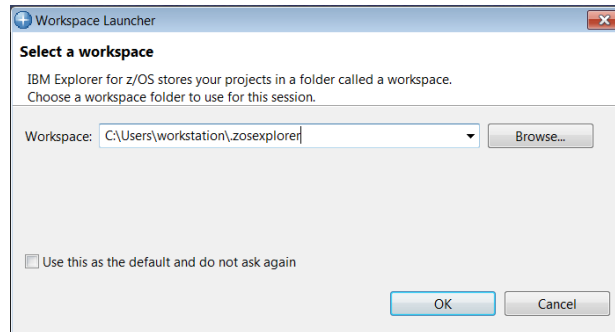


Take the default value by clicking **OK**.

___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:
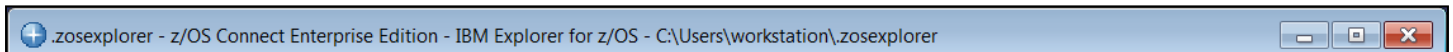


N.B. If a *Welcome* screen is displayed then click the white X beside *Welcome* to close this view.

___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.

Mitch Johnson (mitchj@us.ibm.com)
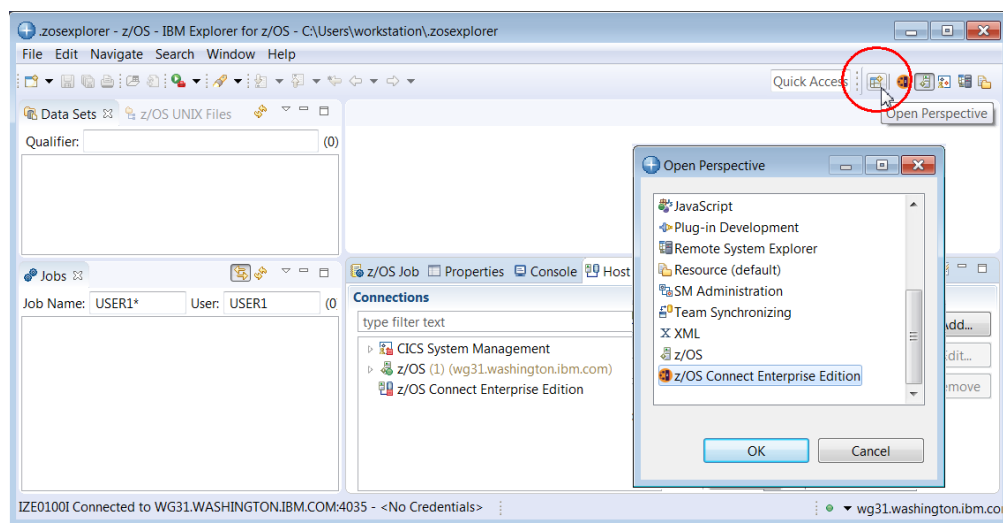
___5. To add a connection to the z/OS Connect Server select *z/OS Connect Enterprise Edition* connection in the *Host connection*s tab in the lower view and then click the **Add** button.
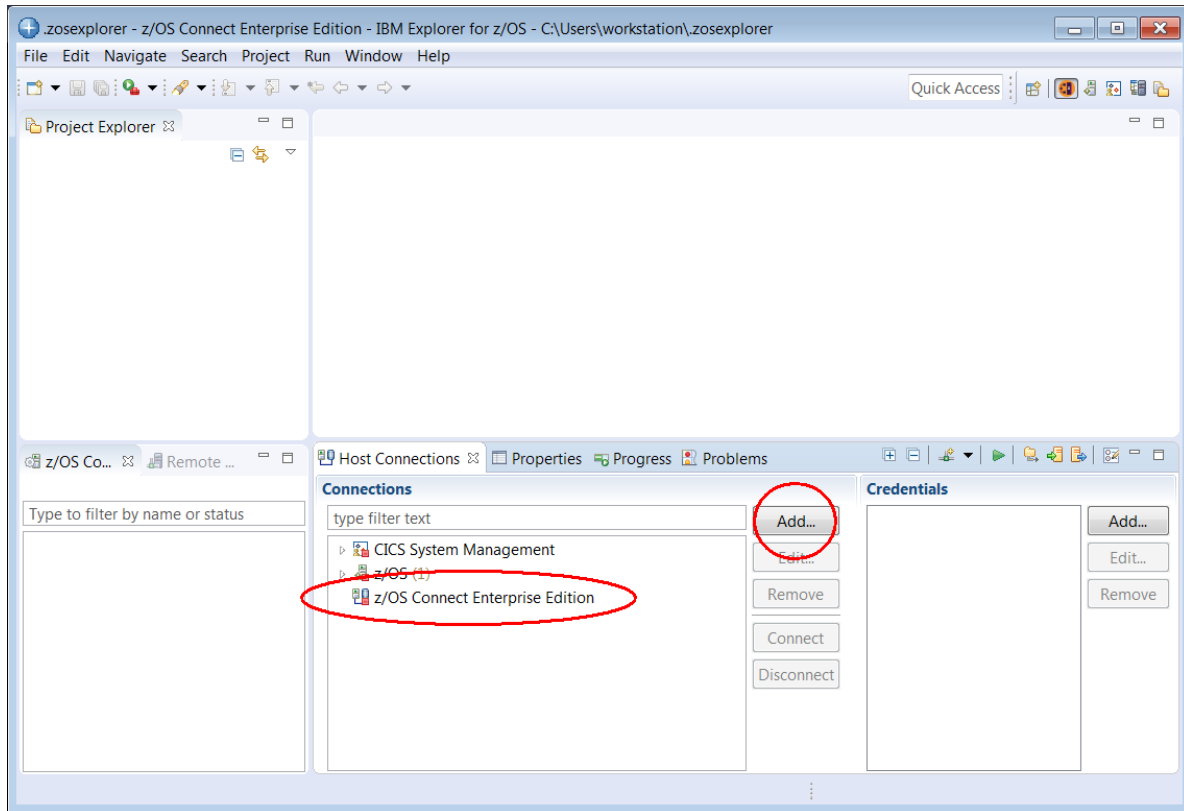


**Tech-Tip:** Eclipse based development tools like z/OS Explorer; provide a graphical interface consisting of multiple views within a single window.

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.

## *Summary*

The next step is the creation of the service and the composing and deployment of the API and then the testing of the API functions.

Mitch Johnson (mitchj@us.ibm.com)

# The MQ for z/OS Service Provider for z/OS Connect

This section provides an opportunity to develop REST interfaces to MQ queues (and topics) using the *MQ for z/OS Service Provider for z/OS Connect* feature which is shipped with z/OS Connect V3.0.21 and later. This feature provides support for two types of services. The first service type is a two-way service where the REST **POST** method is used to put a message on queue. A back-end application processes the message and provides a response message on a reply queue. The second service types are one-way services where REST **POST** and **GET** methods are used to do a MQ PUT and a MQ GET of a message from a single queue.

Switch to the *z/OS Connect Enterprise Edition* perspective. Let's start with a *two-way* service. That is a service where a JSON request message is put on a request queue which triggers some action. The response from that action is placed on a response queue and returned to the client as a response message.

## Create a MQ Two-Way service

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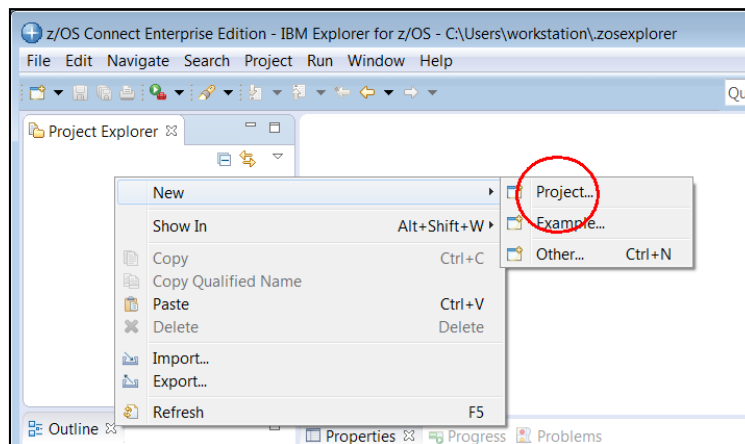
___2. In the *New Project* window, scroll down and open the *z/OS Connect Enterprise Edition* folder and select *z/OS Connect Service Project* and then click the **Next** button.
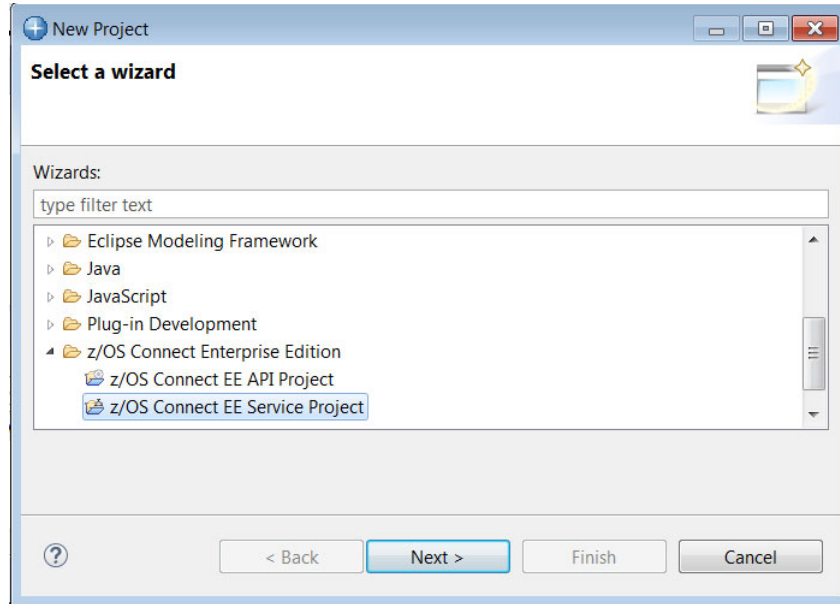


___3. On the *New Project* window enter ***miniloanService*** the *Project name* and use the pull-down arrow to select *MQ Two-Way Service* as the *Project type*. Click **Finish** to continue
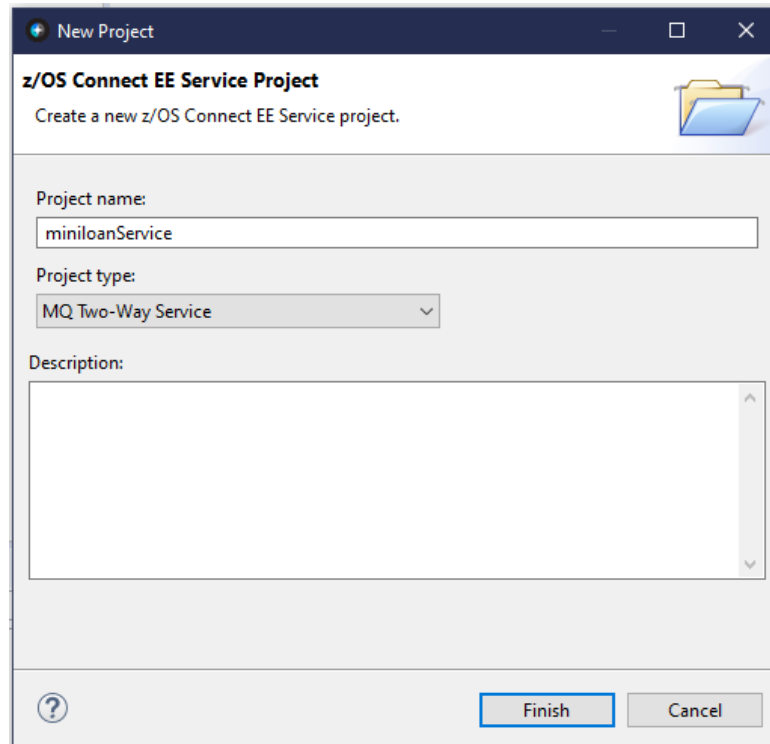
___4. This will open the *Service Project Editor:Definition* window for the *miniloanService*.   For now, disregard the message about the 2 errors detected, they will be addressed shortly.



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

___5. Click the **Create Service Interface** button to create the first service interface required by this API and enter a *Service interface name* of ***miniloanRequest***. Click **OK** to continue.

___6. This will open a *Service Interface Definition* window.

___7. The first step is to import the COBOL copy book (see below) that represents the inbound or request message.

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. Select *MQ Message* and click the *Import COBOL or PL/I data structure into the service interface* icon to start the import process.
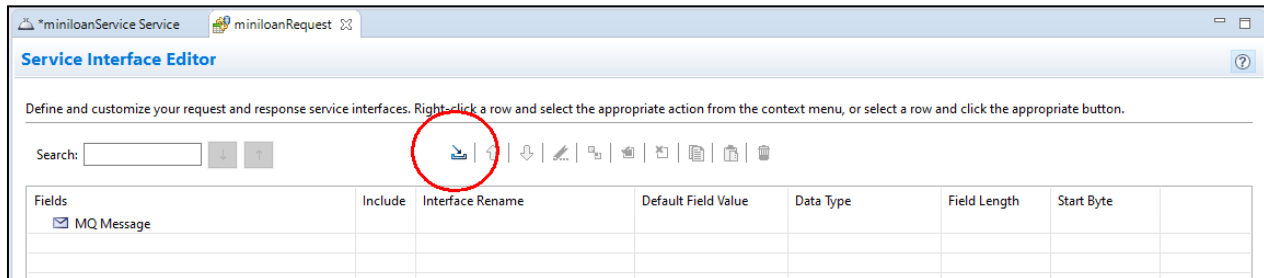


```
        01 MINILOAN-COMMAREA.
           10 name pic X(20).
           10 creditScore pic 9(18).
           10 yearlyIncome pic 9(18).
           10 age pic 9(10).
           10 amount pic   9(18).
           10 approved pic X.
              88 BoolValue value 'T'.
           10 effectDate pic X(8).
           10 yearlyInterestRate pic S9(5).
           10 yearlyRepayment pic 9(18).
           10 uid              pic X(8).
           10 messages-Num pic 9(9).
           10 messages pic X(60) occurs 1 to 10 times
                     depending on messages-Num.
```

___8. 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\MQLab* and then select file *MINILOAN*.cpy and click **Open** to import this file into this project. Click the **Add to Import List** button to continue.
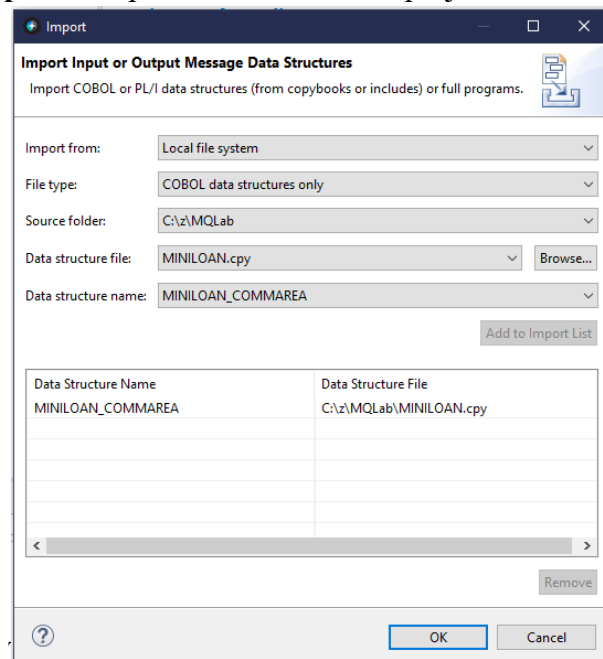
___9. Click **OK** and when you expand *MQ Message*, and *MINILOAN_COMMAREA* you will see the COBOL 'variables' that have been imported into the service project.
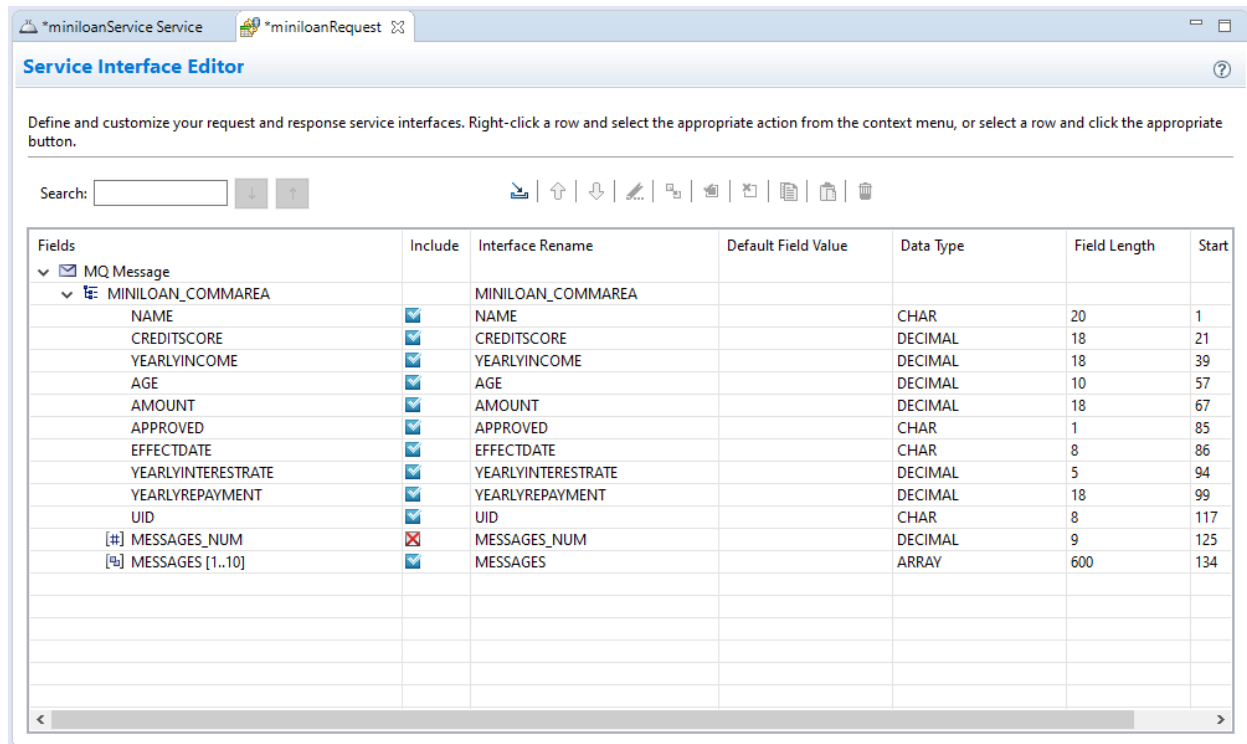


N.B. by default all the variables/fields are automatically selected to be include in the interface (*Include* column) and their default interface names are derived from the COBOL source shown earlier.

