akana
by Perforce

SOLA REST API Designer User Guide Release 6.4.2

Revision Date: August 2017

Contents

About Rest API Designer

Overview

SOLA now provides an easy means by which you can prepare your SOLA SOAP based services to be accessed via REST. The new REST API Designer feature allows you to specify REST specific information such as:

- Version – Multiple versions can be active at the same time.
- Resource Name
- URI parameters
- Query String Parameters
- HTML Method – Currently supported are: GET, PUT, POST, DELETE

Once you have completed the design of your REST API the required metadata to support it at runtime will be placed into the SOLA Template alongside the traditional SOAP metadata. This means, as before, that your new REST APIs can be migrated through your various development stages in the same manner that your SOAP services are currently migrated.

In addition to passing input parameters on the URI or Query String you may also POST them as either a JSON or XML payload (providing you specify POST as your HTML method). At runtime, you can receive your results back as either JSON or XML depending on the value set in the 'ACCEPT' HTML Header.

Currently, the following SOLA plugin types are supported for REST access:

- COMMAREA
- Container
- IMS
- DB2 Stored Procedures

Previously DB2 Stored Procedures did not require the use of a metadata template and none was created. Now, during REST design, a metadata template will be created for DB2 Stored Procedure types.

HTTP Headers Used

There are a variety of HTTP Headers that can impact the behavior of the API:

Header Name	***Header Value***	***Example***
Authorization	Basic Auth format	BASIC VXNlck5hbWU6UGFzc3dvcmQ=
Accept	Response Format	application/json or application/xml
Content-Type	Payload Format	application/json or application/xml
REST-API-KEY	API Specific Key (generated)	RS-BARIM#D1-0866
REST-ACTION	Declare Host/Client Code Page	/CCP:*<ClientCodePage>*/HCP:*<HostCodePage>* /CCP:UTF-8/HCP:1140

Of these, the only one that is required in the REST-API-KEY. All others are optional unless there is a security policy placed on the API which will be covered in the next section.

Note that, if the 'Accept' header is not specified, content type will be JSON.

Security Handling

Regardless of the presence of a security policy, if an Authorization header is included in the HTTP Headers the username and password contained will be verified. For LDAP credentials the verification will be performed against the LDAP server registered through SOLA. Mainframe credentials are verified directly.

In the case that either the program or method has a User Name and Password policy attached then the policy will be enforced and needs valid credentials passed via an “Authorization” HTTP header for the request to be processed.

If the program or method has a policy of Encryption (either in or out), SAML, or x509 certificate attached then, in addition to enforcing basic authorization, SOLA also requires the request to be received over an SSL/TLS Channel. Otherwise is will be rejected.

Working with the Rest API Designer

The example that follows will demonstrate how to use the REST API Designer to quickly enable an existing SOLA SOAP base API for REST access.

Note: In order to REST enable a legacy artifact you first must go through the standard SOLA Import and Analysis steps outlined in the SOLA Developer User guide. For the purposes of this example, we will start with an existing SOAP based API already contained in the SOLA registry.

The REST API Designer uses the HTTP protocol in an easy and effective way.

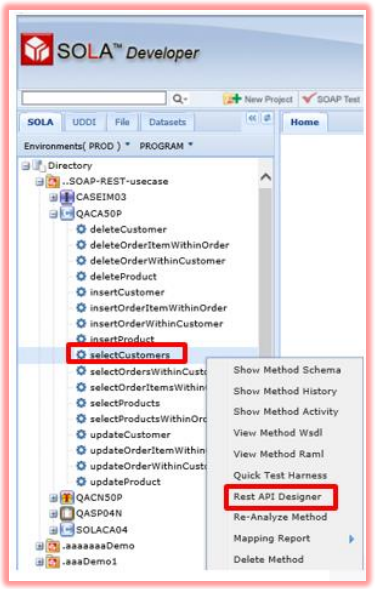
Application state and functionality are organized into resources. Resources can represent physical things, such as a specific Storage Processor (SP); logical things, such as a specific collection of customers and their orders. Each resource has a unique Universal Resource Identifier (URI), and each resource instance has a unique ID. For example, you can identify the Customer collection with this URI: /

To use request parameters, append the parameters to the request URI. The first request parameter appended to the URI begins with a '?' character. Additional request parameters begin with '&' instead of '?'. You can combine request parameters and can use them in any order. If a request parameter is repeated, all but the last one is ignored.

Example #1:

Let's begin by selecting the operation/method that will be used to create the REST service.

Right Click on the 'selectCustomers' method and again on the Rest API Designer option.

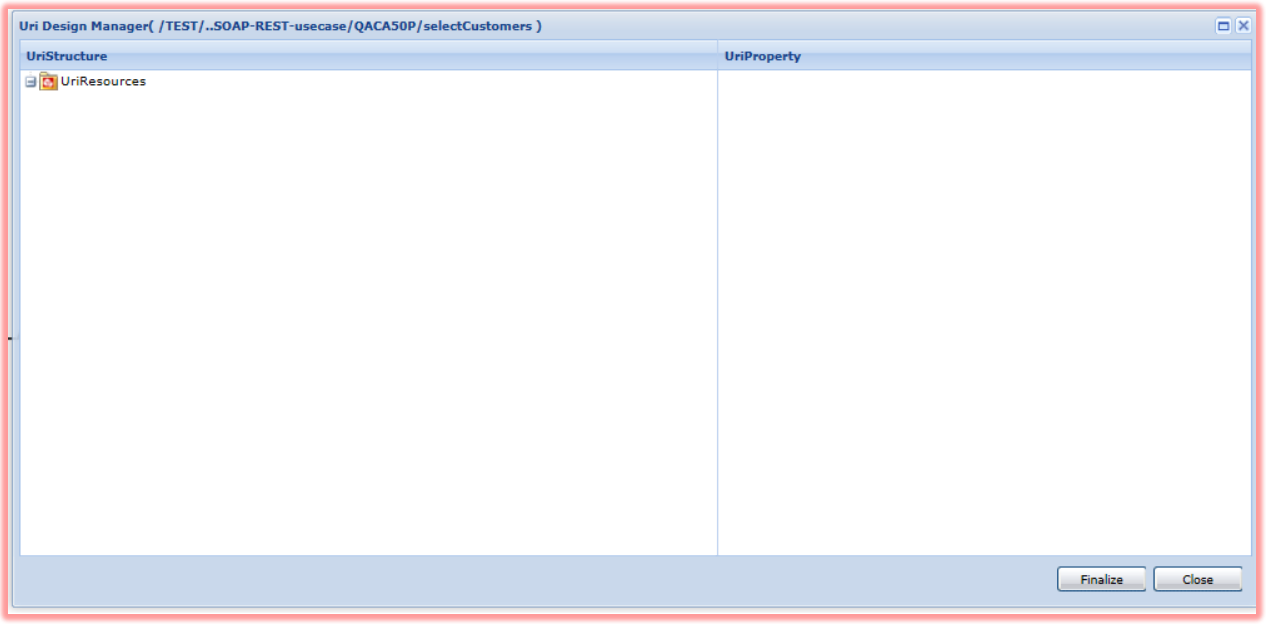


The URI Design Manager pop-up pane is displayed.

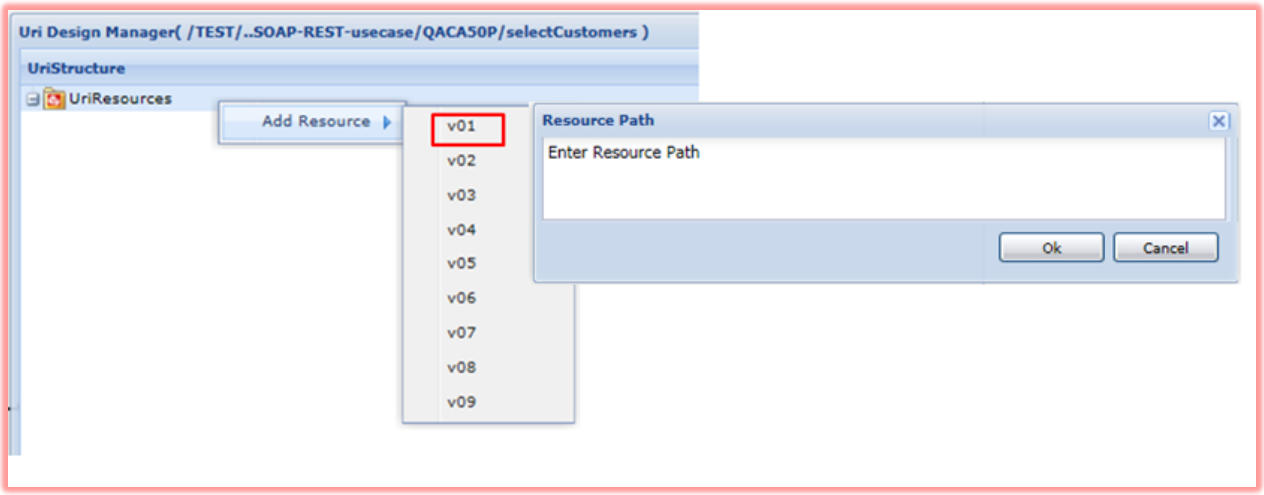
Let's review this.

