Tutorial

MDRest4i

REST Framework & SDK
from
Midrange Dynamics

Version 12

Published May 24, 2023

Table of Contents

1 Introduction

1.1 Overview

MDRest4i iCore is an RPG ILE framework that enable RPG programmers to build REST API's and Consumers quickly and easily using only RPG ILE and SQL.

MDRest4i SDK is a development kit that generates REST API's and Consumers in RPGLE from SWAGGER, using the MDRest4i iCore RPGLE functions.

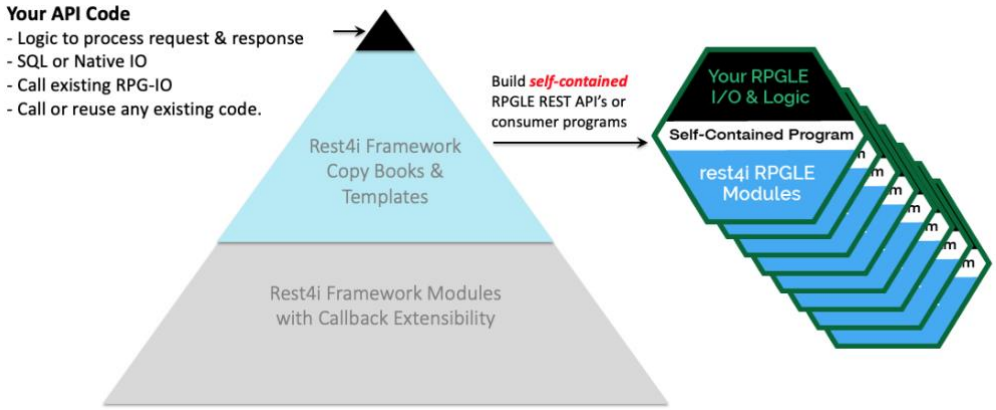


Figure 1 - MDRest4i iCore Layered Architecture

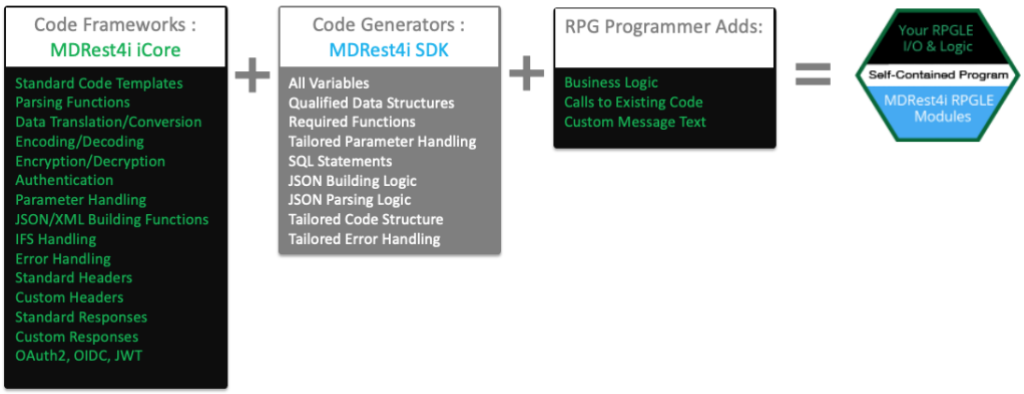


Figure 2 - MDRest4i Productivity Concept

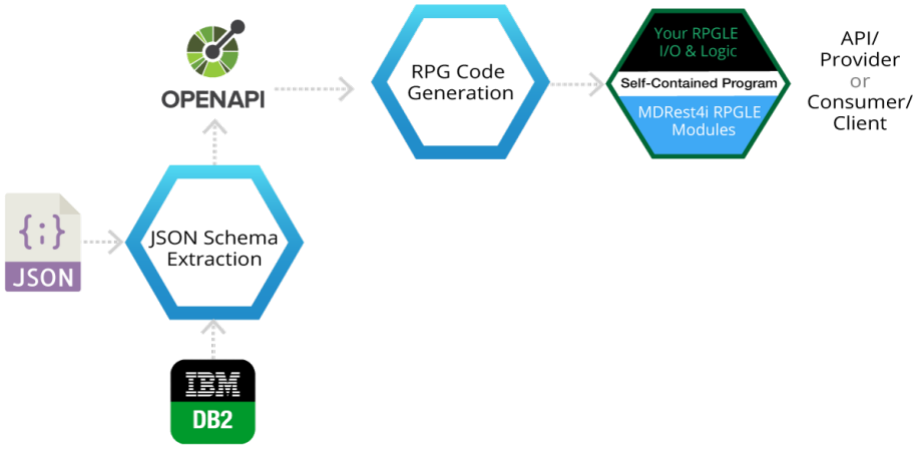


Figure 3 - MDRest4i SDK Alternate Spec Source

In Figures 1, 2 and 3 above, the layered nature of MDRest4i is exposed. Developers don't need to see common logic that rarely changes(this is handled by the iCore Framework), and only need use RPGLE & SQL for their primary purpose: ***Business Logic and Input/output of mission critical data,*** while using the MDRest4i iCore copybooks to allow developers to customize the framework for a company's specific standards or requirements.

RPG ILE program templates and examples for both consumers and API's in RPG are provided with the MDRest4i framework to kick-start manual development.

MDRest4i SDK is supplied with some IBM i commands that automate code generation of RPG ILE REST API and consumer stubs.

MDREst4i SDK Console (web UI) increases productivity further with a set of REST API's (built with MDRest4i) that generate the RPG ILE REST API's and consumer stubs, where more complex JSON payloads are required, and provides automated API documentation.

This tutorial will show how to:

- Generate the RPG ILE REST API's and Consumer stubs in RPG ILE using the MDRest4i SDK generator commands.
- Generate the RPG ILE REST API's and consumer stubs in RPG ILE from SWAGGER, using the MDRest4i SDK pre-built REST API - MDRGENXAPI
- Add RPG and embedded SQL manually to generated RPG ILE stubs
- Secure a REST service using Java Web Tokens (and implementation of OAuth) in RPG ILE
- Handle pagination in RPG ILE for a stateless REST API
- Debug an RPGLE REST API

1.2 Pre-Requisites & Setup

1.2.1 Required Software & Servers

- MDRest4i iCore is installed on the IBM i, with a valid license applied (for sections 4 & 5)
- MDRest4i SDK Web Application is installed and configured on Windows/Linux/MacOS (for section 8)
- An IBM i HTTP Apache Instance is configured and running, that exposes library MDRSTxxxx on the server where MDRest4i iCore is installed

1.2.2 Libraries, Files and IFS Folders Used in the Tutorial

For the tutorial examples we use a library called MDRTUTLIB. Please check if the following exist. If they do not then please create them manually or use your own names. If multiple users are going through the tutorial simultaneously, please adjust the library, source file, and source member names accordingly.

Library: **MDRTUTLIB**

Source PF: **MDRTUTLIB/QRPGLESRC**
IFS Folder: **/mdrest4i/mdrtutorial**

1.2.3 MDRest4i Tutorial Code Examples

This entire tutorial covers 16 examples, 5 of which are in section 8 which covers RPGLE API/Consumer generation from SWAGGER. You can download all of the SWAGGER, RPGLE and JSON example code for the tutorial plus a few extra examples from here:

(https://europe.mdcms.ch/downloads/MDRest4i-v12.0-Tutorial-swagger-rpgle-json-examples.zip)

Download and unzip this MDRest4i-v12.0-Tutorial-swagger-rpgle-json-examples for use during the tutorial exercises below

2 REST Concepts

If you are familiar with REST concepts and programming you can skip this section and go straight to MDRest4i iCore Framework Architecture

2.1 Overview

REST is an architectural style described by six constraints. These constraints, applied to the architecture, were originally communicated by Roy Fielding in his doctoral dissertation (see https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm) and define the basis of RESTful-style:

- **Uniform Interface**
  The uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently. For example some HTML, XML or JSON that represents some database records
- **Stateless**
  As REST is an acronym for Representational State Transfer, statelessness is key. Essentially, what this means is that the necessary state to handle the request is contained within the request itself, whether as part of the URI, query-string parameters, body, or headers.
- **Cacheable**
  On the World Wide Web, clients (your web browser for example) can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.
- **Client-Server**
  The uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered.
- **Layered System**
  A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.
- Code on Demand

  Servers are able to temporarily extend or customize the functionality of a client by transferring logic (as part of the response) to it that it can execute. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

In general the following also applies to REST:

- Preferred format is JSON
- Services are catalogued and easily integrated
- Self-descriptive Messages

Complying with these constraints, and thus conforming to the REST architectural style, will enable any kind of distributed hypermedia system (IBM i, Mobile, Web etc..) to have desirable emergent properties, such as performance, scalability, simplicity, modifiability, visibility, portability and reliability.

NOTE: The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful. MDRest4i has templates for each HTTP Request, where most of the work is done behind the scenes and allows for a greater turn around time in development.

2.2 REST Consumer(client) Model

Normally when we connect to a server, we use a client application PC. Then we have a client-server scenario; where one machine is clearly the server and the PC is clearly the client.

When discussing web services, the problem becomes a little bit more complicated, because there may be two servers talking to each other and neither appears to be a client. To simplify this the server program making the request is described as the consumer.

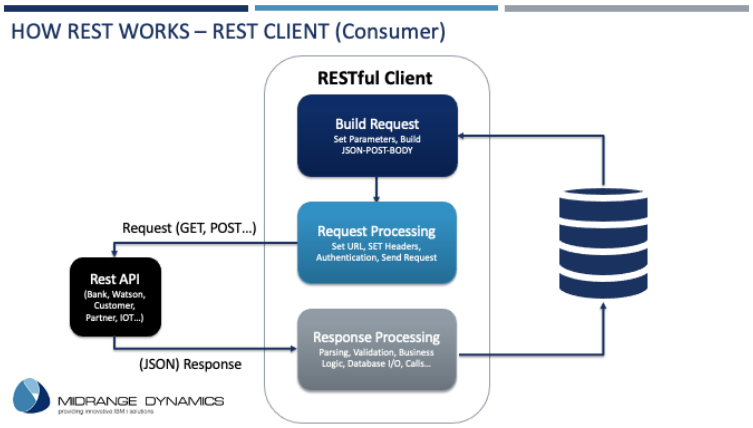


Figure 4 - REST Client/Consumer Flow

Figure 2 above describes the generic flow of a REST consumer.

2.3 REST Provider(service) Model

The Provider or Provider is the service that processes the request and sends the response. Normally we just call it a service although it may also be referred to as a Microservice.

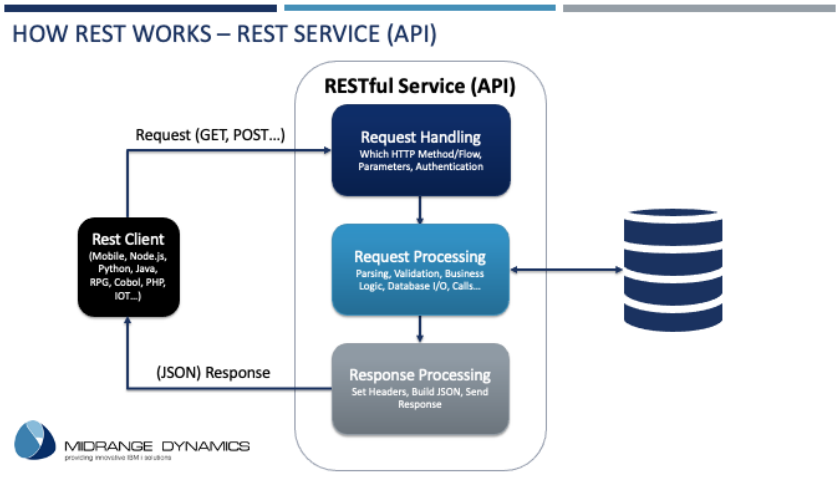


Figure 5 - REST Service/API Flow

Figure 3 above describes the generic flow of a REST service.

The MDRest4i framework provides templates and commands for building both REST services and consumers.

2.4 REST Methods and Safety

HTTP verbs comprise a major portion of the "uniform interface" constraint of REST and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. The table below describes

HTTP Verb/Method	CRUD	Description
GET	Read	The HTTP GET method is used to **read** (or retrieve) a representation of a resource. According to the design of the HTTP specification, GET (along with HEAD) requests are used only to read data and not change it. **Do not expose unsafe operations via GET—it should never modify any resources on the server.**
POST	Create	The POST verb is most-often utilized to **create** new resources. **POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Making two identical POST requests will most-likely result in two resources containing the same information.**
PUT	Update/ Replace	PUT is most-often utilized for **update** capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.
PATCH	Update/ Modify	PATCH is used for **modify** capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource.
DELETE	Delete	DELETE is pretty easy to understand. It is used to **delete** a resource identified by a URI.

Table 1 - REST Methods

In order to maintain the safety of your service please consider the possibility that a service could be invoked several times, possibly with the same data. Ensure your logic caters for this scenario if appropriate.

For a more in-depth tutorial on the details of REST:
https://www.restapitutorial.com/index.html

2.5 HTTP Message Components

REST architectural style is implemented with the HTTP protocol. At is core HTTP uses messages to communicate between client and server.

Table 2 below defines the main parts of any HTTP/REST message that must be handled by REST service or consumer programs.

COMPONENT	POST	GET	PUT	PATCH	DELETE
Path or URI (*uniform resource identifier*) – comprised of a IP address or Host Name, port Number, context path (relative directory on the server where the service can be found)	**M**	**M**	**M**	**M**	**M**

Query String – usually used for Parameters		**O**			**M**
Path Parameters	**O**	**O**	**O**	**O**	**O**
Headers – the HTTP server adds some, some are mandatory and may be custom. For example "Content-Type: application/json; charset=UTF-8"	**M**	**M**	**M**	**M**	**M**
Response Body – usually JSON	**M**	**M**	**M**	**M**	**M**
Request Body – usually JSON	**M**		**M**	**M**	

M=Mandatory O=Optional

Table 2 - REST Program Components

The logic used in a Provider (service) is mirrored on the consumer side and vice-versa. For example a REST Provider that receives a GET method request

(http://yourserver:yourport/yourlib/yourapi?yourparm=value)

will use logic to build/write a JSON response and send this back to the consumer. The logic in the consumer will then use logic to PARSE this JSON response fro the service. The consumer program making a POST request, will build/write a JSON request body, and the POST service that receives the request will parse the request body, and then build/write a JSON response. The consumer will then parse this response.

3 MDRest4i iCore Framework

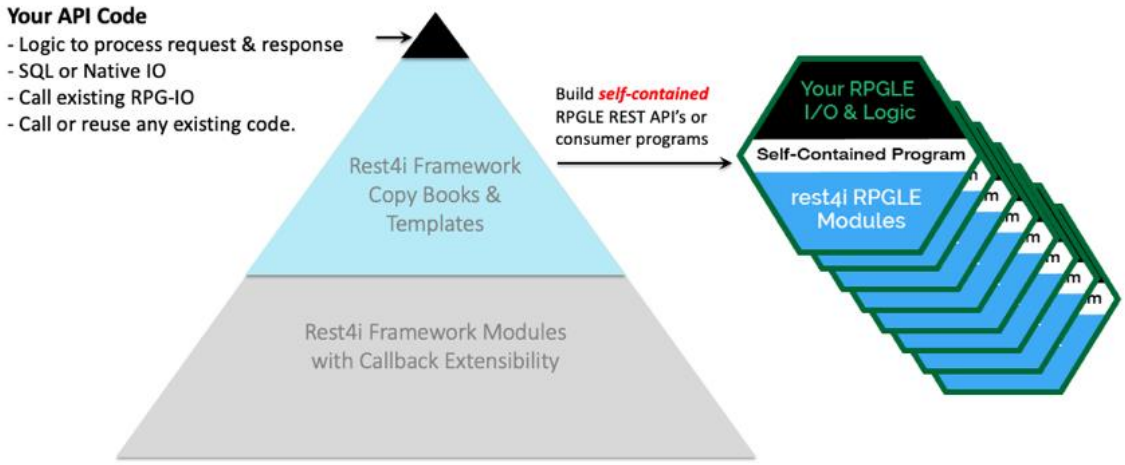


Figure 6 - MDRest4i Framework Architecture

MDRest4i uses a set of interlocking engines that perform a variety of functions. These engines are implemented as ILE modules bound into service or consumer programs using binding directories, exposed as exported procedure Interfaces, making the engines easier to install and use.

Copybooks are used to import the procedure interfaces and various other commonly used declarations into MDRest4i programs. Each MDRest4i module has its own copybook with all the procedure definitions, for rapid inclusion into your code. These copybooks can be found in MDRST/QRPGLESRC.

The MDRest4i programs will include only the copybooks for the engines that are being used into the PGM Stack and the ILE binder will ignore the functions that are not defined. In addition, Comments are only included in the object when “LXR_CommentInc” is Defined.

3.1 Master Templates

To simplify building of REST Consumers and Providers, a master template for a API/Provider program, master template for a Consumer program, and a are provided in MDRest4i. The SDK generators use these templates to create the RPGLE programs.

These templates are made up of a combination of modules, copybooks, and binding directories. They control the standard flow of a Provider or a consumer and contain the generic logic suitable for each type, such as writing the log files, parsing the request, chunking the response etc.

For full details and reference of the MDRest4i framework are documented in the ***MDRest4i_11.0_User-and-Reference-Guide.***

3.1.1 REST API/Provider/Service Program Template

The logic flow for a REST service is as follows:

- Fetch Request Details:
- Request Type
- Query Parameters
- Request body (for PUT, POST, PATCH)
- Load supplied custom Headers
- Extract Authorization Header if necessary

- Process Request:
- Process the JSON/XML from request body (if supplied) to load values in appropriate variables
- Load query parameters in appropriate variables
- Process the business logic (using query parameters or the info from request body)
- Extract and process any Headers from the request

- Process Response:
- Write the response in XML or JSON form using the appropriate MDRest4i procedures
- Update standard or custom headers

3.1.2 REST Consumer Program Template

The logic flow for a consumer program is generally:

- Build & send a request (data out) including:
- Headers
- Query string (url/browser address bar)
- Builds JSON Request body (POST/PUT/PATCH) JSON
- Receive and Process Response (data in)
- Headers
- Parses Response Body (POST/PUT/PATCH) JSON/XML
- Business Logic & DB I/O

MDREst4i uses a template module – MDRCNST, to implement this model

The three main sub-procedures used in a consumer program built with MDRest4i are:

- **Buildrequest:** the logic used to create the requestBody (JSON/XML), set http header values and build query parameter strings, to be sent to the API is written in this procedure.

- **Initialize**: All of the connection details for the request are set here such as host, path, port number, authentication, proxy etc.
- **GskConsume**: this procedure executes the actual request to the APi, including setting up the SSL tunnel, sending the HTTP message, chunking, EBCDIC-UTF-8 conversion, GZIP/UNZIP, receiving the response, parsing headers and response body into memory.

The examples in this tutorial

3.2 Debugging a Provider/service

To follow the logic cycle of the API execution or to fix a bug it will be necessary to debug the RPG ILE program. This has been simplified in MDRest4i with two methods, both using a switch in one of the copybooks.

Your REST services created with MDREst4i run under an HTTP server instance. To find the CGI jobs used to execute your services in the server, use the following command:

```
WRKACTJOB JOB(YOURHTTPINSTANCENAME)
```

One of the functions running in that server instance will be:

```
PGM-QZSRCGI
```

There may however be several of these running. To determine which one is used to execute your service, you can either set the header in the service response to show the CGi job used to execute the service, or hold your service job using a switch in MDRest4i. Either method works to determine the job being used.

3.3 Setting x-mdrsrvjob HTTP header (returns the CGI job number used for the request)

When testing the APi/Service with POSTMAN, SOAPUI or your browser, the response headers can be viewed. MDRest4i can add a header to the response of a service, which contains the following information:

- Program Name – name of the Service/API Object
- Program Library – name of the library the object is running from
- Job Name – name of the HTTP server instance & job name
- Job Number – job number of the CGI job that the service is executed under

To add this header to the response of your service do the following:

Set the switch ng_mdrsrvjob in the z_custominit subroutine of a service. By default this is set to *off

```
Begsr Z_CustomInit;
ng_mdrsrvjob = *on
n_saveIFSSwitch = *off;
Endsr;
```

This will return the following header:

x-mdrsrvjob: YOURAPI-YOURLIBRARY-YOURSERVERINSTANCE-IBMJOBNO

To debug the API run STRSRVJOB command along with job details retrieved. For example:

```
STRSRVJOB JOB(154180/QTMHHTTP/LXRDKM)
```

QTMHHTTP is the default user that runs an HTTP cgi job

After that use the STRDBG command along with program and the corresponding library combination. For example:

```
STRDBG PGM(LXRTUTORIAL/CLIENTS) UPDPROD(*YES) OPMSRC(*YES)
```

Call the service from whichever client you are using: SOAPUI, POSTMAN browser. This will bring up the job in debug mode.

Add the break points in the program at required places using F6 function key

If the control reaches the breakpoint, which you have added, the program will be displayed in debug mode and then you can use function keys F10/F12 to control the execution of the program.

Once your debugging is complete, execute ENDBG and ENDSRVJOB on command line subsequently.

3.4 Holding the HTTP CGi job Completely (where CGi job cannot be pre-determined)

To hold and debug the API, we will have to write "/define trace" statement before the copy statement for MDRESTDFN in the generated API source and then compile it in the REST services library mapped to one of the servers.

```
0017.01  /define trace
0018.00   // Standard MDRest4i Copybooks for REST Provider0
019.00  /copy qrpglesrc,MDRESTDFN
```

To get the service job details, use

```
WRKACTJOB JOB(YOURHTTPINSTANCENAME)
```

and look for this function:

```
Function        Status
PGM-QZSRCGI      MSGW
```

Then take option 2 against this job, and press the F4 function key. The following display will give you the job number

```
                              Change Job (CHGJOB)

 Type choices, press Enter.

 Job name . . . . . . . . . . . . > LXRDKM         Name, *
   User . . . . . . . . . . . . .  >   QTMHHTTP    Name
   Number . . . . . . . . . . . .  >   154180      000000-999999
 Job priority (on JOBQ) . . . . .   *SAME          0-9, *SAME
 Output priority (on OUTQ)  . . .   5              1-9, *SAME
 Print device . . . . . . . . . .   PRT01          Name, *SAME, *USRPRF...
 Output queue . . . . . . . . . .   *DEV           Name, *SAME, *USRPRF, *DEV...
   Library  . . . . . . . . . . .                  Name, *LIBL, *CURLIB
 Run priority . . . . . . . . . .   25             1-99, *SAME
```

To debug the API run STRSRVJOB command along with job details retrieved from the above process on your server. For example:

```
STRSRVJOB JOB(154180/QTMHHTTP/LXRDKM)
```

After that use the STRDBG command along with program and the corresponding library combination. For example:

```
STRDBG PGM(MDRTUTLIB/CLIENTS) UPDPROD(*YES) OPMSRC(*YES)
```

Add the break points in the program at required places using F6 function key.