___10. 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 *MESSAGES* field from the interface.

___11. Select field *MESSAGES* and right mouse button click and select the *Exclude field from interface* option on the list of options.

___12. The other technique would have been to select field *MESSAGES* and use the *Exclude selected fields(s) from the interface* tool icon (the red X).

___13. Notice that the check boxes besides these this field are now unchecked. (You could have simply unchecked the box to accomplish the same results.)

___14. Next select field *APPROVED* and set its default value to **F** by using the right mouse button technique or the *Edit selected field* icon (the pencil) in the tool bar.



___15. Use these same techniques to change the default value for the *YEARLYINTERESTRATE* field to **00005**.

___16. Next remove the *APPROVED, YEARLYINTERESTRATE* and *UID* fields from the service interface by unchecking  the boxes beside these fields under the *Include* column.

___17. Rename the other interface names as shown below:

- *NAME*  to **name**
- *CREDITSCORE* to **creditScore**
- *YEARLYINCOME* to **yearlyIncome**
- *AGE* to **age**
- *AMOUNT* to **amount**
- *EFFECTDATE* to **effectiveDate**
- *YEARLYREPAYMENT* to **yearlyRepayment**

N.B. if the interface names are not renamed, the original interface names will appear in subsequent screen shots.

When finished your service definition interface should look like this.



___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 *miniloanRequest.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 message will include different fields in the interface, so a response service interface needs to be created.

___20. Click the **Create Service Interface** button to create the response service interface required by this API and enter a *Service interface name* of ***miniloanResponse***. Click **OK** to continue.

___21. This will open a *Service Interface Editor* window.

___22. On the *Service Interface Definition* window select *MQ Message* and click the *Import COBOL or PL/I data structure into the service interface* icon to start the import process.

___23. 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\MQLab* and then select file *MINILOAN*.cpy and click **Open** to import this file into this project. Click the **Add to Import List** button to continue.
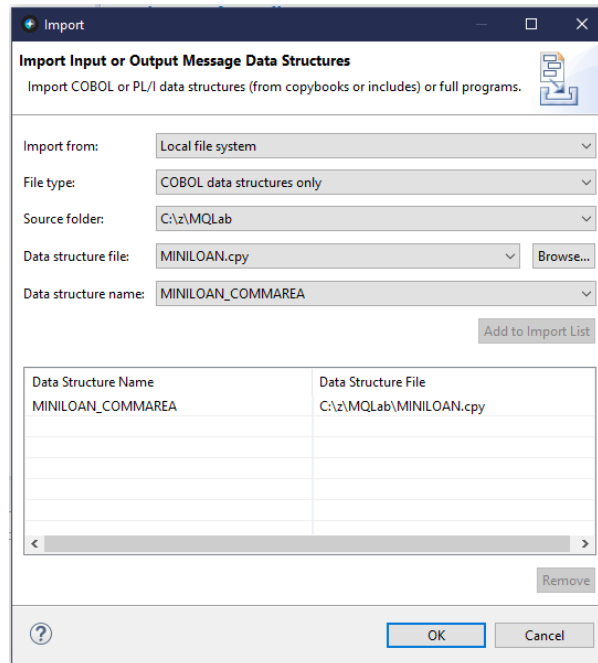


___24. Click **OK** and when you expand *MQ Message*, and *MINILOAN_COMMAREA* you will see the COBOL 'variables' that have been imported into the service project.

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

___25. Rename the interface names for the interface names as shown below:

- *NAME* to **name**
- *CREDITSCORE* to **creditScore**
- *YEARLYINCOME* to **yearlyIncome**
- *AGE* to **age**
- *AMOUNT* to **amount**
- *APPROVED* to **approved?**
- *EFFECTDATE* to **effectiveDate**
- *YEARLYINTERESTRATE* to **yearlyInterestRate**
- *YEARLYREPAYMENT* to **yearlyRepayment**
- *UID* to **uid**
- *MESSAGES* to **messages**

N.B. if the interface names are not renamed, the original interface names will appear in subsequent screen shots.

___26.  When finished the response interface should look like this:

_____27. Close the *miniloanResponse* view by clicking on the white X.

_____28. Use the pull-down arrow beside *Response service interface* to select *miniloanResponse.si*.



_____29. Next, we need to provide JNDI names for the JMS properties.  Click on the *Configuration* tab at the bottom of the *Overview* window to display the *Configuration* window. Enter **jms/qmgrCf**  in the area beside *Connection factory JNDI name*,  **jms/request** in the area beside *Request Destination JNDI name* **jms/response** in the area beside *Reply destination JNDI name* and **30000** in the area beside *Wait interval*.



**Tech Tip:** The values for these attributes must match connection factories and destinations (queues) defined in the server.xml,  see an example of the configuration elements on page 37.

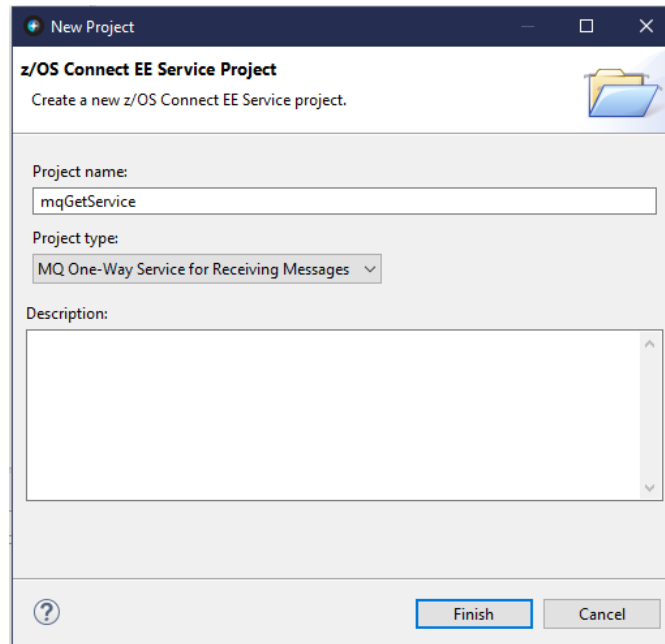___30. Use the *Definition* tab at the bottom of the return to the return to the *Definition* view and click on the *Advanced Options* button. This will display the *Advanced Service Data Conversion Option* view. Check the box beside the *Omit empty fields* output options.  This option will remove the blank or empty messages returned in the response message.
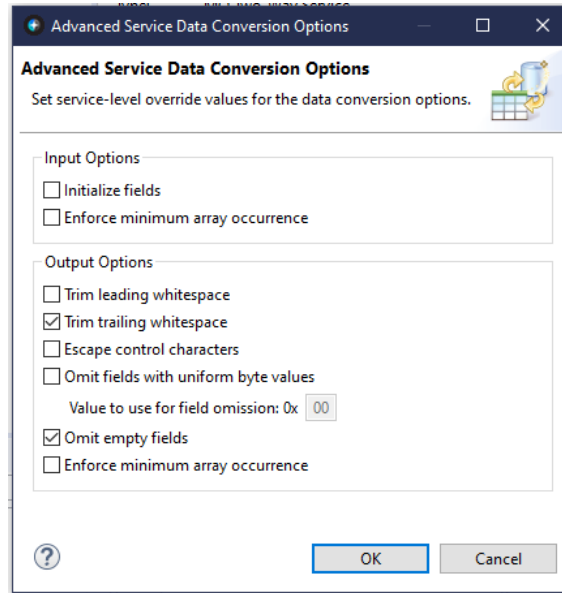


___31. Close all *miniloanService* related views

## *Create a MQ One-Way Service for Receiving Messages*

___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.

___3. On the *New Project* window enter **mqGetService** the *Project name* and use the pull-down arrow to select *MQ One-Way Service for Receiving Messages* as the *Project type*. Click **Finish** to continue

___4. This will open the *Service Project Editor:Definition* window for the *mqGetService*. For now, disregard the message about the 1 error detected, it will be addressed shortly.



___5. Click the **Create Service Interface** button to create the first service interface required by this API and enter a *Service interface name* of **mqGetServiceResponse**. Click **OK** to continue.

___6. This will open a *Service Interface Definition* window.



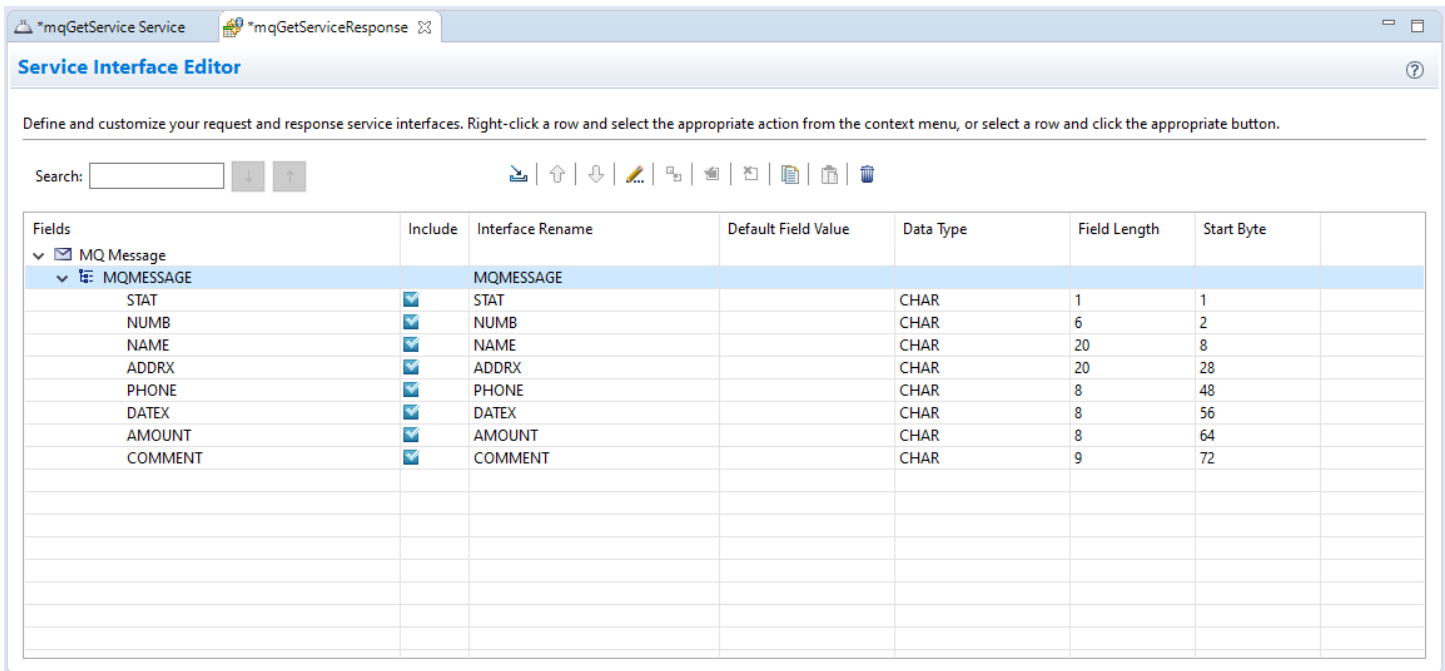___7. The next step is to import the COBOL copy book (see below) that represents the message.

```
        01 MQMESSAGE.
                10  stat        PIC X(1).
                10  numb        PIC X(6).
                10  name        PIC X(20).
                10  addrx       PIC X(20).
                10  phone       PIC X(8).
                10  datex       PIC X(8).
                10  amount      PIC X(8).
                10  comment     PIC X(9).
```

___8. On the *Service Interface Definition* window select *MQ Message* and click the *Import COBOL or PL/I data structure into the service interface* icon to start the import process.

___9. On the *Import* 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\MQLab* and then select file *FILEAMQ*.cpy and click **Open** to import this file into this project. Click the **Add to Import List** button to continue.

___10.  Click **OK** and when you expand *MQ Message*, and *MQMESSAGE* you will see the COBOL 'variables' that have been imported into the service project.

| Fields | Include | Interface Rename | Default Field Value | Data Type | Field Length | Start Byte | |
|---|---|---|---|---|---|---|---|
| ∨ ☑ MQ Message | | | | | | | |
| ∨ ▦ MQMESSAGE | | MQMESSAGE | | | | | |
| STAT | ☑ | STAT | | CHAR | 1 | 1 | |
| NUMB | ☑ | NUMB | | CHAR | 6 | 2 | |
| NAME | ☑ | NAME | | CHAR | 20 | 8 | |
| ADDRX | ☑ | ADDRX | | CHAR | 20 | 28 | |
| PHONE | ☑ | PHONE | | CHAR | 8 | 48 | |
| DATEX | ☑ | DATEX | | CHAR | 8 | 56 | |
| AMOUNT | ☑ | AMOUNT | | CHAR | 8 | 64 | |
| COMMENT | ☑ | COMMENT | | CHAR | 9 | 72 | |

N.B. by default all the variables/fields are automatically selected to be include in the interface (*Include* column) and their default interface names are derived from the COBOL source shown earlier.
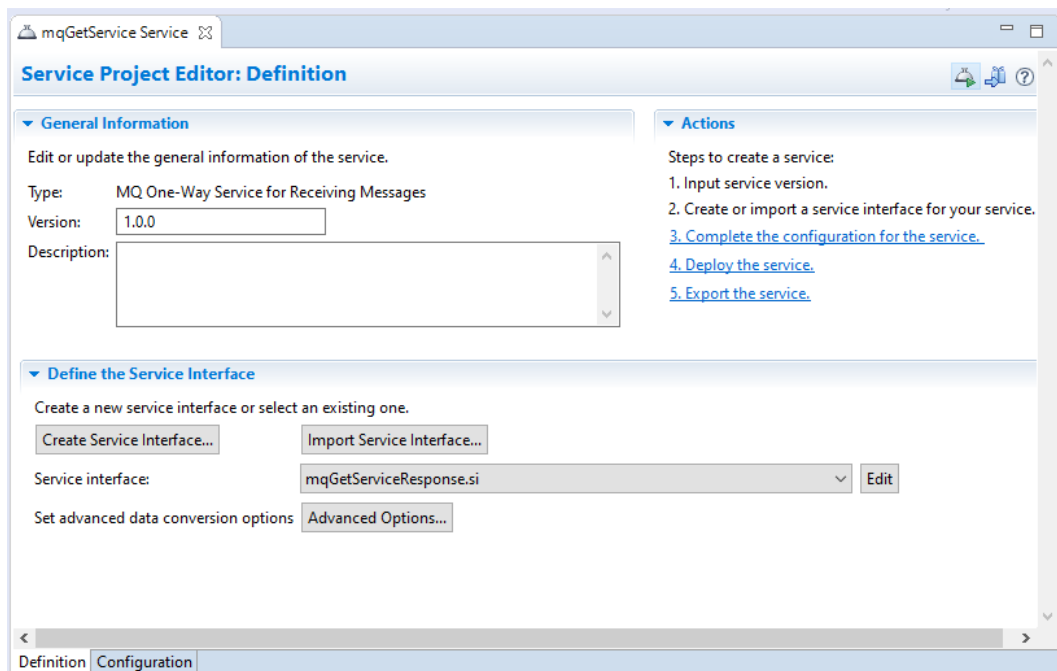
___11. In this window, you can edit and change the property name (e.g. *Interface name*) or exclude specific fields entirely from the interface. In this case the interface names will be changed, and no fields will be eliminated from the interface of the request message.

___12. Rename the interface names as shown below:

- STAT to *status*

- NUMB to *number*

- NAME  to *name*

- ADDRX to *address*

- PHONE to *phone*

- DATEX to *date*

- AMOUNT to *amount*

- COMMENT to *comment*


N.B. if the interface names are not renamed, the original interface names will appear in subsequent screen shots.

___13. Close the Service Interface Definition window by clicking on the white X in the tab being sure to save the changes. Note that you can use he **Edit** buttons to return to the *Service Interface Definition* window for this interface.

___14. Next, we need to provide JNDI names for the JMS properties.  Click on the Configuration tab at the bottom of the *Overview* window to display the *Configuration* window. Enter ***jms/qmgrCf***  in the area beside *Connection factory JNDI name* and *jms/default* in the area beside *Destination JNDI name*



___15. Use the *Definition* tab at the bottom of the return to the return to the *Definition* view.

> **Tech Tip:** The values for these attributes must be defined in the server.xml,  see the example of the configuration elements on page 53.