The UriDesign Manager bar begins with defining your selected operation/service which in this case was chosen from the TEST environment in project …SOAP-REST-usecase. The program is QACA50P and operation is selectCustomers.

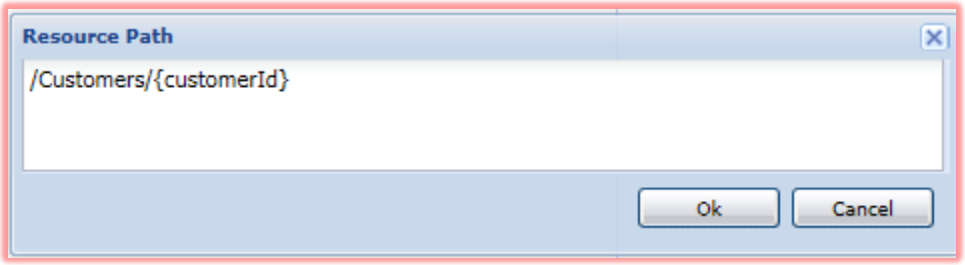
Under the UriDesign Manager bar are two sections, UriStructure which defines each API and the UriProperty (currently being developed).



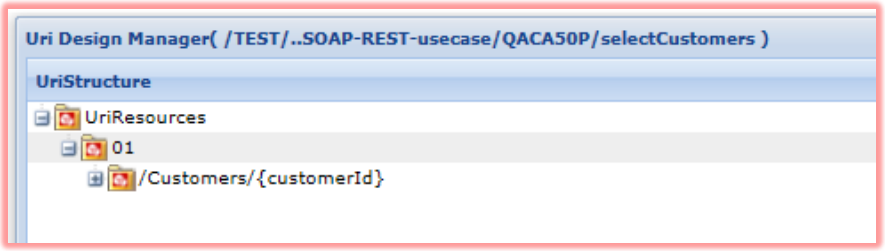
Right click on the uriResources to enter the version number of the API that will be created. In this case it will be the first version, though any version can be chosen. The Resource Path dialog box will be displayed and you will enter the Resource Path.

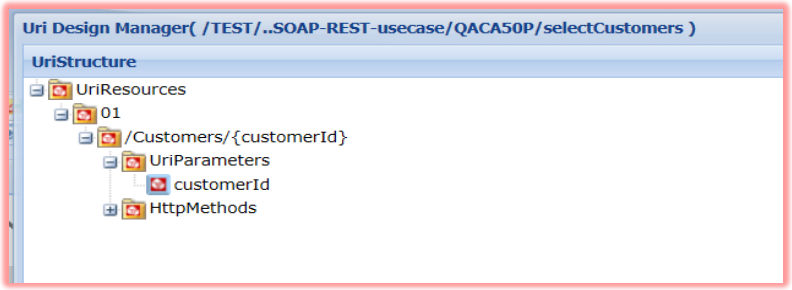


Type /Customers/{customerId} and click OK.

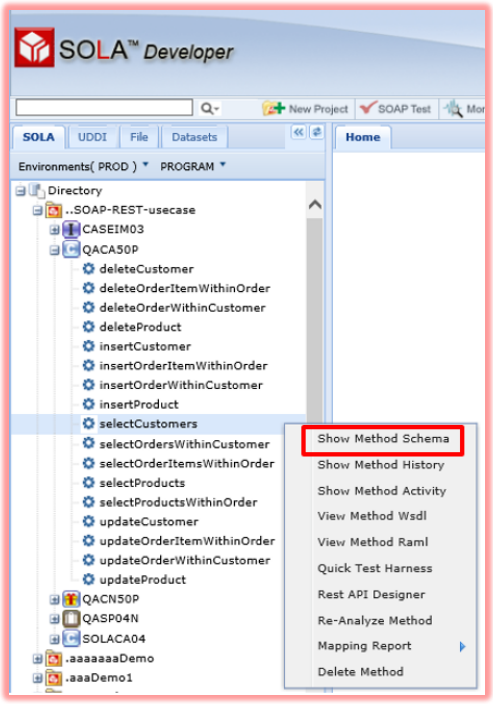


The Customers resource has now been added using UriParameter of “customerId” and automatically extracted into the UriParameters folder which you can see by clicking on the + sign beside the path i.e. /Customers/{customerId}.

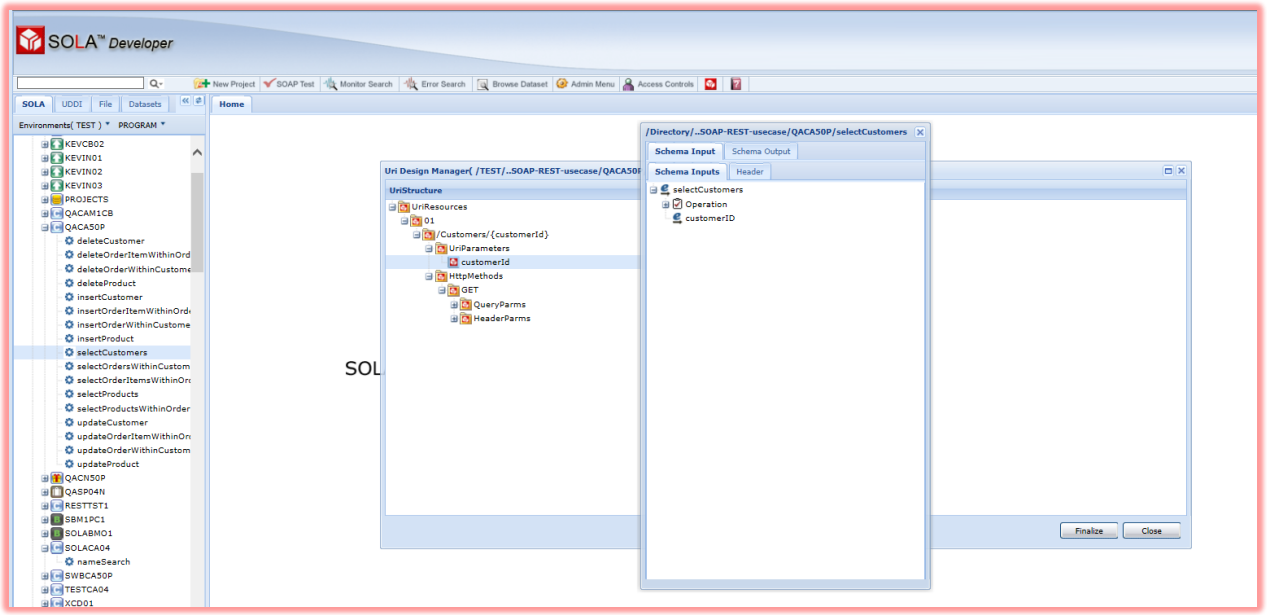




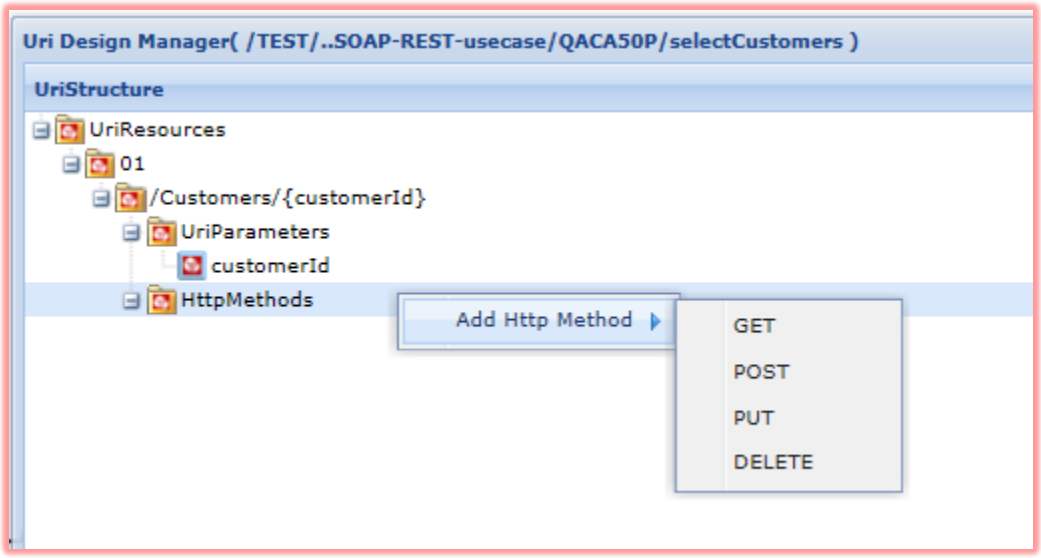
You will now access the method schema in the Directory tree on the left and right click on the method, and from the drop down select ‘Show Method Schema’.