Now go back to WRKACTJOB screen, where job appears in MSGW status. Take option 7 on this job and press enter you will get the message below:

```
                        Additional Message Information

Message ID . . . . . :   RNQ5335       Severity . . . . . . . :   00
Message type . . . . :   Inquiry
Date sent  . . . . . :   15/03/18      Time sent  . . . . . . :   04:01:08

Message . . . . :   DSPLY  CLIENTS
Cause . . . . . :   This is an inquiry message originated from RPG procedure
  CLIENTS in program MDRTUTLIB/CLIENTS. The program is expecting a character
  input field with a maximum length of 1.
```

```
Recovery  . . . :   Enter a valid response according to the type of data
  expected by the program.  The program will tolerate five incorrect responses
  before signalling an error condition.  There have been 0 incorrect responses
  so far.  If you do not want to enter any data, just press Enter to resume
  processing of the program.

                                                                      Bottom
Type reply below, then press Enter.
Reply  . . . .
```

Type any single character (e.g. “r”) on the above highlighted command line and press enter.

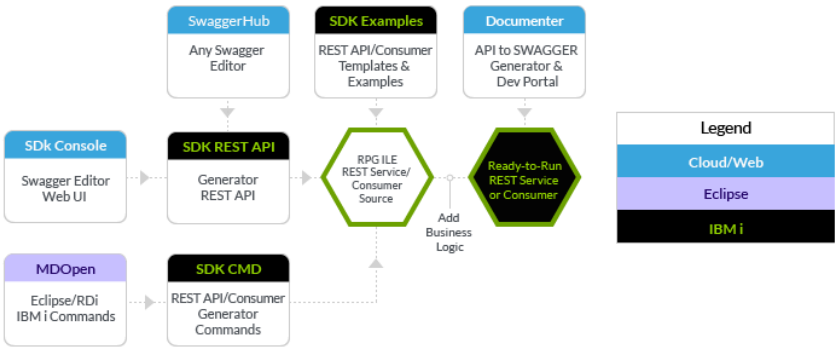
If the control reaches the breakpoint, which you have added, the program will be displayed in debug mode and then you can use function keys F10/F12 to control the execution of the program.

Once your debugging is complete, execute ENDBG and ENDSRVJOB on command line subsequently.

4 MDRest4i SDK (Software Development Kit)

4.1 Overview

MDREst4i SDK (Software Development Kit) is a set of tools that accelerate the creation, editing and documenting of REST services and consumers.



MDRest4i SDK generators combine parameters provided by the user and the master templates, to build consumer and Provider (service) RPGLE source code.

MDRESt4i SDK has three types of code generators available:

- MDRGENPRD – IBM i command for creating a basic REST service/Provider
- MDRGENCNS – IBM I command for creating a basic REST consumer/client
- MDRGENXAPI –REST Service that accepts SWAGGER to create an advanced REST service/consumer
- MDRGNAPI – REST Service that accepts SWAGGER to create an advanced REST service/consumer. This

API generates 2 types service/consumer, where first generate with all in one pgm and second is with procedures externalized.

These compiled source members are self-contained REST consumers or services, which a developer can then add all the necessary business logic to make the service/consumer fully functional.

Here we will use a built in commands supplied with MDRest4i SDK to build REST APIs(Provider) or REST Clients(Consumer).

This basic approach can be adapted as a starting to point to build virtually any REST API/Client with any or multiple methods. The MDRest4i SDK web UI is designed to allow SWAGGER to be used to design and then convert the SWAGGER definition into a REST API or consumer, using a REST API built with MDRest4i. This is covered in section 4.4 below

4.2 Command MDRGENPRD – Provider/Service Examples

This section of the tutorial covers 9 examples of using the IBM i Command MDRGENPRD to create, edit and run REST services.

Note: For the whole of section 4.2 have MDRST in your library list

4.2.1 GET Method Service - HelloWorld example

Let's build a very simple example of a REST service that accepts parameters from a browser request and sends back a JSON response resource.

First of all prompt the command MDRST/MDRGENPRD from the command line using below parameters:

```
                  MDRest4i Generator - Provider (MDRGENPRD)

 Type choices, press Enter.

 Target Library . . . . . . . . . > MDRTUTLIB     Name
Target Source File . . . . . . .   QRPGLESRC     Name
Target Source Member . . . . . . > HELLOWORLD    Name
Member Text  . . . . . . . . . .   'REST Pgm Generated by MDRest4i'

Get Method Required? . . . . . > Y             Y=Yes , N-No
Put Method Required? . . . . . .  N             Y=Yes , N-No
Post Method Required?  . . . . .  N             Y=Yes , N-No
Patch Method Required? . . . . .  N             Y=Yes , N-No
Delete Method Required?  . . . .  N             Y=Yes , N-No
Options Method Required? . . . .  N             Y=YES , N-NO
Data Format  . . . . . . . . . .  J             J=JSON, X-XML
```

- Now select the method using "Y"(as per the required HTTP method) . Here we are going to use the GET method in our API so we are selecting "Y" in Get method option. We can create the API in XML also by selecting Data Format "X". Otherwise default set as "J" for JSON. After this press enter to see the next parameters.

```
Response Processing Method . . .   USR           IFS, DBF, USR
```

- Now here are three provisions, "IFS" for reading response from the IFS file, "DBF" for reading response from MDRJSONF DB file and the last is "USR" which will read logic inside the API for sending response to client. "USR" set as default. After that, press enter to see the next parameters.

```
Processing Type  . . . . . . . .   N             D=DB, C-Call, N-None
```

- We have three provisions as well. If you want to process the DB file then select "D", for calling program, module and service program, select "C" and last is "N" in case you don't want to process any DB file or any external program. The default is "N". Press enter to see next parameters.

```
Parm Required? . . . . . > Y              Y=Parm Reqd, N-Parm Not Reqd
```

- If query parameters are required, enter "Y" on "parm required" and press enter. Press Page down key to enter the parameters.

```
List of parameters:
  Parameter Name . . . . . > name
  Mandatory Y/N? . . . . . > Y              Character value
  Array Y/N? . . . . . > N                  Character value
  Array Dim  . . . . . >                    1-99
  Length . . . . . > 30                     1-999999
  Data Type  . . . . . > A                  A, B, D, F, G, I, N, P, S...
  Decimal Positions  . . . . .              0-63

  Parameter Name . . . . . > title
Mandatory Y/N? . . . . . > N              Character value
Array Y/N? . . . . . > N                  Character value
Array Dim  . . . . . >                    1-99
Length . . . . . > 5                      1-999999
Data Type  . . . . . > A                  A, B, D, F, G, I, N, P, S...
Decimal Positions  . . . . .              0-63
            + for more value
s
```

In target library we will specify the library name for the source file where the member will be generated.

In target source file we will specify the source file name where the API source will be generated.

In target source member we will specify the source member name that will be generated.

In this example, we have used the HTTP GET method, so below parameter should be set "Y".

```
Get Method Required? . . . . .   Y
```

In this example, we don't want to read response through the MDRJSONF or IFS file, so we have used USR method:

```
Response Processing Method . . . . . .    USR                          IFS, DBF, USR
```

Here we don't want to process any DB file or any external program. That's why we are creating API by using "N".

```
Processing Type  . . . . . .   N            D=DB, C-Call, N-None
```

Now we want to pass parameters to the API, so parameter below should be set "Y".

```
Parm Required? . . . . . > Y
```

Here we are supplying two parameters "name" & "title" where "name" is mandatory and "title" is an optional parameter. We can create the parameters with the array also, for this we need to set "Array Y/N" is "Y". After that, we can supply the array dimension in the "Array Dim" otherwise it will create array with the dimension of 5 and if we will provide the value in the "Array Dim" parm, it will create array param with the supplied dimension.

```
List of parameters:
  Parameter Name . . . . . > name
  Mandatory Y/N? . . . . . > Y              Character value
```

```
Array Y/N? . . . . . . . . . . > N            Character value
Array Dim  . . . . . . . . . . >              1-99
Length . . . . . . . . . . . . > 50           1-999999
Data Type  . . . . . . . . . . > A            A, B, D, F, G, I, N, P, S...
Decimal Positions  . . . . . .                0-63

Parameter Name . . . . . . . . > title
Mandatory Y/N? . . . . . . . . > N            Character value
Array Y/N? . . . . . . . . . . > N            Character value
Array Dim  . . . . . . . . . . >              1-99
Length . . . . . . . . . . . . > 5            1-999999
Data Type  . . . . . . . . . . > A            A, B, D, F, G, I, N, P, S...
Decimal Positions  . . . . . .                0-63
```

Once the command completes, the source member below is generated under the specified source file and library combinations and is also compiled into the library. The name of the program is HELLOWORLD.

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0003.00 h bnddir('LXRGLOBAL':'BNDZIP')
0004.00  * ©=2019 Midrange Dynamics=============================================
0005.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0006.00  * ©=2019 Midrange Dynamics=============================================
0007.00  * h decedit(',')
0008.00  * ©=2019=Midrange Dynamics=============================================
0009.00  * ©=2019=Midrange Dynamics==========================================
0010.00  * Create Date :
0011.00  * Created By  : Midrange Dynamics
0012.00  * Description : This is the MDRest4i RESTfull Webservice
0013.00  * Service Type: RESTfull Webservice
0014.00  * Input       : GET Request - URL PARMS
0015.00  * Output      : Error/warning - JSON Component
0016.00  * ©=2019=Midrange Dynamics==========================================
0017.00
0018.00  // Standard MDRest4i Copybooks for REST Service
0019.00  /copy qrpglesrc,mdrusercpy
0020.00
0021.00  // Variable Definitions
0022.00 d w_string        s          20480
0023.00 d w_str2          s          20480
0024.00
0025.00  // Write query parameter definitions
0026.00 d  p_name         s             50A
0027.00 d  n_name         s               n
0028.00 d  p_title        s              5A
0029.00 d  n_title        s               n
0030.00
0031.00  // Indicate subroutine overrides defined locally
0032.00  /define LXR_CustomInit
0033.00
0034.00  /define LXR_CheckParms
0035.00  /define LXR_SetMethod
0036.00  /define LXR_SetParms
0037.00  /define LXR_ProcGET
0038.00
0039.00  /copy qrpglesrc,mdrestdfn
0040.00  /copy qrpglesrc,lxrrestc
0041.00
0042.00  /undefine LXR_CustomInit
0043.00
0044.00  /undefine LXR_CheckParms
0045.00  /undefine LXR_SetMethod
0046.00  /undefine LXR_SetParms
0047.00  /undefine LXR_ProcGET                                    0059.00
0048.00
0049.00  /Free
0050.00    // ===========================================================================
```

```
0051.00    //   Customized initialization
0052.00    // ===================================================================
0053.00    Begsr Z_CustomInit;
0054.00
0055.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0056.00      // set w_maxrsplen to the expected maximum response length
0057.00      // uncomment the statement below and set the value as required
0058.00      // w_maxrsplen =15542880;
0059.00
0060.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0061.00      // set w_maxreqlen to the expected maximum request body length
0062.00      // w_maxreqlen =15542880;
0063.00
0064.00      // If request or response value is greater than 16MB,
0065.00      // declare the keyword "alloc(*teraspace)" in control spec of this pro
0066.00
0067.00      // In order to determine the actual request/response size, set the
0068.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0069.00      // The REST API will then only return the request/response size.
0070.00      // The indicator "n_getrspsize" will allocate 500MB memory from heap
0071.00      // to process the reqeust and return the response size.
0072.00      // If the expected response size is much lesser, set the max expected
0073.00      // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0074.00
0075.00      // Use this IFS switch to log the API data in an IFS Folder
0076.00      // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0077.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
0078.00      n_saveIFSSwitch = *off;
0079.00
0080.00      // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0081.00      // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0082.00      ng_mdrJobHdr = *off;
0083.00
0084.00      // Extracts request query parameters to d_Qparm data structure
0085.00      n_extractQryPrms = *off;
0086.00
0087.00    Endsr;
0088.00
0089.00    // ===================================================================
0090.00    //    Check and Populate Parms  - Empty Declaration
0091.00    // ===================================================================
0092.00    Begsr z_checkParms;
0093.00
0094.00      if %trim(d_parm(1).s_value) <>  '*notFound';
0095.00        p_name = %trim(d_parm(1).s_value);
0096.00        n_name = *On;
0097.00      endif;
0098.00      if %trim(d_parm(2).s_value) <>  '*notFound';
0099.00        p_title = %trim(d_parm(2).s_value);
0100.00        n_title = *On;
0101.00      endif;
0102.00
0103.00    Endsr;
0104.00    // ===================================================================
0105.00    //    Populate Required Parameters List
0106.00    // ===================================================================
0107.00    Begsr z_setParms;
0108.00      // Populate the List of Expected Parms:
0109.00      d_parm(1).s_Name= 'name';
0110.00      d_parm(1).s_Required= *on  ;
0111.00      d_parm(2).s_Name= 'title';
0112.00      d_parm(2).s_Required= *off ;
0113.00    Endsr;
0114.00    // ===================================================================
0115.00    //    Populate Parms Required RETURN (HTTP TYPE - OPTION)
0116.00    // ===================================================================
```

```
0117.00    begsr z_setMethod;
0118.00      // Required and Optional Parameters:
0119.00      beginObject(' ');
0120.00      addChar('Get':'Mandatory Parms: name ::Optional Parms: title');
0121.00      endObject(' ');
0122.00      n_ParmError = *on;
0123.00    Endsr;
0124.00    // ========================================================================
0125.00    //    Override Get Method
0126.00    // ========================================================================
0127.00    Begsr z_ProcGET;
0128.00
0129.00    // Logic from custom copybook
0130.00    // added by stu
0131.00
0132.00      // Write your database interaction logic here
0133.00      Exsr Z_PrcSndRsp;
0134.00
0135.00
0136.00    Endsr;
0137.00    // ========================================================================
0138.00    //   // Process Send Response
0139.00    // ========================================================================
0140.00    Begsr Z_PrcSndRsp;
0141.00
0142.00    Endsr;
0143.00  /End-Free
0144.00    // ========================================================================
0145.00    //   LXR Generic Procedures
0146.00    // ========================================================================
0147.00  /copy qrpglesrc,lxrrestp
```

Now we will manually add logic to in z_PROCGET subroutine as well as specifying the JSON response. To add your own logic to the generated program for the GET method, the default LXR_PROCGET subroutine in the copybook was overridden during the generation, because the GET method was selected. For this a compiler directive "/define-/undefine" is used. It is important to note that this "/define LXR_PROCGET" is added ***before*** the "/copy LXRRESTC", and "/undefine LXR_PROCGET" is added ***after*** the "/copy LXRRESTC". Otherwise the compiler will include the subroutine defined in LXRREST copybook and therefore it will cause compilation failure.

The source below contains the original generated source, ***plus*** the code added manually (highlighted in red).

Comments (highlighted in green below) were added for better understanding of this REST GET service example.

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0003.00 h bnddir('LXRGLOBAL':'BNDZIP')
0004.00  * ©=2019 Midrange Dynamics=============================================
0005.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0006.00  * ©=2019 Midrange Dynamics=============================================
0007.00  * h decedit(',')
0008.00  * ©=2019=Midrange Dynamics=============================================
0009.00  * ©=2019=Midrange Dynamics==========================================
0010.00  * Create Date :
0011.00  * Created By  : Midrange Dynamics
0012.00  * Description : This is the MDRest4i RESTfull Webservice
0013.00  * Service Type: RESTfull Webservice
0014.00  * Input       : GET Request - URL PARMS
0015.00  * Output      : Error/warning - JSON Component
0016.00  * ©=2019=Midrange Dynamics=========================================
0017.00
0018.00  // Standard MDRest4i Copybooks for REST Service
0019.00  /copy qrpglesrc,mdrusercpy
0020.00
0021.00  // Variable Definitions
```

```
0022.00 d w_string        s          20480
0023.00 d w_str2          s          20480
0023.00 d w_message       s            100
0024.00
0025.00  // Write query parameter definitions
0026.00 d  p_name         s             50A
0027.00 d  n_name         s               n
0028.00 d  p_title        s              5A
0029.00 d  n_title        s               n
0030.00
0031.00  // Indicate subroutine overrides defined locally
0032.00  /define LXR_CustomInit
0033.00
0034.00  /define LXR_CheckParms
0035.00  /define LXR_SetMethod
0036.00  /define LXR_SetParms
0037.00  /define LXR_ProcGET
0038.00
0039.00  /copy qrpglesrc,mdrestdfn
0040.00  /copy qrpglesrc,lxrrestc
0041.00
0042.00  /undefine LXR_CustomInit
0043.00
0044.00  /undefine LXR_CheckParms
0045.00  /undefine LXR_SetMethod
0046.00  /undefine LXR_SetParms
0047.00  /undefine LXR_ProcGET
0048.00
0049.00  /Free
0050.00    // ====================================================================
0051.00    //   Customized initialization
0052.00    // ====================================================================
0053.00    Begsr Z_CustomInit;
0054.00
0055.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0056.00      // set w_maxrsplen to the expected maximum response length
0057.00      // uncomment the statement below and set the value as required
0058.00      // w_maxrsplen =15542880;
0059.00
0060.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0061.00      // set w_maxreqlen to the expected maximum request body length
0062.00      // w_maxreqlen =15542880;
0063.00
0064.00      // If request or response value is greater than 16MB,
0065.00      // declare the keyword "alloc(*teraspace)" in control spec of this pro
0066.00
0067.00      // In order to determine the actual request/response size, set the
0068.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0069.00      // The REST API will then only return the request/response size.
0070.00      // The indicator "n_getrspsize" will allocate 500MB memory from heap
0071.00      // to process the reqeust and return the response size.
0072.00      // If the expected response size is much lesser, set the max expected
0073.00      // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0074.00
0075.00      // Use this IFS switch to log the API data in an IFS Folder
0076.00      // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0077.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
0078.00      n_saveIFSSwitch = *off;
0079.00
0080.00      // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0081.00      // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0082.00      ng_mdrJobHdr = *on;
0083.00
0084.00      // Extracts request query parameters to d_Qparm data structure
0085.00      n_extractQryPrms = *off;
0086.00
```

```
0087.00
Endsr;
0089.00
0090.00    // ===================================================================
0091.00    //    Check and Populate Parms  - Empty Declaration
0092.00    // ===================================================================
0093.00    Begsr z_checkParms;
0094.00
0094.01      // In this subroutine, we are checking if the specific query parameter
0094.02      // is supplied and if so, set the value in that parameter and the
0094.03      // corresponding indicator is set *On
0094.04
0095.00      if %trim(d_parm(1).s_value) <>  '*notFound';
0096.00        p_name = %trim(d_parm(1).s_value);
0097.00        n_name = *On;
0098.00      endif;
0099.00      if %trim(d_parm(2).s_value) <>  '*notFound';
0100.00        p_title = %trim(d_parm(2).s_value);
0101.00        n_title = *On;
0102.00      endif;
0103.00
0104.00    Endsr;
0105.00    // ===================================================================
0106.00    //    Populate Required Parameters List
0107.00    // ===================================================================
0108.00    Begsr z setParms;
0109.00      // Populate the List of Expected Parms:
0110.00      // In this case, we are expecting name (in small case) as the mandatory
0111.00      // parameters and title as the optional parameter. When both are supplied,
0112.00      // the query string will be like "../HELLOWORLD?name=Bob&title=Mr." but
0113.00      // if only mandatory parm is supplied, it is like ../HELLOWORLD?name=Bob
0114.00
0115.00      d parm(1).s Name= 'name';
0116.00      // The name is required and hence below indicator is being set as *On
0117.00      d_parm(1).s_Required= *on  ;
0118.00
0119.00      d parm(2).s Name= 'title';
0120.00      // The title is optional and hence below indicator is being set as *Off
0121.00      d_parm(2).s_Required= *off ;
0122.00
0123.00      // The REST service validates if the mandatory parameter (i.e. the one
0124.00      // with s_required indicator set to *On) is supplied or not. If it's
0125.00      // not supplied, the parameter error is sent as the response and no
0126.00      // further processing happens
0127.00    Endsr;
0128.00    // ===================================================================
0129.00    //    Populate Parms Required RETURN (HTTP TYPE - OPTION)
0130.00    // ===================================================================
0131.00    begsr z setMethod;
0132.00      // Required and Optional Parameters:
0133.00
0134.00      // This subroutine gets called when all the mandatory parameters are not
0135.00      // supplied. Below message will be displayed for this API but you can
0136.00      // give your own message that you like
0137.00
0138.00      beginObject(' ');
0139.00      addChar('Get':'Mandatory Parms: name ::Optional Parms: title');
0140.00      endObject(' ');
0141.00      n_ParmError = *on;
0142.00    Endsr;
0143.00    // ===================================================================
0144.00    //    Override Get Method
0145.00    // ===================================================================
0146.00    Begsr z_ProcGET;
0147.00
0148.00    // Logic from custom copybook
0149.00
0150.00
0151.00      // Write your database interaction logic here
0152.00      Exsr Z_PrcSndRsp;
0153.00
0154.00
0155.00    Endsr;
0156.00    // ===================================================================
```

```
0157.00    //   // Process Send Response
0158.00    // ====================================================================
0159.00    Begsr Z_PrcSndRsp;
0160.00
0161.00      // The control reached here means the mandatory parameters have been
0162.00      // supplied but we have to check if the optional parameters have been
0163.00      // supplied or not and accordingly write the logic to send response
0164.00
0165.00      // In this example, if the parameter "name" was supplied, the value
0166.00      // is set in "p name" variable and the indicator "n name" is set *On
0167.00      // Likewise, if "title" is supplied, the variables "p title" and
0168.00      // "n_title" are set.
0169.00
0170.00      if n_title = *On;
0171.00        w_message = 'Hello ' + %trim(p_title) + ' ' + %trim(p_name);
0172.00      else;
0173.00        // optional parameter not provided, use "user" as default value
0174.00        w_message = 'Hello user ' + %trim(p_name);
0175.00      endif;
0176.00
0177.00      // We have to first return HTTP headers and then the response
0178.00      // Response has to be sent in the format provided in http header.
0179.00      // By default, the content-type is set to "json" and therefore
0180.00      // the response has to be in JSON format
0181.00
0182.00      // Below function with blank parameter writes non-labeled opening
0183.00      // curly brace
0184.00
0185.00      beginObject(' ');
0186.00
0187.00      // Below function writes label value pair e.g. below
0188.00      // "message" : "Hello Mr. John"
0189.00
0190.00      addChar('message':%trim(w message));
0191.00
0192.00      endObject();
0193.00
0194.00      // No further processing is needed. The system will send your response
0195.00      // and exit the program.
0196.00    Endsr;
0197.00  /End-Free
0198.00    // ====================================================================
0199.00    //   LXR Generic Procedures
0200.00    // ====================================================================
0201.00  /copy qrpglesrc,lxrrestp
```

Now let's compile and call helloworld. Here are the example links to call hello world.

Paste the following URL into your browser:

http://youribmiserver:yourportnumber/yourlibrary/helloworld

You will get this response:

```
{"Get": "Mandatory Parms: name :: Optional Parms: title"}
```

The response above is produced by line 139.00

Now paste the following URL into your browser (note the case of parameters as we gave during API creation, the both parameters "name" and "title" was in lower case):

http://youribmiserver:yourportnumber/yourlibrary/helloworld?name=Bob&title=Mr

we get this response:

```
{
"message": "Hello Mr Bob"
}
```

The response above is produced by line 185.00 to 192.00 inclusive.