## *Create a MQ One-Way Service for Sending Messages*

___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.

___3. On the *New Project* window enter *mqPutService* the *Project name* and use the pull-down arrow to select *MQ One-Way Service for Sending Messages* as the *Project type*. Click **Finish** to continue
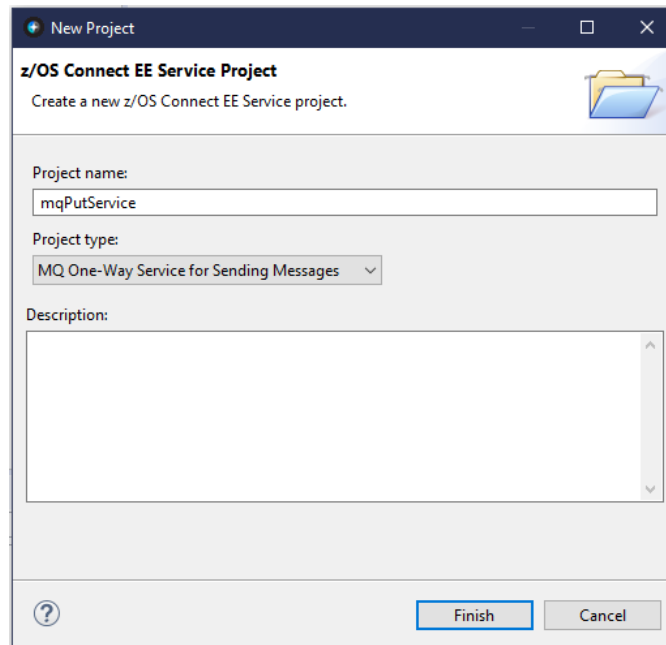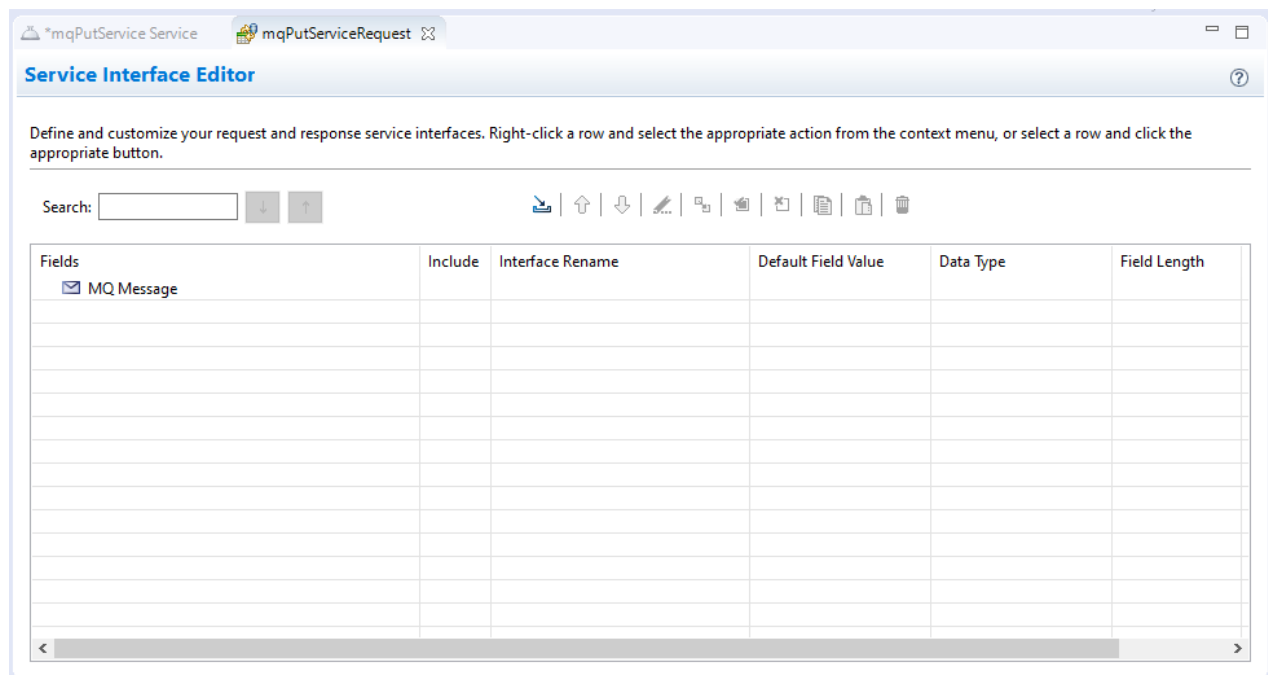
___4. This will open the *Service Project Editor:Definition* window for the *mqPutService*.   For now, disregard the message about the 1 error detected, it will be addressed shortly.
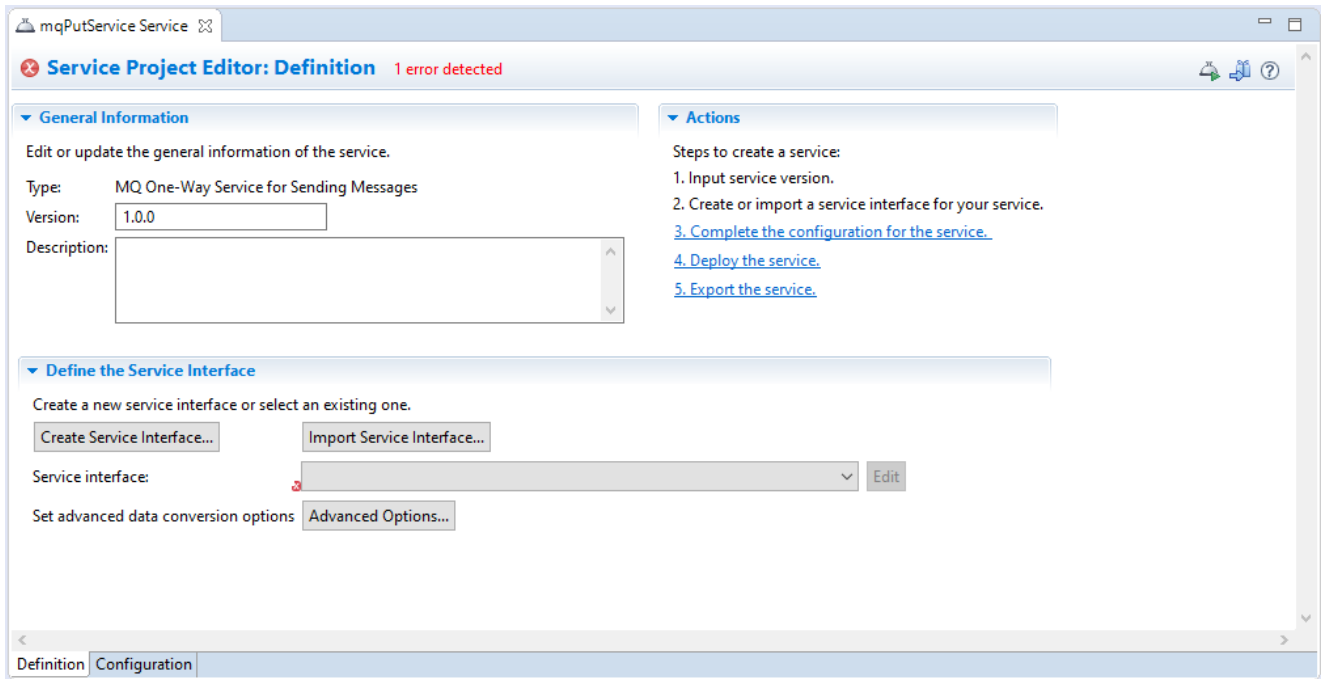


___5. Click the **Create Service Interface** button to create the first service interface required by this API and enter a *Service interface name* of ***mqPutServiceRequest***. Click **OK** to continue.

___6. This will open a *Service Interface Definition* window.

___7. The next step is to import the COBOL copy book (see below) that represents the message.
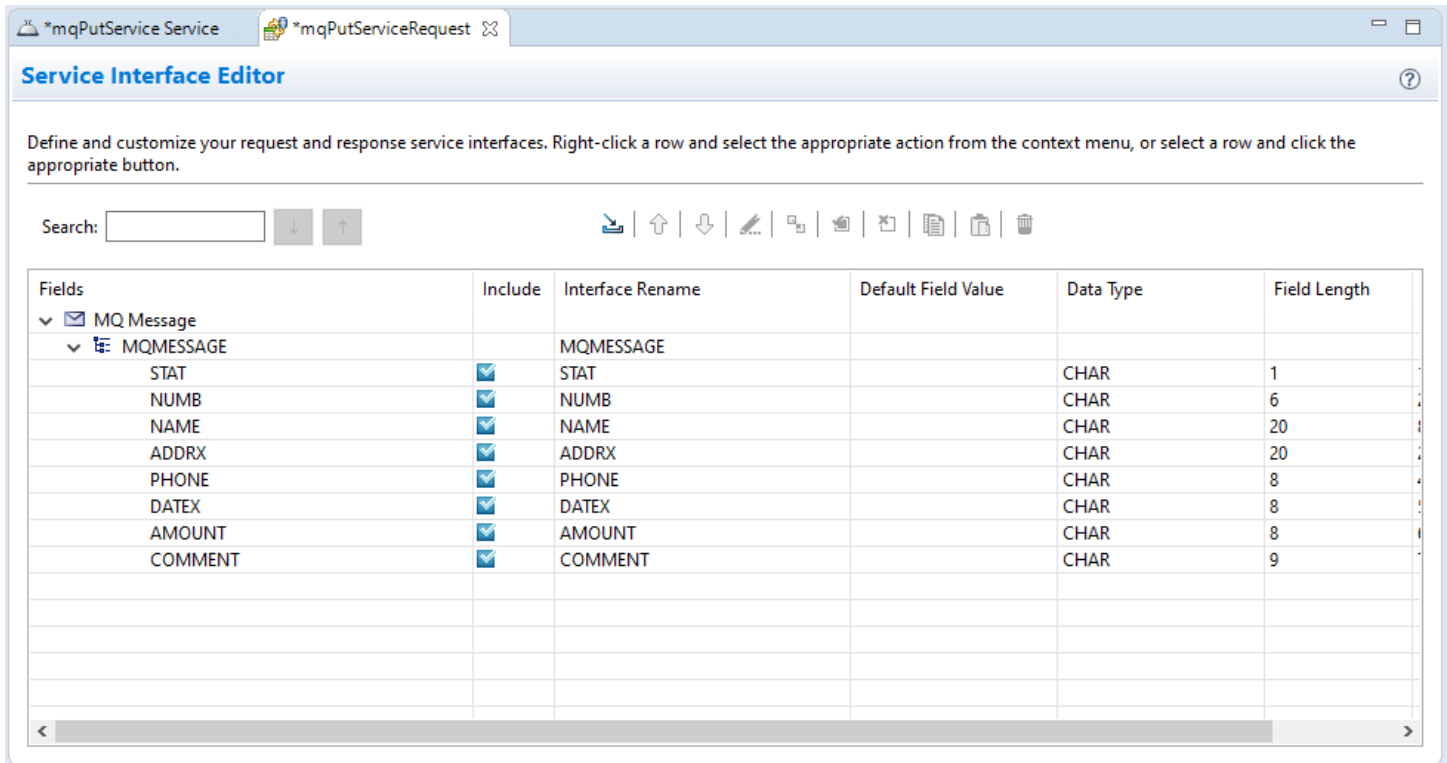
```
01 MQMESSAGE.
       10   stat        PIC X(1).
       10   numb        PIC X(6).
       10   name        PIC X(20).
       10   addrx       PIC X(20).
       10   phone       PIC X(8).
       10   datex       PIC X(8).
       10   amount      PIC X(8).
       10   comment     PIC X(9).
```

___8. On the *Service Interface Definition* window select *MQ Message* and click the *Import COBOL or PL/I data structure into the service interface* icon to start the import process.

___9. On the *Import* 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\MQLab* and then select file *FILEAMQ*.cpy and click **Open** to import this file into this project. Click the **Add to Import List** button to continue.

___10. Click **OK** and when you expand *MQ Message*, and *MQMESSAGE* you will see the COBOL 'variables' that have been imported into the service project.
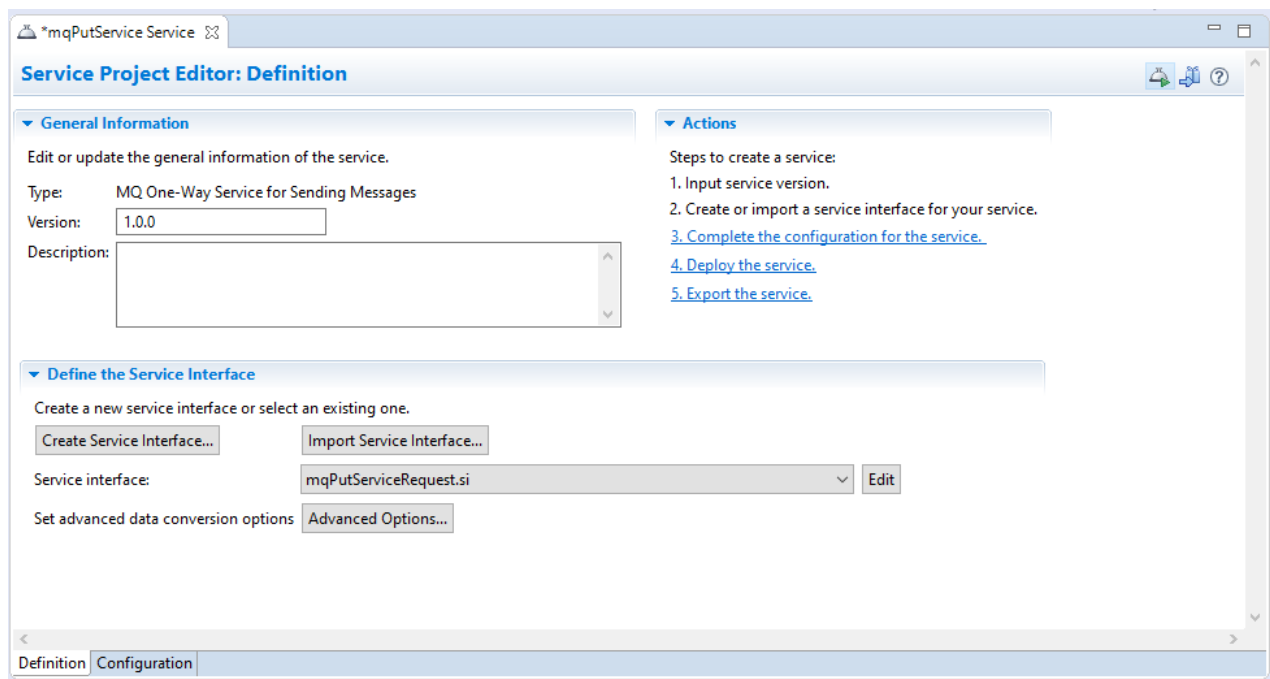


N.B. by default all the variables/fields are automatically selected to be include in the interface (*Include* column) and their default interface names are derived from the COBOL source shown earlier.

___11. In this window, you can edit and change the property name (e.g. *Interface name*) or exclude specific fields entirely from the interface. In this case the interface names will be changed, and no fields will be eliminated from the interface of the request message.

___12. Rename the interface names as shown below:

- STAT to *status*
- NUMB to *number*
- *NAME* to *name*
- ADDRX to *address*
- PHONE to *phone*
- DATEX to *date*
- AMOUNT to *amount*
- COMMENT to *comment*

___13. Close the *Service Interface Definition* window by clicking on the white X in the tab being sure to save the changes. Note that you can use the **Edit** buttons to return to the *Service Interface Definition* window for this interface.

___14. Next, we need to provide JNDI names for the JMS properties.  Click on the Configuration tab at the bottom of the *Overview* window to display the *Configuration* window. Enter **jms/qmgrCf**  in the area beside *Connection factory JNDI name* and *jms/default* in the area beside *Destination JNDI name*



---

> **Tech Tip:** The values for these attributes must be defined in the server.xml,  see the example of the configuration elements on page 53.

Save and close the service

## *Export and deploy the Service Archive files*

Before a service interface can be used it must be exported to create Service Archive (SAR) file and deployed as a SAR file to the server. The exported SAR is used in developing an API in the z/OS Connect Toolkit. This section describes the process for exporting and deploying SAR files.