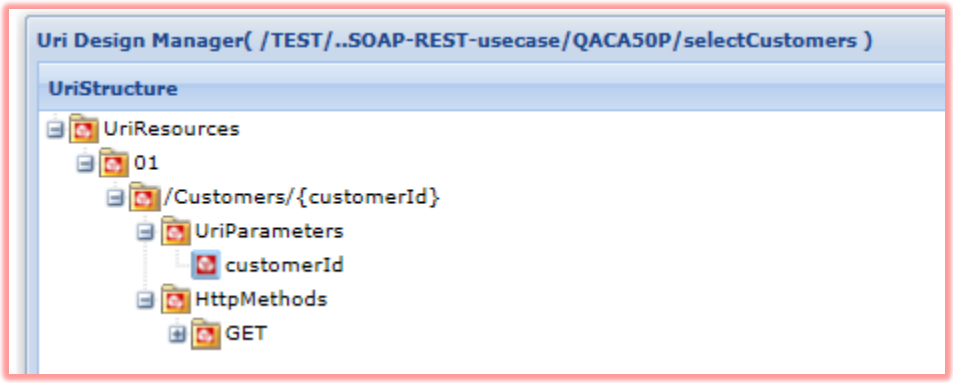
From the Schema Inputs panel drag and drop the ‘customerId’ over to the ‘customerId’ of the UriParameter to link them. You will see a green ‘+’ sign when you are properly hovering over the folder.



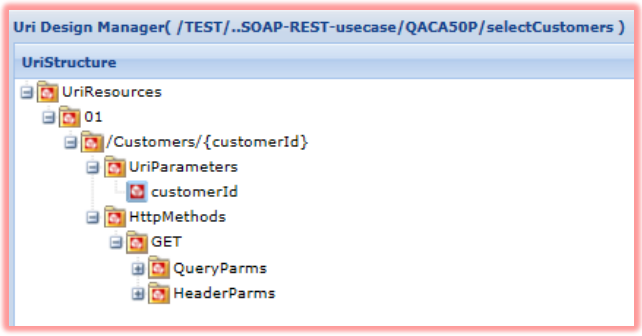
Next you will define which HTTP method you will be using for this API. In this case it will be GET. Right click on HttpMethods and add the GET method.

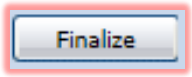


The HttpMethod GET has now been added.

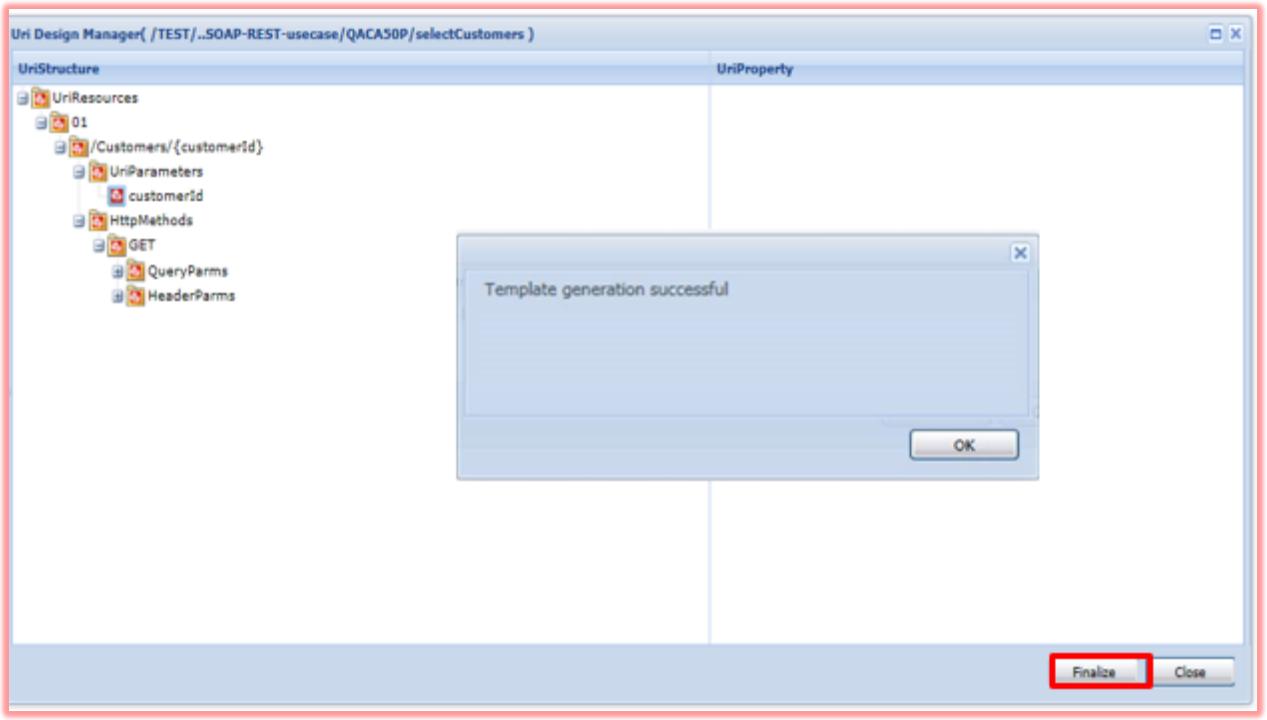


Clicking the + icon next to GET will drop down further to reveal the QueryParms and HeaderParms. We will review these further in a later example.



We are now ready to click 

You will get the message ‘Template generation successful’.



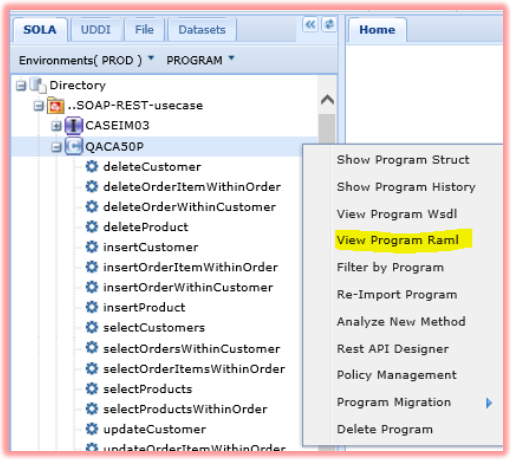
That’s it. A RAML Definition document has been created. You now can access your legacy program via a Rest API that looks something like this (in this case assume CustomerId = ‘16’):

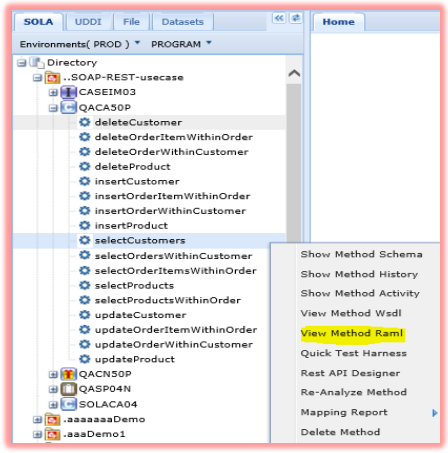
/V1/Customers?customerId=16

As explained earlier the output from this API will either be XML or JSON depending on your HTTP Headers and will follow the general schema that was established during the SOLA Analysis phase.