Let's explain how it all fits together.

You are provided with the following source members as templates:

LXRRESTC	MDRest4i Generic C Specs for Rest Services	Main routines and subroutines controlling the REST service execution
MDRESTDFN	MDRest4i Generic D Specs for Rest Services	Standard variable & prototype definitions for REST service
LXRRESTP	MDRest4i Generic P Specs for Rest Services	Standard procedures used in REST service

You are required to populate the following subroutines as needed:

- z_setParms– Populate the name of expected parameters as well as if they are mandatory. To override this subroutine, "/define LXR_setParms" is required before including LXRRESTC copybook.
- z_setMethod – Populate Services Methods and Requirement. To override this subroutine, "/define LXR_SetMethod" is required before including LXRRESTC copybook.
- z_Proc***() – The actual progam logic will reside in this Subroutine as well as the return output creation. To override these subroutines, appropriate define compiler directive is required.
- z_CustomInit – If you want to override this subroutine in your program for any initialization before the REST service starts running, add "/define LXR_CustomInit" before including LXRRESTC copybook. Then define z_CustomInit subroutine in the program to add any initialization of standard variables.

If you want to debug the service, add the statement with "/define trace" in your program before including MDRESTDFN copybook or initialize n_trace with *On in z_CustomInit subroutine. For more info please see Debugging the service in section 3.2 in the document above. Or look at the HTTP headers in the response in the browser. This can be viewed looking in Developer mode in Chrome browser for example, selecting the network tab, refresh the page, and looking at the response headers for the resource helloworld.

If you want to log the incoming http requests in an IFS file, initialize n_saveIFSSwitch indicator with *On in z_CustomInit subroutine. The log files will be created with the name "LXRLOGS_" followed by the system timestamp. The logs are by default created in the directory specified in DFTLOGDIR data area under MDRST library. If you want to change the IFS location, initialize the variable w_saveIFSPath with the directory path along with the file name(e.g. w_saveIFSPath = '/home/temp/LxrLogs.txt').

If you want the specific rest service should only be allowed for the relevant http method, override the other subroutines and write the statement "SetHttpStatus(405: 'Wrong method');" in that subroutine. The starting subroutines for the different requests are z_getMethod, z_postMethod, z_putMethod, z_patchMethod, z_delMethod, z_optionsMethod.

4.2.2 GET Method Service with Database reads

For this example we use the LXCLIENT table included in the library MDRST. Please copy this data table to your library [YOURLIB].

First of all prompt the command MDRST/MDRGENPRD from the command line and use below parameters to run the command:

```
MDRest4i Generator - Provider (MDRGENPRD)
```

```
Type choices, press Enter.

Target Library . . . . . > MDRTUTLIB     Name
Target Source File . . . . . QRPGLESRC     Name
Target Source Member . . . . . > CLIENTS       Name
Member Text  . . . . . 'REST Pgm Generated by MDRest4i'

Get Method Required? . . . . . > Y             Y=Yes , N-No
Put Method Required? . . . . . N             Y=Yes , N-No
Post Method Required?  . . . . . N             Y=Yes , N-No
Patch Method Required? . . . . . N             Y=Yes , N-No
Delete Method Required?  . . . . N             Y=Yes , N-No
Options Method Required? . . . . N             Y=YES , N-NO
Data Format  . . . . . J             J=JSON, X-XML
```

- Now select the method using "Y"(as per the required HTTP method) . Here we are going to use the GET method in our API so we are selecting "Y" in Get method option. We can create the API in XML also by selecting Data Format "X". Otherwise default set as "J" for JSON. After this press enter to see the next parameters.

```
Response Processing Method . . .   USR           IFS, DBF, USR
```

- Now here are three provisions, "IFS" for reading response from the IFS file, "DBF" for reading response from MDRJSONF DB file and the last is "USR" which will read logic inside the API for sending response to client. "USR" set as default. After that, Press enter to see the next parameters.

```
Processing Type  . . . . . > D             D=DB, C-Call, N-None
```

- We have three provisions as well. If you want to process the DB file then select "D", for calling program, module and service program, select "C" and last is "N" in case you don't want to process any DB file or any external program. The default is "N". Press enter to see next parameters.

```
Include Paging Logic?  . . . . .   N             Y=Include Paging, N-Not Reqd
```

- If we want to use paging logic in the API, select "Y" above Otherwise default is "N". Press page down key to see next parameters.

```
Default DB File  . . . . . > LXCLIENT      Name, *NONE
  Library  . . . . . > MDRST         Name, *LIBL
Parm Required? . . . . . > Y             Y=Parm Reqd, N-Parm Not Reqd
```

- Now provide the DB file name and library, select 'Y" in "parm required" and press enter to enter the parameters.

```
List of parameters:
  Parameter Name . . . . . > id
  Mandatory Y/N? . . . . . > N             Character value
              + for more values
```

In target library we will specify the library name where member would be generated.

In target source file we will specify the source file name.

In target source member we will specify the source member name.

In this example, we have used GET method, so below parameter should be set "Y".

```
Get Method Required? . . . . . Y
```

In this example, we don't want to read response through the MDRJSONF or IFS file so, we have used USR method:

```
Response Processing Method . .    USR                          IFS, DBF, USR
```

To use the database I/O, the below parameter should be set "D".

```
Processing Type  . . . . . . . . > D             D=DB, C-Call, N-Not Reqd
```

Here we are supplying database details.

```
Default DB File  . . . . . . . . > LXCLIENT      Name, *NONE
  Library  . . . . . . . . . . . > MDRST         Name, *LIBL
```

If we want to have the parameters in API, then below parameter should be set "Y".

```
Parm Required? . . . . . . . . . > Y
```

Here we are supplying one parameter "id" with mandatory option "N". It means it's an optional parameter.

```
List of parameters:
 Parameter Name . . . . . . . . > id
 Mandatory Y/N? . . . . . . . . > N             Character value
```

Once the command completes the execution, the source member below is generated under the specified source file and library combinations and it gets compiled as well. The name of the program is CLIENTS.

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0004.00 h bnddir('LXRGLOBAL':'BNDZIP')
0005.00  * ©=2019 Midrange Dynamics=============================================
0006.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0007.00  * ©=2019 Midrange Dynamics=============================================
0008.00  * h decedit(',')
0009.00  * ©=2019=Midrange Dynamics=============================================
0010.00  * ©=2019=Midrange Dynamics==========================================
0011.00  * Create Date :
0012.00  * Created By  : Midrange Dynamics
0013.00  * Description : This is the MDRest4i RESTfull Webservice
0014.00  * Service Type: RESTfull Webservice
0015.00  * Input       : GET Request - URL PARMS
0016.00  * Output      : Error/warning - JSON Component
0017.00  * ©=2019=Midrange Dynamics==========================================
0018.00
0019.00  // Standard MDRest4i Copybooks for REST Service
0020.00  /copy qrpglesrc,mdrusercpy
0021.00
0022.00  // Variable Definitions
0023.00 d w_string        s          20480
0024.00 d w_str2          s          20480
0025.00  // Define Work Variables for SQL building
0026.00 d                 ds
0027.00 d   s_statement                1024    varying
0028.00 d   s_statement1               1024    overlay(s_statement:+3)
0029.00 d   s_firstRec                    n
0030.00 d w_stmtcnt       s            200
0032.00  // Custom Variables for response schema
0033.00 d d_S             ds                   qualified
0034.00 d lx_id                          9B 0
```

```
0035.00 d lx_clidno                      15A
0036.00 d lx_clname                      30A
0037.00 d lx_clsurname...
0038.00 d                                30A
0039.00 d lx_cltitle                     10A
0040.00 d lx_clphone                     15A
0041.00 d lx_clemail                    100A
0042.00 d lx_claddr1                     30A
0043.00 d lx_claddr2                     30A
0044.00 d lx_claddr3                     30A
0045.00 d lx_clpcode                     15A
0046.00 d lx_cllang                       1A
0047.00
0048.00 d w_Count         s              10i 0
0049.00
0050.00   // Write query parameter definitions
0051.00 d  p_id           s               9B 0
0052.00 d  n_id           s                n
0053.00
0054.00  // Indicate subroutine overrides defined locally
0055.00  /define LXR_CustomInit
0056.00
0057.00  /define LXR_CheckParms
0058.00  /define LXR_SetMethod
0059.00  /define LXR_SetParms
0050.00  /define LXR_ProcGET
0061.00
0062.00  /copy qrpglesrc,mdrestdfn
0063.00  /copy qrpglesrc,lxrrestc
0064.00
0065.00  /undefine LXR_CustomInit
0066.00
0067.00  /undefine LXR_CheckParms
0068.00  /undefine LXR_SetMethod
0069.00  /undefine LXR_SetParms
0070.00  /undefine LXR_ProcGET
0071.00
0072.00  /Free
0073.00    // ====================================================================
0074.00    //   Customized initialization
0075.00    // ====================================================================
0076.00    Begsr Z_CustomInit;
0077.00
0078.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0079.00      // set w_maxrsplen to the expected maximum response length
0080.00      // uncomment the statement below and set the value as required
0081.00      // w_maxrsplen =15542880;
0082.00
0083.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0084.00      // set w_maxreqlen to the expected maximum request body length
0084.00      // w_maxreqlen =15542880;
0086.00
0087.00      // If request or response value is greater than 16MB,
0089.00      // declare the keyword "alloc(*teraspace)" in control spec of this pro
0090.00
0091.00      // In order to determine the actual request/response size, set the
0091.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0092.00      // The REST API will then only return the request/response size.
0093.00      // The indicator "n_getrspsize" will allocate 500MB memory from heap
0094.00      // to process the reqeust and return the response size.
0095.00      // If the expected response size is much lesser, set the max expected
0096.00      // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0097.00
0097.00      // Use this IFS switch to log the API data in an IFS Folder
0098.00      // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0099.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
```

```
0100.00       n_saveIFSSwitch = *off;
0101.00
0102.00       // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0103.00       // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0104.00       ng_mdrJobHdr = *off;
0105.00
0106.00       // Extracts request query parameters to d_Qparm data structure
0107.00       n_extractQryPrms = *off;
0108.00
0109.00     Endsr;
0103.00
0104.00     // ====================================================================
0105.00     //    Check and Populate Parms  - Empty Declaration
0106.00     // ====================================================================
0107.00     Begsr z_checkParms;
0108.00
0109.00       if %trim(d_parm(1).s_value) <>  '*notFound';
0110.00         p_id = %dec(%trim(d_parm(1).s_value):9:0);
0111.00         n_id = *On;
0112.00       endif;
0113.00
0114.00     Endsr;
0115.00     // ====================================================================
0116.00     //    Populate Required Parameters List
0117.00     // ====================================================================
0118.00     Begsr z_setParms;
0119.00       // Populate the List of Expected Parms:
0120.00       d_parm(1).s_Name= 'id';
0121.00       d_parm(1).s_Required= *off ;
0122.00     Endsr;
0123.00     // ====================================================================
0124.00     //    Populate Parms Required RETURN (HTTP TYPE - OPTION)
0125.00     // ====================================================================
0126.00     begsr z_setMethod;
0127.00       // Required and Optional Parameters:
0128.00       beginObject(' ');
0128.00       addChar('Get':'Optional Parms: id');
0128.00       endObject(' ');
0129.00       n_ParmError = *on;
0130.00     Endsr;
0131.00     // ====================================================================
0132.00     //    Override Get Method
0133.00     // ====================================================================
0134.00     Begsr z_ProcGET;
0135.00
0136.00     // Logic from custom copybook
0137.00
0138.00       Exsr z_PrcResponse;
0139.00
0140.00     Endsr;
0141.00     // ====================================================================
0142.00     //    Process Response
0143.00     // ====================================================================
0144.00     Begsr z_PrcResponse;
0145.00
0146.00       n_BgnKey = *Off;
0147.00       clear d_S;
0148.00        // Main SQL statement for cursors
0149.00       s_statement =
0150.00       'select ID, CLIDNO, CLNAME, CLSURNAME, CLTITLE, CLPHONE, CLEMAIL,'+
0151.00       ' CLADDR1, CLADDR2, CLADDR3, CLPCODE, CLLANG from LXCLIENT';
0152.00
0153.00       w_stmtcnt = 'select count(*) from LXCLIENT';
0154.00
0155.00        // Below is an optional parameter in query string
0156.00       if n_id = *On;
```

```
0157.00         s_statement = %trim(s_statement) + ' where ID = '+ %char(p_id);
0158.00
0159.00         w_stmtcnt = %trim(w_stmtcnt) + ' where ID = '+ %char(p_id);
0160.00       endif;
0161.00
0162.00        // Start preparing cursors
0163.00       exec sql prepare w_stmtcnt from :w_stmtcnt;
0164.00
0165.00       exec sql prepare s_statement1 from :s_statement1;
0166.00
0167.00       if sqlstt <> c_SqlSuccess;
0168.00         ErrorJSON('No data to return');
0169.00         leavesr;
0170.00       endif;
0171.00
0172.00       // Process SQL data
0173.00       exec sql declare c2_curser cursor for w_stmtcnt;
0174.00       exec sql open c2_curser;
0175.00       // Load the count of records
0176.00       exec sql fetch c2_curser into :w_count;
0177.00       exec sql declare c1_curser cursor for s_statement1;
0178.00       exec sql open c1_curser;
0179.00       // Prepare an array for multiple record selection
0180.00       exec sql fetch c1_curser into :d_S;
0181.00       if sqlstt = c_SqlSuccess or sqlstt < c_SqlEOF;
0182.00         // Build Response Container
0183.00         if w_count > 1;
0184.00           beginObject(' ');
0185.00           beginArray('LXCLIENT');
0186.00         endif;
0187.00       else;
0188.00         ErrorJSON('No data to return');
0189.00         leavesr;
0291.00       endif;
0292.00
0293.00       dow sqlstt < c_SqlEOF;
0294.00         Exsr Z_ProcessSend;
0295.00         exec sql fetch c1_curser into :d_S;
0295.00       enddo;
0296.00       if w_count > 1;
0297.00         endArray();
0298.00         endObject();
0299.00       endif;
0200.00
0201.00       // Close the cursors
0202.00       exec sql close c1_curser;
0203.00       exec sql close c2_curser;
0204.00
0205.00     Endsr;
0206.00     // ===================================================================
0207.00     //    Process the DB file
0208.00     // ===================================================================
0209.00     Begsr Z_ProcessSend;
0210.00
0211.00       beginObject(' ');
0212.00
0213.00       // Execute any data manipulations here
0213.00
0214.00       // Add a JSON object for each field from IO/SQL
0215.00       addIntr('id' : d_s.lx_id);
0216.00       addchar('clidno': %trim(d_s.lx_clidno));
0217.00       addchar('clname': %trim(d_s.lx_clname));
0218.00       addchar('clsurname': %trim(d_s.lx_clsurname));
0219.00       addchar('cltitle': %trim(d_s.lx_cltitle));
0200.00       addchar('clphone': %trim(d_s.lx_clphone));
0201.00       addchar('clemail': %trim(d_s.lx_clemail));
```

```
0202.00       addchar('claddr1': %trim(d_s.lx_claddr1));
0203.00       addchar('claddr2': %trim(d_s.lx_claddr2));
0204.00       addchar('claddr3': %trim(d_s.lx_claddr3));
0205.00       addchar('clpcode': %trim(d_s.lx_clpcode));
0206.00       addchar('cllang': %trim(d_s.lx_cllang));
0211.00       endObject();
0213.00     Endsr;
0214.00   /End-Free
0215.00     // ========================================================================
0216.00     //   LXR Generic Procedures
0217.00     // ========================================================================
0218.00   /copy qrpglesrc,lxrrestp
```

Colour Legend:

Orange	Header Specifications
Green	Copybooks for the MDRest4i engines.
Blue	Work field definitions on the basis of the File Fields along with prefix lx_.
DarkRed	See Z_PrcResponse Routine below
Purple	See Z_PrcResponse Routine below
Pink	See Z_PrcResponse Routine below
Brown	See Z_ProcessSend Routine below
Royal Blue	See Z_SetParms Routine below

Z_PrcResponse Routine: In this routine we fetch the data by building an SQL query based on the parameters supplied in the command. If no values are supplied in the parameters of the command, then the API will return the whole data set based upon the SQL query. In the above example, we are using one parameter and that is optional. So, there are two possibilities (e.g. one parameter/ no parameters) and accordingly the query is first created without any "where" clause as highlighted in dark red. Then, there is a condition to check if the parameter is supplied or not and accordingly add the "where" clause in SQL query as highlighted in purple. To validate the file has multiple records to be returned via JSON array, the command likewise writes the code to count the records via sq. query as highlighted in pink color.

Z_ProcessSend Routine: Please refer the highlighted brown color section through which we write the response (Add a JSON object for each field obtained via SQL I/O).

Z_SetParms Routine: Since we are using the optional parameter, so the required indicator has been set *Off, please refer the highlighted royal blue color.

4.2.3 POST Method Service with generated logic for database Updates

Let's create an example program to parse a request Body from a POST request as well as writing a response. The command will create the member, write the data structure definition with all the necessary fields as the subfields of the DS. The processing logic will fetch the value in each subfield from the request body, write them to the database file and then send the response.

First of all prompt the command MDRST/MDRGENPRD from the command line and use the parameters below to run the command:

```
                  MDRest4i Generator - Provider (MDRGENPRD)

Type choices, press Enter.

Target Library . . . . . . . . . > MDRTUTLIB     Name
Target Source File . . . . . . .   QRPGLESRC     Name
Target Source Member . . . . . . > CLIENTPST    Name
Member Text  . . . . . . . . . .   'REST Pgm Generated by MDRest4i'

Get Method Required? . . . . . .   N             Y=Yes , N-No
Put Method Required? . . . . . .   N             Y=Yes , N-No
Post Method Required?  . . . . . > Y             Y=Yes , N-No
Patch Method Required? . . . . .   N             Y=Yes , N-No
Delete Method Required?  . . . .   N             Y=Yes , N-No
Options Method Required? . . . .   N             Y=YES , N-NO
Data Format  . . . . . . . . . .   J             J=JSON, X-XML
```

- Now select the method using “Y”(as per the required HTTP method) . Here we are going to use the POST method in our API so we are selecting “Y” in Post method option. We can create the API in XML also by selecting Data Format “X”. Otherwise default set as “J” for JSON. After this press enter to see the next parameters.

```
Requestbody Parse Method . . . .   USR           IFS, DBF, USR
```

- Now here are three provisions, “IFS” for writing JSON request to the IFS file, “DBF” for writing JSON request to MDRJSONF DB file and the last is “USR” which will add logic inside the API. “USR” set as default. After that, Press enter to see the next parameters.

```
Response Processing Method . . .   USR           IFS, DBF, USR
```

- Now here are three provisions, “IFS” for reading response from the IFS file, “DBF” for reading response from MDRJSONF DB file and the last is “USR” which will read logic inside the API for sending response to client. “USR” set as default. After that, Press enter to see the next parameters.

```
Processing Type  . . . . . . . . > D             D=DB, C-Call, N-None
```

- Three provisions here as well. If you want to process the DB file then select “D”, for calling program, module and service program, select “C” and last is “N” in case you don’t want process any DB file or any external program. The default is “N”. Press enter to see next parameters.

```
Include Paging Logic?  . . . . .   N             Y=Include Paging, N-Not Reqd
```

- Page down to see the remaining parameters.

```
Default DB File  . . . . . . . . > LXCLIENT      Name, *NONE
  Library  . . . . . . . . . . . > MDRST         Name, *LIBL
Parm Required? . . . . . . . . .   N             Y=Parm Reqd, N-Parm Not Reqd
```

In the target library we will specify the library name where member would be generated.

In the target source file we will specify the source file name.

In the target source member we will specify the source member name.

In this example, we have used the HTTP POST method, so parameter below should be set “Y”.

```
Post Method Required?  . . . . . > Y            Y=Yes , N-No
```

In this example, we don't want to write request body in the IFS file or DB file, so we are using option USR

```
Requestbody Parse Method . . . .   USR          IFS, DBF, USR
```

In this example we don't want to send response from IFS file or DB file, so we are using option "USR" method:

```
Response Processing Method . . .   USR          IFS, DBF, USR
```

We then set "Include DBF I/O" as "D" and press enter. It will open another parameter "Default DB File" and "Library". We entered the file name as "LXCLIENT" and library as "MDRST". This setting means the generated POST API would expect the incoming request body having single record with all the fields of the DB file where the record would be written and response sent.