___1. First 'export' the service interface as a SAR 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 *miniloanService* 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.
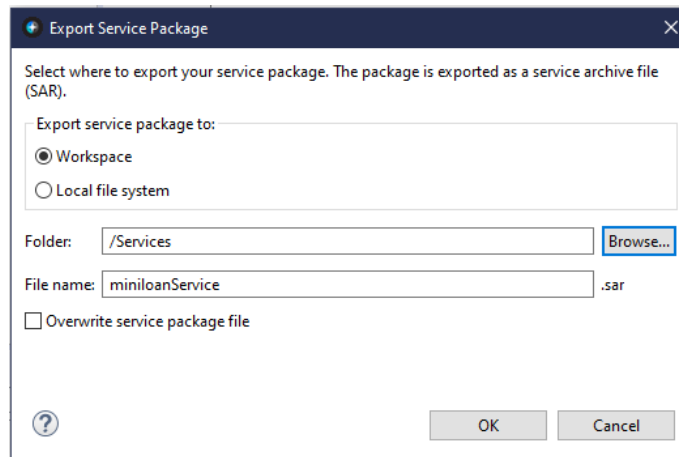


___4. Select the *miniloanService* again projects and right mouse button click again and on the pop-up selection select **z/OS Connect** ➔ **Deploy Service to z/OS Connect Server**. On the *Deploy Service* window select the target server (*wg31:9453*) and click **OK** twice to continue.
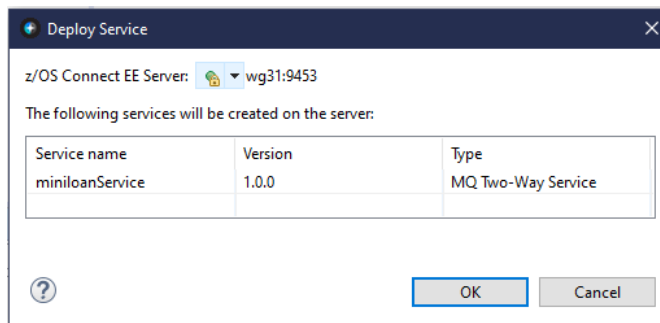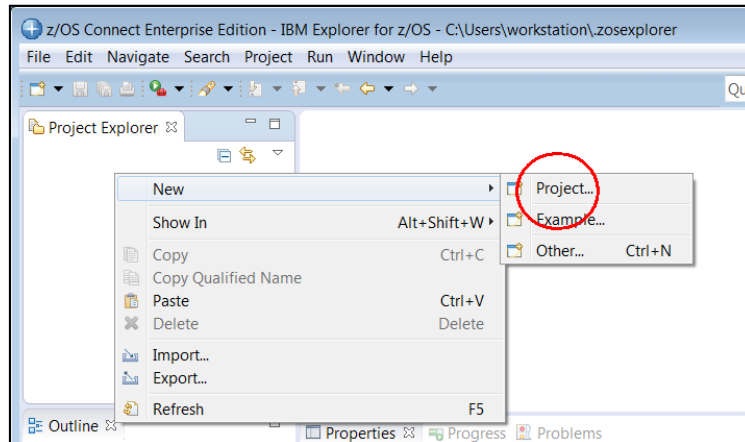


___5. Repeat these steps to deploy and export the services *mqGetService* and *mqPutService*
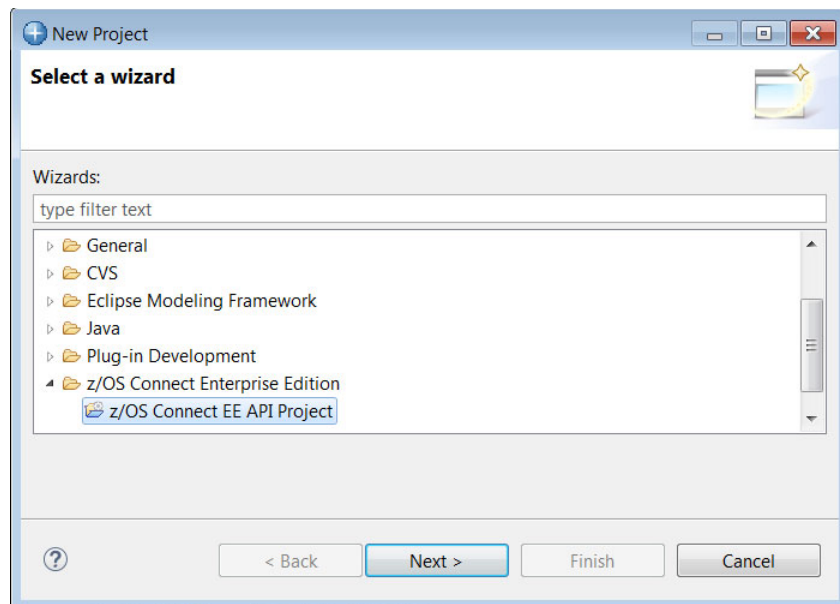
# Create the MQ API project

In this section an API that uses both MQ a two-way service and two one-way services will be used.
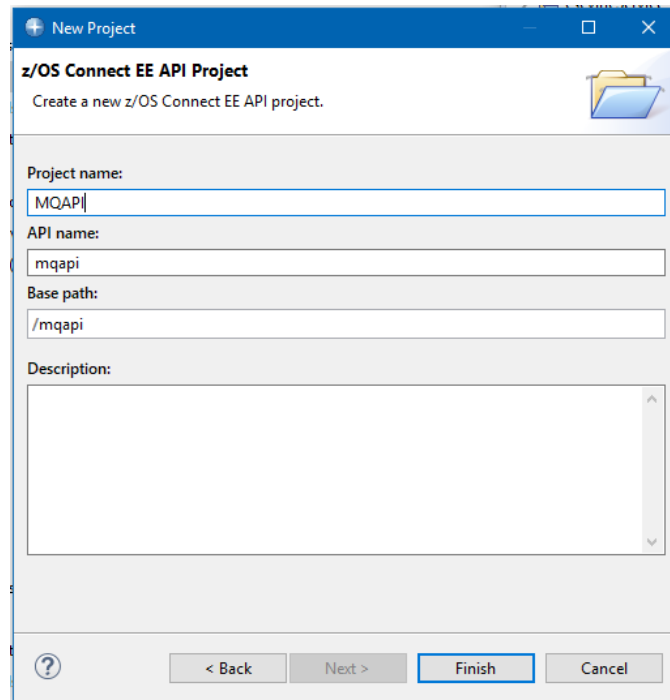
___1. In the *z/OS Connect Enterprise Edition* perspective of the z/OS Explorer create a new API project by clicking the right mouse button and selecting *New → Project*:

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

___3. Enter **MQAPI** for the *Project name.* Be sure the *API name* is set to **mqapi** and the *Base path is* set to
   */mqapi*.   Click **Finish** to continue.



> **Important:**  The values are somewhat arbitrary, but they do relate to later tasks.  If you use
> the values and cases as supplied, then the subsequent commands and the use of subsequent
> URLs will work seamlessly.

## *Import the SAR files*

___1. In the z/OS Explorer in the *z/OS Connect Enterprise Edition* perspective in the the *Project Explorer* view (upper left), right-click on the *MQAPI* project, then select *z/OS Connect* and then *Import z/OS Connect Services* (see below):
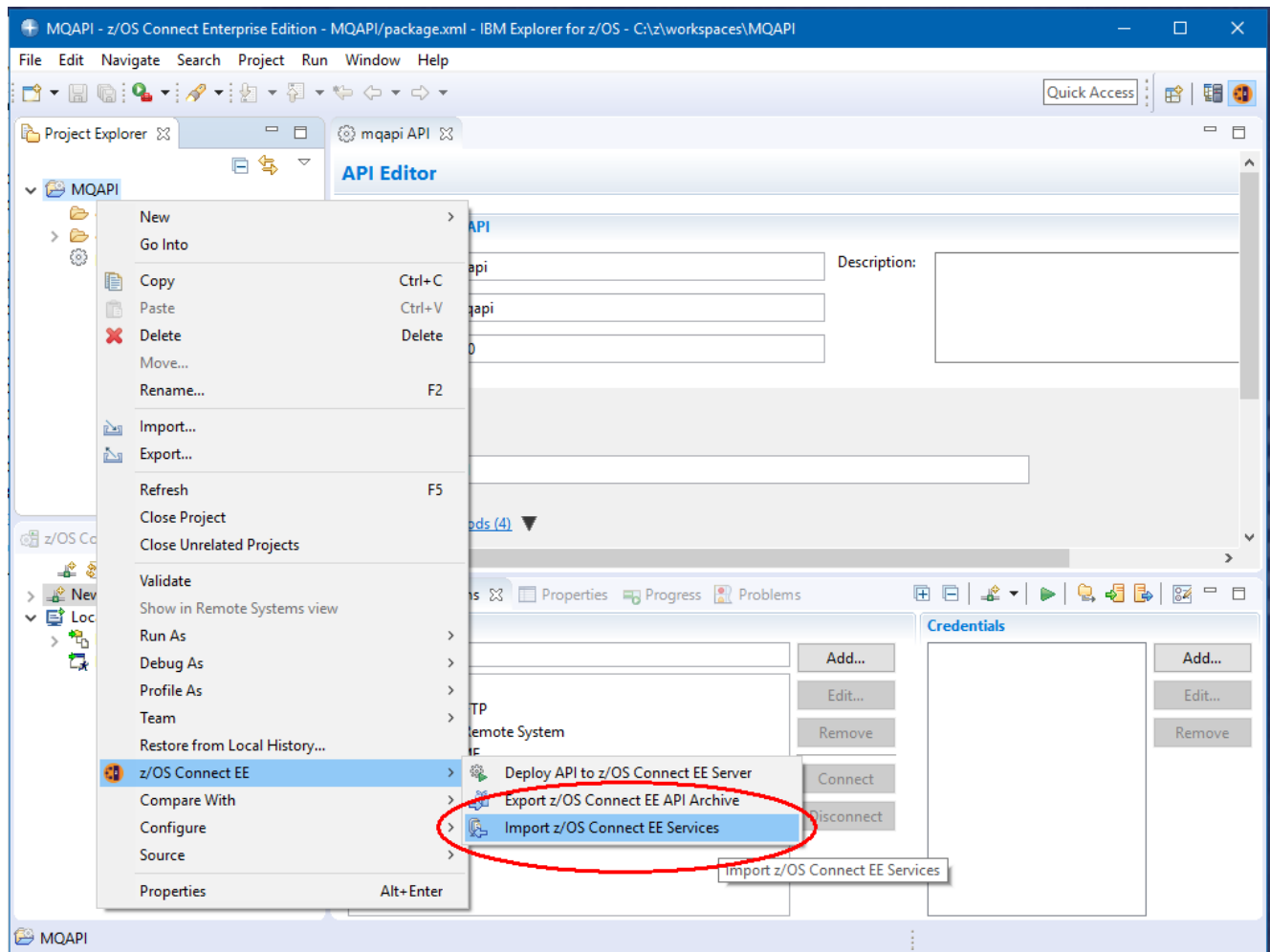


> **Tech-Tip:** If the API Editor view is closed, it can be reopened by double clicking the *package.xml* file in the API project.

___2. In the *Import z/OS Connect Services* window click on the **Workspace** button.  Expand the *Services* folder and select the three services and click the **Ok** button:
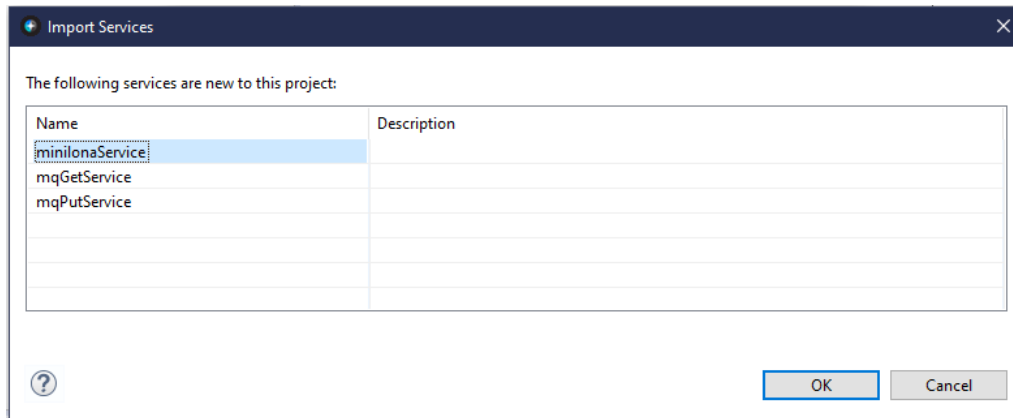


___3. The three service archive files should appear in the *Import z/OS Connect Services* screen.  Click the **OK** button to import them into the workspace.
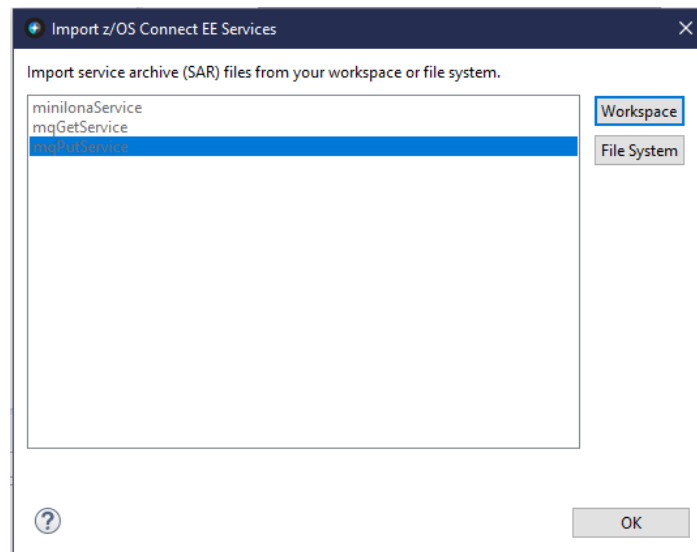


___4. In the *Project Explorer* view (upper left), expand the *services* folder to see the the imported service:
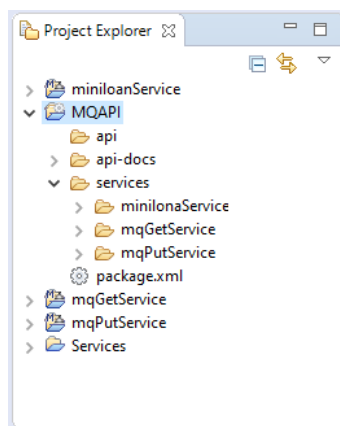
# Compose the API for two-way MQ service

___1. A *MQ for z/OS Service Provider for z/OS Connect* in a two-way service requires only one method, the **POST** method ('put' a message on a queue). The **PUT** method is not supported by the *MQ for z/OS Service provider for z/OS Connect* so it should be deleted**.** The **GET** and **DELETE** methods are also not needed for a two-way service so they can be deleted also. The view may need to be adjusted by dragging the view boundary lines.
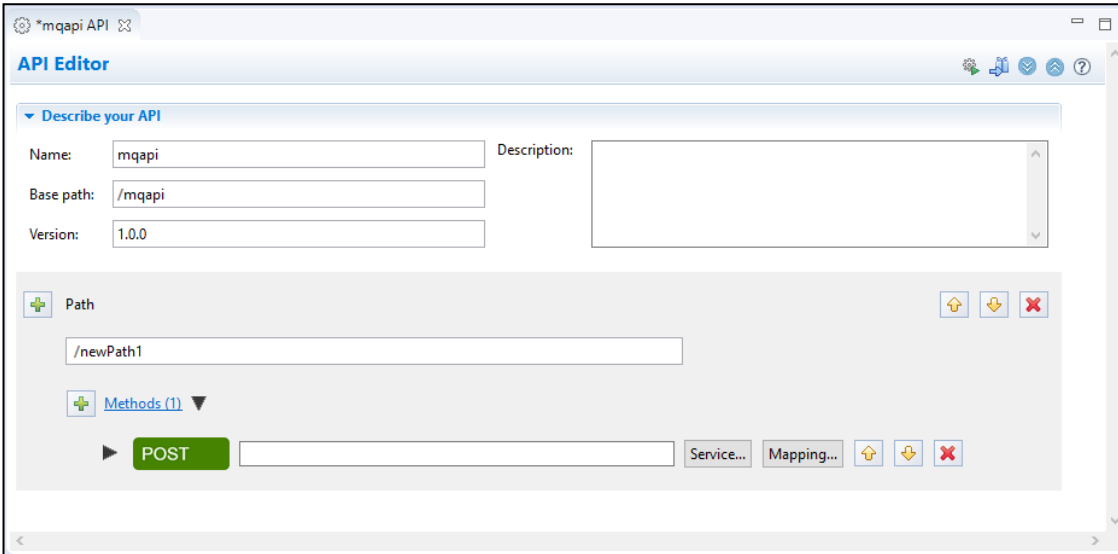


> **Tech-Tip:** The *API Editor* view above can be reopened if closed by double clicking on *package.xml*.

___2. Start by entering */loan* as the *Path* string in the *z/OS Connect API Editor* view. Click on the **Service** button to the right of the **POST** method. Then on the *Select a z/OS Connect Service* window select *miniloanService* and click **OK**. This will populate the field to the right of the method.
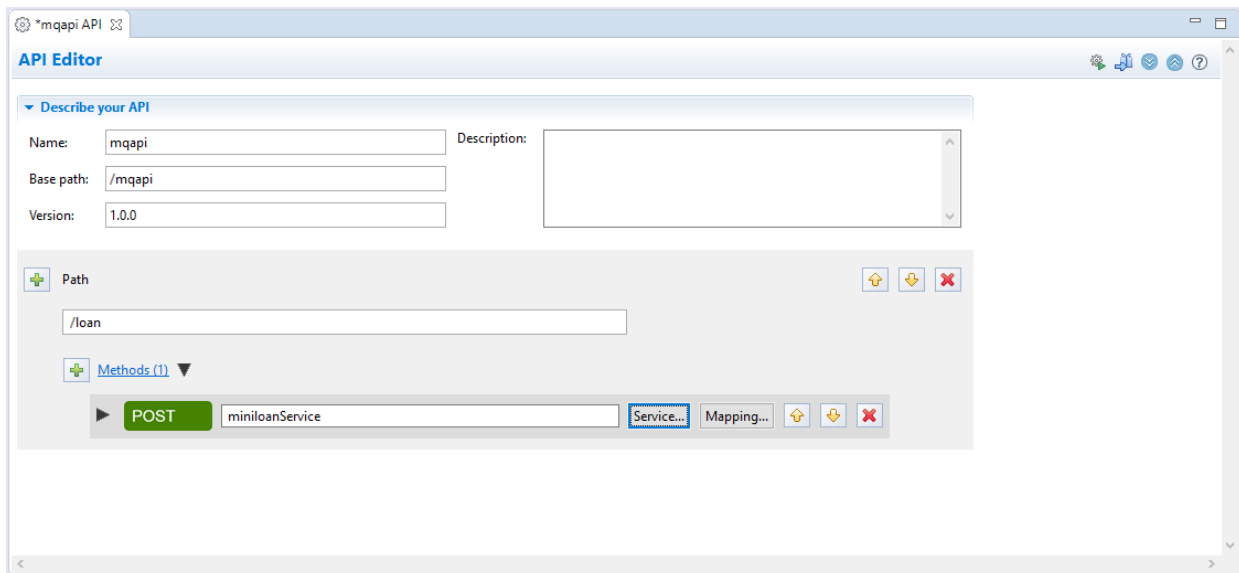
___3. Next, click on the **Mapping** button beside the **POST** method and then select *Open Request Mapping*:
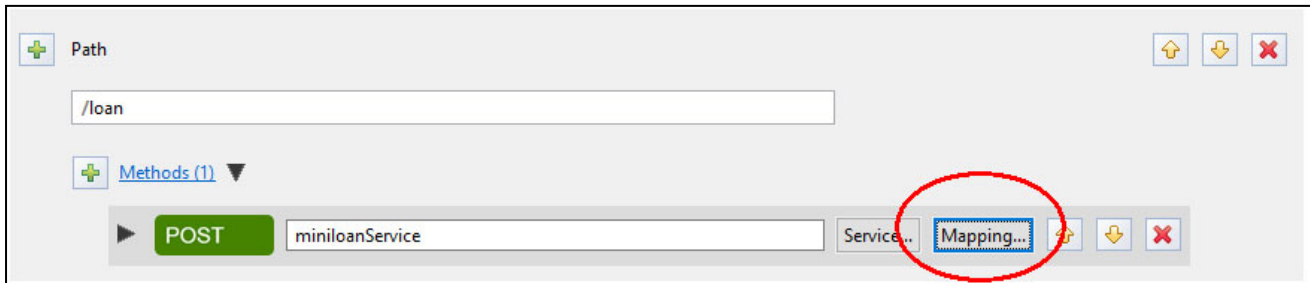