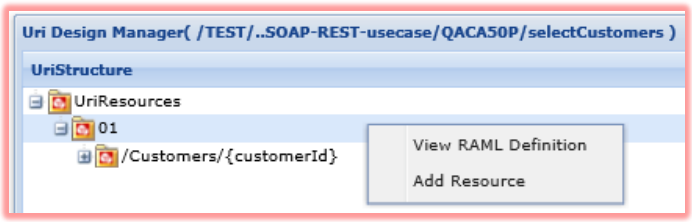
RAML Definition Document

Now you can access the RAML (Rest-ful API Modeling Language) definition by opening and saving it with the .raml extension. The RAML definition document can be accessed from either the Directory tree or the Uri Design Manager panel. You can view the RAML at the Program level or at the Method level from the Directory tree as seen here:





First you will right click on the Resource version (01) and you will be presented with a drop down as seen below. Click 'View RAML Definition'.

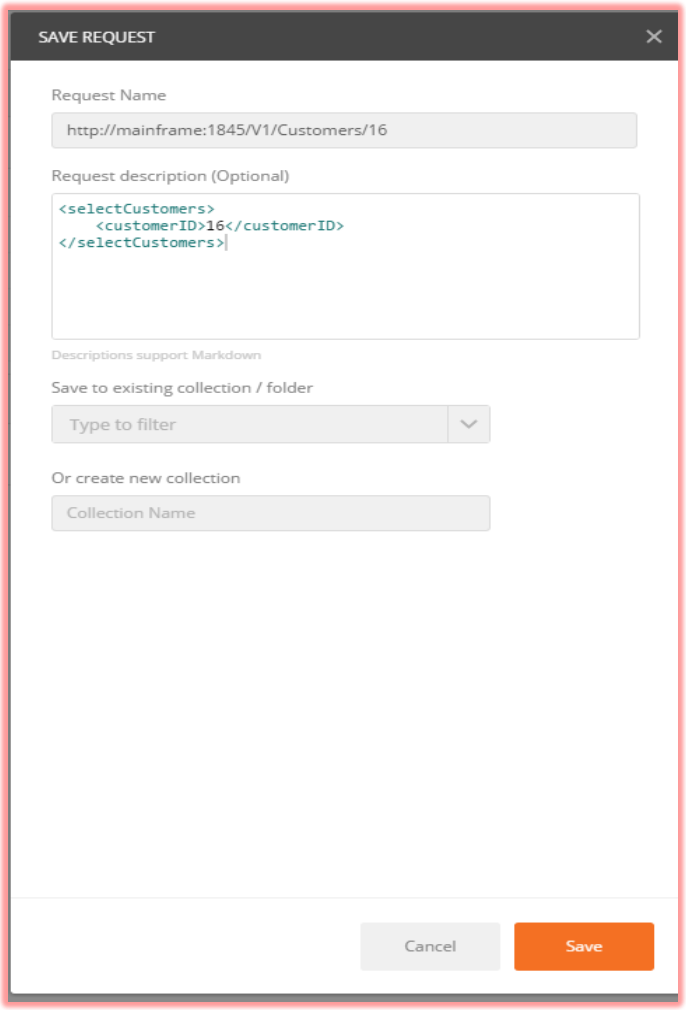


```
#%RAML 1.0
title: SOLA6 Rest API for Method( /PROD/..SOAP-REST-usecase/QACA50P/insertCustomer )
version: 00
baseUri: HTTP://MAINFRAME:1845
protocols: [ HTTP, HTTPS ]
types:
  insertCustomer:
    type: !include http://njstage-gx980:1445/sola/xsd/v1.1/inputs/PROD/..SOAP-REST-usecase/QACA50P/insertCustomer
  insertCustomerResponse:
    type: !include http://njstage-gx980:1445/sola/xsd/v1.1/output/PROD/..SOAP-REST-usecase/QACA50P/insertCustomer

/v01:

  /Customers:

    post:

      headers:
        REST-API-KEY:
          description: SOLA Rest DB2 & TSQ access key
          type: string
          enum: [ RS-T#ORD002-1109 ]
          required: true

        REST-ACTION:
          description: Runtime Charset Information
          type: string
          enum: [ /CCP:UTF-8/HCP:037 ]
          required: true

      responses:
        200:
          body:
            application/json:
              type: insertCustomerResponse
```

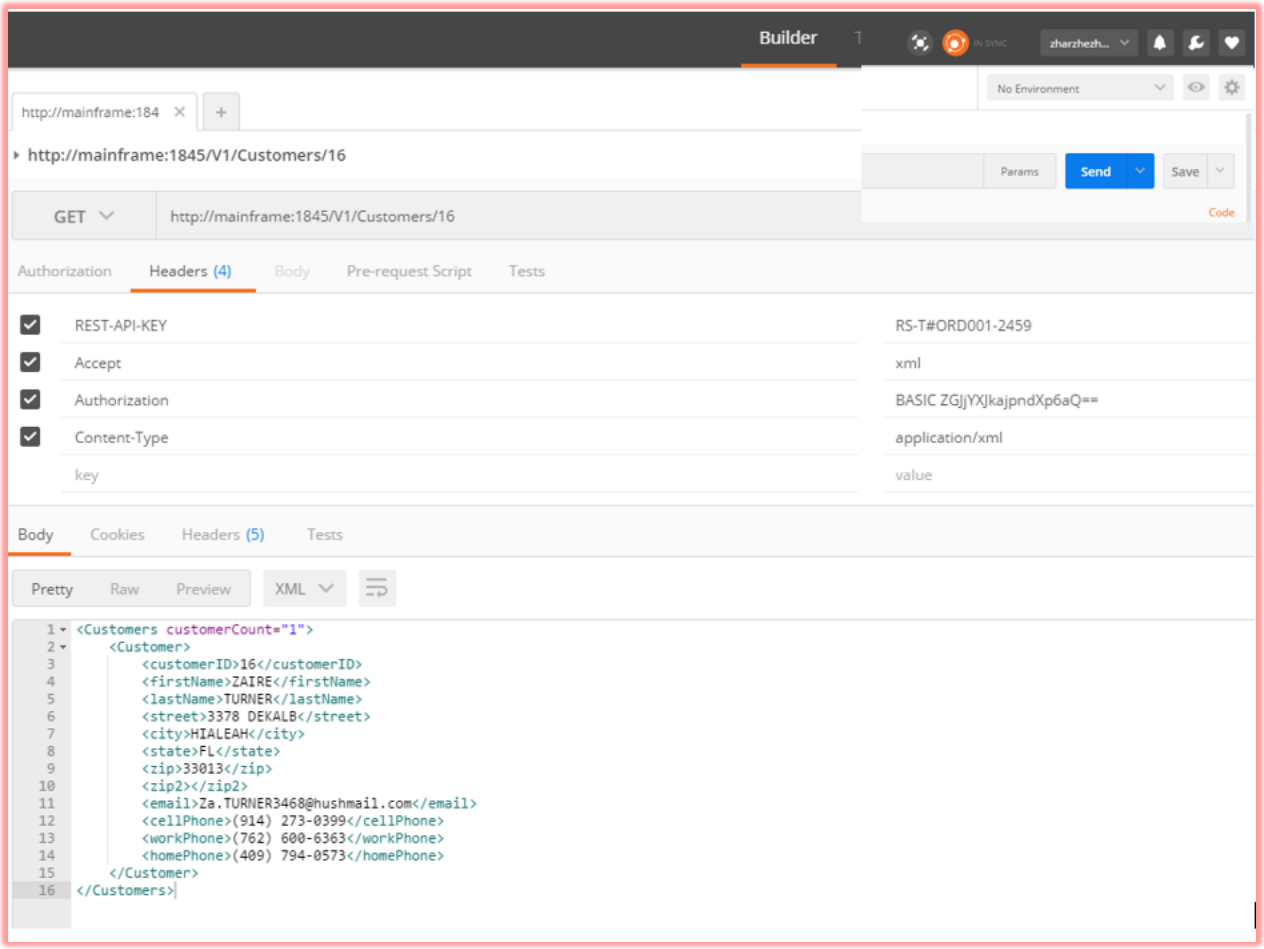
The RAML definition document will be displayed in your IE Browser.
An example of how the RAML definition can be used to create collections for testing in open source software such as Postman will follow in the next section.