```
Processing Type  . . . . . > D            D=DB, C-Call, N-None
Default DB File  . . . . . > LXCLIENT
  Library  . . . . . >   MDRST
```

Once the command execution completes, the source member below is generated under the specified source file and library combinations, and the program will be compiled into the target library. The name of the program is CLIENTPST.

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0004.00 h bnddir('LXRGLOBAL':'BNDZIP')
0005.00  * ©=2019 Midrange Dynamics=============================================
0006.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0007.00  * ©=2019 Midrange Dynamics=============================================
0008.00  * h decedit(',')
0009.00  * ©=2019=Midrange Dynamics=============================================
0010.00  * ©=2019=Midrange Dynamics==========================================
0011.00  * Create Date :
0012.00  * Created By  : Midrange Dynamics
0013.00  * Description : This is the MDRest4i RESTfull Webservice
0014.00  * Service Type: RESTfull Webservice
0015.00  * Input       : GET Request - URL PARMS
0016.00  * Output      : Error/warning - JSON Component
0017.00  * ©=2019=Midrange Dynamics==========================================
0018.00
0019.00  // Standard MDRest4i Copybooks for REST Service
0020.00  /copy qrpglesrc,mdrusercpy
0031.00
0032.00  // Variable Definitions
0033.00 d w_string        s          20480
0034.00 d w_str2          s          20480
0035.00  // Define Work Variables for SQL building
0036.00 d                 ds
0037.00 d   s_statement                1024    varying
0038.00 d   s_statement1               1024    overlay(s_statement:+3)
0039.00 d   s_firstRec                    n
0040.00 d w_stmtcnt       s            200
0042.00  // Custom Variables for response schema
0043.00 d d_S             ds                   qualified
0044.00 d lx_id                          9B 0
0044.00 d lx_clidno                     15A
0045.00 d lx_clname                     30A
0046.00 d lx_clsurname...
```

```
0047.00 d                                30A
0048.00 d lx_cltitle                     10A
0049.00 d lx_clphone                     15A
0050.00 d lx_clemail                    100A
0051.00 d lx_claddr1                     30A
0052.00 d lx_claddr2                     30A
0053.00 d lx_claddr3                     30A
0054.00 d lx_clpcode                     15A
0055.00 d lx_cllang                       1A
0056.00 d w_Count         s              10i 0
0057.00
0058.00   // Write query parameter definitions
0059.00
0060.00  // Indicate subroutine overrides defined locally
0061.00  /define LXR_CustomInit
0062.00
0063.00  /define LXR_ProcPost
0064.00
0065.00  /copy qrpglesrc,mdrestdfn
0065.00  /copy qrpglesrc,lxrrestc
0066.00
0067.00  /undefine LXR_CustomInit
0068.00
0069.00  /undefine LXR_ProcPost
0070.00
0071.00  /Free
0072.00    // ====================================================================
0073.00    //   Customized initialization
0074.00    // ====================================================================
0075.00    Begsr Z_CustomInit;
0076.00
0067.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0068.00      // set w_maxrsplen to the expected maximum response length
0069.00      // uncomment the statement below and set the value as required
0070.00      // w_maxrsplen =15542880;
0071.00
0072.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0073.00      // set w_maxreqlen to the expected maximum request body length
0074.00      // w_maxreqlen =15542880;
0075.00
0076.00      // If request or response value is greater than 16MB,
0077.00      // declare the keyword "alloc(*teraspace)" in control spec of this pro
0078.00
0079.00      // In order to determine the actual request/response size, set the
0080.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0081.00      // The REST API will then only return the request/response size.
0082.00      // The indicator "n_getrspsize" will allocate 500MB memory from heap
0083.00      // to process the reqeust and return the response size.
0084.00      // If the expected response size is much lesser, set the max expected
0085.00      // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0086.00
0087.00      // Use this IFS switch to log the API data in an IFS Folder
0088.00      // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0089.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
0090.00      n_saveIFSSwitch = *off;
0091.00
0092.00      // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0093.00      // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0094.00      ng_mdrJobHdr = *off;
0095.00
0096.00      // Extracts request query parameters to d_Qparm data structure
0097.00      n_extractQryPrms = *off;
0098.00
0099.00    Endsr;
0105.00
0106.00    // ====================================================================
```

```
0107.00    //    Override Post Method
0108.00    // ========================================================================
0109.00    Begsr z_ProcPost;
0110.00
0111.00    // Logic from custom copybook
0112.00
0113.00      Exsr z_LoadReqBody;
0114.00      Exsr z_PrcDBWrite;
0115.00      Exsr z_PrcPostRsp;
0116.00
0117.00    Endsr;
0118.00    // ========================================================================
0119.00    //    Process Request Body
0120.00    // ========================================================================
0121.00    Begsr z_LoadReqBody;
0122.00
0123.00      // Load the reqeust body
0124.00      clear d_S;
0125.00
0126.00      // Add an statement for each expected DB Field
0128.00      d_s.lx_id = JPathN('id');
0128.00      d_s.lx_clname = JPathV ('clname');
0129.00      d_s.lx_clsurname = JPathV ('clsurname');
0130.00      d_s.lx_cltitle = JPathV ('cltitle');
0131.00      d_s.lx_clphone = JPathV ('clphone');
0132.00      d_s.lx_clemail = JPathV ('clemail');
0133.00      d_s.lx_claddr1 = JPathV ('claddr1');
0134.00      d_s.lx_claddr2 = JPathV ('claddr2');
0135.00      d_s.lx_claddr3 = JPathV ('claddr3');
0136.00      d_s.lx_clpcode = JPathV ('clpcode');
0137.00      d_s.lx_cllang = JPathV ('cllang');
0138.00
0139.00    endsr;
0140.00    // ========================================================================
0141.00    //    Write the Requested Data
0142.00    // ========================================================================
0143.00    Begsr z_PrcDBWrite;
0144.00
0145.00      exec sql insert into LXCLIENT( id, clidno, clname, clsurname, cltitle,
0146.00      clphone, clemail, claddr1, claddr2, claddr3, clpcode, cllang)
0147.00      Values( :d_s.lx_clidno, :d_s.lx_clname, :d_s.lx_clsurname,
0148.00      :d_s.lx_cltitle, :d_s.lx_clphone, :d_s.lx_clemail,
0149.00      :d_s.lx_claddr1, :d_s.lx_claddr2, :d_s.lx_claddr3,
0150.00      :d_s.lx_clpcode, :d_s.lx_cllang);
0151.00
0152.00    Endsr;
0153.00    // ========================================================================
0154.00    //    Process Post Response
0155.00    // ========================================================================
0156.00    Begsr z_PrcPostRsp;
0157.00
0158.00      // Write the response
0159.00
0160.00      beginObject(' ');
0161.00      if sqlcod = *zeros;
0162.00        addchar('status' : 'data successfully written');
0163.00      else;
0164.00        addchar('status' : 'data insert fail check joblog for details');
0165.00      endif;
0166.00      beginObject('requestData');
00166.00     addIntr('id' : d_s.lx_id);
0167.00      addchar('clidno': %trim(d_s.lx_clidno));
0168.00      addchar('clname': %trim(d_s.lx_clname));
0169.00      addchar('clsurname': %trim(d_s.lx_clsurname));
0170.00      addchar('cltitle': %trim(d_s.lx_cltitle));
0171.00      addchar('clphone': %trim(d_s.lx_clphone));
```

```
0172.00       addchar('clemail': %trim(d_s.lx_clemail));
0173.00       addchar('claddr1': %trim(d_s.lx_claddr1));
0174.00       addchar('claddr2': %trim(d_s.lx_claddr2));
0175.00       addchar('claddr3': %trim(d_s.lx_claddr3));
0176.00       addchar('clpcode': %trim(d_s.lx_clpcode));
0177.00       addchar('cllang': %trim(d_s.lx_cllang));
0178.00       endObject();
0179.00       endObject();
0180.00
0181.00     endsr;
0182.00   /End-Free
0207.00     // ========================================================================
0208.00     //   LXR Generic Procedures
0209.00     // ========================================================================
0211.00   /copy qrpglesrc,lxrrestp
```

Colour Legend:

Orange	Header Specifications
Green	Copybooks for the MDRest4i engines.
Blue	Work field definitions on the basis of the File Fields along with prefix lx_.
Purple	See Z_LoadRequestBody Routine below
Lignt Green	See Z_PrcDBWrite Routine below
Pink	See Z_PrcPostRSP Routine below

Z_PROCPOST Routine: This subroutine gets the control when the request type is “POST”. This routine has calls th three subroutines “z_PrcDBWrite”, “z_PrcPostRsp”, “z_LoadReqBody”.

Z_LoadReqBody Routine: In this subroutine, the data structure is cleared first and then “JGet” function is used to fetch the value of relevant field from the request body. If the field is numeric or date field, appropriate type-casting is applied.

Z_PrcDBWrite Routine: In this subroutine, we have a very simple statement where we insert the record in the DB file using the data structure populated from “z_LoadReqBody” subroutine.

Z_PrcPostRSP Routine: In this subroutine, first the procedure "beginObject" is called with blank value. This will open the top level curly brace in the response. We then call “addChar” function to write the status of data write request. Subsequently, we open an inner object, by calling the same “beginObject” procedure but this time with the label “requestData”. This will open a curly brace labeled with "requestData" in response. Then, we are then calling “addChar”, “addInt”, “addDec” procedures depending on the data type of the field which is being written in response.

Now let's compile CLINTPST program and use details below in SOAPUI/POSTMAN (or any client software able to POST JSON to a REST API) to call the API.

Before the test, kindly ensure the library which has the DB file processed in above MDRGENPRD, has been added to the configuration file for the HTTP server instance you are using in this test. This is an example of the library list setting in the httpd.conf file:

SetEnv QIBM_CGI_LIBRARY_LIST "DBLIB; MDRST" MDRST need not be in the library list unless objects used by your added API logic are in that library.

SOAPUI/POSTMAN Testing specs

Method : POST

Media Type : application/json

EndPoint : http://youribmiserver:yourportnumber

Resource : /MDRTUTLIB /clientpst

POST Json Body :

```
{
    "id": 7,
    "clidno": "621572419",
    "clname": "Richard",
    "clsurname": "Gail",
    "cltitle": "Mr",
    "clphone": "07856432194",
    "clemail": "rgail@gmail.com",
    "claddr1": "4 fox road",
    "claddr2": "Post Town",
    "claddr3": "Plantville",
    "clpcode": "1001",
    "cllang": "E"
}
```

we get this response:

```
{
      "status": "data successfully written",
      "requestData": {
            "id": 7,
            "clidno": "621572419",
            "clname": "Richard",
            "clsurname": "Gail",
            "cltitle": "Mr",
            "clphone": "07856432194",
            "clemail": "rgail@gmail.com",
            "claddr1": "4 fox road",
            "claddr2": "Post Town",
            "claddr3": "Plantville",
            "clpcode": "1001",
            "cllang": "E"
      }
}
```

We can also run a query over the underlying LXCLIENT file to see the added record.

4.2.3.1 POST Method Service with Manual logic for Database Updates

Let's create an example program in which we will insert records into a database file using the POST method from a client or consumer.

First of all prompt the command MDRST/MDRGENPRD from the command line and use the parameters below to run the command:

```
                  MDRest4i Generator - Provider (MDRGENPRD)

Type choices, press Enter.

Target Library . . . . . . . . . > MDRTUTLIB    Name
Target Source File . . . . . .    QRPGLESRC    Name
Target Source Member . . . . . > CLIENTPST1    Name
Member Text  . . . . . . . . . . > 'Test Member Generated by MDRest4i'

Get Method Required? . . . . . .   N            Y=Yes , N-No
Put Method Required? . . . . . .   N            Y=Yes , N-No
Post Method Required?  . . . . . > Y            Y=Yes , N-No
Patch Method Required? . . . . .   N            Y=Yes , N-No
Delete Method Required?  . . . .   N            Y=Yes , N-No
Options Method Required? . . . .   N            Y=YES , N-NO
Data Format  . . . . . . . . . .   J            J=JSON, X-XML
```

- Now select the method using "Y"(as per the required HTTP method) . Here we are going to use the POST method in our API so we are selecting "Y" in Post method option. We can create the API in XML also by selecting Data Format "X". Otherwise default set as "J" for JSON. After this press enter to see the next parameters.

```
Requestbody Parse Method . . . .   USR          IFS, DBF, USR
```

- Now here are three provisions, "IFS" for writing JSON request to the IFS file, "DBF" for writing JSON request to MDRJSONF DB file and the last is "USR" which will add logic inside the API. "USR" set as default. After that, Press enter to see the next parameters.

```
Response Processing Method . . .   USR          IFS, DBF, USR
```

- Now here are three provisions, "IFS" for reading response from the IFS file, "DBF" for reading response from MDRJSONF DB file and the last is "USR" which will read logic inside the API for sending response to client. "USR" set as default. After that, Press enter to see the next parameters.

```
Processing Type  . . . . . . . . > N            D=DB, C-Call, N-None
```

- Three provisions here as well. If you want to process the DB file then select "D", for calling program, module and service program, select "C" and last is "N" in case you don't want process any DB file or any external program. The default is "N". Press enter to see next parameters.

```
Parm Required? . . . . . . . . .   N            Y=Parm Reqd, N-Parm Not Reqd
```

In the target library we will specify the library name where member would be generated.

In the target source file we will specify the source file name.

In the target source member we will specify the source member name.

In this example, we have used the HTTP POST method, so parameter below should be set “Y”.

```
Post Method Required?  . . . . . > Y
```

We also set “Processing Type” as “N” which means we will manually add the required logic for database operations. Once the command execution completes, the source member below is generated under the specified source file and library combinations, and the program will be compiled into the target library. The name of the program is CLIENTPST1.

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0004.00 h bnddir('LXRGLOBAL':'BNDZIP')
0005.00  * ©=2019 Midrange Dynamics=============================================
0006.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0007.00  * ©=2019 Midrange Dynamics=============================================
0008.00  * h decedit(',')
0009.00  * ©=2019=Midrange Dynamics============================================
0010.00  * ©=2019=Midrange Dynamics==========================================
0011.00  * Create Date :
0012.00  * Created By  : Midrange Dynamics
0013.00  * Description : This is the MDRest4i RESTfull Webservice
0014.00  * Service Type: RESTfull Webservice
0015.00  * Input       : GET Request - URL PARMS
0016.00  * Output      : Error/warning - JSON Component
0017.00  * ©=2019=Midrange Dynamics==========================================
0018.00
0019.00  // Standard MDRest4i Copybooks for REST Service
0020.00  /copy qrpglesrc,mdrusercpy
0032.00
0033.00  // Variable Definitions
0034.00 d w_string        s          20480
0035.00 d w_str2          s          20480
0036.00
0037.00   // Write query parameter definitions
0038.00
0039.00  // Indicate subroutine overrides defined locally
0040.00  /define LXR_CustomInit
0041.00
0042.00  /define LXR_ProcPost
0043.00
0044.00  /copy/qrpglesrc,mdrestdfn
0044.00  /copy qrpglesrc,lxrrestc
0045.00
0046.00  /undefine LXR_CustomInit
0047.00
0048.00  /undefine LXR_ProcPost
0049.00
0050.00  /Free
0051.00    // ====================================================================
0052.00    //   Customized initialization
0053.00    // ====================================================================
0054.00    Begsr Z_CustomInit;
0055.00
0067.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0068.00      // set w_maxrsplen to the expected maximum response length
0069.00      // uncomment the statement below and set the value as required
0070.00      // w_maxrsplen =15542880;
0071.00
0072.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0073.00      // set w_maxreqlen to the expected maximum request body length
0074.00      // w_maxreqlen =15542880;
```

```
0075.00
0076.00      // If request or response value is greater than 16MB,
0077.00      // declare the keyword "alloc(*teraspace)" in control spec of this pro
0078.00
0079.00      // In order to determine the actual request/response size, set the
0080.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0081.00      // The REST API will then only return the request/response size.
0082.00      // The indicator "n_getrspsize" will allocate 500MB memory from heap
0083.00      // to process the reqeust and return the response size.
0084.00      // If the expected response size is much lesser, set the max expected
0085.00      // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0086.00
0087.00      // Use this IFS switch to log the API data in an IFS Folder
0088.00      // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0089.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
0090.00      n_saveIFSSwitch = *off;
0091.00
0092.00      // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0093.00      // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0094.00      ng_mdrJobHdr = *off;
0095.00
0096.00      // Extracts request query parameters to d_Qparm data structure
0097.00      n_extractQryPrms = *off;
0098.00
0099.00    Endsr;
0081.00
0082.00    // ========================================================================
0083.00    //    Override Post Method
0084.00    // ========================================================================
0085.00    Begsr z_ProcPost;
0086.00
0087.00    // Logic from custom copybook
0088.00
0089.00
0090.00    Endsr;
0091.00  /End-Free
0116.00    // ========================================================================
0117.00    //   LXR Generic Procedures
0118.00    // ========================================================================
0120.00  /copy qrpglesrc,lxrrestp
```

Now we will manually add logic in z_ProcPost subroutine as well as specifiying the JSON response. Here, we want to be able to write multiple records in DB file and therefore we will load the received entries via using“JPathV” function inside the generated program for the POST method, the default z_PROCPOST subroutine in the copybook was overridden during the generation, because the POST method was selected. For this a compiler directive “/define-/undefine” is used. It is important to note that this “/define LXR_PROCPOST” is added ***before*** the “/copy LXRRESTC”, and “/undefine LXR_PROCPOST” is added ***after*** the “/copy LXRRESTC”. Otherwise the logic added manually in the z_PROCPOST subroutine by the user would cause compilation failure due to duplicate definition of subroutine.

The source below contains the original generated source, ***plus*** the z_ProcPost subroutine logic added manually (highlighted in red), work fields used in program added manually (highlighted in blue).

Comments (highlighted in green below) were added for better understanding of this REST POST service example.

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0004.00 h bnddir('LXRGLOBAL':'BNDZIP')
```

```
0005.00  * ©=2019 Midrange Dynamics=============================================
0006.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0007.00  * ©=2019 Midrange Dynamics=============================================
0008.00  * h decedit(',')
0009.00  * ©=2019=Midrange Dynamics=============================================
0010.00  * ©=2019=Midrange Dynamics==========================================
0011.00  * Create Date :
0012.00  * Created By  : Midrange Dynamics
0013.00  * Description : This is the MDRest4i RESTfull Webservice
0014.00  * Service Type: RESTfull Webservice
0015.00  * Input       : GET Request - URL PARMS
0016.00  * Output      : Error/warning - JSON Component
0017.00  * ©=2019=Midrange Dynamics==========================================
0018.00
0019.00  // Standard MDRest4i Copybooks for REST Service
0020.00  /copy qrpglesrc,mdrusercpy
0032.00
0033.00  // Variable Definitions
0034.00 d w_string        s          20480
0035.00 d w_str2          s          20480
0036.00
0037.00  // Data Structure Definitions
0041.00 d w_Pos           S              5S 0
0043.00
0058.00 d i_Lxclient      S              3S 0 Inz(0)
0059.00
0060.00   // Data Struclure Definitions
0061.00 d d_S             DS                  Inz Qualified
0061.00 d id                               9B 0
0062.00 d clidno                          15A
0063.00 d clName                          30A
0064.00 d clSurName                       30A
0065.00 d clTitle                         10A
0066.00 d clPhone                         15A
0067.00 d clEmail                        100A
0068.00 d clAddr1                         30A
0069.00 d clAddr2                         30A
0070.00 d clAddr3                         30A
0071.00 d clPCode                         15A
0072.00 d clLang                           1A
0073.00
0074.00   // Write query parameter definitions
0075.00
0076.00  // Indicate subroutine overrides defined locally
0077.00  /define LXR_CustomInit
0078.00
0079.00  /define LXR_ProcPost
0080.00
0081.00  /copy qrpglesrc,mdrestdfn
0081.00  /copy qrpglesrc,lxrrestc
0082.00
0083.00  /undefine LXR_CustomInit
0084.00
0085.00  /undefine LXR_ProcPost
0086.00
0087.00  /Free
0051.00    // ====================================================================
0052.00    //   Customized initialization
0053.00    // ====================================================================
0054.00    Begsr Z_CustomInit;
0055.00
0067.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0068.00      // set w_maxrsplen to the expected maximum response length
0069.00      // uncomment the statement below and set the value as required
0070.00      // w_maxrsplen =15542880;
0071.00
```

```
0072.00       // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0073.00       // set w_maxreqlen to the expected maximum request body length
0074.00       // w_maxreqlen =15542880;
0075.00
0076.00       // If request or response value is greater than 16MB,
0077.00       // declare the keyword "alloc(*teraspace)" in control spec of this pro
0078.00
0079.00       // In order to determine the actual request/response size, set the
0080.00       // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0081.00       // The REST API will then only return the request/response size.
0082.00       // The indicator "n_getrspsize" will allocate 500MB memory from heap
0083.00       // to process the reqeust and return the response size.
0084.00       // If the expected response size is much lesser, set the max expected
0085.00       // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0086.00
0087.00       // Use this IFS switch to log the API data in an IFS Folder
0088.00       // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0089.00       // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
0090.00       n_saveIFSSwitch = *off;
0091.00
0092.00       // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0093.00       // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0094.00       ng_mdrJobHdr = *off;
0095.00
0096.00       // Extracts request query parameters to d_Qparm data structure
0097.00       n_extractQryPrms = *off;
0098.00
0099.00     Endsr;
0118.00
0119.00     // ========================================================================
0120.00     //    Override Post Method
0121.00     // ========================================================================
0122.00     Begsr z_ProcPost;
0123.00
0124.00       // Logic from custom copybook
0102.00       i_Lxclient = JGetArrayDim('Lxclient');
0103.00
0104.00       // Below statement used to write "{" in json response
0105.00       beginObject(' ');
0106.00       if i_Lxclient > 1;
0107.00         beginArray('recordsAdded');
0108.00       endif;
0109.00       For w_Pos = 1 to i_Lxclient;
0110.00         clear d_s;
0111.00         // Receving the values in Data structure using JPathV
0112.00
0112.00         d_s.id     =  %dec(JPathN('Lxclient(' + %char(w_pos) + ').Id'):9:0);
0113.00         d_s.clIdNo    =  JPathV('Lxclient(' + %char(w_pos) + ').clIdNo');
0114.00         d_s.clName    =  JPathV('Lxclient(' + %char(w_pos) + ').clName');
0115.00         d_s.clSurName =  JPathV('Lxclient(' + %char(w_pos) + ').clSurName');
0116.00         d_s.clTitle   =  JPathV('Lxclient(' + %char(w_pos) + ').clTitle');
0117.00         d_s.clPhone   =  JPathV('Lxclient(' + %char(w_pos) + ').clPhone');
0118.00         d_s.clEmail   =  JPathV('Lxclient(' + %char(w_pos) + ').clEmail');
0119.00         d_s.clAddr1   =  JPathV('Lxclient(' + %char(w_pos) + ').clAddr1');
0120.00         d_s.clAddr2   =  JPathV('Lxclient(' + %char(w_pos) + ').clAddr2');
0121.00         d_s.clAddr3   =  JPathV('Lxclient(' + %char(w_pos) + ').clAddr3');
0122.00         d_s.clPCode   =  JPathV('Lxclient(' + %char(w_pos) + ').clPCode');
0123.00         d_s.clLang    =  JPathV('Lxclient(' + %char(w_pos) + ').clLang');
0124.00
0125.00         // Insert the record in Lxclient file
0126.00         Exec Sql
0127.00           Insert InTo Lxclient(id,clIdNo,clName,clSurName,clTitle,clPhone,
0128.00             clEmail,clAddr1,clAddr2,clAddr3,clPCode,clLang)values :d_S;
0129.00
0130.00           If SqlCode = 0;
0131.00             SetHttpStatus(201: 'Lxclient Record Inserted');
```

```
0132.00
0136.00
0137.00              // Write the Json response
0138.00              if i_Lxclient > 1;
0139.00                beginObject(' ');
0140.00              endif;
0141.00              addChar('message':'Record added in lxclient table');
0142.00              addIntr('id_number':d_S.id);
0143.00              addChar('client_name':%trim(d_s.clName));
0144.00              endObject();
0145.00            endif;
0146.00          endfor;
0147.00          if i_Lxclient > 1;
0148.00            endArray();
0149.00          endif;
0150.00          // Below statement used to write "}" in json response
0151.00          endObject();
0152.00      Endsr;
0184.00    /End-Free
0256.00      // ====================================================================
0257.00      //   LXR Generic Procedures
0258.00      // ====================================================================
0260.00    /copy qrpglesrc,lxrrestp
```

Now let's compile this CLIENTPST1 program and use details below in SOAPUI/POSTMAN (or any client software able to POST JSON to a REST API) to call the API.

Note: Before the test kindly ensure YOURLIB has been added to the configuration file for the HTTP server instance you are using in this test. This is an example of the library list setting in the httpd.conf file:

SetEnv QIBM_CGI_LIBRARY_LIST "YOURLIB; MDRST

MDRST need not be in the library list unless objects used by your added API logic are in that library.