___4. In the mapping view that opens, go to the right side of the mapping (which represents the COPYBOOK fields), and if necessary, click the little + signs to expand *MINILOAN_COMMAREA*. You should see fields that correspond to the fields exposed when the service was defined (see below).
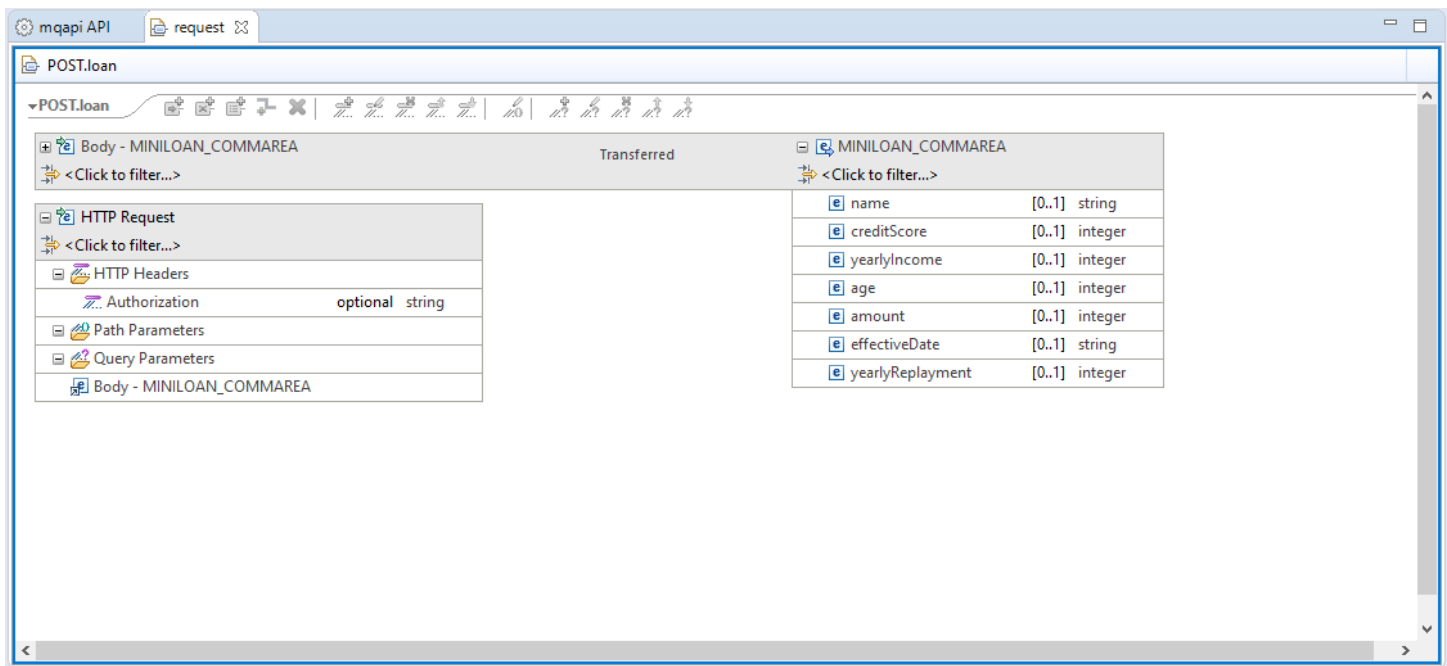
Fields can be removed from the response message (i.e. the reply from the application) by using the response mapping. At this time, all fields should be returned so no response mapping changes are required. If time allows change the response mapping to remove a field or two and test the results.
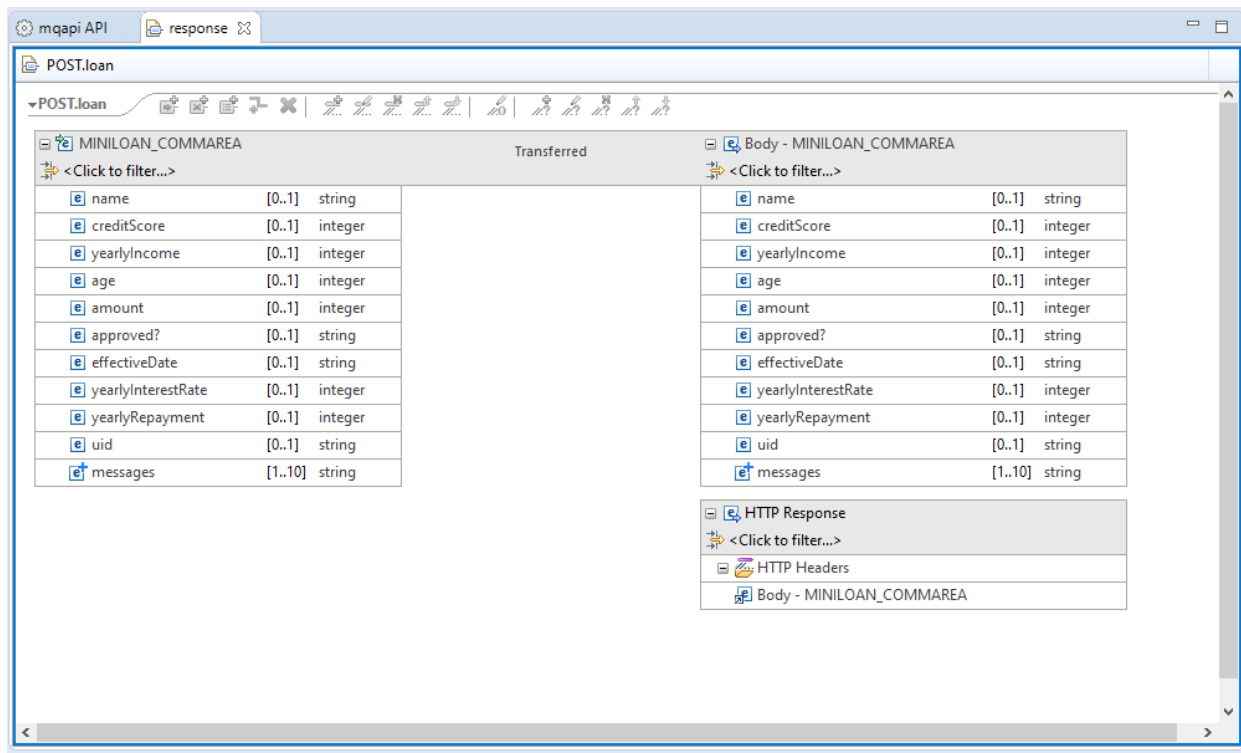


___5. In the API Editor view of *mqapi API* click on the **Mapping** button again but this time select *Define Response Codes* to add an additional HTTP response code. A HTTP response code of *200* will indicate the loan request was approved while a HTTP response code of *406* will indicate the loan request was not approved (406 was arbitrarily chosen).
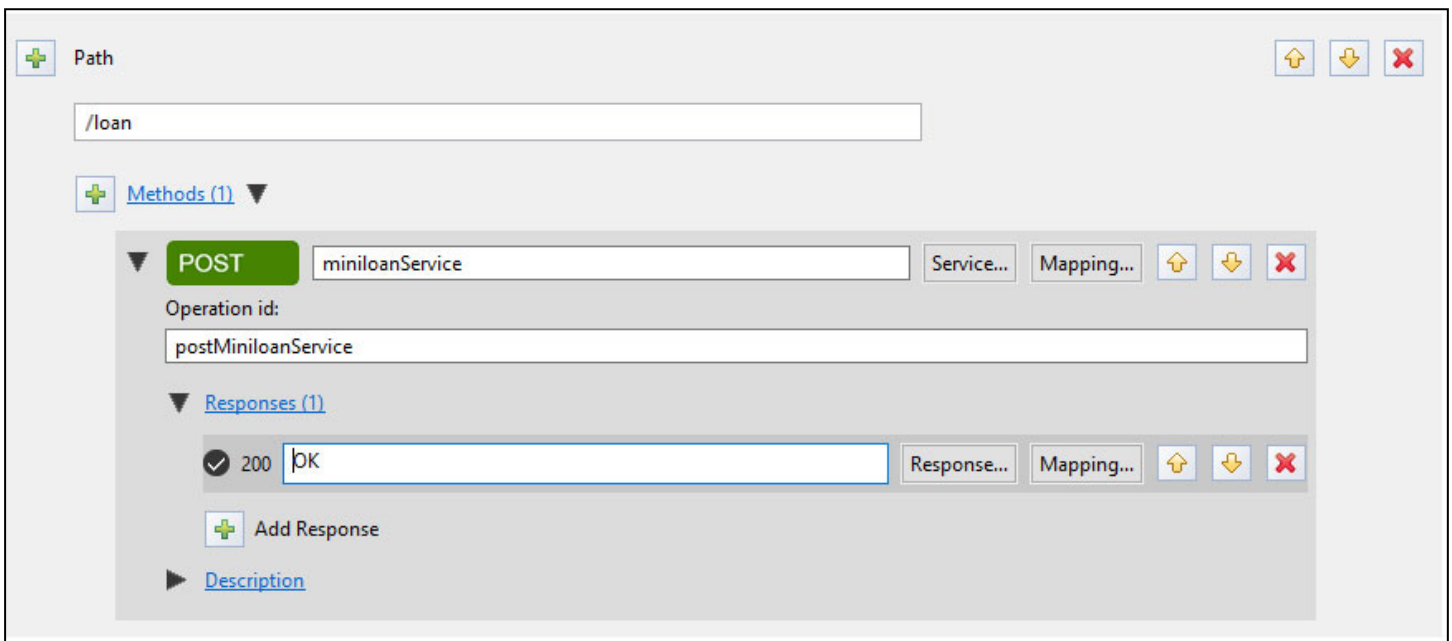
___6. Click the green plus sign beside *Add Response* to display the *Add Response* screen. Use the pull-down arrows to select a *Response code* of *406 – Not Acceptable*, select *MINILOAN_COMMAREA/approved?* for the field in the response message to be compared, select the equal sign for the comparison operand and enter *F* for the comparison value.



___7. The *API Editor* should contain something like the below:



___8. Close all open views to save all changes.

## Summary

You created the API, which just a base path and with the **POST** method along with the request and response mapping required for this method.  This API will now be deployed into z/OS Connect.

## *Deploy the two-way API to a z/OS Connect Server*

Review the z/OS Connect server.xml updates required for the *MQ for z/OS Connect Service Provider for z/OS Connect* before deploying the API.

___1. Suport for this API is added to the z/OS Connect server's *server.xml* file by the including the *mq.xml* file (see below).

```
<server description="MQ Service Provider">

 <featureManager>
      <feature>zosconnect:mqService-1.0</feature>          1
</featureManager>

 <variable name="wmqJmsClient.rar.location"
         value="/shared/mqm/V9R0M1/java/lib/jca/wmq.jmsra.rar"/>
 <wmqJmsClient nativeLibraryPath="/shared/mqm/V9R0M1/java/lib"/>

  <connectionManager id="ConMgr1" maxPoolSize="5"/>

<jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
      connectionManagerRef="ConMgr1">
      <properties.wmqJMS transportType="BINDINGS"            2
            queueManager="QMZ1" />
</jmsConnectionFactory>


<jmsQueue id="request" jndiName="jms/request">
     <properties.wmqJms
        baseQueueName="ZCONN2.TRIGGER.REQUEST"              3
        targetClient="MQ"
        CCSID="37"/>
</jmsQueue>

<jmsQueue id="response" jndiName="jms/response">
     <properties.wmqJms
        baseQueueName="ZCONN2.TRIGGER.RESPONSE"             4
        targetClient="MQ"
        CCSID="37"/>
 </jmsQueue>
</server>
```

*1.* The *featureManager* element identifies the Liberty features required by the *MQ for z/OS Service Provider for z/OS Connect.*

2. The *jmsConnectionFactory* element associates the JMS connection factory(*jndiName*) with the target queue manager and details on how to connect to this queue manager.

3. The jmsQueue elements provide details that associate the JMS destination (*jndiName*) with the request queue (*baseQueueName*) and its JMS/MQ properties.

4. The jmsQueue elements provide details that associate the JMS destination (*jndiName*) with the resonse queue (*baseQueueName*) and its JMS/MQ properties.

Mitch Johnson (mitchj@us.ibm.com)

___2. In the *Project Explorer* view (upper left), right-mouse click on the *MQAPI* folder, then select *z/OS Connect→ Deploy API to z/OS Connect Server*.
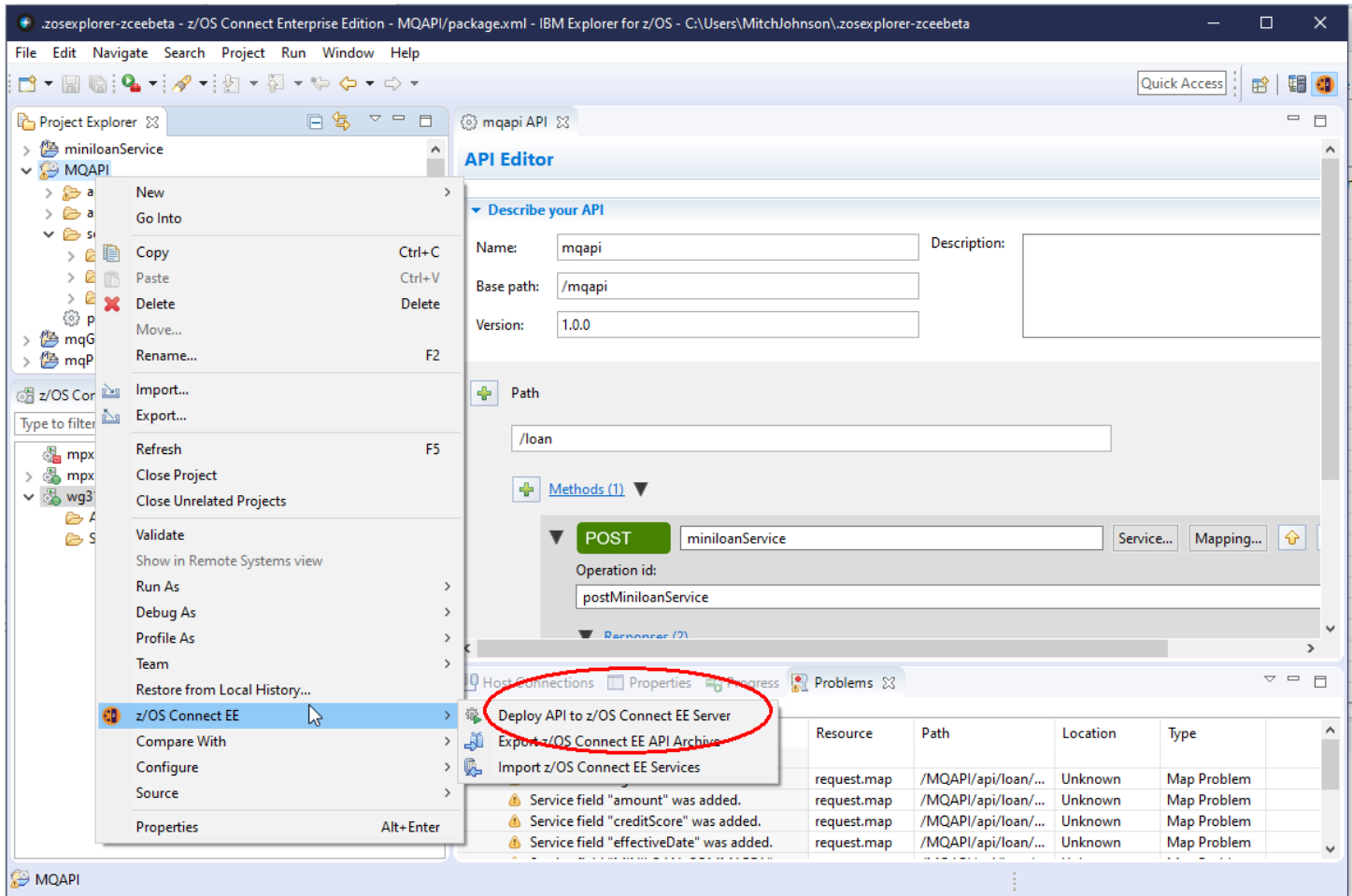
___3. Since z/OS Explorer is connected to only one z/OS Connect server there is only one choice (*wg31:9453*). If z/OS Explorer had multiple host connections to z/OS Connect servers then the pull-down arrow would allow a selection to which server to deploy. Click **OK** on this screen to continue.
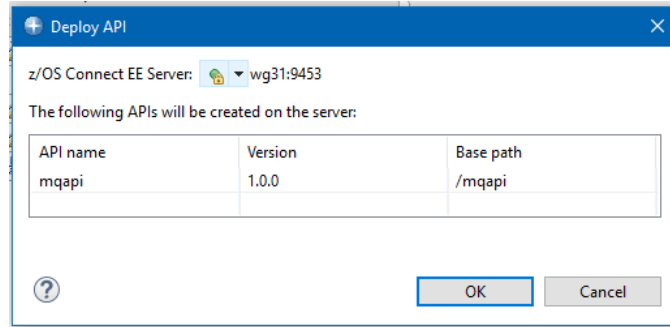


___4. The API artifacts will be transferred to z/OS and copied into the */var/ats//zosconnect/servers/zceesrv1/resources/zosconnect/apis* directory.
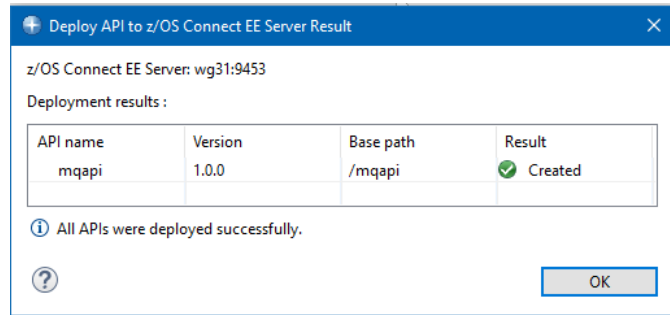


## Test the MQ Miniloan Reply/Response APIs

The application used to the test the MQ API is a CICS application that uses rules for approving loan requests. The CICS transaction that invokes the application is started when a message is written to a request queue. The arrival of a message triggers the CICS transaction which starts a program which reads the message from the request queue. The application uses the information in the message to determine if a loan can be approved. The results are returned in a message in a reply queue including the explanation if the loan is denied.