API Testing – Postman

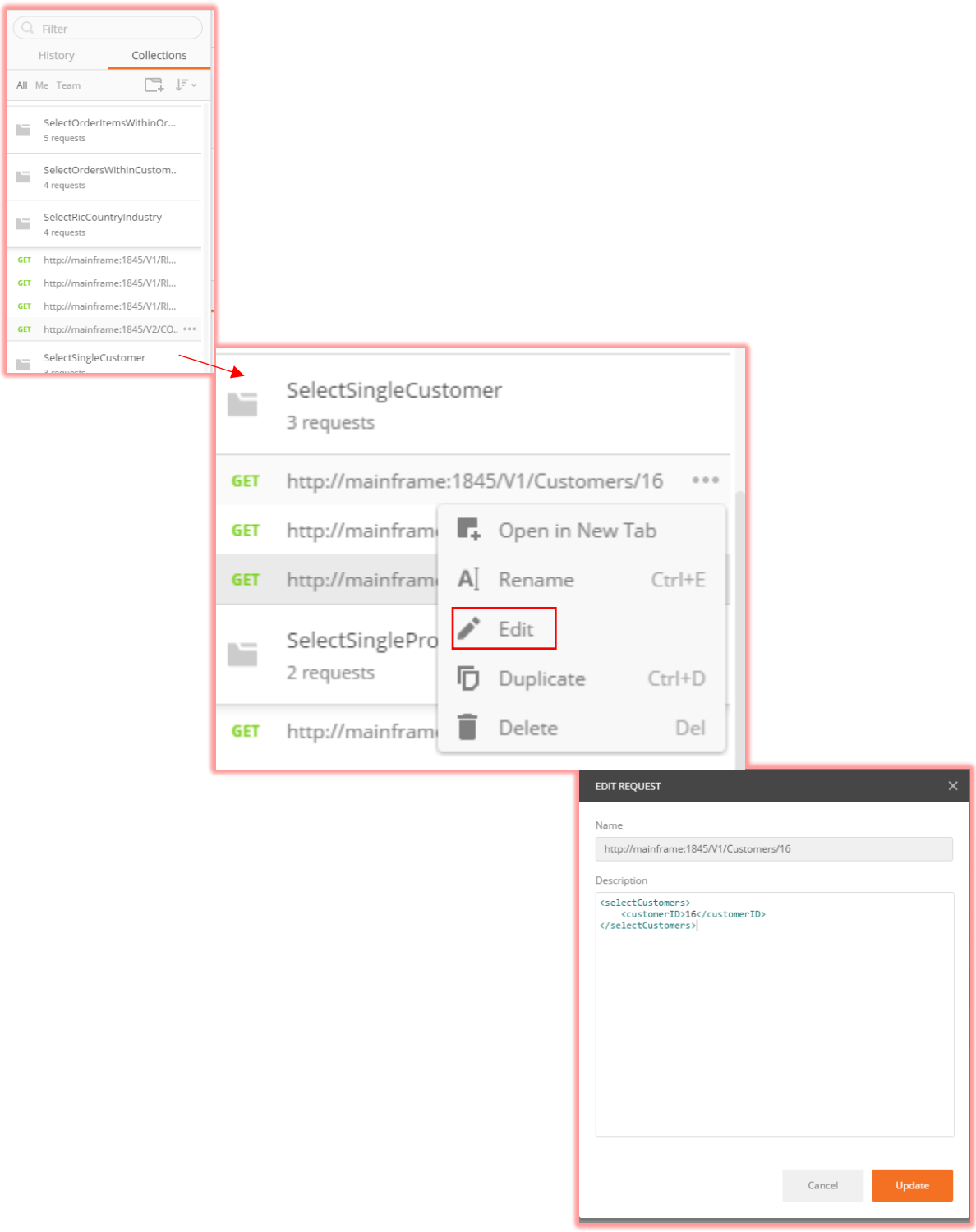
Let's begin a short review of setting up a test of this API using Postman. We would first Save the Request Name and Description to a Collection, either an existing or new Collection.



Now in the Builder space you will setup the HTTPHeader and all other Header and Resource information pertaining to the request. Once you have completed the setup of the required definitions click SEND and you will receive the Response in the Body section of the Builder screen.



You can edit the Request from the Collections panel on the left side of the Builder screen by clicking on the '**...**' beside the http header. From the drop down select the function, in this case Edit.



Example #2

In this example we will take a COMMAREA program which has already been analyzed using SOLA and enable it as a REST API. This particular program takes, as inputs, four fields:

- Sales Person ID
- Client (lead) name
- Search Type ('N' or 'S')
- Access Method (determines the scope of data that can be viewed)

Of these four fields, two of them have been declared as 'Defaults'. This means that they are not intended to be provided on the API as inputs but rather will be taken care of internally by the SOLA runtime code.

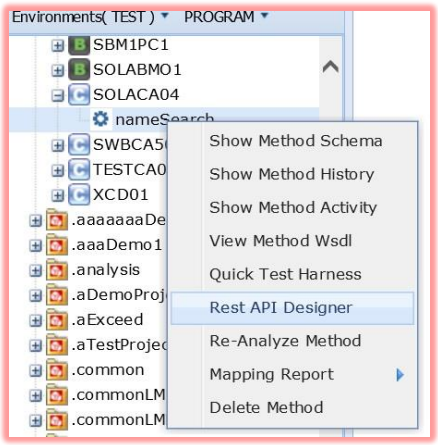
In this case 'SearchType' has been defaulted to 'N' indicating that the search will be conducted based on a Client/Lead name and 'AccessMethod' to 'F' which indicates that Firm wide data will be searched.

It will return, among other things:

- Return Code
- Return Message
- Host system ID
- Error Codes
- Number of matches found
- An array of Client/Lead data containing Name, Phone Number etc.

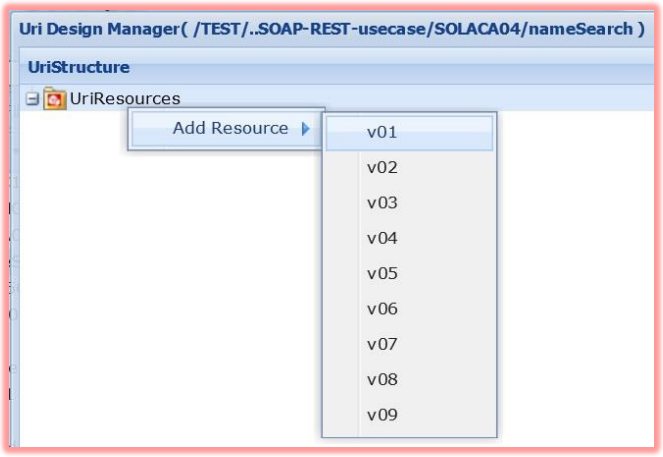
So let's get started.

First, locate the API/Method in the SOLA directory. In this case it is an API hosted by program 'SOLACA04' called 'nameSearch'. Right mouse click on the API name and, from the drop down select 'Rest API Designer':

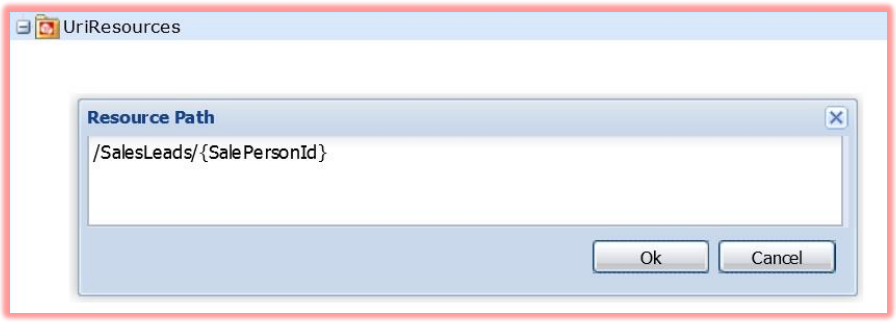


The Rest Designer will appear. If this is the first Rest API created for this service, you will see an empty folder called UriResources. Right mouse click on the folder and add a new resource

at the version you wish to generate. You do not have to start with version 01 although that is what is shown in this example:

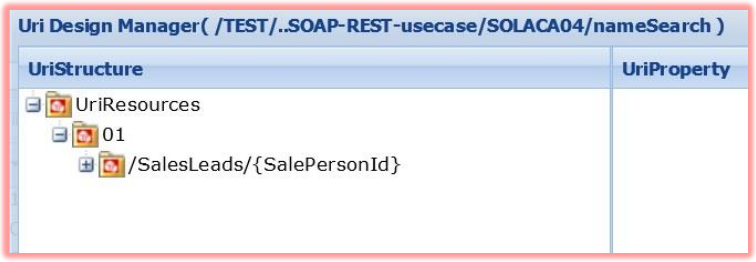


In the resulting dialog enter the resource you want to create. It can be as simple as a single path node or multiple. If you intend to transfer input data on the URI path, then you must specify a name for the parameter enclosed in braces '{}'. In the example below the resource consists of a resource name of 'SalesLeads' with a single parameter pass in on the URI called 'SalePersonId'. Note that the names chosen are completely arbitrary and do not have to correlate to any previously created parameter names. They are just place holders for later use. So for instance you could have specified '/Sales/Leads/{SalesId}' as you URI resource. In this case it is simply '/SalesLeads/{SalePersonId}'.