SOAPUI/POSTMAN Testing specs

Method : POST

Media Type : application/json

EndPoint : http://youribmiserver:yourportnumber

Resource : /MDRTUTLIB /clientpst1

POST Json Body :

```
{
      "Lxclient": [{

                  "Id": 30,
                  "clIdNo": "6207215024011",
                  "clName": "David",
                  "clSurName": "Gail",
                  "clTitle": "Mr",
                  "clPhone": "07856432194",
                  "clEmail": "david@gmail.com",
                  "clAddr1": "4 fox road",
                  "clAddr2": "Post Town",
                  "clAddr3": "Plantville",
                  "clPCode": "1001",
                  "clLang": "E"
            },
            {
                  "Id": 31,
                  "clIdNo": "6207215024022",
                  "clName": "Steve",
                  "clSurName": "Hussy",
                  "clTitle": "Mr",
                  "clPhone": "08856932187",
                  "clEmail": "steve@gmail.com",
                  "clAddr1": "5 fox road",
```

```
                "clAddr2": "Animal Town",
                "clAddr3": "Buckburg",
                "clPCode": "1002",
                "clLang": "F"
        }
    ]
}
```

we get this response:

```
{
    "recordsAdded": [{
                "message": "Record added in lxclient table",
                "id_number": 30,
                "client_name": "David"
        },
        {
                "message": "Record added in lxclient table",
                "id_number": 31,
                "client_name": "Steve"
        }
    ]
}
```

We can also run a query over the underlying LXCLIENT file to see the added records. If the file was already having the records, the "id_number" might be different in response.

4.2.4 Service generation with call to an external program

Let's create an example of REST service in which we will call another program or the procedure of the specific module or service program.

Prompt the command MDRST/MDRGENPRD from the command line and use the parameters below to run the command:

```
                  MDRest4i Generator - Provider (MDRGENPRD)

Type choices, press Enter.

Target Library . . . . . > MDRTUTLIB    Name
Target Source File . . . . . QRPGLESRC     Name
Target Source Member . . . . . > CALLPGM       Name
Member Text  . . . . . > 'Test Member Generated by MDRest4i'

Get Method Required? . . . . . > Y             Y=Yes , N-No
Put Method Required? . . . . .   N             Y=Yes , N-No
Post Method Required?  . . . .   N             Y=Yes , N-No
Patch Method Required? . . . .   N             Y=Yes , N-No
Delete Method Required?  . . .   N             Y=Yes , N-No
Options Method Required? . . .   N             Y=YES , N-NO
Data Format  . . . . . .   J             J=JSON, X-XML
```

- Now select the method using "Y"(as per the required HTTP method) . Here we are going to use the GET method in our API so we are selecting "Y" in Get method option. We can create the API in XML also by selecting Data Format "X". Otherwise default set as "J" for JSON. After this press enter to see the next parameters.

```
Response Processing Method . . .   USR           IFS, DBF, USR
```

- Now here are three provisions, "IFS" for reading response from the IFS file, "DBF" for reading response from MDRJSONF DB file and the last is "USR" which will read logic inside the API for sending response to client. "USR" set as default. After that, Press enter to see the next parameters.

```
Processing Type  . . . . . > C             D=DB, C-Call, N-None
```

- We have three provisions as well. If you want to process the DB file then select “D”, for calling program, module and service program, select “C” and last is “N” in case you don’t want to process any DB file or any external program. The default is “N”. Press enter to see next parameters.

```
Object Name  . . . . . . . . . . > CLIENTS       Name, *NONE
  Library  . . . . . . . . . . . >   MDRTUTLIB   Name, *LIBL
```

- Now here you need to supply object name and library of program, module or service program which you want to call inside the API.

```
Object Type  . . . . . . . . . . > *PGM          *MODULE, *SRVPGM, *PGM
```

- In this way If you want to call the program, provide “Object Type” as “*PGM”, If you want to call module Object, type will be “*MODULE” and in this way it will ask for procedure name which you use in the API and if you want to process service program, provide “Object Type” as *SRVPGM” you need to supply procedure name also.

```
Parm Required? . . . . . . . . . > P             P=Call, Q-Query & Call, N-None
List of parameters:
  Parameter Name . . . . . . . . > custid
  Mandatory Y/N? . . . . . . . . > Y             Character value
  Array Y/N? . . . . . . . . . . > N             Character value
  Array Dim  . . . . . . . . . .                 1-99
  Length . . . . . . . . . . . . > 7             1-999999
  Data Type  . . . . . . . . . . > A             A, B, D, F, G, I, N, P, S...
  Decimal Positions  . . . . . .                 0-63

  Parameter Name . . . . . . . . > custname
  Mandatory Y/N? . . . . . . . . > Y             Character value
  Array Y/N? . . . . . . . . . . > N             Character value
  Array Dim  . . . . . . . . . .                 1-99
  Length . . . . . . . . . . . . > 30            1-999999
  Data Type  . . . . . . . . . . > A             A, B, D, F, G, I, N, P, S...
  Decimal Positions  . . . . . .                 0-63
              + for more values
```

In the target library, source file and source member are the entries where the member will be generated.

In this example, we have specified the parameter below to “C” (i.e. external call). Then, we entered the object name, object library and object type. Here, we typed the object type as *PGM and therefore the command will generate the logic to call the specified external program. The field “Parm Required” is set to “P” and that means directly supply the query parameters in the program/procedure call. If we type “Q”, same thing happens but in this case, another set of variables with “w_” prefix is created. This variable gets used on the call statement so that you can perform any intermediate calculation from the value received and maintained in the query parameter.

```
Processing Type  . . . . . . . . > C             D=DB, C-Call, N-None
Object Name  . . . . . . . . . . > CLIENTS       Name, *NONE
  Library  . . . . . . . . . . . >   MDRTUTLIB   Name, *LIBL
Object Type  . . . . . . . . . . > *PGM          *MODULE, *SRVPGM, *PGM
Parm Required? . . . . . . . . . > P             P=Call, Q-Query & Call, N-None
```

While providing the parameter names, you have to also provide their data types. If a parameter is marked as optional (i.e. required = “N”), the keyword option(*nopass) will be attached on the prototype declaration.

Below is the generated source from the above command. The section specific to the program call (i.e. prototype declaration and the actual call) has been highlighted in green color.

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0003.00 h bnddir('LXRGLOBAL':'BNDZIP')
0004.00  * ©=2019 Midrange Dynamics=============================================
0005.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0006.00  * ©=2019 Midrange Dynamics=============================================
0007.00  * h decedit(',')
0008.00  * ©=2019=Midrange Dynamics=============================================
0009.00  * ©=2019=Midrange Dynamics==========================================
0010.00  * Create Date :
0011.00  * Created By  : Midrange Dynamics
0012.00  * Description : This is the MDRest4i RESTfull Webservice
0013.00  * Service Type: RESTfull Webservice
0014.00  * Input       : GET Request - URL PARMS
0015.00  * Output      : Error/warning - JSON Component
0016.00  * ©=2019=Midrange Dynamics==========================================
0017.00
0018.00  // Standard MDRest4i Copybooks for REST Service
0019.00  /copy qrpglesrc,mdrusercpy
0020.00
0021.00 d clients         Pr                  ExtPgm('CLIENTS')
0022.00 d  p_custid                    7A
0023.00 d  p_custname                 30A
0024.00
0025.00
0026.00  // Write query parameter definitions
0027.00 d  p_custid       s              7A
0028.00 d  n_custid       s               n
0029.00 d  p_custname     s             30A
0030.00 d  n_custname     s               n
0031.00
0032.00  // Indicate subroutine overrides defined locally
0033.00  /define LXR_CustomInit
0034.00
0035.00  /define LXR_CheckParms
0036.00  /define LXR_SetMethod
0037.00  /define LXR_SetParms
0038.00  /define LXR_ProcGET
0051.00
0039.00
0040.00  /copy qrpglesrc,mdrestdfn
0041.00  /copy qrpglesrc,lxrrestc
0042.00
0043.00  /undefine LXR_CustomInit
0044.00
0045.00  /undefine LXR_CheckParms
0046.00  /undefine LXR_SetMethod
0047.00  /undefine LXR_SetParms
0048.00  /undefine LXR_ProcGET
0049.00
0050.00  /Free
0051.00    // ====================================================================
0052.00    //   Customized initialization
0053.00    // ====================================================================
0054.00    Begsr Z_CustomInit;
0055.00
0056.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0057.00      // set w_maxrsplen to the expected maximum response length
0058.00      // uncomment the statement below and set the value as required
0059.00      // w_maxrsplen =15542880;
0060.00
```

```
0061.00       // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0062.00       // set w_maxreqlen to the expected maximum request body length
0063.00       // w_maxreqlen =15542880;
0064.00
0065.00       // If request or response value is greater than 16MB,
0066.00       // declare the keyword "alloc(*teraspace)" in control spec of this pro
0067.00
0068.00       // In order to determine the actual request/response size, set the
0069.00       // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0070.00       // The REST API will then only return the request/response size.
0071.00       // The indicator "n_getrspsize" will allocate 500MB memory from heap
0072.00       // to process the reqeust and return the response size.
0073.00       // If the expected response size is much lesser, set the max expected
0074.00       // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0075.00
0076.00       // Use this IFS switch to log the API data in an IFS Folder
0077.00       // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0078.00       // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
0079.00       n_saveIFSSwitch = *off;
0080.00
0081.00       // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0082.00       // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0083.00       ng_mdrJobHdr = *off;
0084.00
0085.00       // Extracts request query parameters to d_Qparm data structure
0086.00       n_extractQryPrms = *off;
0087.00
0088.00     Endsr;
0089.00
0090.00     // ===========================================================================
0091.00     //    Check and Populate Parms  - Empty Declaration
0092.00     // ===========================================================================
0093.00     Begsr z_checkParms;
0094.00
0095.00       if %trim(d_parm(1).s_value) <>  '*notFound';
0096.00         p_custid = %trim(d_parm(1).s_value);
0097.00         n_custid = *On;
0098.00       endif;
0099.00       if %trim(d_parm(2).s_value) <>  '*notFound';
0100.00         p_custname = %trim(d_parm(2).s_value);
0101.00         n_custname = *On;
0102.00       endif;
0103.00
0104.00     Endsr;
0105.00     // ===========================================================================
0106.00     //    Populate Required Parameters List
0107.00     // ===========================================================================
0108.00     Begsr z_setParms;
0109.00       // Populate the List of Expected Parms:
0110.00       d_parm(1).s_Name= 'custid';
0111.00       d_parm(1).s_Required= *on  ;
0112.00       d_parm(2).s_Name= 'custname';
0113.00       d_parm(2).s_Required= *on  ;
0114.00     Endsr;
0115.00     // ===========================================================================
0116.00     //    Populate Parms Required RETURN (HTTP TYPE - OPTION)
0117.00     // ===========================================================================
0118.00     begsr z_setMethod;
0119.00       // Required and Optional Parameters:
0120.00       beginObject(' ');
0121.00       addChar('Get':'Mandatory Parms: custid,custname');
0122.00       endObject(' ');
0123.00       n_ParmError = *on;
0124.00     Endsr;
0125.00     // ===========================================================================
0126.00     //    Override Get Method
```

```
0127.00    // ========================================================================
0128.00    Begsr z_ProcGET;
0129.00
0130.00    // Logic from custom copybook
0131.00    // added by stu
0132.00
0133.00
0134.00      clients(p_custid:p_custname);
0135.00
0136.00
0137.00      Exsr Z_PrcSndRsp;
0138.00
0139.00
0140.00    Endsr;
0141.00    // ========================================================================
0142.00    //   // Process Send Response
0143.00    // ========================================================================
0144.00    Begsr Z_PrcSndRsp;
0145.00
0146.00    Endsr;
0147.00  /End-Free
0148.00    // ========================================================================
0149.00    //   LXR Generic Procedures
0150.00    // ========================================================================
0151.00  /copy qrpglesrc,lxrrestp
```

Above is the automatically generated member where you can make the necessary adjustments according to the processing requirements.

4.2.5 REST Service generation using IFS file

MDRest4i has functions that will automatically parse incoming JSON (requestBody in Provider or response in consumer) through the IFS file. In this example we will write the request body JSON directly into an IFS file and send the JSON response from the same IFS file.

```
                  MDRest4i Generator - Provider (MDRGENPRD)

 Type choices, press Enter.

 Target Library . . . . . .   MDRTUTLIB    Name
Target Source File . . . . . .  QRPGLESRC    Name
Target Source Member . . . . .  IFSPRD       Name
Member Text  . . . . . .  'REST Pgm Generated by MDRest4i'

Get Method Required? . . . . .  N            Y=Yes , N-No
Put Method Required? . . . . .  N            Y=Yes , N-No
Post Method Required?  . . . . > Y            Y=Yes , N-No
Patch Method Required? . . . .  N            Y=Yes , N-No
Delete Method Required?  . . .  N            Y=Yes , N-No
Options Method Required? . . .  N            Y=YES , N-NO
Data Format  . . . . . .  J            J=JSON, X-XML
```

- Now select the method using “Y”(as per the required HTTP method) . Here we are going to use the POST method in our API so we are selecting “Y” in Post method option. We can create the API in XML also by selecting Data Format “X”. Otherwise default set as “J” for JSON. After this press enter to see the next parameters.

```
Requestbody Parse Method . . . > IFS          IFS, DBF, USR
```

- Now here are three provisions, “IFS” for writing JSON request to the IFS file, “DBF” for writing JSON request to MDRJSONF DB file and the last is “USR” which will add logic inside the API. “USR” set as default. After that, Press enter and page down to enter the ifs file name.

```
IFS File for Requestbody . . . .   /home/MDRest4i/Testjson.json
```

- Here we need to supply complete IFS path including the file name where we want to write JSON request body as per above.

```
Response Processing Method . . .   IFS           IFS, DBF, USR
```

- Now here are three provisions, “IFS” for reading response from the IFS file, “DBF” for reading response from MDRJSONF DB file and the last is “USR” which will read logic inside the API for sending response to client. “USR” set as default. After that, Press enter to see the next parameters.

```
IFS File for Response  . . . . .   /home/MDRest4i/Testjson.json
```

- We need to supply complete IFS path including the file name where from we want read JSON for sending the response.

In the target library we will specify the library name where member would be generated.

In the target source file we will specify the source file name.

In the target source member we will specify the source member name.

In this example, we have used the HTTP POST method, so parameter below should be set “Y”.

```
Post Method Required?  . . . . . > Y
```

In this example, we want to write request body in the IFS file so we need to select “IFS” and provide the complete IFS file including the IFS file name. In this way API will receive JSON request body through SOAP UI or POSTMAN and write this in the given IFS file.

```
Requestbody Parse Method . . . . > IFS           IFS, DBF, USR
 IFS File for Requestbody . . . . > '/home/MDRest4i/Testjson.json'
```

In this example We want to send response from the IFS file, so we need to supply “IFS” and also need to provide the complete IFS file name including file name like below:

```
Response Processing Method . . . . . .    IFS                          IFS, DBF, USR
 IFS File for Response  . . . . . > '/home/MDRest4i/Testjson.json'
```

Once the command execution completes, the source member below is generated under the specified source file and library combinations, and the program will be compiled into the target library. The name of the program is IFSPRD.

Examples is below:

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0003.00 h bnddir('LXRGLOBAL':'BNDZIP')
```

```
0004.00   * ©=2019 Midrange Dynamics=========================================
0005.00   * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0006.00   * ©=2019 Midrange Dynamics=========================================
0007.00   * h decedit(',')
0008.00   * ©=2019=Midrange Dynamics=========================================
0009.00   * ©=2019=Midrange Dynamics=========================================
0010.00   * Create Date :
0011.00   * Created By  : Midrange Dynamics
0012.00   * Description : This is the MDRest4i RESTfull Webservice
0013.00   * Service Type: RESTfull Webservice
0014.00   * Input       : GET Request - URL PARMS
0015.00   * Output      : Error/warning - JSON Component
0016.00   * ©=2019=Midrange Dynamics=========================================
0017.00
0018.00   // Standard MDRest4i Copybooks for REST Service
0019.00   /copy qrpglesrc,mdrusercpy
0020.00
0021.00   // Variable Definitions
0022.00  d w_string        s          20480
0023.00  d w_str2          s          20480
0024.00  d w_workstr       s        5242880
0025.00  d w_datalen       s             10i 0
0026.00  d w_success       s              1
0027.00  d w_dbfsts        s             50
0028.00
0029.00
0030.00   // Indicate subroutine overrides defined locally
0031.00   /define LXR_CustomInit
0032.00
0033.00   /define LXR_ProcPost
0034.00
0035.00   /copy qrpglesrc,mdrestdfn
0036.00   /copy qrpglesrc,lxrrestc
0037.00
0038.00   /undefine LXR_CustomInit
0039.00
0040.00   /undefine LXR_ProcPost
0041.00
0042.00   /Free
0043.00     // ===============================================================
0044.00     //   Customized initialization
0045.00     // ===============================================================
0046.00     Begsr Z_CustomInit;
0047.00
0048.00       // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0049.00       // set w_maxrsplen to the expected maximum response length
0050.00       // uncomment the statement below and set the value as required
0051.00       // w_maxrsplen =15542880;
0052.00
0053.00       // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0054.00       // set w_maxreqlen to the expected maximum request body length
0055.00       // w_maxreqlen =15542880;
0056.00
0057.00       // If request or response value is greater than 16MB,
0058.00       // declare the keyword "alloc(*teraspace)" in control spec of
0058.00       // this pro
0059.00
0060.00       // In order to determine the actual request/response size, set the
0061.00       // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0062.00       // The REST API will then only return the request/response
              // size. The indicator "n_getrspsize" will allocate 500MB
              // memory from heap
0064.00       // to process the reqeust and return the response size.
0065.00       // If the expected response size is much lesser, set the max
0065.00       // expected size in "w_debug_max_rsp_size" e.g. 10485760
0065.00       // (10MB) size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
```

```
0067.00
0068.00       // Use this IFS switch to log the API data in an IFS Folder
0069.00       // If you set n_saveIFSSwitch to *on; Then set the log
0069.00       // filepath e.g:
0070.00       // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/' +
0070.00       // 'LogFileddmmyyyy.txt'
0071.00       n_saveIFSSwitch = *off;
0072.00
0073.00       // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-
0073.00       // MYSERVERJOB-nnnnnn
0074.00       // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP
0073.00       // Instanc
0075.00       ng_mdrJobHdr = *off;
0076.00
0077.00       // Extracts request query parameters to d_Qparm data structure
0078.00       n_extractQryPrms = *off;
0079.00       n_parse = *off;
0080.00
0081.00     Endsr;
0082.00
0083.00     // ===============================================================
0084.00     //    Override Post Method
0085.00     // ===============================================================
0086.00     Begsr z_ProcPost;
0087.00
0088.00       // Get request Body
0089.00       GetReqBody(w_workstr);
0090.00       // Write request to the specified IFS file
0091.00       WriteIFS('/home/MDRest4i/Testjson.json': w_workstr:
0092.00       %len(%trim(w_workstr)));
0093.00
0094.00       // Read IFS file and send back response
0095.00       readIFS('/home/MDRest4i/Testjson.json': w_workstr:
0096.00       %size(w_workstr));
0097.00       w_datalen = %len(%trim(w_workstr));
0098.00       wlong(w_workstr:w_datalen);
0099.00
0100.00       // Logic from custom copybook
0101.00
0102.00
0103.00     Endsr;
0104.00   /End-Free
0105.00     // ===============================================================
0106.00     //   LXR Generic Procedures
0107.00     // ===============================================================
0108.00   /copy qrpglesrc,lxrrestp
```

I have highlighted code of IFS file processing with the bright green colour.

For a detailed explanation of this, please see the Reference-Guide's section "17.3 IFS Data Functions"

4.2.6 Rest Service generation using MDRJSONF file

MDRestt4i has functions that will automatically parse incoming JSON (requestBody in Provider or response in consumer) through the DB file (MDRJSONF) file. In this way we can write request body in MDRJSONF DB file and send response to the consumer from MDRJSONF file. Example is below:

```
                  MDRest4i Generator - Provider (MDRGENPRD)

 Type choices, press Enter.

 Target Library . . . . . . . . . > MDRTUTLIB     Name
 Target Source File . . . . . . .   QRPGLESRC     Name
```

```
Target Source Member . . . . . > DBFPRD        Name
Member Text  . . . . . .        'REST Pgm Generated by MDRest4i'

Get Method Required? . . . . .   N             Y=Yes , N-No
Put Method Required? . . . . .   N             Y=Yes , N-No
Post Method Required?  . . . . > Y             Y=Yes , N-No
Patch Method Required? . . . .   N             Y=Yes , N-No
Delete Method Required?  . . .   N             Y=Yes , N-No
Options Method Required? . . .   N             Y=YES , N-NO
Data Format  . . . . . .         J             J=JSON, X-XML
```

- Now select the method using “Y”(as per the required HTTP method). Here we are going to use the POST method in our API so we are selecting “Y” in Post method option. We can create the API in XML also by selecting Data Format “X”. Otherwise default set as “J” for JSON. After this press enter to see the next parameters.

```
Requestbody Parse Method . . . . > DBF          IFS, DBF, USR
Library Name for DBF Request . . > MDRTUTLIB    Name
```

- Now here are three provisions, “IFS” for writing JSON request to the IFS file, “DBF” for writing JSON request to MDRJSONF DB file and the last is “USR” which will add logic inside the API. “USR” set as default.

```
Response Processing Method . . . > DBF          IFS, DBF, USR
Library Name for DBF Response  . > MDRTUTLIB    Name
                                                                More...
F3=Exit    F4=Prompt    F5=Refresh    F12=Cancel    F13=How to use this display
F24=More keys
```

- We need to supply response method “DBF” and provide the library name where MDRJSONF DB file exist with the loaded JSON.

In the target library we will specify the library name where member would be generated.

In the target source file we will specify the source file name.

In the target source member we will specify the source member name.

In this example, we have used the HTTP POST method, so parameter below should be set “Y”.

```
Post Method Required?  . . . . . > Y
```

In this example, we want to write request body in the MDRJSONF DB file so we have selected method “DBF” for this we need to also supply the library name of DBF(where MDRJSONF file exist).

```
Requestbody Parse Method . . . . > DBF          IFS, DBF, USR
Library Name for DBF Request . . > MDRTUTLIB    Name
```

In this example we want to send response using the MDRJSONF DB file. So we need to provide method “DBF” here. In this way we need to provide library name of DBF(where MDRJSONF file exist).

```
Response Processing Method . . . > DBF          IFS, DBF, USR
Library Name for DBF Response  . > MDRTUTLIB    Name
```

Once the command execution completes, the source member below is generated under the specified source file and library combinations, and the program will be compiled into the target library. The name of the program is DBFPRD.