The rules for rejecting a loan can be for any one of the following:

- If the credit score of the borrower is less than 300.
- If the yearly repayment amount is more than 30% of the borrower's income.
- If the income of the borrower is less than $24,000.
- If the age of the borrower is more than 65 years.
- The loan amount is more than $1,000,000.

.

___1. Enter URL ***https://wg31.washington.ibm.com:9453/zosConnect/apis*** in the Firefox browser and you should see something like the window below. The API *miniloan* is now displayed. This is because this API was just 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 **Advanced** button to continue. Scroll down and then click on the **Accept the Risk and Continue** button. Next you may see a prompt you for a userid and
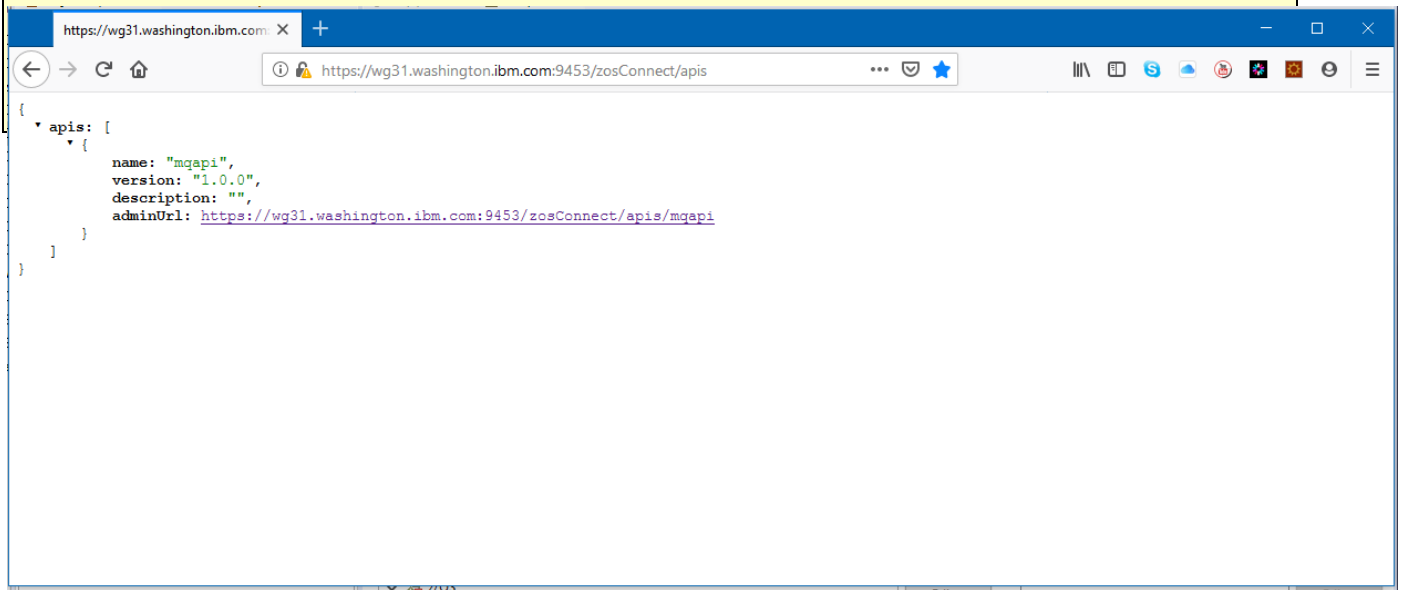


**Tech Tip:** It is very important to access the z/OS Connect server from a browser prior to any testing using the Swagger UI. Accessing a z/OS Connect URL from a browser starts an SSL handshake between the browser and the server. If this handshake has not performed prior to performing any test the test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

___2. If you click on *adminUrl* URL the window below should be displayed.
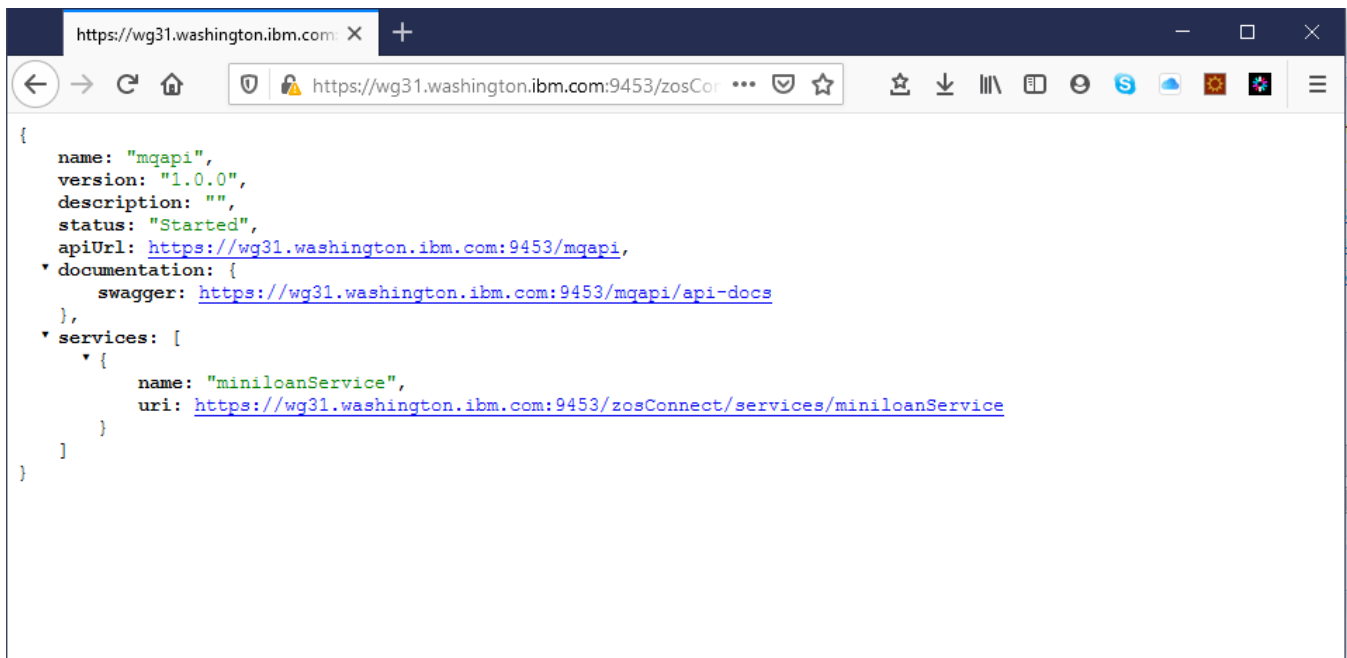


___3. Finally click on the *swagger* URL and you should see the Swagger document associated with this API.
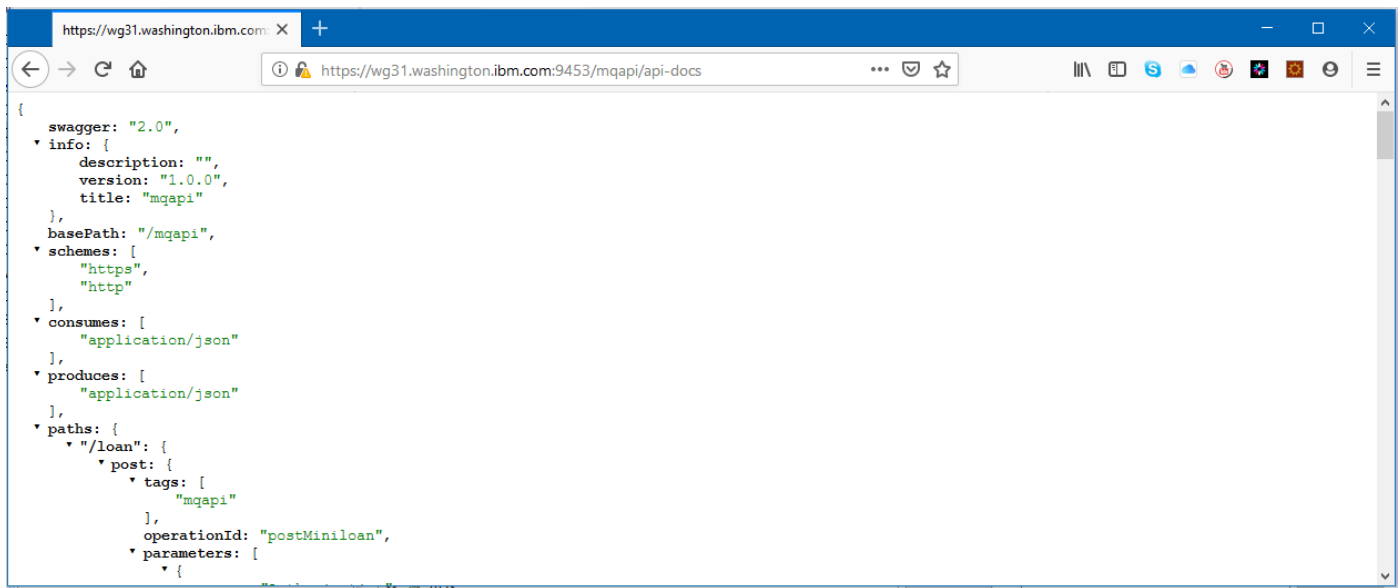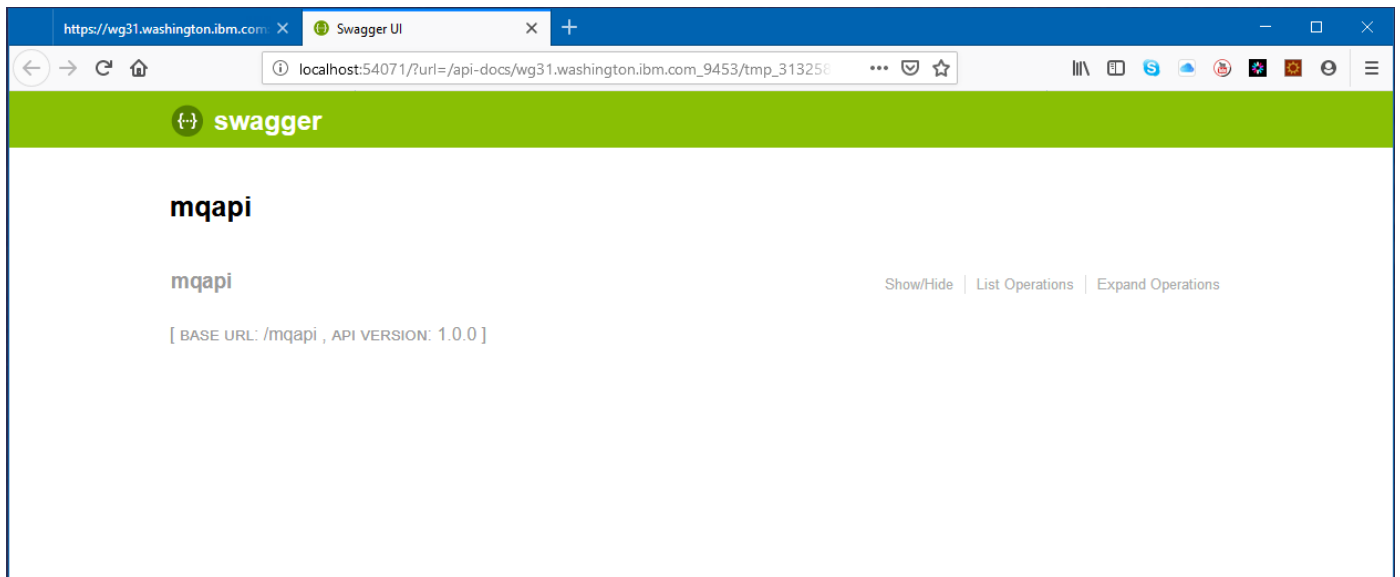


___4. Explore this Swagger document and you will see the results of the request and response mapping performed earlier.  This document can be used by a developer or other tooling to develop REST clients for this specific API.
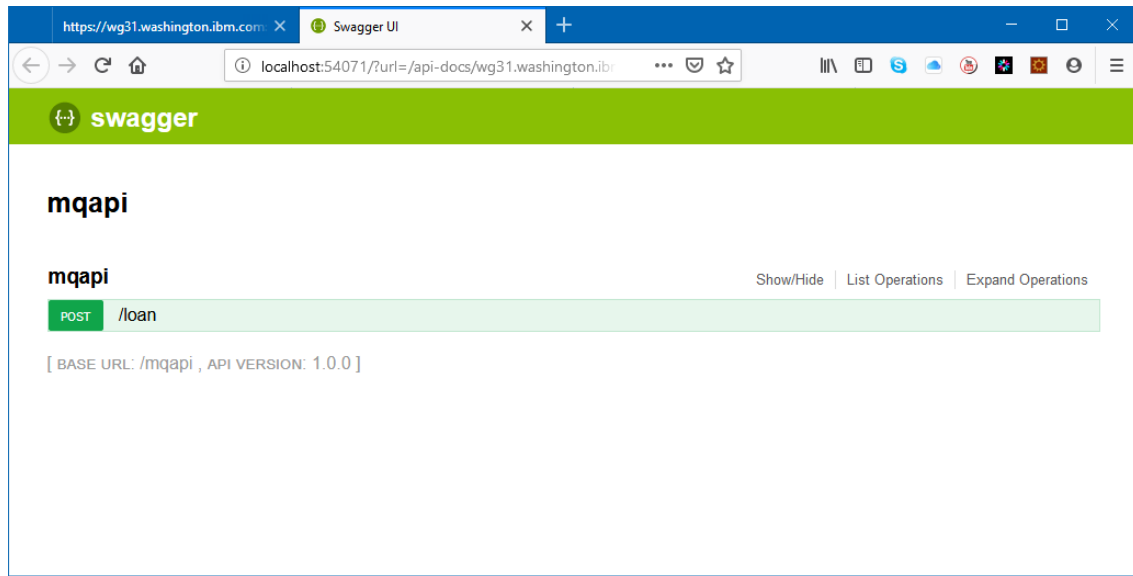
___5. In the lower left-hand side of the *z/OS Connect Explorer* perspective there is view entitled *z/OS Connect Servers*. Expand *wg31:9453* and the expand the *APIs* folder. You should see a list of the APIs installed in the server.
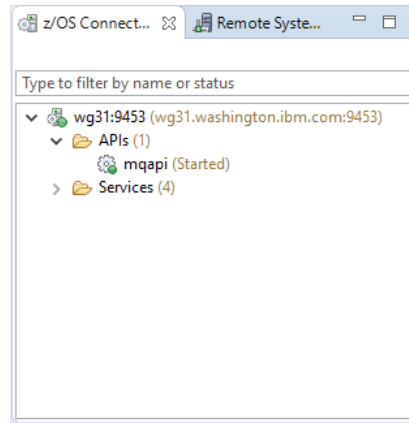
___6. Right click the mouse button on *mqapi* and select *Open in Swagger UI*. Click OK if an informational prompt appears. This will open a Firefox window showing a *Swagger* test client (see below).

___7. Click the *List Operations* and the browser should show a list of the available HTTP methods like this:



___8. Expand the *Post* method by clicking on the path beside it (e.g. */loan)* and scroll down until the method *Parameters* are displayed as shown below:

___9. Enter *John* in the area under *name*, *100* in the area under *creditScore*, *10000* in the area under *yearlyIncome*, *99* in the area under *age*, *1000000* in the area under *amount*, *12/12/21* in the area under *effectDate* and *1000* in the area under *yearlyRepayment*. Finally enter **Basic RnJlZDpmcmVkcHdk** in the area beside *Authorization*.

Mitch Johnson (mitchj@us.ibm.com)

___10. Click **Try it out!**.  You should see a *406* in the *Response Code* area and a display of the response from the loan application in the *Response Body*. This is the message from the application returned in the reply queue. The loan was not approved (*F* in the *approved* field) and an explanation in the *messages* array.



Note that only the *MESSAGE_NUM* field is not returned in the interface and null *MESSAGES* have been suppressed.

Mitch Johnson (mitchj@us.ibm.com)

___11. Change the request field as shown below and press **Try it Out!** again.



___12. This time the loan should be approved based on the value in the *approved* field with a HTTP response of 200.



___13. (Optional) Start a CICS execution diagnostic trace on transaction MINC, e.g. **CEDX MINC** and trace the flow in CICS. Note that the Swagger-UI client will eventually fail with a timed-out message:

Mitch Johnson (mitchj@us.ibm.com)

```
{

  "errorMessage": "BAQR0405I: The asynchronous request under URL
    https://wg31.washington.ibm.com:9453/miniloan/loan has timed out after 30,000 milliseconds."

}
```

Mitch Johnson (mitchj@us.ibm.com)

## *Compose the API project for a One-Way Service*

In this section the MQAPI API will be enhanced by add support for MQ one-way services. MQ one-way service only supports the **POST** and **GET** methods. The POST method puts JSON request message on a queue after converting the JSON request message to a non-JSON format. The **GET** method is used get messages from a queue and convert the contents to a JSON response message.

___1. The *MQ for z/OS Service Provider for z/OS Connect* in a one-way service which supports the **POST** ('put' a message on a queue) and **GET** (a 'get' of a message from a queue)**.** The view may need to be adjusted by dragging the view boundary lines. Start by click in plus sign beside the *Path* for the */loan* request.