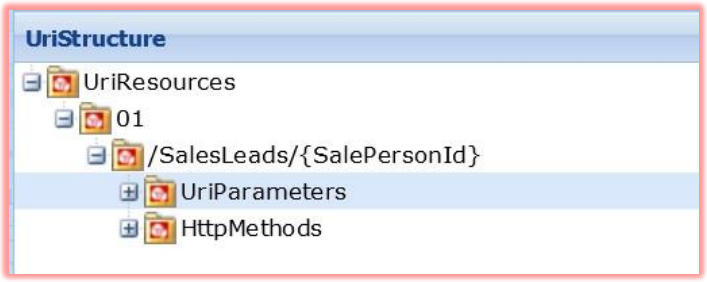


Click on the 'Ok' button.

You will now see your newly created resource as shown below:

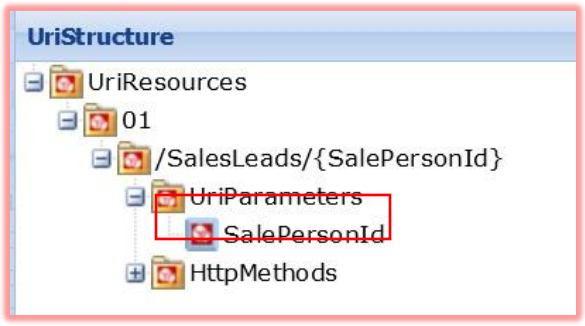


Click on the '+' to expand additional information:



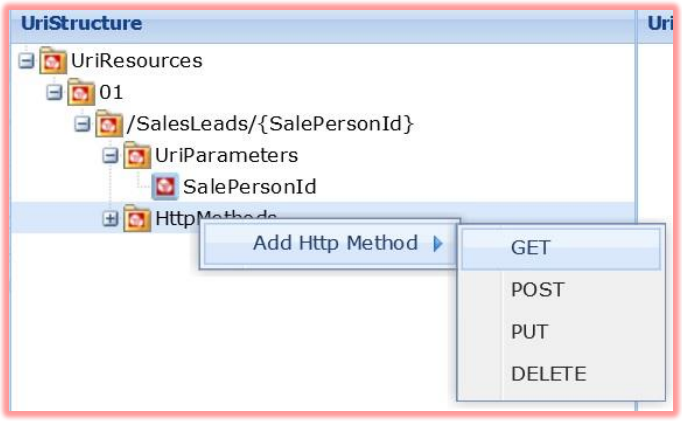
Since we have placed a parameter on the URI path {**SalePersonId**} it is automatically extracted into the UriParameters folder.

You should now see the parameter within the folder as shown:

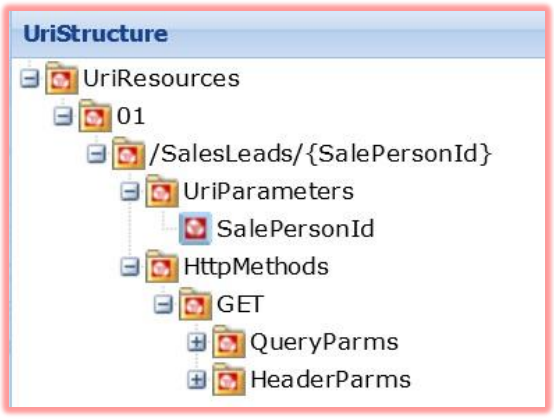


Next we need to add an HTTP method. Supported methods are 'GET', 'POST', 'PUT' and 'DELETE'.
Note: The method you select will be enforced at runtime. So if you pick 'GET' (which is done in this example) and a request for this API arrives using a 'POST' the request will be rejected.

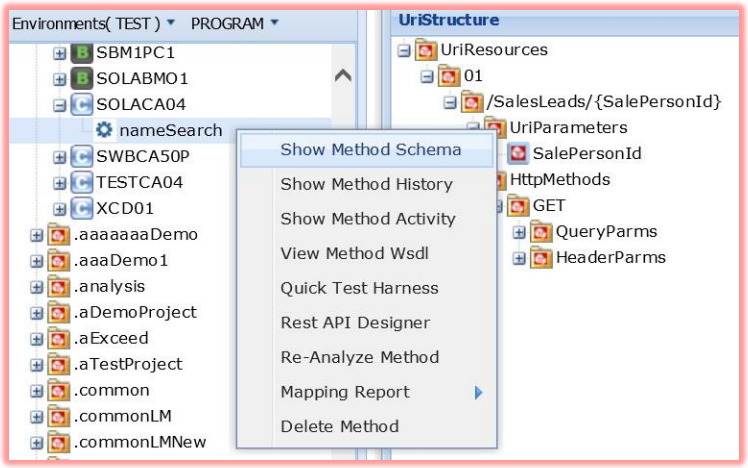
To select the method simply right click on the HttpMethods folder, select 'Add Http Method' and select the appropriate method from the drop down. In this case we are going to choose 'GET':



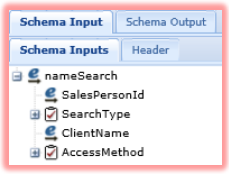
You will now see a 'GET' folder under the HttpMethods. If you click on '+' to expand the folder, it would now look as follows:



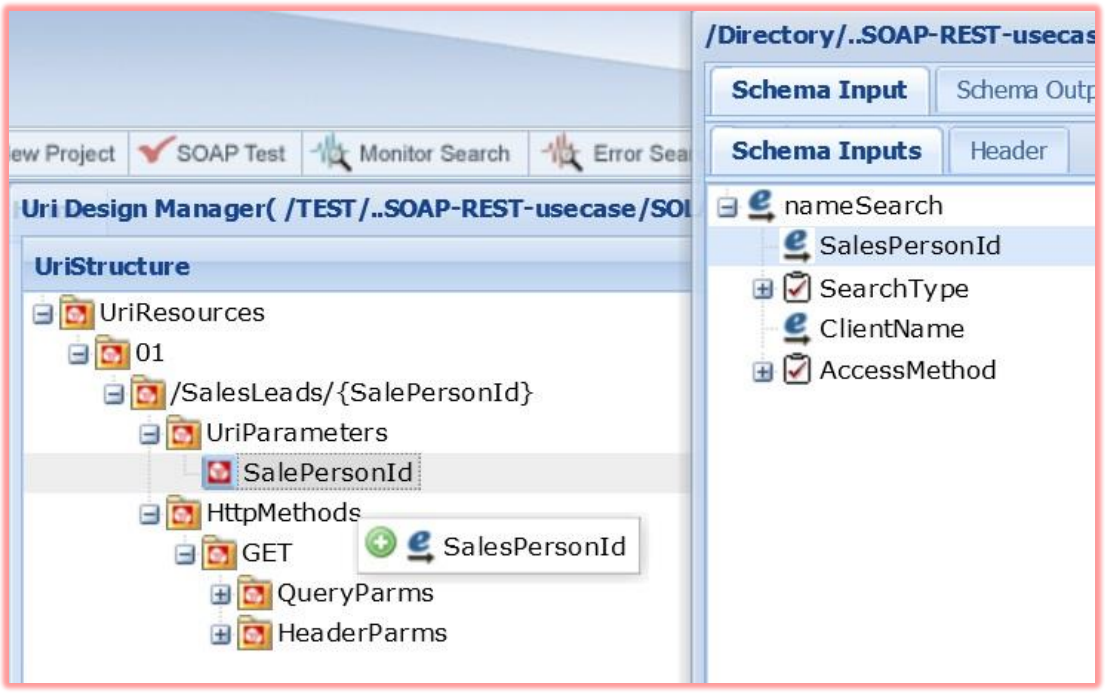
That completes the 'design' of the API. Now it is time to declare how we want to map our input schema items to this API. To do this right click on the API name in the SOLA directory once again and select from the drop down 'Show Method Schema':