Generated code is below:

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0003.00 h bnddir('LXRGLOBAL':'BNDZIP')
0004.00  * ©=2019 Midrange Dynamics=========================================
0005.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0006.00  * ©=2019 Midrange Dynamics=========================================
0007.00  * h decedit(',')
0008.00  * ©=2019=Midrange Dynamics=========================================
0009.00  * ©=2019=Midrange Dynamics=========================================
0010.00  * Create Date :
0011.00  * Created By  : Midrange Dynamics
0012.00  * Description : This is the MDRest4i RESTfull Webservice
0013.00  * Service Type: RESTfull Webservice
0014.00  * Input       : GET Request - URL PARMS
0015.00  * Output      : Error/warning - JSON Component
0016.00  * ©=2019=Midrange Dynamics=========================================
0017.00
0018.00  // Standard MDRest4i Copybooks for REST Service
0019.00  /copy qrpglesrc,mdrusercpy
0020.00
0021.00  // Variable Definitions
0022.00 d w_string        s          20480
0023.00 d w_str2          s          20480
0024.00 d w_workstr       s        5242880
0025.00 d w_datalen       s             10i 0
0026.00 d w_success       s              1
0027.00 d w_dbfsts        s             50
0028.00
0029.00
0030.00  // Indicate subroutine overrides defined locally
0031.00  /define LXR_CustomInit
0032.00
0033.00  /define LXR_ProcPost
0034.00
0035.00  /copy qrpglesrc,mdrestdfn
0036.00  /copy qrpglesrc,lxrrestc
0037.00
0038.00  /undefine LXR_CustomInit
0039.00
0040.00  /undefine LXR_ProcPost
0041.00
0042.00  /Free
0043.00    // ===============================================================
0044.00    //   Customized initialization
0045.00    // ===============================================================
0046.00    Begsr Z_CustomInit;
0047.00
0048.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0049.00      // set w_maxrsplen to the expected maximum response length
0050.00      // uncomment the statement below and set the value as required
0051.00      // w_maxrsplen =15542880;
0052.00
0053.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0054.00      // set w_maxreqlen to the expected maximum request body length
0055.00      // w_maxreqlen =15542880;
0056.00
0057.00      // If request or response value is greater than 16MB,
0058.00      // declare the keyword "alloc(*teraspace)" in control spec of
```

```
0058.00      // this pro
0059.00
0060.00      // In order to determine the actual request/response size, set the
0061.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0062.00      // The REST API will then only return the request/response
0062.00      // size. The indicator "n_getrspsize" will allocate 500MB
0062.00      // memory from heap
0064.00      // to process the reqeust and return the response size.
0065.00      // If the expected response size is much lesser, set the max
0065.00      // expected size in "w_debug_max_rsp_size" e.g. 10485760
0065.00      // (10MB) size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0067.00
0068.00      // Use this IFS switch to log the API data in an IFS Folder
0069.00      // If you set n_saveIFSSwitch to *on; Then set the log
0069.00      // filepath e.g:
0070.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/' +
0070.00      // 'LogFileddmmyyyy.txt'
0071.00      n_saveIFSSwitch = *off;
0072.00
0073.00      // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-
0073.00      // MYSERVERJOB-nnnnnn
0074.00      // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP
0073.00      // Instanc
0075.00      ng_mdrJobHdr = *off;
0076.00
0077.00      // Extracts request query parameters to d_Qparm data structure
0078.00      n_extractQryPrms = *off;
0079.00      n_parse = *off;
0080.00
0081.00    Endsr;
0082.00
0083.00    // ===============================================================
0084.00    //    Override Post Method
0085.00    // ===============================================================
0086.00    Begsr z_ProcPost;
0087.00
0086.00      CreateJSONF('MDRTUTLIB':w_success);
0087.00      if w_success = 'N';
0088.00        ClearJsonF('MDRTUTLIB':'REQ':w_success:w_dbfsts);
0089.00      endif;
0090.00      JsonToDB('MDRTUTLIB':'REQ':w_success);
0091.00
0092.00      // Response processing
0093.00
0094.00      // Below is the dummy update which you need to remove and add
0095.00      // appropriate logic to process the request body  and load the
0096.00      // DB file with the response data
0097.00      exec sql update MDRTUTLIB/mdrjsonf set reqrsp = 'RSP' where
0098.00      reqrsp = 'REQ';
0099.00
0100.00      JsonFromDB('MDRTUTLIB':'RSP':w_success);
0099.00
0100.00      // Logic from custom copybook
0101.00
0102.00
0103.00    Endsr;
0104.00  /End-Free
0105.00    // ===============================================================
0106.00    //   LXR Generic Procedures
0107.00    // ===============================================================
0108.00  /copy qrpglesrc,lxrrestp
```

4.2.7 REST Service generation with call to a procedure in service program

Let's create an example of REST service in which we will call an external procedure from the service program.

Prompt the command MDRST/MDRGENPRD from the command line and use the parameters below to run the command:

```
                  MDRest4i Generator - Provider (MDRGENPRD)

Type choices, press Enter.

Target Library . . . . . . . . . > MDRTUTLIB    Name
Target Source File . . . . . .   QRPGLESRC     Name
Target Source Member . . . . . > CALLPRC       Name
Member Text  . . . . . . . . . . > 'Test Member Generated by MDRest4i'

Get Method Required? . . . . . > Y             Y=Yes , N-No
Put Method Required? . . . . .   N             Y=Yes , N-No
Post Method Required?  . . . .   N             Y=Yes , N-No
Patch Method Required? . . . .   N             Y=Yes , N-No
Delete Method Required?  . . .   N             Y=Yes , N-No
Options Method Required? . . .   N             Y=YES , N-NO
Data Format  . . . . . . . . .   J             J=JSON, X-XML
```

- Now select the method using “Y”(as per the required HTTP method). Here we are going to use the GET method in our API so we are selecting “Y” in Get method option. We can create the API in XML also by selecting Data Format “X”. Otherwise default set as “J” for JSON. After this press enter to see the next parameters.

```
Response Processing Method . . .   USR           IFS, DBF, USR
```

- Now here are three provisions, “IFS” for reading response from the IFS file, “DBF” for reading response from MDRJSONF DB file and the last is “USR” which will read logic inside the API for sending response to client. “USR” set as default. After that, Press enter to see the next parameters.

```
Processing Type  . . . . . > C             D=DB, C-Call, N-None
```

- We have three provisions as well. If you want to process the DB file then select “D”, for calling program, module and service program, select “C” and last is “N” in case you don’t want to process any DB file or any external program. The default is “N”. Press enter to see next parameters.

```
Object Name  . . . . . > SRV1          Name, *NONE
  Library  . . . . . >   MDRTUTLIB    Name, *LIBL
```

- Here we need to supply object name and library of program, module or service program which you want to call inside the API. Press the page down key to enter object type.

```
Object Type  . . . . . > *SRVPGM       *MODULE, *SRVPGM, *PGM
```

- Here we need to provide object type. So if you want to call the program, provide “Object Type” as “*PGM”, If you want to call module, need to provide Object type “*MODULE” and and if you want to process service program, provide “Object Type” as *SRVPGM”. In this this way if you supply object type *MODULE or *SRVPGM, it will ask for procedure name which you want to call inside the REST API. As per below:

```
Procedure Name . . . . . > ADDNUM
Parm Required? . . . . . > Q             P=Call, Q-Query & Call, N-None
List of parameters:
  Parameter Name . . . . . > custid
  Mandatory Y/N? . . . . . > Y             Character value
```

```
  Array Y/N? . . . . . . . . . . > N              Character value
  Array Dim  . . . . . . . . . .                  1-99
  Length . . . . . . . . . . . . > 7              1-999999
  Data Type  . . . . . . . . . . > A              A, B, D, F, G, I, N, P, S...
  Decimal Positions  . . . . . .                  0-63

  Parameter Name . . . . . . . . > custname
  Mandatory Y/N? . . . . . . . . > N              Character value
  Array Y/N? . . . . . . . . . . > N              Character value
  Array Dim  . . . . . . . . . .                  1-99
  Length . . . . . . . . . . . . > 30             1-999999
  Data Type  . . . . . . . . . . > A              A, B, D, F, G, I, N, P, S...
  Decimal Positions  . . . . . .                  0-63
              + for more values
```

In the target library, source file and source member are the entries where the member will be generated.

In this example, we have specified the parameter below to “C” (i.e. external call). Then, we entered the object name, object library and object type. As the object type is *SRVPGM, another parameter “Procedure Name” is prompted to enter the name of the procedure from the specified service program. The field “Parm Required” is set to “Q” in this case and that means another set of variables with “w_” prefix is also created. This variable is used on the call statement so that you can perform any intermediate calculation from the value received and maintained in the query parameter.

```
Processing Type  . . . . . . . . > C              D=DB, C-Call, N-None
Object Name  . . . . . . . . . . > SRV1.         Name, *NONE
  Library  . . . . . . . . . . . >   MDRTUTLIB   Name, *LIBL
Object Type  . . . . . . . . . . > *SRVPGM       *MODULE, *SRVPGM, *PGM
Procedure Name . . . . . . . . . > ADDNUM
Parm Required? . . . . . . . . . > Q             P=Call, Q-Query & Call, N-None
```

Below is the generated source from the above command. The section specific to the procedure call (i.e. prototype declaration and the actual call) has been highlighted in green color. As you can see, the command has generated the assignment statement on “w_” prefixed variables from “p_” suffixed variables and the reverse after the call statement.

```
0001.00 h bnddir('MDRUSERBND')
0003.00 h bnddir('LXRGLOBAL':'BNDZIP')
0004.00  * ©=2019 Midrange Dynamics=============================================
0005.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0006.00  * ©=2019 Midrange Dynamics=============================================
0007.00  * h decedit(',')
0008.00  * ©=2019=Midrange Dynamics=============================================
0009.00  * ©=2019=Midrange Dynamics==========================================
0010.00  * Create Date :
0011.00  * Created By  : Midrange Dynamics
0012.00  * Description : This is the MDRest4i RESTfull Webservice
0013.00  * Service Type: RESTfull Webservice
0014.00  * Input       : GET Request - URL PARMS
0015.00  * Output      : Error/warning - JSON Component
0016.00  * ©=2019=Midrange Dynamics=========================================
0017.00
0018.00  // Standard MDRest4i Copybooks for REST Service
0019.00  /copy qrpglesrc,mdrusercpy
0030.00
0031.00 d addnum          Pr                  ExtProc('ADDNUM')
0032.00 d  p_custid                    7A
0033.00 d  p_custname                 30A    Options(*Nopass)
```

```
0034.00
0035.00 d  w_custid       s              7A
0036.00 d  w_custname     s             30A
0037.00
0038.00
0039.00   // Write query parameter definitions
0040.00 d  p_custid       s              7A
0041.00 d  n_custid       s               n
0042.00 d  p_custname     s             30A
0043.00 d  n_custname     s               n
0044.00
0045.00  // Indicate subroutine overrides defined locally
0046.00  /define LXR_CustomInit
0047.00
0048.00  /define LXR_CheckParms
0049.00  /define LXR_SetMethod
0050.00  /define LXR_SetParms
0051.00  /define LXR_ProcGET
0052.00
0052.00  /copy qrpglesrc,mdrestdfn
0053.00  /copy qrpglesrc,lxrrestc
0054.00
0055.00  /undefine LXR_CustomInit
0056.00
0057.00  /undefine LXR_CheckParms
0058.00  /undefine LXR_SetMethod
0059.00  /undefine LXR_SetParms
0060.00  /undefine LXR_ProcGET
0061.00
0062.00  /Free
0063.00    // ====================================================================
0064.00    //   Customized initialization
0065.00    // ====================================================================
0066.00    Begsr Z_CustomInit;
0067.00
0056.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0057.00      // set w_maxrsplen to the expected maximum response length
0058.00      // uncomment the statement below and set the value as required
0059.00      // w_maxrsplen =15542880;
0060.00
0061.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0062.00      // set w_maxreqlen to the expected maximum request body length
0063.00      // w_maxreqlen =15542880;
0064.00
0065.00      // If request or response value is greater than 16MB,
0066.00      // declare the keyword "alloc(*teraspace)" in control spec of this pro
0067.00
0068.00      // In order to determine the actual request/response size, set the
0069.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0070.00      // The REST API will then only return the request/response size.
0071.00      // The indicator "n_getrspsize" will allocate 500MB memory from heap
0072.00      // to process the reqeust and return the response size.
0073.00      // If the expected response size is much lesser, set the max expected
0074.00      // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0075.00
0076.00      // Use this IFS switch to log the API data in an IFS Folder
0077.00      // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0078.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
0079.00      n_saveIFSSwitch = *off;
0080.00
0081.00      // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0082.00      // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0083.00      ng_mdrJobHdr = *off;
0084.00
0085.00      // Extracts request query parameters to d_Qparm data structure
0086.00      n_extractQryPrms = *off;
```

```
0087.00
0088.00     Endsr;
0093.00
0094.00     // ===================================================================
0095.00     //    Check and Populate Parms  - Empty Declaration
0096.00     // ===================================================================
0097.00     Begsr z_checkParms;
0098.00
0099.00       if %trim(d_parm(1).s_value) <>  '*notFound';
0100.00         p_custid = %trim(d_parm(1).s_value);
0101.00         n_custid = *On;
0102.00       endif;
0103.00       if %trim(d_parm(2).s_value) <>  '*notFound';
0104.00         p_custname = %trim(d_parm(2).s_value);
0105.00         n_custname = *On;
0106.00       endif;
0107.00
0108.00     Endsr;
0109.00     // ===================================================================
0110.00     //    Populate Required Parameters List
0111.00     // ===================================================================
0112.00     Begsr z_setParms;
0113.00       // Populate the List of Expected Parms:
0114.00       d_parm(1).s_Name= 'custid';
0115.00       d_parm(1).s_Required= *on  ;
0116.00       d_parm(2).s_Name= 'custname';
0117.00       d_parm(2).s_Required= *off ;
0118.00     Endsr;
0119.00     // ===================================================================
0120.00     //    Populate Parms Required RETURN (HTTP TYPE - OPTION)
0121.00     // ===================================================================
0122.00     begsr z_setMethod;
0123.00       // Required and Optional Parameters:
0121.00       beginObject(' ');
0122.00       addChar('Get':'Mandatory Parms: custid ::Optional Parms: custname');
0123.00       endObject(' ');
0125.00       n_ParmError = *on;
0126.00     Endsr;
0127.00     // ===================================================================
0128.00     //    Override Get Method
0129.00     // ===================================================================
0130.00     Begsr z_ProcGET;
0131.00
0132.00     // Logic from custom copybook
0133.00     // added by stu
0134.00
0135.00       w_custid = p_custid;
0136.00       w_custname = p_custname;
0137.00
0138.00       addnum(w_custid:w_custname);
0139.00
0140.00       p_custid = w_custid;
0141.00       p_custname = w_custname;
0142.00
0143.00       Exsr Z_PrcSndRsp;
0144.00
0145.00
0146.00     Endsr;
0147.00     // ===================================================================
0148.00     //   // Process Send Response
0149.00     // ===================================================================
0150.00     Begsr Z_PrcSndRsp;
0151.00
0152.00     Endsr;
0153.00   /End-Free
0178.00     // ===================================================================
```

```
0179.00    //   LXR Generic Procedures
0180.00    // ========================================================================
0181.00  /copy qrpglesrc,lxrrestp
```

4.2.8 Pagination using a REST GET Method Provider/Service

In order to demonstrate pagination in REST service we have to perform below steps:

1. Setup the headers in webserver.
2. Create the REST service and introduce page handling.
3. Call the service from SOAPUI or any other consumer.
4. Setup the web-server to allow specific custom headers

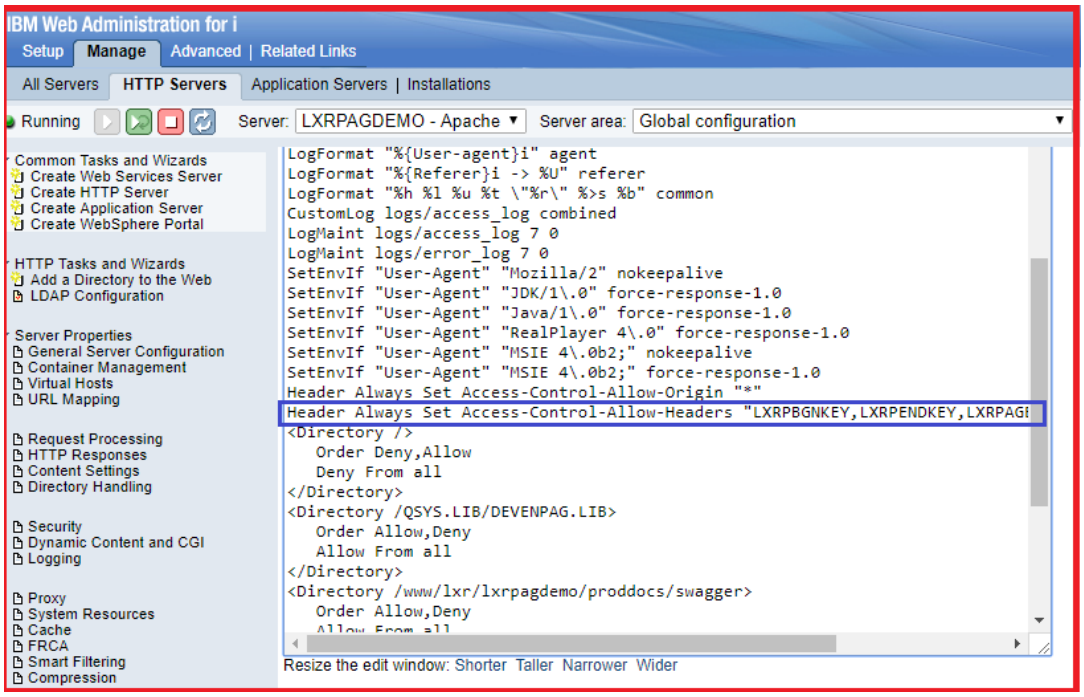
We need to go access the configuration of the specific http server by accessing from the link below in web-browser:

http://yourserver:2001/HTTPAdmin

Click on “Edit Configuration File” in left navigation panel and make sure the new custom headers “LXRPBGNKEY, LXRPENDKEY, LXRPAGTYP, LXRNBRRCD” are added below:

```
Header Set Access-Control-Allow-Origin "*"
SetEnv QIBM_CGI_LIBRARY_LIST "MDRST;YOURLIB"
Header Set Access-Control-Allow-Headers "LXRPBGNKEY,LXRPENDKEY,LXRPAGTYP,LXRNBRRCD"
<Directory />
```

Below is the screenshot where this setup is to be done:



2. Create the REST service capable of page handling

This is similar to previous example except that the API has been requested to generate the logic for retrieving the previous and the next page. In this case, we set the parameter “Include Paging Logic” as “Y” and set “Parm Required” to “N”.

```
Target Library . . . . . . . . . > MDRTUTLIB     Name
Target Source File . . . . . .   QRPGLESRC     Name
Target Source Member . . . . . > CLIENTSPG     Name
Member Text  . . . . . .         'REST Pgm Generated by MDRest4i'
```

```
Get Method Required? . . . . . > Y            Y=Yes , N-No
Put Method Required? . . . . .   N            Y=Yes , N-No
Post Method Required?  . . . .   N            Y=Yes , N-No
Patch Method Required? . . . .   N            Y=Yes , N-No
Delete Method Required?  . . .   N            Y=Yes , N-No
Options Method Required? . . .   N            Y=YES , N-NO
Data Format  . . . . . .  . . .  J            J=JSON, X-XML
```

- Now select the method using “Y”(as per the required HTTP method) . Here we are going to use the GET method in our API so we are selecting “Y” in Get method option. We can create the API in XML also by selecting Data Format “X”. Otherwise default set as “J” for JSON. After this press enter to see the next parameters.

```
Response Processing Method . . .   USR          IFS, DBF, USR
```

- Now here are three provisions, “IFS” for reading response from the IFS file, “DBF” for reading response from MDRJSONF DB file and the last is “USR” which will read logic inside the API for sending response to client. “USR” set as default. After that, Press enter to see the next parameters.

```
Processing Type  . . . . . > D            D=DB, C-Call, N-None
```

- We have three provisions as well. If you want to process the DB file then select “D”, for calling program, module and service program, select “C” and last is “N” in case you don’t want process any DB file or any external program. The default is “N”. Press enter to see next parameters.

```
Include Paging Logic?  . . . . > Y            Y=Include Paging, N-Not Reqd
```

- If we want to include paging logic inside the API, specify “Y” above. Press enter and page down key to enter the DB file name and library.