.
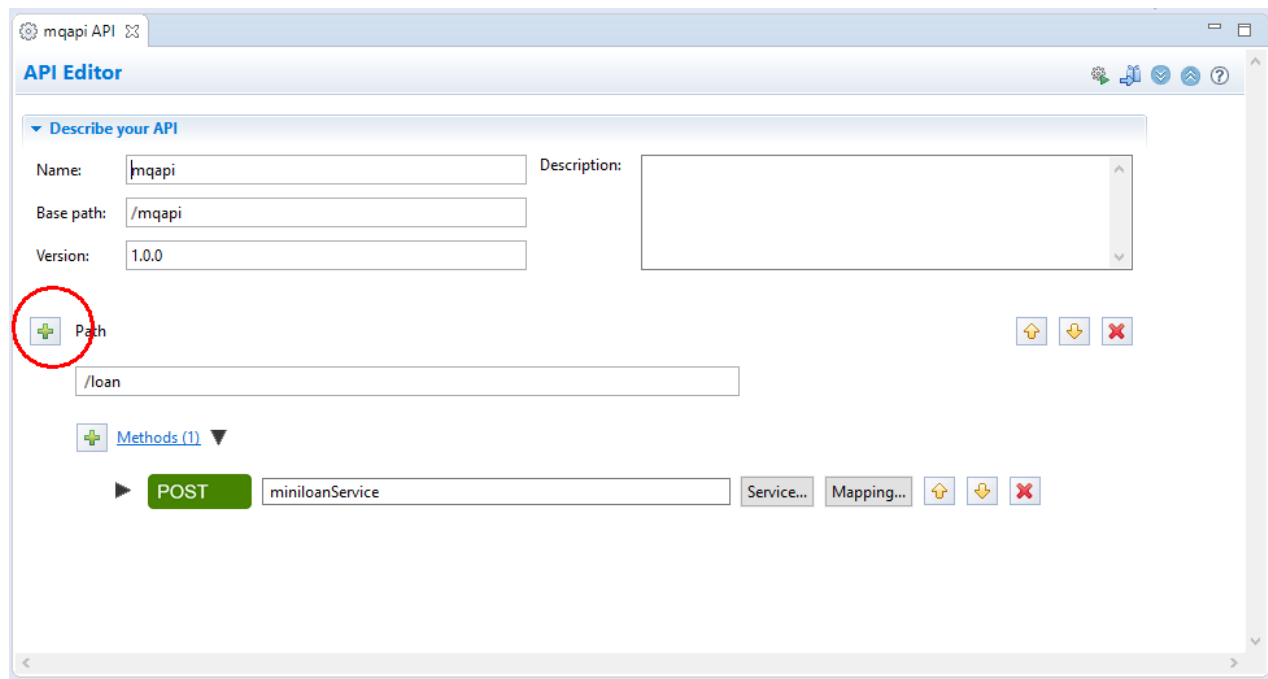
___2. Start by entering */queue* as the *Path* string in the *z/OS Connect API Editor* view and delete the PUT and DELETE methods. Click on the **Service** button to the right of the **POST** method. Then on the *Select a z/OS Connect Service* window select *mqPutService*. Click **OK** to continue. This will populate the field to the right of the method.

___3. Click on the **Service** button to the right of the **GET** method. Then on the *Select a z/OS Connect Service* window select *mqGetService*. Click **OK** to continue. This will populate the field to the right of the method.

___4. When completed the API editor should look something like this.

___5. Next, click on the **Mapping** button beside the **POST** method and then select *Open Request Mapping*:

___6. In the mapping view that opens, go to the right side of the mapping (which represents the COPYBOOK fields), and if necessary click the little + signs to expand *MQMESSAGE*. You should see fields that correspond to the fields defined in the original COBOL copy book *FILEAMQ.cpy* (see below).
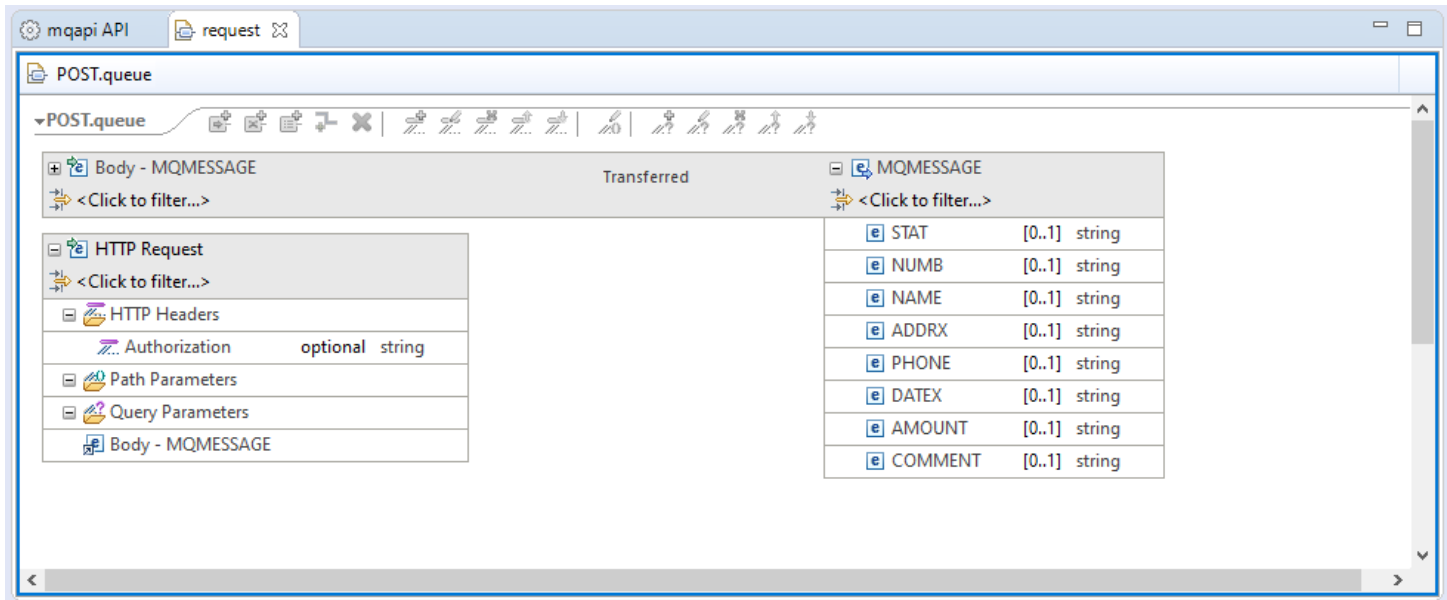


```
        01 MQMESSAGE.
            10   stat        PIC X(1).
            10   numb        PIC X(6).
            10   name        PIC X(20).
            10   addrx       PIC X(20).
            10   phone       PIC X(8).
            10   datex       PIC X(8).
            10   amount      PIC X(8).
            10   comment     PIC X(9).
```

This is where values can be assigned to fields and/or fields can be omitted from the interface.

___7. Use the **Ctrl-S** key sequence to close this view.

___8. Click on the **Mapping** button beside the **POST** method and then select *Open Default Response Mapping*. Note that the response body has no content. A POST (MQPUT) does not a response JSON message.
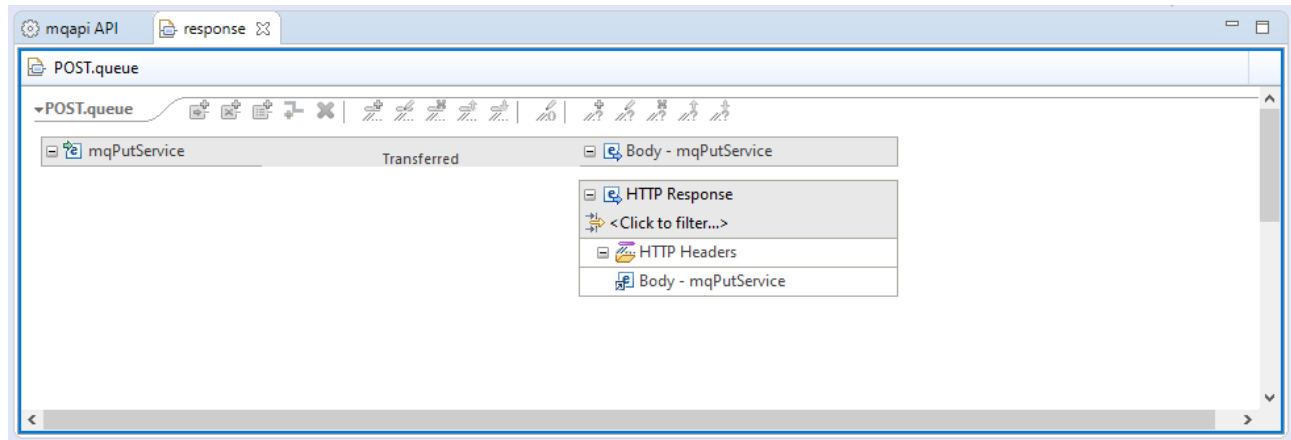


___9. Next, click on the **Mapping** button beside the **GET** method and then select *Open Request Mapping*. Note that there are no fields displayed in the request message.
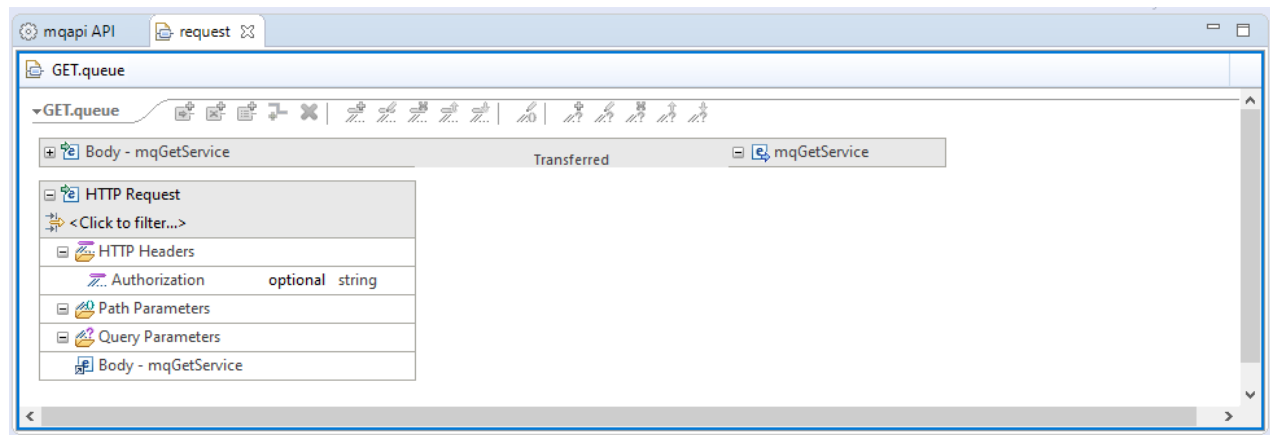


This is because a GET request not use require a JSON request message, only a JSON response message is returned.

___10. Next, click on the **Mapping** button beside the **GET** method and then select *Open Response Mapping*. Note that there are no
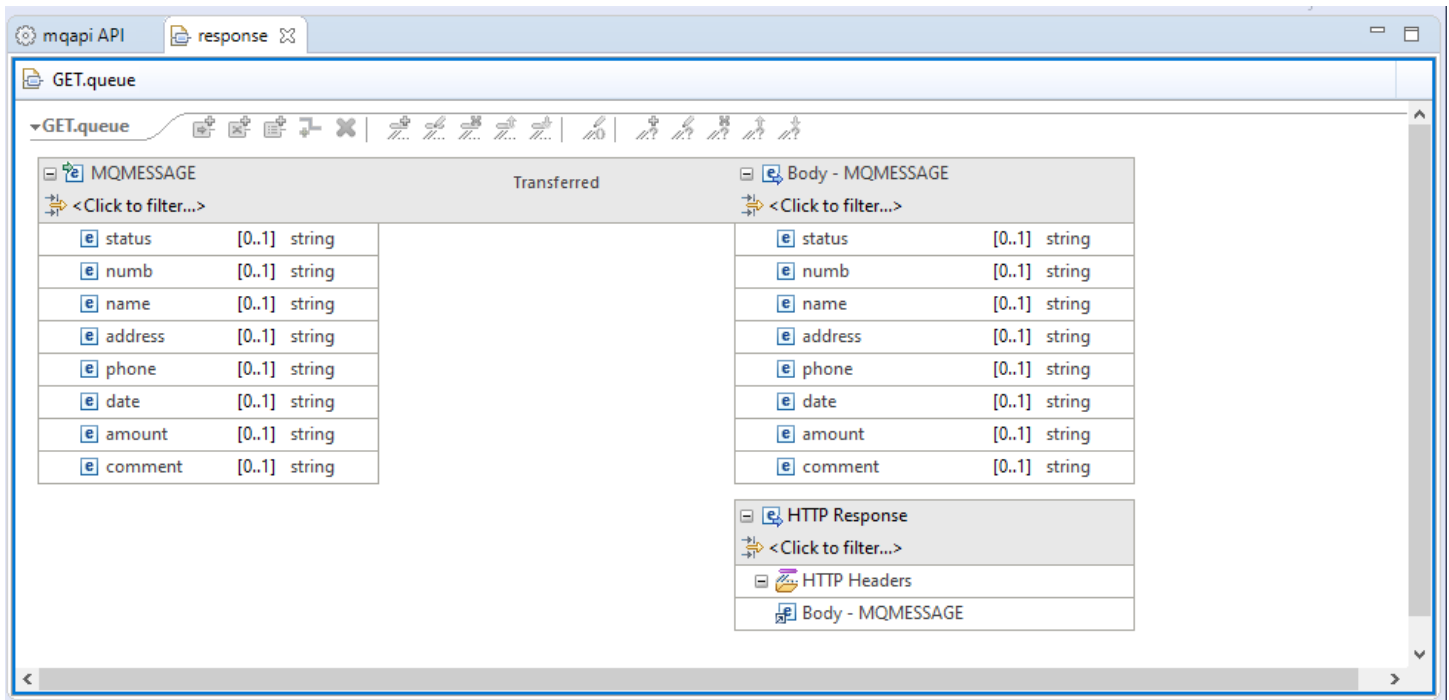


## Summary

You created the API, which just a base path and with the **POST** and **GET** HTTP methods along with the request and response mapping required for each method.  This API will now be deployed into z/OS Connect.

## *Deploy the API to a z/OS Connect Server*

Review the z/OS Connect server.xml updates required for the *MQ for z/OS Connect Service Provider for z/OS Connect* before deploying the API.

___1. Suport for this API is added to the z/OS Connect server's *server.xml* file by the including the *mq.xml* file (see below).

```
<server description="MQ Service Provider">

 <featureManager>
      <feature>zosconnect:mqService-1.0</feature>              (1)
</featureManager>

 <variable name="wmqJmsClient.rar.location"
          value="/shared/mqm/V9R0M1/java/lib/jca/wmq.jmsra.rar"/>
 <wmqJmsClient nativeLibraryPath="/shared/mqm/V9R0M1/java/lib"/>

  <connectionManager id="ConMgr1" maxPoolSize="5"/>

<jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
       connectionManagerRef="ConMgr1">
       <properties.wmqJMS transportType="BINDINGS"           (2)
              queueManager="QMZ1" />
</jmsConnectionFactory>

<jmsQueue id="q1" jndiName="jms/default">
     <properties.wmqJms                                       (3)
         baseQueueName="ZCONN2.DEFAULT.MQZCEE.QUEUE"
         CCSID="37"/>
</jmsQueue>


</server>
```

1. The *featureManager* element identifies the Liberty features required by the *MQ for z/OS Service Provider for z/OS Connect.*

2. The *jmsConnectionFactory* element associates the JMS connection factory(*jndiName*) with the target queue manager and details on how to connect to this queue manager.

3. The jmsQueue elements provide details that associate the JMS destination (*jndiName*) with the default queue (*baseQueueName*) and its JMS/MQ properties.