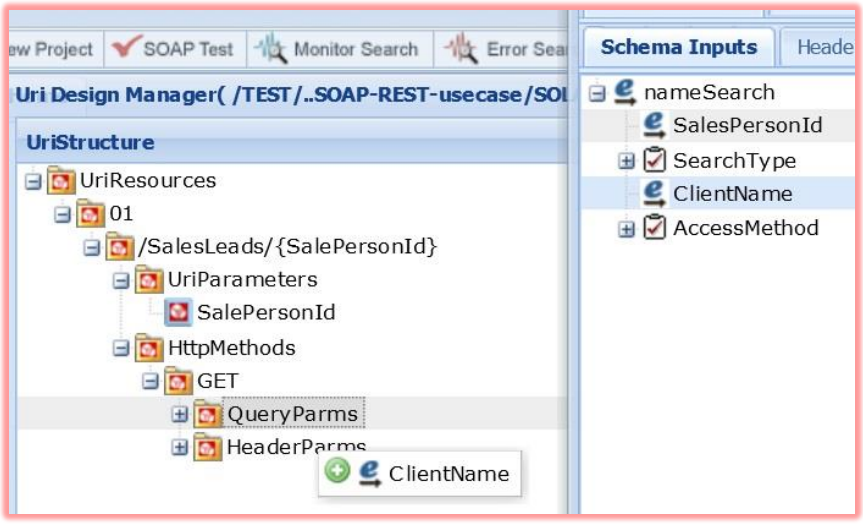
The method schema window will pop up. From this window you can see in the 'Check Mark' icons that both SearchType and AccessMethod are default fields as discussed earlier. This means that you can, and should, ignore these.



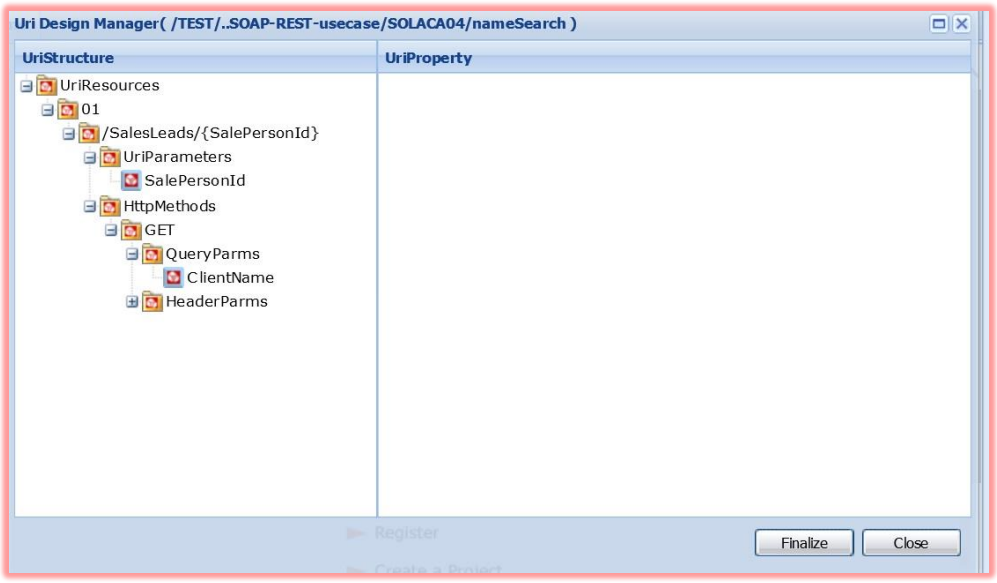
In this case we have already chosen to place the 'SalesPersonId' on the URI path. Making that association is easy. Simply click and drag SalesPersonId from the 'Schema Inputs' tab and hover over the UriParameter 'SalesPersonId' in the Rest API Designer window. When you are properly hovering over the target you will see a **green '+'** sign as shown below. This indicates that you can now release the left mouse button and drop the schema item onto the URI parameter.



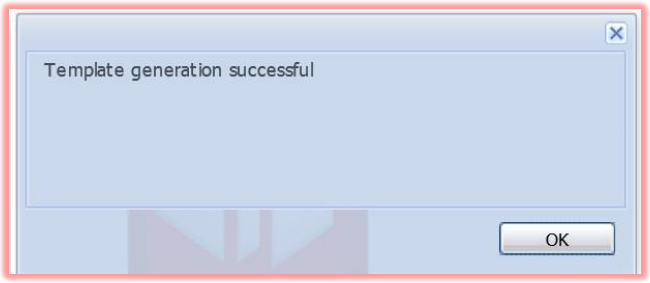
In this example we will make the '**ClientName**' schema item a Query String parameter. To do this we will once again use the drag and drop technique. This time simply drag **ClientName** from the 'Schema Inputs' tab and drop it onto the 'QueryParms' folder. Once again you will see a green '+' sign when you are properly hovering over the folder:



Once dropped the **ClientName** will appear in the folder as shown:



We are now finished building this Rest API. Simply click on the '**Finalize**' button. After a few seconds you should see this:



That's it. You now can access your legacy program via a Rest API that look something like this (in this case assume Sales Person ID is 'SPID001' and Client Name is 'Smith'):

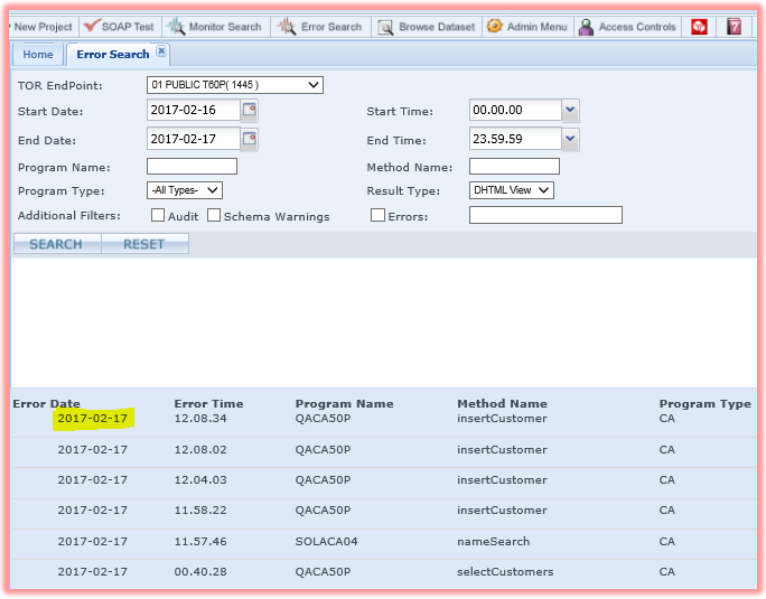
/v1/SalesLeads/SPID001?ClientName=Smith

As explained earlier the output from this API will either be XML or JSON depending on your HTTP Headers and will follow the general schema that was established during the SOLA Analysis phase.

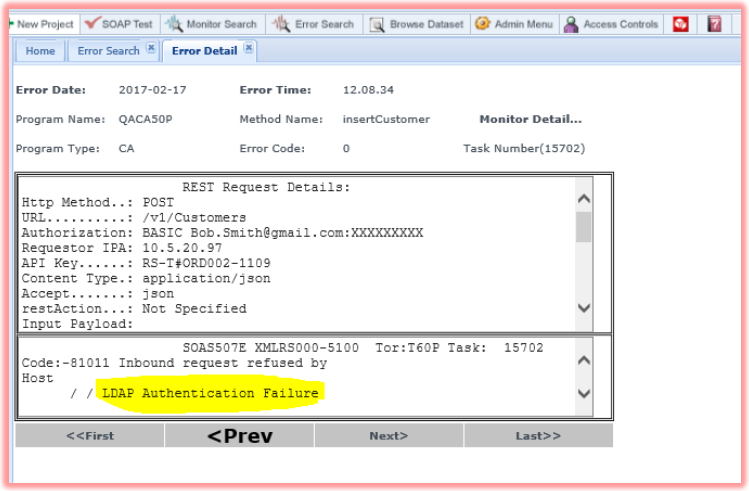
Error Logging

When testing your Rest APIs, you can locate errors in SOLA Developer by clicking on the 'Error Search' tab on the Menu bar. The 'Error Search' tab will open up and you will need to enter search criteria. In our example below we are searching by Date and All Types of programs.

A report of all the programs and errors for that date range will be presented. Look for your program name and click on the date associated with it.