```
Default DB File  . . . . . > LXCLIENT      Name, *NONE
  Library  . . . . . >   MDRST       Name, *LIBL
Parm Required? . . . . . > N            Y=Parm Reqd, N-Parm Not Reqd
```

Once the command completes the execution, the source member below is generated under the specified source file and library combinations and it gets compiled as well. The name of the program is CLIENTSPG. As we can see, two new compiler directives “LXR_NxtPage” and “LXR_PrvPage” have been added which means the default declaration of the subroutines “Z_PrcNxtPage” and “Z_PrcPrvPage” from the copybook LXRRESTC are not included and are required to be defined in this source.

At the beginning of subroutine “z_PrcResponse” (which is the first entry) as well as in the subroutines “Z_PrcNxtPage” and “Z_PrcPrvPage”, the variable “n_BgnKey” is set to *Off. This is an instruction to “Z_ProcessSend” subroutine to save the page begin key when the first record is being processed. The subroutine “z_ProcessSend” is called to process the next entry read from the cursor and in the first iteration when “n_BgnKey” is *off, it sets the page key in “w_pageBgnKey”. The variable “w_pageEndKey” is set in each iteration to record the key values of the last record. The subroutine “z_PrcNxtPage” uses the keys received from the http header parameter to position the pointer after the keys available in “w_pageEndKey”. The subroutine “z_PrcPrvPage” uses the order by clause in reverse order to fetch prior to the key values in “w_pageBgnKey”. It uses scroll cursor to move forward and backward and in the cursor with reverse order data, it goes ahead till the number of rows requested or the end of the data. It then stars reading in reverse order because for sending, it has to send the data in normal DB keys order.

The calling of previous page and next page subroutine is managed automatically by the logic in copybook. The REST service expects four http header parameters “LXRPBGNKEY” (page begin key of last request),

"LXRPENDKEY" (page end key of last request), "LXRPAGETYP" (it would have "P" for previous page and "N" for next page) and "LXRNBRRCD" (number of records to return in http response).

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0004.00 h bnddir('LXRGLOBAL':'BNDZIP')
0005.00  * ©=2019 Midrange Dynamics============================================
0006.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0007.00  * ©=2019 Midrange Dynamics============================================
0008.00  * h decedit(',')
0009.00  * ©=2019=Midrange Dynamics============================================
0010.00  * ©=2019=Midrange Dynamics==========================================
0011.00  * Create Date :
0012.00  * Created By  : Midrange Dynamics
0013.00  * Description : This is the MDRest4i RESTfull Webservice
0014.00  * Service Type: RESTfull Webservice
0015.00  * Input       : GET Request - URL PARMS
0016.00  * Output      : Error/warning - JSON Component
0017.00  * ©=2019=Midrange Dynamics==========================================
0018.00
0019.00  // Standard MDRest4i Copybooks for REST Service
0020.00  /copy qrpglesrc,mdrusercpy
0031.00
0032.00  // Variable Definitions
0033.00 d w_string        s          20480
0034.00 d w_str2          s          20480
0035.00  // Define Work Variables for SQL building
0036.00 d                 ds
0037.00 d   s_statement                1024    varying
0038.00 d   s_statement1               1024    overlay(s_statement:+3)
0039.00 d   s_firstRec                    n
0040.00 d w_stmtcnt       s            200
0042.00  // Custom Variables for response schema
0043.00 d d_S             ds                   qualified
0044.00 d lx_id                          9B 0
0045.00 d lx_clidno                      15A
0046.00 d lx_clname                      30A
0047.00 d lx_clsurname...
0048.00 d                                30A
0049.00 d lx_cltitle                     10A
0050.00 d lx_clphone                     15A
0051.00 d lx_clemail                    100A
0052.00 d lx_claddr1                     30A
0053.00 d lx_claddr2                     30A
0054.00 d lx_claddr3                     30A
0055.00 d lx_clpcode                     15A
0056.00 d lx_cllang                       1A
0057.00 d w_Count         s              10i 0
0058.00 d w_CountRcd      s              10i 0
0059.00
0060.00   // Write query parameter definitions
0061.00
0062.00  // Indicate subroutine overrides defined locally
0063.00  /define LXR_CustomInit
0064.00
0065.00  /define LXR_ProcGET
0066.00  /define LXR_NxtPage
0067.00  /define LXR_PrvPage
0068.00
0069.00  /copy qrpglesrc,mdrestdfn
0069.00  /copy qrpglesrc,lxrrestc
0070.00
0071.00  /undefine LXR_CustomInit
0072.00
```

```
0073.00  /undefine LXR_ProcGET
0074.00  /undefine LXR_NxtPage
0075.00  /undefine LXR_PrvPage
0076.00
0077.00  /Free
0078.00    // ====================================================================
0079.00    //   Customized initialization
0080.00    // ====================================================================
0081.00    Begsr Z_CustomInit;
0082.00
0079.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0080.00      // set w_maxrsplen to the expected maximum response length
0081.00      // uncomment the statement below and set the value as required
0082.00      // w_maxrsplen =15542880;
0083.00
0084.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0085.00      // set w_maxreqlen to the expected maximum request body length
0086.00      // w_maxreqlen =15542880;
0087.00
0088.00      // If request or response value is greater than 16MB,
0089.00      // declare the keyword "alloc(*teraspace)" in control spec of this pro
0090.00
0091.00      // In order to determine the actual request/response size, set the
0092.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0093.00      // The REST API will then only return the request/response size.
0094.00      // The indicator "n_getrspsize" will allocate 500MB memory from heap
0095.00      // to process the reqeust and return the response size.
0096.00      // If the expected response size is much lesser, set the max expected
0097.00      // size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0098.00
0099.00      // Use this IFS switch to log the API data in an IFS Folder
0100.00      // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
0101.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
0102.00      n_saveIFSSwitch = *off;
0103.00
0104.00      // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
0105.00      // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
0106.00      ng_mdrJobHdr = *off;
0107.00
0108.00      // Extracts request query parameters to d_Qparm data structure
0109.00      n_extractQryPrms = *off;
0110.00      n_paging = *on;
0111.00
0112.00    Endsr;
0111.00
0112.00    // ====================================================================
0113.00    //    Override Get Method
0114.00    // ====================================================================
0115.00    Begsr z_ProcGET;
0116.00
0117.00    // Logic from custom copybook
0118.00    // added by stu
0119.00
0120.00      Exsr z_PrcResponse;
0121.00
0122.00    Endsr;
0123.00    // ====================================================================
0124.00    //    Process Response
0125.00    // ====================================================================
0126.00    Begsr z_PrcResponse;
0127.00
0128.00      n_BgnKey = *Off;
0129.00      clear d_S;
0130.00       // Main SQL statement for cursors
0131.00      s_statement =
0132.00      'select ID, CLIDNO, CLNAME, CLSURNAME, CLTITLE, CLPHONE, CLEMAIL,'+
```

```
0133.00       ' CLADDR1, CLADDR2, CLADDR3, CLPCODE, CLLANG from LXCLIENT';
0134.00
0135.00       w_stmtcnt = 'select count(*) from LXCLIENT';
0136.00
0137.00        // Start preparing cursors
0138.00       exec sql prepare w_stmtcnt from :w_stmtcnt;
0139.00
0140.00       exec sql prepare s_statement1 from :s_statement1;
0141.00
0142.00       if sqlstt <> c_SqlSuccess;
0143.00         ErrorJSON('No data to return');
0144.00         leavesr;
0145.00       endif;
0146.00
0147.00       // Process SQL data
0148.00       exec sql declare c2_curser cursor for w_stmtcnt;
0149.00       exec sql open c2_curser;
0150.00       // Load the count of records
0151.00       exec sql fetch c2_curser into :w_count;
0152.00       exec sql declare c1_curser cursor for s_statement1;
0153.00       exec sql open c1_curser;
0154.00       // Prepare an array for multiple record selection
0155.00       exec sql fetch c1_curser into :d_S;
0156.00       if sqlstt = c_SqlSuccess or sqlstt < c_SqlEOF;
0157.00         // Build Response Container
0158.00         if w_count > 1;
0159.00           beginObject(' ');
0160.00           beginArray('LXCLIENT');
0161.00         endif;
0162.00       else;
0163.00         ErrorJSON('No data to return');
0164.00         leavesr;
0165.00       endif;
0166.00
0167.00       dow sqlstt < c_SqlEOF and w_countRcd < w_nbrrcd;
0168.00         Exsr Z_ProcessSend;
0169.00         exec sql fetch c1_curser into :d_S;
0170.00         w_countRcd = w_countRcd+1;
0171.00       enddo;
0172.00       if w_count > 1;
0173.00         endArray();
0174.00         endObject();
0175.00       endif;
0176.00
0177.00       // Close the cursors
0178.00       exec sql close c1_curser;
0179.00       exec sql close c2_curser;
0180.00
0181.00     Endsr;
0182.00     // ===================================================================
0183.00     //    Process the DB file
0184.00     // ===================================================================
0185.00     Begsr Z_ProcessSend;
0186.00
0187.00       if n_BgnKey = *off;
0188.00         // Save the page start key here
0189.00         n_BgnKey = *On;
0190.00         w_pageBgnKey = %char(d_s.lx_id) + '&&' + %trim(d_s.lx_clidno);
0191.00       endif;
0192.00       w_pageEndKey = %char(d_s.lx_id) + '&&' + %trim(d_s.lx_clidno);
0193.00       beginObject(' ');
0194.00
0195.00       // Execute any data manipulations here
0196.00
0197.00       // Add a JSON object for each field from IO/SQL
0198.00       addIntr('id' : d_s.lx_id);
```

```
0199.00       addchar('clidno': %trim(d_s.lx_clidno));
0200.00       addchar('clname': %trim(d_s.lx_clname));
0201.00       addchar('clsurname': %trim(d_s.lx_clsurname));
0202.00       addchar('cltitle': %trim(d_s.lx_cltitle));
0203.00       addchar('clphone': %trim(d_s.lx_clphone));
0204.00       addchar('clemail': %trim(d_s.lx_clemail));
0205.00       addchar('claddr1': %trim(d_s.lx_claddr1));
0206.00       addchar('claddr2': %trim(d_s.lx_claddr2));
0207.00       addchar('claddr3': %trim(d_s.lx_claddr3));
0208.00       addchar('clpcode': %trim(d_s.lx_clpcode));
0209.00       addchar('cllang': %trim(d_s.lx_cllang));
0214.00       endObject();
0216.00     Endsr;
0217.00     // ===================================================================
0218.00     //    Process Next page
0219.00     // ===================================================================
0220.00     Begsr z_PrcNxtPage;
0221.00
0222.00       n_BgnKey = *Off;
0223.00       clear d_S;
0224.00       w_countRcd = *zeros;
0225.00       exec sql close c1_nxt;
0226.00       exec sql close c2_nxt;
0227.00       s_statement =
0228.00       'select ID, CLIDNO, CLNAME, CLSURNAME, CLTITLE, CLPHONE, CLEMAIL,'+
0229.00       ' CLADDR1, CLADDR2, CLADDR3, CLPCODE, CLLANG from LXCLIENT  where'+
0230.00       ' clidno > ' + w_squote + %trim(d_PARM(2).s_Value) + w_squote +
0231.00       ' order by id , clidno';
0232.00
0233.00       w_stmtcnt = 'select count(*) from LXCLIENT  where clidno > ' +
0234.00       w_squote + %trim(d_PARM(2).s_Value) + w_squote;
0235.00
0236.00       exec sql prepare w_nxtcnt from :w_stmtcnt;
0237.00       exec sql prepare s_nxtstm from :s_statement1;
0238.00       if sqlstt <> c_SqlSuccess;
0239.00         ErrorJSON('No More entries to return');
0240.00         leavesr;
0241.00       endif;
0242.00
0243.00       // Process SQL data
0244.00       exec sql declare c2_nxt cursor for w_nxtcnt;
0245.00       exec sql open c2_nxt;
0246.00       // Load the count of records
0247.00       exec sql fetch c2_nxt into :w_count;
0248.00       exec sql declare c1_nxt scroll cursor for s_nxtstm;
0249.00       exec sql open c1_nxt;
0250.00       // Prepare an array for multiple record selection
0251.00       exec sql fetch c1_nxt into :d_S;
0252.00       if sqlstt = c_SqlSuccess or sqlstt < c_SqlEOF;
0253.00         // Build Response Container
0254.00         if w_count > 1;
0255.00           beginObject(' ');
0256.00           beginArray('LXCLIENT');
0257.00         endif;
0258.00       else;
0259.00         ErrorJSON('No more entries to return');
0260.00         leavesr;
0261.00       endif;
0262.00
0263.00       dow sqlstt < c_SqlEOF and w_countRcd < w_nbrrcd;
0264.00
0265.00         Exsr Z_ProcessSend;
0266.00         exec sql fetch c1_nxt into :d_S;
0267.00         w_countRcd = w_countRcd+1;
0268.00       enddo;
0269.00
```

```
0270.00       if w_count > 1;
0271.00         endArray();
0272.00         endObject();
0273.00       endif;
0274.00
0275.00     Endsr;
0276.00     // ===========================================================================
0277.00     //    Process Previous page
0278.00     // ===========================================================================
0279.00     Begsr z_PrcPrvPage;
0280.00
0281.00       n_BgnKey = *Off;
0282.00       clear d_S;
0283.00       w_countRcd = *zeros;
0284.00       exec sql close c1_prv;
0285.00       exec sql close c2_prv;
0286.00       s_statement =
0287.00       'select ID, CLIDNO, CLNAME, CLSURNAME, CLTITLE, CLPHONE, CLEMAIL,'+
0288.00       ' CLADDR1, CLADDR2, CLADDR3, CLPCODE, CLLANG from LXCLIENT  where'+
0289.00       ' clidno < ' + w_squote + %trim(d_PARM(2).s_Value) + w_squote +
0290.00       ' order by id desc , clidno desc';
0291.00
0292.00       w_stmtcnt = 'select count(*) from LXCLIENT  where clidno < ' +
0293.00       w_squote + %trim(d_PARM(2).s_Value) + w_squote;
0294.00
0295.00       exec sql prepare w_prvcnt from :w_stmtcnt;
0296.00       exec sql prepare s_prvstm from :s_statement1;
0297.00       if sqlstt <> c_SqlSuccess;
0298.00         ErrorJSON('No More entries to return');
0299.00         leavesr;
0300.00       endif;
0301.00
0302.00       // Process SQL data
0303.00       exec sql declare c2_prv cursor for w_prvcnt;
0304.00       exec sql open c2_prv;
0305.00       // Load the count of records
0306.00       exec sql fetch c2_prv into :w_count;
0307.00       exec sql declare c1_prv scroll cursor for s_prvstm;
0308.00       exec sql open c1_prv;
0309.00       // Prepare an array for multiple record selection
0310.00       exec sql fetch c1_prv into :d_S;
0311.00       if sqlstt = c_SqlSuccess or sqlstt < c_SqlEOF;
0312.00         // Build Response Container
0313.00         if w_count > 1;
0314.00           beginObject(' ');
0315.00           beginArray('LXCLIENT');
0316.00         endif;
0317.00         dow sqlstt < c_SQLEOF and w_countRcd <=w_nbrrcd;
0318.00
0319.00           w_countRcd = w_countRcd + 1;
0320.00           exec sql fetch next from c1_prv into :d_S;
0321.00         enddo;
0322.00       else;
0323.00         ErrorJSON('No more entries to return');
0324.00         leavesr;
0325.00       endif;
0326.00
0327.00       exec sql fetch prior from c1_prv into :d_S;
0328.00       w_countRcd = *zeros;
0329.00
0330.00       dow sqlstt < c_SqlEOF and w_countRcd < w_nbrrcd;
0331.00
0332.00         Exsr Z_ProcessSend;
0333.00         exec sql fetch prior from c1_prv into :d_S;
0334.00         w_countRcd = w_countRcd+1;
0335.00       enddo;
```

```
0336.00
0337.00      if w_count > 1;
0338.00        endArray();
0339.00        endObject();
0340.00      endif;
0341.00
0342.00    Endsr;
0343.00  /End-Free

0368.00    // ========================================================================
0369.00    //   LXR Generic Procedures
0370.00    // ========================================================================
0372.00  /copy qrpglesrc,lxrrestp
```

Above source can also be generated manually and BELOW specific changes in above source are for page handling.

1. Define one variable "w_CountRcd" for counting the records being processed in the current page/request. This is highlighted in yellow color.

2. add the compiler directives "/define LXR_NxtPage" and "/define LXR_PrvPage" before LXRRESTC to instruct the compiler that we are defining these subroutines to handle paging. This is highlighted in yellow color.

3. In z_PrcResponse subroutine, insert the statement "n_BgnKey=*Off" to always set this indicator to *Off in the beginning of the first/next/previos page requests. This is highlighted in yellow color.

4. Add the statement "Leavesr" when SQL doesn't return any data. This is highlighted in yellow color.

5. Add the source statements for handling the initial page in subroutine "Z_ProcessSend". This is highlighted in turquoise color.

6. Add the statement "Exsr Z_ProcessSend" before fetch and increment the record counter (i.e. variable w_countRcd) either before or after the fetch statement. These additions are highlighted in yellow color.

7. In subroutine "z_ProcessSend", check if the variable "n_BgnKey = *Off" (means this subroutine is called the first time in the current page request. If that's the case, set this indicator to *On and assign the current database keys in variable "w_pageBgnKey" with each key separated by &&. In this example, we only need one key because the file has "Id" as the only key but we have added "clidno" as well as "clname" just to depict how to send multiple keys. In the source member above, we have used only two keys (i.e. “id” and “clidno”). After the "Endif" statement, assign the current key values to "w_pageEndKey" variable the same way. This statement always gets executed so that it keeps the key values of last loaded record in the page whereas "w_pageBgnKey" is expected to contain only the first record key values. All this logic is highlighted in yellow color.

8. We then copy the whole content of restructured version of "z_PrcResponse" subroutine to create the new subroutine "z_PrcNxtPage".

9. In "z_PrcNxtPage" subroutine, we reset the variable w_countRcd to zero. This is highlighted in green color.

10. We change the cursor names from "c1_curser" and "c2_cursor" to "c1_nxt" and "c2_nxt" respectively all across the subroutine. We also rename the prepare statements from "w_stmtcnt" and "s_statement1" to "w_nxtcnt" and "s_nxtstm" respectively. These changes are highlighted in green color.

11. We add the where and order by clauses in SQL query. Please note that, when next page is requested, the values from "w_pageEndKey" as received from the consumer are set in d_parm data structure by the standard logic and when the previous page is requested, the values from "w_pageBgnKey" is set likewise. This is highlighted in green color.

12. The cursor declaration is now done with "scroll" keyword (even though it is required only in previous page request. The remaining logic of "z_PrcNxtPage" is kept unchanged. This is highlighted in green color.

13. We then copy the complete logic of "z_PrcNxtPage" to create another new subroutine "z_PrcPrvPage".

14. We change the cursor names from "c1_nxt" and "c2_nxt" to "c1_prv" and "c2_prv" respectively all across the subroutine. We also rename the prepare statements from "w_nxtcnt" and "s_nxtstm" to "w_prvcnt" and "s_prvstm" respectively. This is highlighted in pink color.

15. The order by clause is changed this time to fetch the data in descending order. This is done so that we can move forward by the specific number of records in the page. This is highlighted in pink color.

16. If the records are fetched, we add the additional logic to move forward by the specified number of records in one page. Then, read backwards and start loading from there. As the data is sorted in descending order, moving forwards means we are trying to reach the location from where the record loading should start and then use "fetch prior" clause and load the data. This is highlighted in pink color.

4.2.8.1 Paging in a REST Consumer built with MDRest4i

In order to test the working of next/previous page via REST consumer, set “w_pagetyp” variable to “N” or “P” to retrieve the next or previous page in the consumer code. The first call to GskConsume will set the values in “w_bgnkey”, “w_endkey” by what’s received from the REST service. As an example, below would be the logic to fetch the next page first time and previous page second time.

```
tg_InitializePointer = %paddr(Initialize);
tg_BuildReqPointer = %paddr(BuildRequest);
tg_ClosedownPointer  = %paddr(Closedown);

GskConsume();
// Process the next request for next page
Exsr Z_LoadCustomHdr;
w_pagetyp = 'N';
GskConsume();

// Process the next request for previous page
Exsr Z_LoadCustomHdr;
w_pagetyp = 'P';

GskConsume();
```

The subroutine “Z_LoadCustomHdr” should have the logic below to load the key values. The values in “w_nbrrcd” (i.e. number of records to return from the REST service) should be set in the beginning of the program:

```
Begsr Z_LoadCustomHdr;

  GetHdr('LXRPBGNKEY':w_BgnKey);
  GetHdr('LXRPENDKEY':w_EndKey);
  GetHdr('LXRNBRRCD':w_NbrRcdc);

  if w_nbrrcdc <> *blanks and %check('0123456789':%trim(w_nbrRcdc)) = 0;
    monitor;
      w_nbrrcd = %dec(%trim(w_nbrrcdc):4:0);
    on-error;
      w_nbrrcd = *zeros;
    endmon;
  endif;

Endsr;
```

The last change is required in the “Initialize” procedure where “AddHttpHeader” procedure should be used as highlighted below so that the REST service receives these values and processes the request accordingly.

```
0180.00 p Initialize      b                   export
```

```
0181.00 d Initialize      pi
0182.00  /Free
0183.00
0184.00    //Set Import values
0185.00    wg_contentType = 'application/json; charset=UTF-8';
0185.00    wg_httpsMethod = 'GET';
0185.00
0185.00    wg_maxAttempt = 25;
0185.00    wg_mSecDelay = 500000;
0185.00
0186.00    wg_hostName = 'dev.mdcms.ch';
0187.00    wg_PortNumber = 4522;
0187.00    wg_servicePath = '/tut/CLIENTSPG';
0188.00    wg_serverIP= ' ';
0188.01    w_nbrrcd = 4;    // Means we are instructing the REST serice to return 4
records
0189.00    AddHttpHeader('LXRPBGNKEY':%trim(w_bgnkey));
0190.00    AddHttpHeader('LXRPENDKEY':%trim(w_endkey));
0191.00    AddHttpHeader('LXRPAGETYP':%trim(w_pagetyp));
0192.00    AddHttpHeader('LXRNBRRCD':%char(w_nbrrcd));
0193.00
```

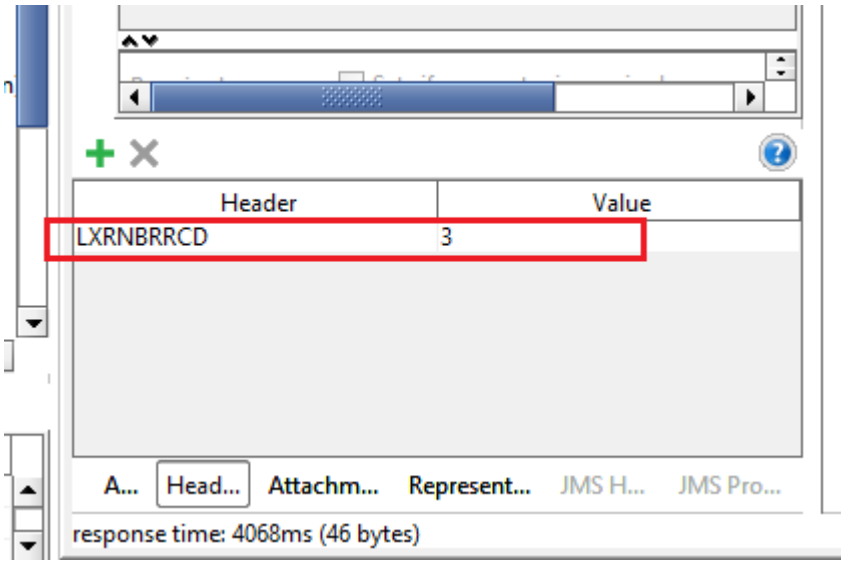
We now compile the generated REST service to create the object in the library mapped on webserver.

4.2.8.2 Test Pagination in the service from SOAPUI.

Use the webservice URL like below in SOAPUI for calling the REST service:

http://yourserver:xxxx/yourlib/clientspg

Click on “Headers” at the bottom of SOAPUI window as highlighted and then click on “+” button to add two custom header “LXRNBRRCD” as appearing in below screenshot. Here, we are requesting the REST API to return only 3 records in the page:



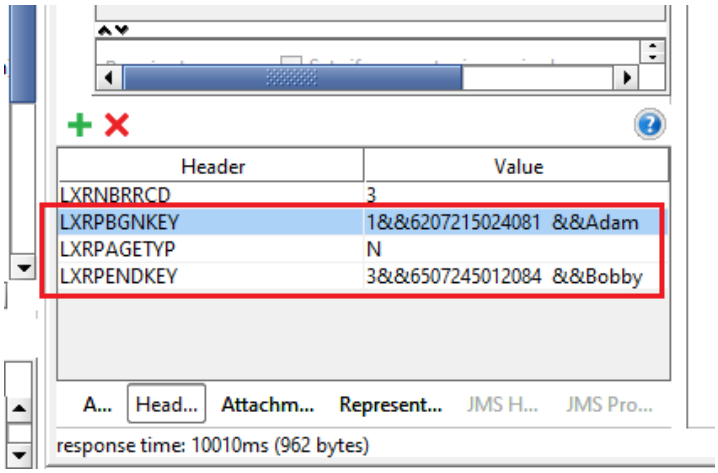
Below is the response:

```
{"LXCLIENT": [
      {
      "id": "1",
      "clidno": "6207215024081",
      "clname": "Adam",
      "clsurname": "ArtichokePaAT",
      "cltitle": "Mr",
      "clphone": "0115556666",
```

```
      "clemail": "adam@vegies.co.za",
      "claddr1": "1 Tree Road",
      "claddr2": "Plantville",
      "claddr3": "6068 SLIDING PaAT DOOR WHITE",
      "clpcode": "RIGHT",
      "cllang": ""
   },
      {
      "id": "2",
      "clidno": "90**********",
      "clname": "Brandon",
      "clsurname": "Whitehead",
      "cltitle": "Mr",
      "clphone": "0115556666",
      "clemail": "brandon@demo.com",
      "claddr1": "1 Address Street",
      "claddr2": "Post Town",
      "claddr3": "",
      "clpcode": "9999",
      "cllang": ""
   },
      {
      "id": "3",
      "clidno": "6507245012084",
      "clname": "Bobby",
      "clsurname": "Butcher",
      "cltitle": "Mr",
      "clphone": "0845557892",
      "clemail": "bb@beefsurprise.co.za",
      "claddr1": "1 Brown Bear Boulevard",
      "claddr2": "Animal Town",
      "claddr3": "",
      "clpcode": "9993",
      "cllang": ""
   }
]}
```


If we now click on “Headers” tab at the bottom of response window, it shows the page begin and end key values in LXRPBGNKEY and LXRPENDKEY custom headers. We will now use these keys to fetch the next page.