> **Tech Tip:** You can override MQ provider properties in the service archive file by using an *zosconnect_services* element like the one below for service *mqPut*.
>
> ```
> <zosconnect_services>
>         <service name="mqPut">
>           <property name="destination" value="jms/default"/>
>           <property name="useCallerPrincipal" value="false"/>
>     </service>
> </zosconnect_services>
> ```

___2. In the *Project Explorer* view (upper left), right-mouse click on the *MQAPI* folder, then select *z/OS Connect→ Deploy API to z/OS Connect Server*.
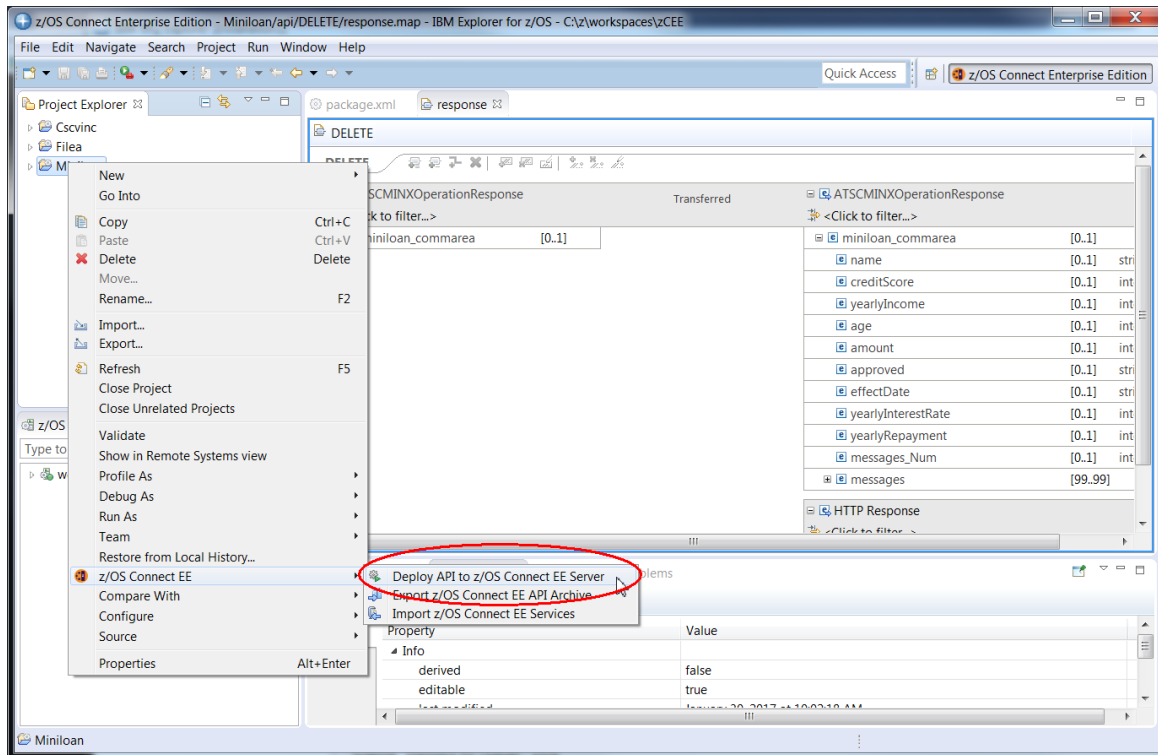


___3. Since z/OS Explorer is connected to only one z/OS Connect server there is only one choice (*wg31:9453*). If z/OS Explorer had multiple host connections to z/OS Connect servers then the pull-down arrow would allow a selection to which server to deploy. Check the box beside *Update existing APIs* and click **OK** on this screen to continue.
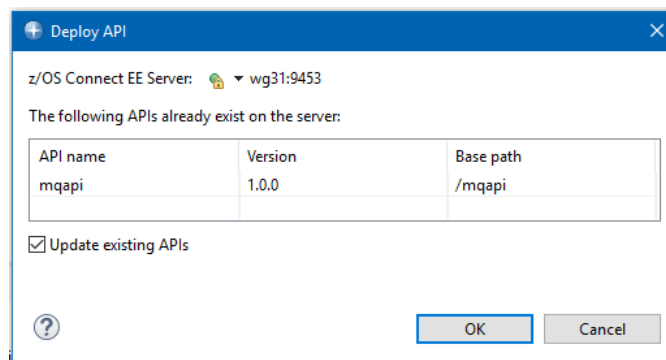
___4.   The API artifacts will be transferred to z/OS and copied into the
        */var/ats//zosconnect/servers/zceesrv1/resources/zosconnect/apis* directory.
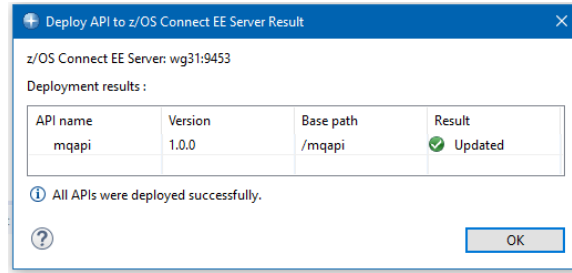


## Test the MQ One Way Service API

A MQ "one way" service provides a REST interface for putting to and getting messages from a queue (or topic). The supported REST methods are; **POST** ('put' a message), **GET** ('get' a message).

___1. Enter URL ***https://wg31.washington.ibm.com:9453/zosConnect/apis/mqapi*** in the Firefox browser and you should see the window below.  The new services should not be displayed.

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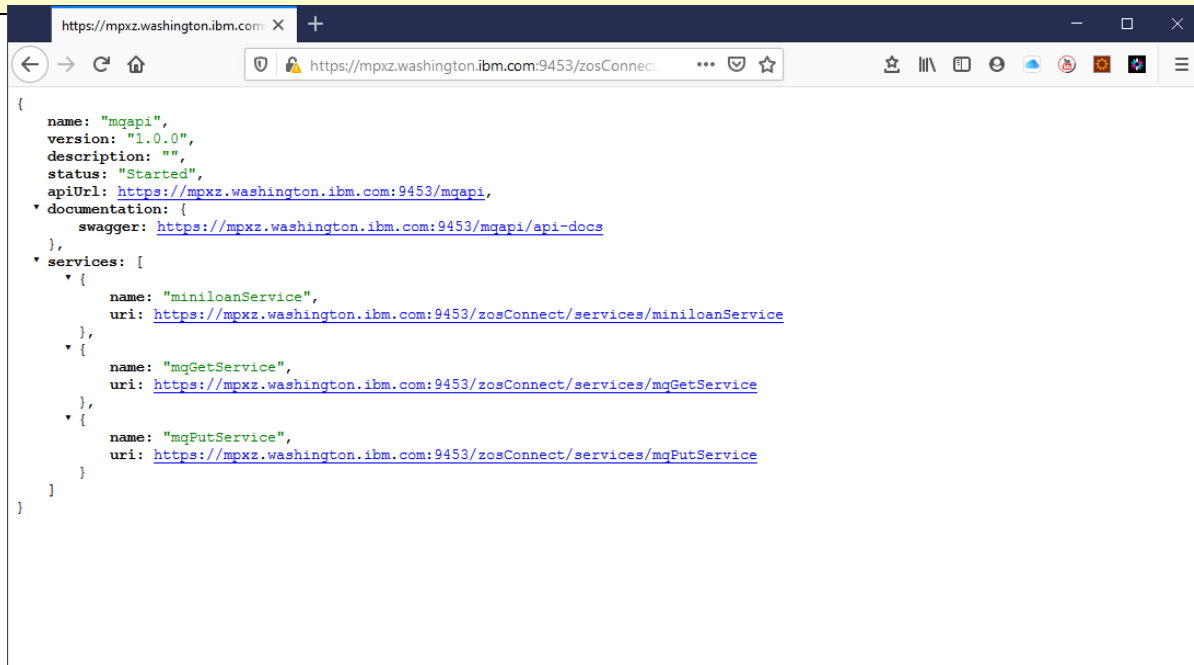
> **Tech Tip:** It is very important to access the z/OS Connect server from a browser prior to any testing using the Swagger UI. Accessing a z/OS Connect URL from a browser starts an SSL handshake between the browser and the server. If this handshake has not performed prior to performing any test the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

___2. Finally click on the *swagger* URL and you should see the Swagger document associated with this API.

___3. Explore this Swagger document and you will see the results of the request and response mapping performed earlier. This 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 Servers*. Expand *wg31:9453* and the expand the *APIs* folder. You should see a list of the APIs installed in the server.

___5. Right click the mouse button on *mqapi* 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).
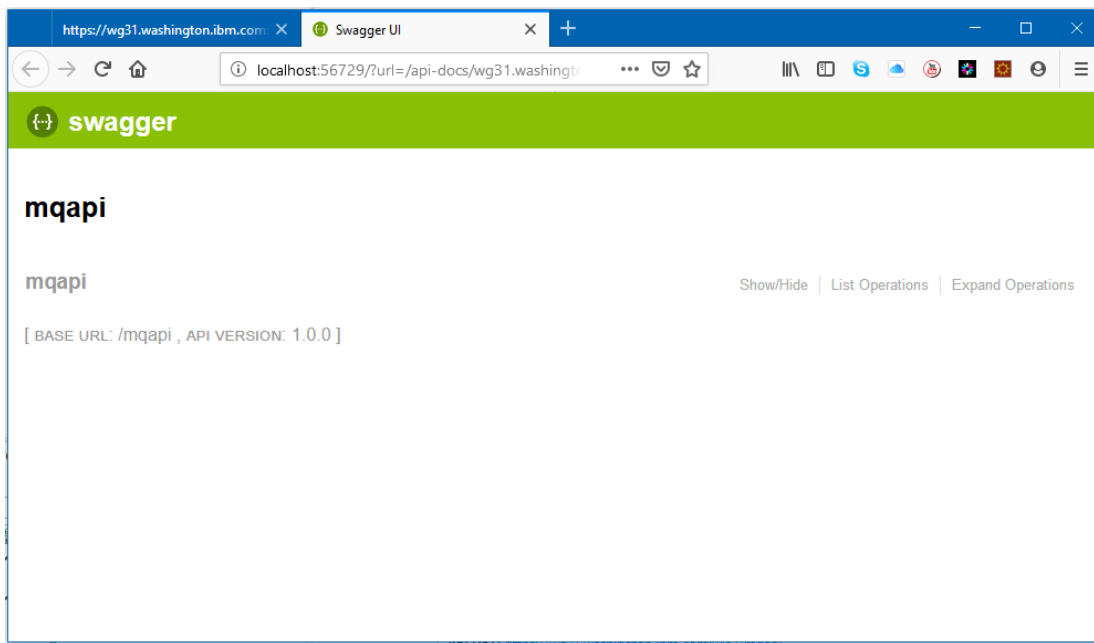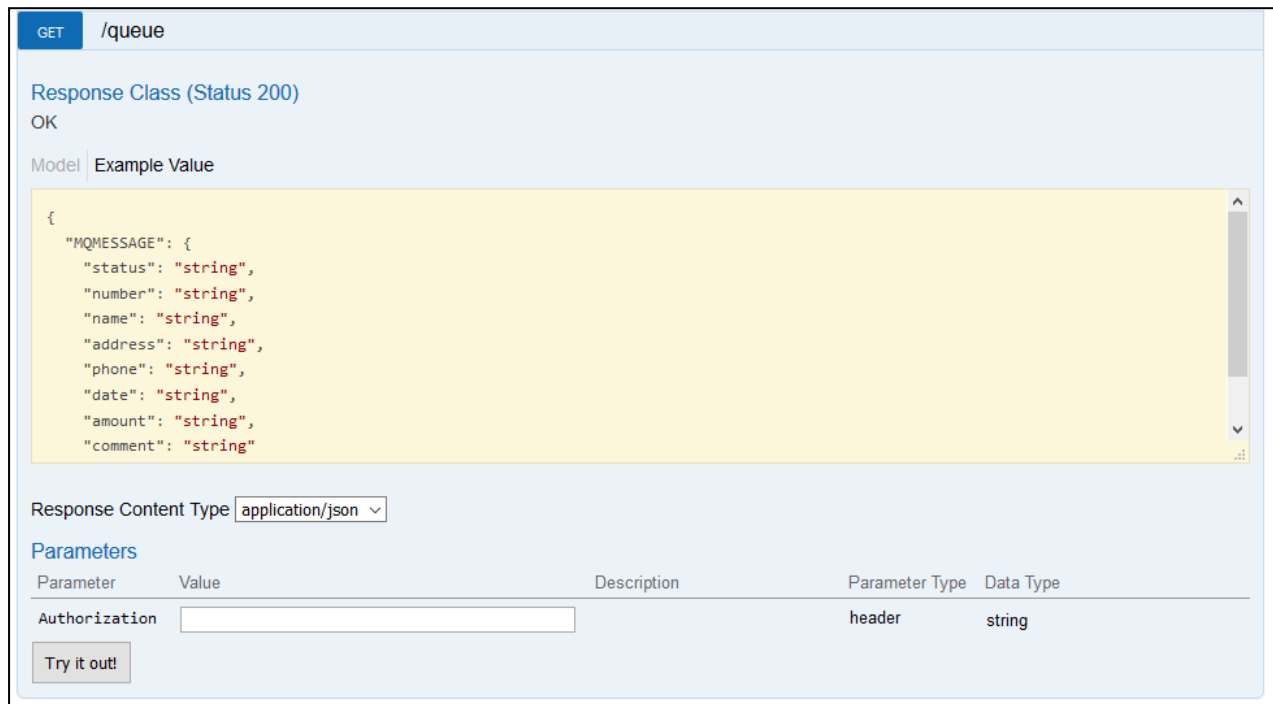


___6. Click the *List Operations* and the browser should show a list of the available HTTP methods for this API.
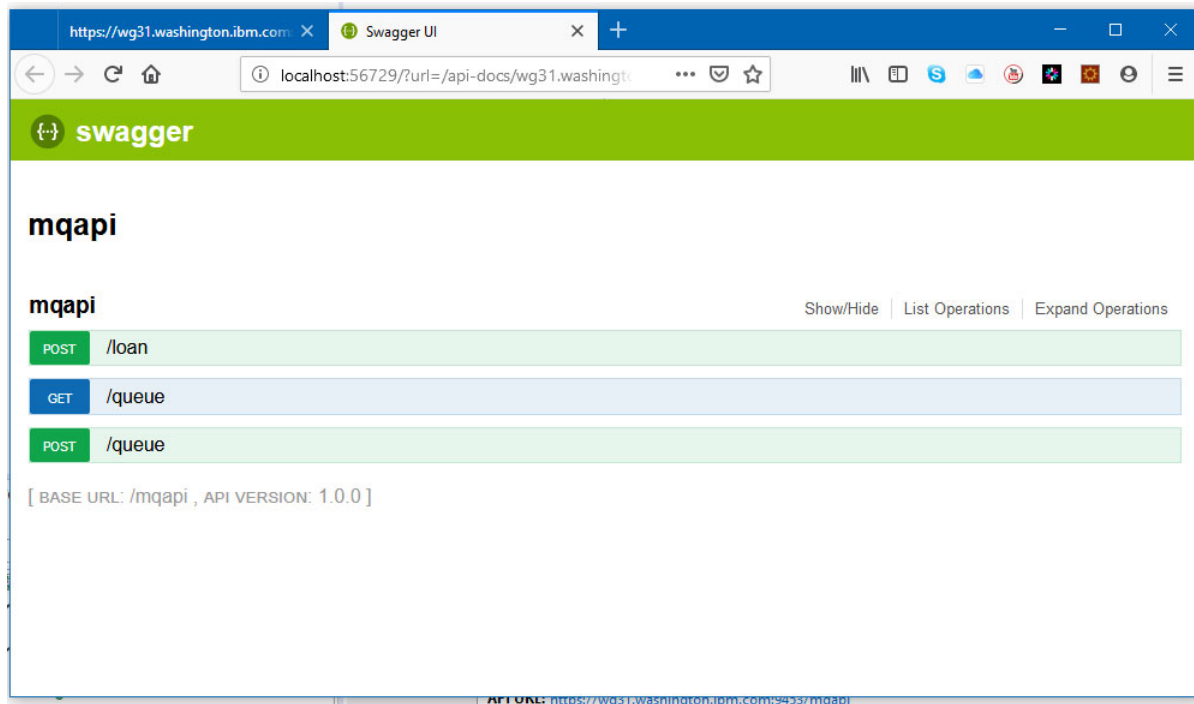
___7. Expand the **GET** method by clicking on the path beside it (e.g. */queue)* and scroll down until the method *Parameters* are displayed as shown below:

Mitch Johnson (mitchj@us.ibm.com)

___8. Enter **_Basic RnJlZDpmcmVkcHdk_** in the area beside *Authorization* and click **Try it out!**. Scroll down to display the response in the *Response Body*. This is the message from the application returned in the reply queue. This was the message retrieved by a destructive get. If you continue to click the **Try it out!** button you will see the next message.

```
Response Body

  {
    "MQMESSAGE": {
      "date": "10 04 73",
      "number": "001222",
      "amount": "$3349.99",
      "address": "BOBLINGEN, GERMANY",
      "phone": "70315551",
      "name": "D.J.VOWLES",
      "comment": "Y",
      "status": ""
    }
  }

Response Code

  200
```

___9. Expand the **POST** method by clicking on the path beside it (e.g. */queue)* and scroll down until the method *Parameters* are displayed as shown below. Enter values for each field (see examples below) and press the **Try it Out!** button.

Mitch Johnson (mitchj@us.ibm.com)

| POST | /queue |
|------|--------|

## Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|-----------|-------|-------------|----------------|-----------|
| Authorization | | | header | string |
| **postMqPutService_request** | [-] | **request body** | body | Model \| Example Value |

MQMESSAGE [-]

staus
```
Y
```
number
```
948478
```
name
```
Don Bagwell
```
address
```
Raleigh NC
```
phone
```
0065
```
date
```
26 11 81
```
amount
```
$100.0
```
comment
```
******
```

Example Value:
```
{
  "MQMESSAGE": {
    "staus": "string",
    "number": "string",
    "name": "string",
    "address": "string",
    "phone": "string",
    "date": "string",
    "amount": "string",
    "comment": "string"
  }
}
```

Parameter content type: application/json

## Response Messages

| HTTP Status Code | Reason | Response Model | Headers |
|------------------|--------|----------------|---------|
| 200 | OK | | |

[Try it out!]

___10. Scroll down to display the response in the *Response Body*. Note that there is no response message. The MQ Service provider does not return a reply message for the *POST* method. If you continue to click the **Try it out!** button you will see the same results over and over.

| Try it out! | Hide Response |

**Curl**

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'Authorization: Basic RnJlZDpmcmcm'
```

**Request URL**

```
https://wg31.washington.ibm.com:9453/fileaqueue/mq
```

**Request Headers**

```
{
  "Accept": "application/json",
  "Authorization": "Basic RnJlZDpmcmcmVkcHdk"
}
```

**Response Body**

```
no content
```

**Response Code**

```
204
```

**Response Headers**

```
{
  "content-language": "en-US",
  "expires": "Thu, 01 Dec 1994 16:00:00 GMT",
  "cache-control": "no-cache=\"set-cookie, set-cookie2\"",
  "content-type": null
}
```

**Tech-Tip:** An HTTP code of 204 the server processed the request but returned no content. For an explanation of HTTP codes see URL
https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

___11. Confirm by using the *MQExplorer* tool on the desktop and browse the messages in the *ZCONN2.DEFAULT.MQZCEE.QUEUE* queue.

- Select *QMZ1* under **Queue Managers** and right click the mouse button

- Select the *Connect* option.

- Once connected, expand the *Queues* folder, and select *ZCONN2.DEFAULT.MQZEE.QUEUE* and right click the mouse button.

**Tech Tip:** The message is first in the list because it has a higher priority than the other messages.

___12. Do a few more **GET** and **POST** requests and observe the changes in the queue.

## Summary

You have added the *MQ for z/OS Service Provider  for z/OS Connect* to the z/OS Connect server and configured two services.  One service (Miniloan) supports a reply/response application using two queues and the other service (Queue)) is a one-way service where the same queue is used for POSTs and GETs.