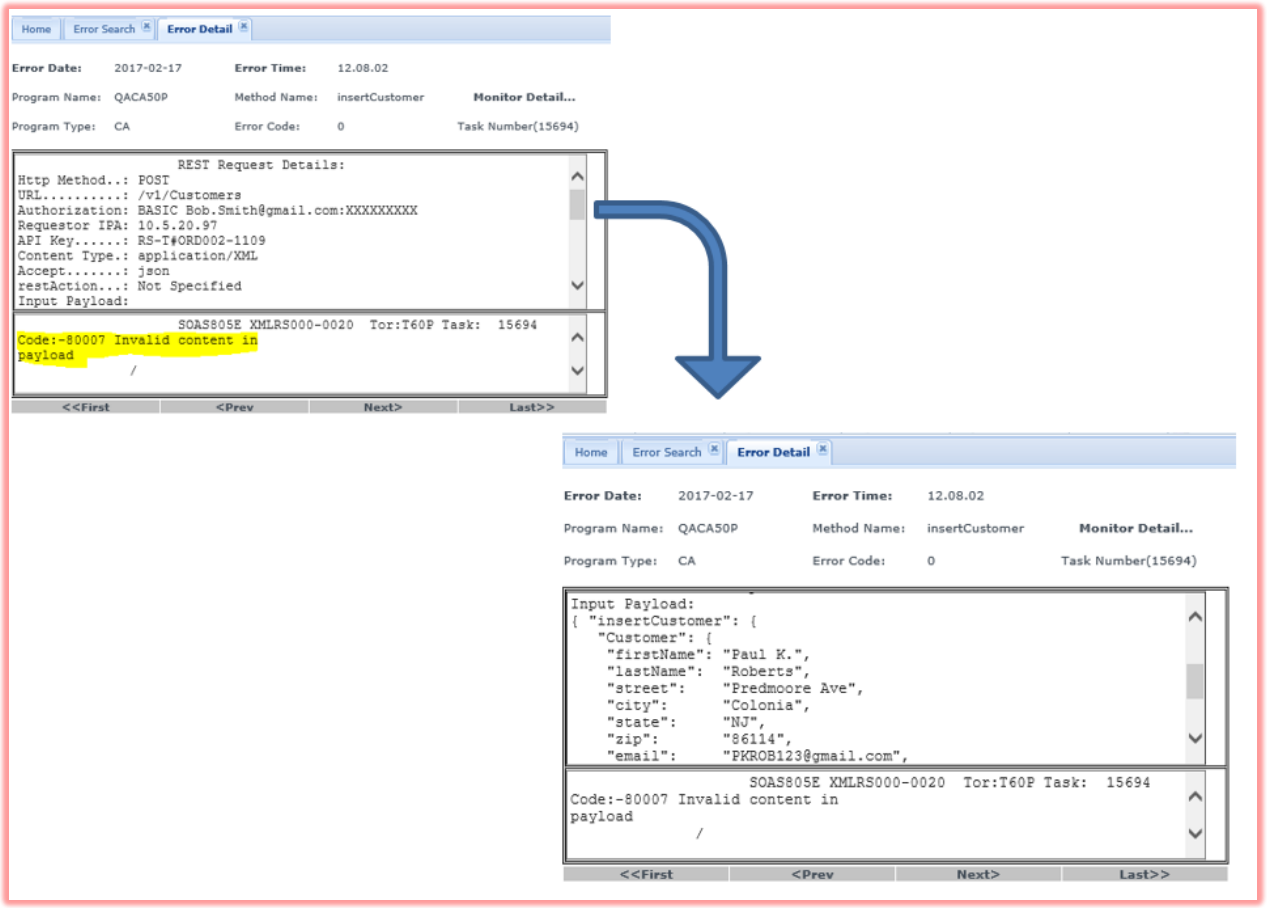


In our example below this POST - Inbound Request was refused due to an Authentication failure.

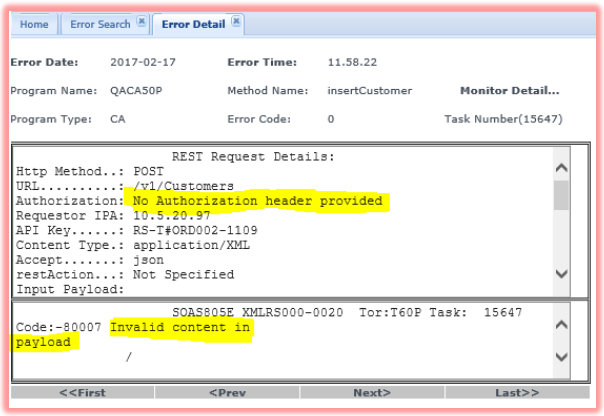


In yet another example of the Error Logging feature we have a Payload that contains invalid content.

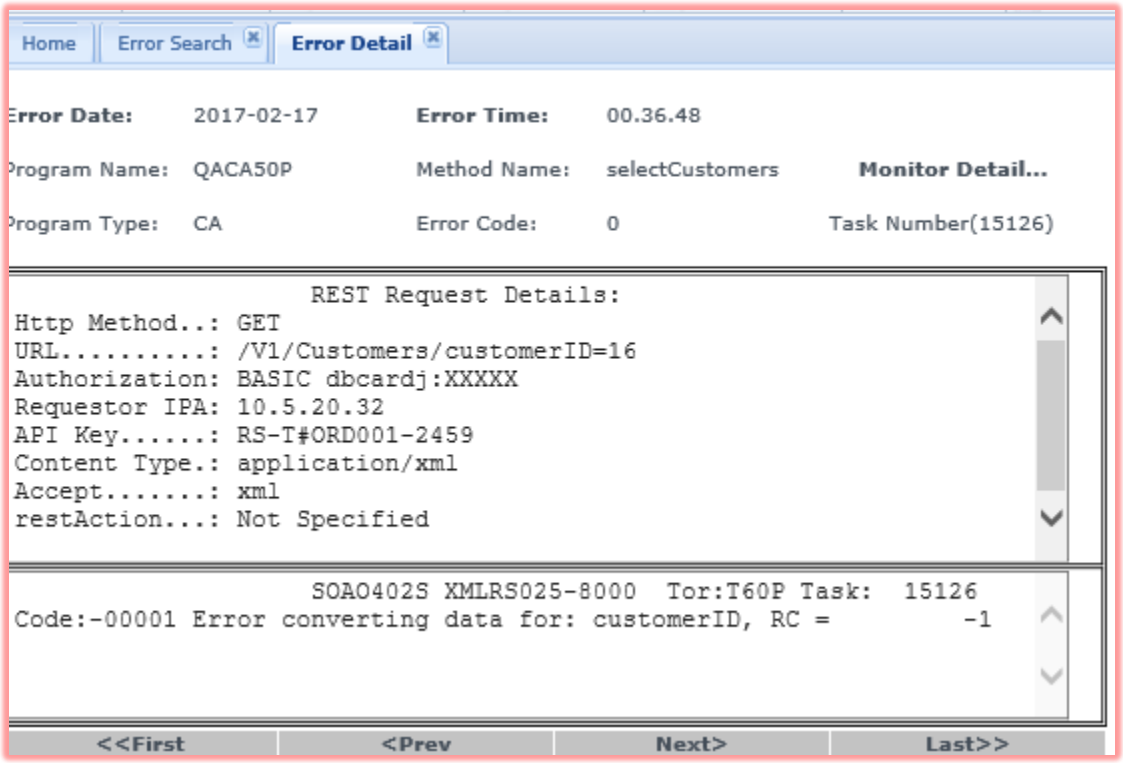
Scroll down within the Request Details and the Input Payload is displayed.



In the following example the POST Request failed because 'No Authorization header was provided'.



In the last of our Error Logging examples we have a URL that was defined improperly; it should have been coded as “/V1/Customers/16”:



CICS Analyzers for REST

XMLRSAN1 - This will look for a Basic Authorization Header (BAH). If it is found it assumes that the credentials are Mainframe username and password and will attempt to start the CICS transaction under those credentials. If it finds no BAH then it will look in the mapping facility for an entry of DEFAULT. If found it will attempt to start the transaction under the '**tranid**' and '**username'** specified in the mapping facility. If no mapping facility entry for DEFAULT is found it will start the transaction under the CICS default '**userid**'.

XMLRSAN2 - This will also look for the Basic Authorization Header. If it is found, it will ALWAYS look into the mapping facility for the '**userid**' contained. If the userid is found, it will start the '**tranid**' under the credentials specified in the mapping facility. If not, it will look next for a default entry and if found it will run under those credentials. If not found it will run under the CICS default.