To fetch the next page, we add three more custom headers. The header “LXRPAGETYP” is set to “N” means return the next page and the headers “LXRPBGNKEY”/”LXRPENDKEY” are setup with the value which was returned in response above. This setting is shown in below screenshot:

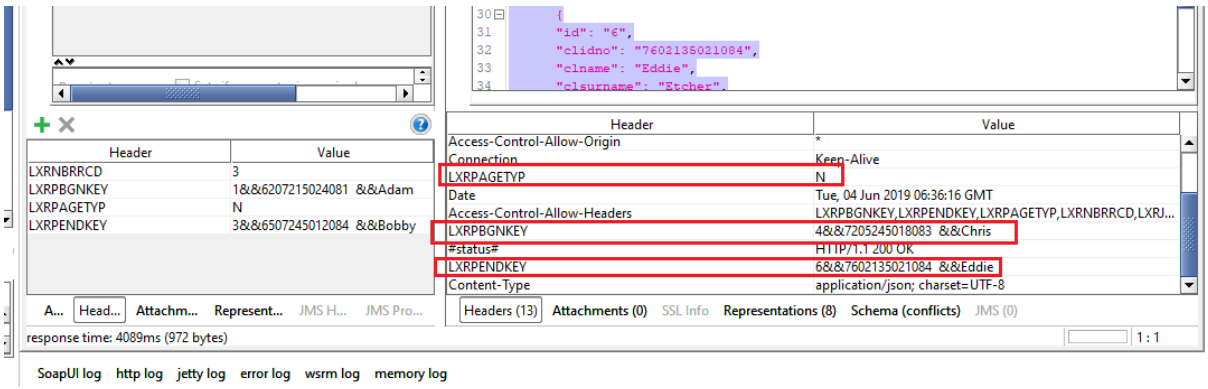


Below is the response in SOAPUI:

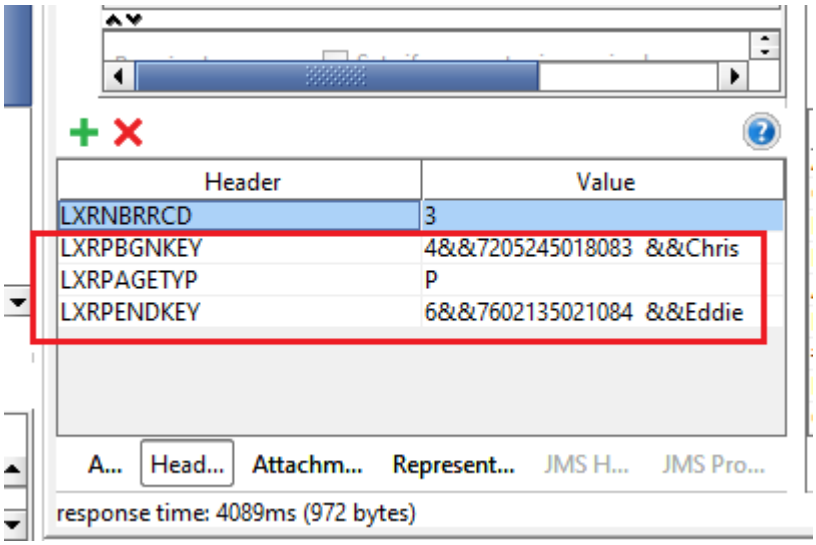

```
{"LXCLIENT": [
      {
      "id": "4",
      "clidno": "7205245018083",
      "clname": "Chris",
      "clsurname": "Consultant",
      "cltitle": "Mr",
```

```
      "clphone": "0845557834",
      "clemail": "chris@askus.co.za",
      "claddr1": "2 Cougar Crescent",
      "claddr2": "Animal Town",
      "claddr3": "",
      "clpcode": "9998",
      "cllang": ""
   },
      {
      "id": "5",
      "clidno": "8103245012083",
      "clname": "David",
      "clsurname": "O'Brian //Sullivan\\\\\\Johnson\"",
      "cltitle": "Mr",
      "clphone": "0814562347",
      "clemail": "david@chaufferservices.co.za",
      "claddr1": "5 Dinosaur Drive",
      "claddr2": "Animal Town",
      "claddr3": "",
      "clpcode": "9998",
      "cllang": ""
   },
      {
      "id": "6",
      "clidno": "7602135021084",
      "clname": "Eddie",
      "clsurname": "Etcher",
      "cltitle": "Mr",
      "clphone": "0875553478",
      "clemail": "eddie@edgyengraving.co.za",
      "claddr1": "8 Elephant Avenue",
      "claddr2": "Animal Town",
      "claddr3": "",
      "clpcode": "9998",
      "cllang": ""
   }
]}
```


Below are the received header values. As we can see, “LXRPAGETYP” contains the same value which was supplied for the next page and the headers “LXRPBGNKEY” and “LXRPENDKEY” contain the begin and end keys of the received page records.



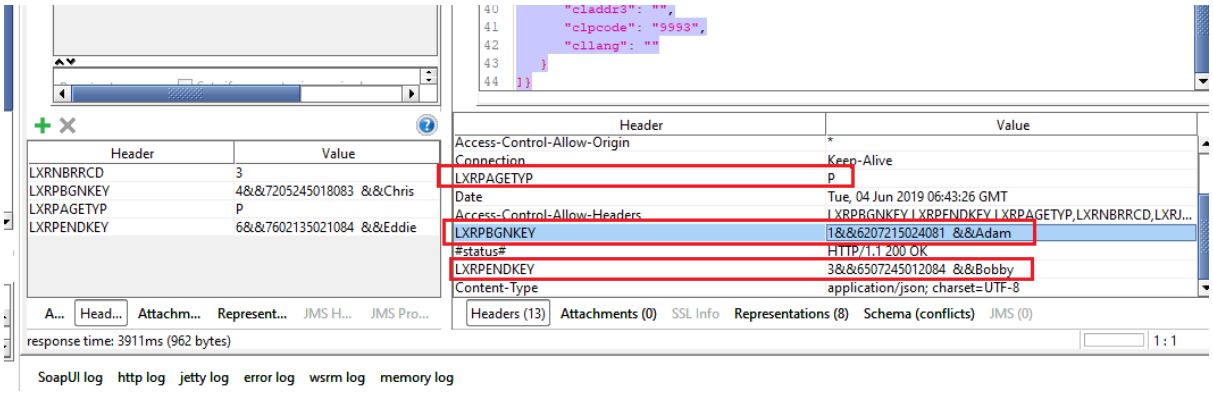
We will now fetch the previous page. For this, we have to set the custom header “LXRPAGETYP” to “P” and LXRPBGNKEY, LXRPENDKEY are to be set what we received on the current page as shown in above screenshot. Below is the final setting of headers for sending the previous page request:



We now submit the request and below response is received:


```
{"LXCLIENT": [
      {
      "id": "1",
      "clidno": "6207215024081",
      "clname": "Adam",
      "clsurname": "ArtichokePaAT",
      "cltitle": "Mr",
      "clphone": "0115556666",
      "clemail": "adam@vegies.co.za",
      "claddr1": "1 Tree Road",
      "claddr2": "Plantville",
      "claddr3": "6068 SLIDING PaAT DOOR WHITE",
      "clpcode": "RIGHT",
      "cllang": ""
   },
      {
      "id": "2",
      "clidno": "90**********",
      "clname": "Brandon",
      "clsurname": "Whitehead",
      "cltitle": "Mr",
      "clphone": "0115556666",
      "clemail": "brandon@demo.com",
      "claddr1": "1 Address Street",
      "claddr2": "Post Town",
      "claddr3": "",
      "clpcode": "9999",
      "cllang": ""
   },
      {
      "id": "3",
      "clidno": "6507245012084",
      "clname": "Bobby",
      "clsurname": "Butcher",
      "cltitle": "Mr",
      "clphone": "0845557892",
      "clemail": "bb@beefsurprise.co.za",
      "claddr1": "1 Brown Bear Boulevard",
      "claddr2": "Animal Town",
      "claddr3": "",
      "clpcode": "9993",
      "cllang": ""
   }
]}
```


The received response headers are shown in below screenshot:



This way, using the received values in LXRPBGNKEY and LXRPENDKEY headers and setting the header LXRPAGETYP to “P” or “N”, we can fetch the previous or next page. The header LXRNBRRCD allows to set how many records you want to fetch. If this header is not specified, the default value is 100 records in one request.

4.3 Command MDRGENCNS – Consumer Examples

4.3.1 Consumer GET Method Program

Now we will create a consumer program, which will make REST request to the Provider service for the program we created above i.e. CLIENTS

First of all prompt the command MDRST/MDRGENCNS from the command line and use below parameters to run the command:

```
                  MDRest4i Generator - Consumer (MDRGENCNS)

 Type choices, press Enter.

 Target Library . . . . . .   MDRTUTLIB     Name
Target Source File . . . . . .  QRPGLESRC     Name
Target Source Member . . . . .  CLIENTCON     Name
Member Text  . . . . . .  'REST Pgm Generated by MDRest4i'

SSL/Non SSL? . . . . . .  N             Y=SSL , N-Non SSL
Single or Multiple Service . . .  S             M=Multiple, S=Single
```

- If we want to create SSL consumer, supply “Y” in “SSL/Non SSL” parm. Otherwise ‘N” set as default for creation the NON SSL consumer. After that we can create single or multiple consumer program using this. So if we want to create multiple consumer program select “M” for multiple. You can create 10 consumer program using this. S is default set for creating the single consumer program then press enter to see next parameters.

```
HTTP Method T=PATCH D=DEL  . . .  G             G=GET, P=PUT, S=POST
```

- Using this command we can create consumer program using HTTP method GET, PUT, POST, PATCH and DELETE. So if we can supply these values for creating the consumer programs (G= GET, P= PUT, S=POST, T=PATCH and D=DELETE. G is set as default for creating GET consumer. Press enter to see next parameters.

```
Parse response method  . . . . .  USR           IFS, DBF, USR
```

- If we want to write incoming response to an IFS file, specify the “IFS” and provide the complete IFS path where we want to write the incoming response, if we want to write response to MDRJSONF file so we need to supply “DBF” and also provide the library name where MDRJSONF

file exist. Otherwise it set “USR” by default. But here we don’t want to write incoming response to DBF or IFS that’s why we used USR. Press enter to see the next parameters.

```
URL for Consumer Service . . . . > 'http://yourserver:yourport/yourlib/clients'
```

- Specify the complete URL with the http:// or https:// as per above then press page down key to see next parameters.

```
IFS Path to store the Output . . > '/MDRest4i/logs/'


 File name on the IFS Directory   > TESTGEN
 List of parameters:
   Parameter Name . . . . . . . . > ID
   Length . . . . . . . . . . . . > 9              1-999999
   Data Type  . . . . . . . . . . > B              A, D, I, N, P, S, T, Z, *, B
   Decimal Positions  . . . . . . > 0              0-63
   Parameter Type . . . . . . . .   P              P=Parm, E=Entry Parm, B=Both
              + for more values
```

In the target library we will specify the library name where member should be generated.

In the target source file we will specify the source file name. In the target source member we will specify the source member name. In this example, we are going to use a GET method. This is because the Provider/Service we are sending the request to, only responds to an HTTP GET method request. So the parameters below should be set “G”.

```
HTTP Method T=PATCH D=DEL  . . . G              G=GET, P=PUT, S=POST
```

In this consumer we don’t want to write response in DB file (MDRJSONF) or IFS file. Therefore I Have used USR method below:

```
Parse response method  . . . . .   USR           IFS, DBF, USR
```

In the “URL for Consumer Service” we will have to put the url path of the Provider service.

```
URL for Consumer Service . . . . > http://yourserver:yourport/yourlib/clients'
```

In the “IFS Path to store the Output” we entere the IFS path loction where the response output log file will be written.

```
IFS Path to store the Output . . > '/MDRest4i/youruser/Clientsdtl'
```

Here we are supplying one parameter.

```
List of parameters:
  Parameter Name . . . . . . . . > ID
  Length . . . . . . . . . . . . > 9              1-999999
  Data Type  . . . . . . . . . . > B              A, D, I, N, P, S, T, Z, *, B
```

```
Decimal Positions  . . . . . .   0            0-63
Parameter Type . . . . . . . .   P            P=Parameter, E=Entry Parameter
```

A type "P" parameter create a work field with "w_" prefix in the code, and also creates the query string in the wg_urlparm variable of the initialize procedure. By default, the numeric variable is initialized to 9999999 and the developer is expected to change this value as per the logic.

Once the command completes the execution, the source member below is generated under the specified source file and library combinations and it gets compiled as well. The name of the program is CLIENTSCON.

```
0001.00 h dftactgrp(*no) actgrp(*new)
0003.00 h bnddir('LXRGLOBAL': 'BNDZIP')
0004.00  *****************************************************************
0005.00  *  MDRest4i SSL Consumer Template GET JSON
0006.00  *****************************************************************
0007.00
0008.00  /copy qrpglesrc,mdrestdfn
0013.00
0021.00 d Initialize      pr
0022.00
0023.00 d BuildRequest    pr
0024.00 d CloseDown       pr
0025.00
0026.00
0027.00 d  w_id           s              9B 0
0028.00
0029.00  /free
0030.00
0031.00    InitVariant();
0033.00    tg_InitializePointer = %paddr(Initialize);
0034.00    tg_BuildReqPointer = %paddr(BuildRequest);
0035.00    tg_ClosedownPointer  = %paddr(Closedown);
0036.00    GskConsume();
0037.00
0038.00    if wg_httpStatus = '200';
0039.00
0040.00      // Change above condition to appropriate success status codes as
0041.00      // applicable to your REST service and add the processing logic here
0042.00
0043.00      // You may also call "GetErrorWarnings" procedure with one parameter
0044.00      // of 1024a attribute to get any errors/warnings reported in execution
0045.00
0046.00    Endif;
0047.00
0048.00    // Cleanup the memory allocated dynamically from heap
0036.00    cleantree();
0050.00
0051.00    *Inlr = *On;
0052.00
0053.00  /End-Free
0054.00  *-------------------------------------------------------------
0055.00  * End of Mainline Section
0056.00  *-------------------------------------------------------------
0057.00
0058.00  * Procedure Interfaces
0059.00  *-------------------------------------------------------------
0095.00
0096.00 p CloseDown       b                   export
0097.00 d CloseDown       pi
0098.00  /Free
0099.00
0100.00    //Enter service closedown instructions here
0101.00  /End-Free
```

```
0102.00 p CloseDown       e
0103.00
0104.00 p BuildRequest    b                   export
0105.00 d BuildRequest    pi
0106.00  /Free
0107.00
0108.00    //Enter the request body  here
0109.00  /End-Free
0110.00 p BuildRequest    e
0111.00
0112.00 p Initialize      b                   export
0113.00 d Initialize      pi
0114.00  /Free
0115.00
0116.00    //Set parameter values
0117.00    w_ID = 9999999;
0118.00    // Set Import Values
0119.00    wg_contentType = 'application/json; charset=UTF-8';
0120.00    wg_httpsMethod = 'GET';
0121.00
0122.00    wg_maxAttempt = 25;
0123.00    wg_mSecDelay = 500000;
0124.00
0125.00    wg_hostName = 'dev.mdcms.ch';
0126.00    wg_serverIP= ' ';
0127.00    wg_portNumber = 2525;
0128.00
0129.00    wg_servicePath = '/tut/clients';
0130.00    wg_urlparm = 'id='+ %char(w_ID);
0131.00
0132.00    // Set below indicator to *Off if you don't want to save logs in IFS
0133.00    ng_saveRestSwitch = *On;
0134.00    wg_saveRESTpath ='/MDRest4i/logs/client.txt';
0135.00
0136.00    // The setting "ng_ssl=*On" enables SSL request. Otherwise, it's non SSL
0137.00    ng_ssl = *Off;
0138.00
0139.00    // If the REST service requires authentication, set ng_authsend = *On
0140.00    // and then use one of the three subsequent variables to send auth info.
0141.00    // In order to send the basic authentication via userId and pwd, set the
0142.00    // variables wg_username and wg_password. If however, it's the "Bearer"
0143.00    // token or some other auth string, set the value in "wg_authstring".
0144.00    // e.g. wg_authstring = Bearer <token value>
0145.00
0146.00    ng_authsend = *on;      //Process basic authorization
0146.01    wg_userName = 'user1';  //User ID for basic authorization
0146.02    wg_password = 'mypwd';  //Password for basic authorization
0149.00    wg_authstring = *blanks;
0150.00
0151.00    // If the REST service is accessible via proxy, set ng_proxy = *on
0152.00    // and set the proxy details in subsequent variables. If the proxy
0153.00    // host is in IP address format, set the value in "wg_proxyIP" and if
0154.00    // it's the host name format, set the hostname in the "wg_proxyhost"
0155.00    // variable without any http/https prefix and there shouldn't be any
0156.00    // slash before or after the host name or IP address. The port number
0157.00    // of the proxy host should be set in "wg_proxyport" variable below.
0158.00    // If the proxy is on SSL channel, set "ng_proxySSL" to *On.
0159.00    ng_proxy = *Off;
0160.00    ng_proxyssl = *Off;
0161.00    wg_proxyhost = *blanks;
0162.00    wg_proxyIp = *blanks;
0163.00    wg_proxyport = *zeros;
0164.00    wg_proxyusr = *blanks;
0165.00    wg_proxypwd = *blanks;
0166.00
0119.00  // If you want to load config via MRCNSDTLF file, please uncomment
```

```
0120.00  // the below line. In this case you need to make sure you have loaded
0121.00  // (Host, Port, path etc.) in the MRCNSDTLF file with the key of This
0122.00  // Program name. Add below file in the 'F' spec in this program
0123.00  // mrcnsdtlf if   e           k disk    Usropn
0124.00  // /copy qrpglesrc,mrsetcon
0125.00
0167.00  /End-Free
0168.00 p Initialize      e
0169.00
0170.00  * ------------------------------------------------------------------------
0171.00  *  Procedure - FixReprocess
0172.00  * ------------------------------------------------------------------------
0173.00 p FixReprocess    b                   export
0174.00 d FixReprocess    pi
0175.00  /free
0176.00  /end-free
0177.00 p FixReProcess    e
0178.00  * ------------------------------------------------------------------------
0179.00  *  Procedure - InitVariant
0180.00  * ------------------------------------------------------------------------
0181.00 p InitVariant     b
0182.00 d InitVariant     pi
0183.00  /free
0184.00    w_OpenCurly = %char(c_OpenCurly);
0185.00    w_CloseCurly = %char(c_CloseCurly);
0186.00    w_OpenSquare = %char(c_OpenSquare);
0187.00    w_CloseSquare = %char(c_CloseSquare);
0188.00    w_EscapeChar = %char(c_EscapeChar);
0189.00    w_Power = %char(c_Power);
0190.00    w_Tilde1 = %char(c_Tilde1);
0191.00    w_Exclaim = %char(c_Exclaim);
0192.00    w_Hash = %char(c_Hash);
0193.00    w_Pipe1 = %char(c_Pipe1);
0194.00    w_Accent = %char(c_Accent);
0195.00    w_Doller = %char(c_Doller);
0196.00    w_AtSign = %char(c_AtSign);
0197.00
0198.00  /end-free
0199.00 p InitVariant     e
```

You can also edit the IFS path location and url for the Provider program, plus the query parameters
After the execution of consumer program, you can review the response in the IFS location specified in the program as below.

```
************Beginning of data**************
###REQUEST##START###

 ###REQUEST##HTTP-HEADER###
Request Length = 199

 ###REQUEST##DETAILS###
GET http://yourserver:yourport/yourlib/clients?id=1 HTTP/1.1
Accept-Encoding: deflate
Host: yourserver:yourport
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)



 ###REQUEST##END###

 ###RESPONSE##START###
Response Status = 200
Response Length = 626

 #RESPONSE#HTTP-HEADER#
HTTP/1.1 200 OK
Date: Thu, 15 Mar 2018 04:01:07 GMT
Server: Apache
Access-Control-Allow-Origin: *
```

```
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept, RRN
Keep-Alive: timeout=300, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/json; charset=UTF-8

126

#RESPONSE#JSON-BODY#
{
"ID": "1"
,"CLIDNO": "6207215024081"
,"CLNAME": "Adam"
,"CLSURNAME": "Artichoke"
,"CLTITLE": "Mr"
,"CLPHONE": "0115556666"
,"CLEMAIL": "adam@vegies.co.za"
,"CLADDR1": "1 Tree Road"
,"CLADDR2": "Plantville"
,"CLADDR3": ""
,"CLPCODE": "1002"
,"CLLANG": ""
}
###RESPONSE##END###
 ************End of Data********************
```

Colour Legend:

Orange	Header Specifications
Blue	Copybooks for the MDRest4i engines
Blue	Setting of the REST service path and port number
Green	IFS file name with the path where consumer log will be saved
Cyan	Parameters specified in the command and used in the URL as parms

4.3.2 Consumer POST method using IFS file

MDRESt4i has functions that will automatically create the outgoing JSON (for a requestBody) from an IFS file, and write the incoming JSON (response) to an IFS file.

Here is an example of the MDRGENCNS command used to build a consumer template that uses this IFS concept.

```
MDRest4i Generator - Consumer (MDRGENCNS)

 Type choices, press Enter.

 Target Library . . . . . .   QTEMP         Name
Target Source File . . . . . .   QRPGLESRC     Name
Target Source Member . . . . . > IFSCNS        Name
Member Text  . . . . . .   'REST Pgm Generated by MDRest4i'
```

```
SSL/Non SSL? . . . . . . . . . . .   N             Y=SSL , N-Non SSL
Single or Multiple Service . . .   S             M=Multiple, S=Single
```

- If we want to create SSL consumer, supply “Y” in “SSL/Non SSL” parm. Otherwise ‘N” set as default for creation the NON SSL consumer. After that we can create single or multiple consumer program using this. So if we want to create multiple consumer program select “M” for multiple. We can create 10 consumer programs using this. “S” is default set for creating the single consumer program then press enter to see next parameters.

```
HTTP Method T=PATCH D=DEL  . . . > S             G=GET, P=PUT, S=POST
```

- Using this command we can create consumer program using HTTP method GET, PUT, POST, PATCH and DELETE. So if we can supply these values for creating the consumer programs (G= GET, P= PUT, S=POST, T=PATCH and D=DELETE. G is set as default for creating GET consumer.

```
Build request body method  . . . > IFS           IFS, DBF, USR
```

- Here are three provisions for sending request body to the API. If we want to send request body via IFS file, need to specify “IFS” in “Build request body method and provide the complete IFS path name where JSON available. If we want send request body via MDRJSONF DB file, need to specify “DBF” in this case we need to supply library name also where MDRJSONF file exist with the loaded JSON request.

```
 IFS File for Requestbody . . . .   /home/MDRest4i/Testjson.json
```

- Provide above the complete IFS path including the file name as per above.

```
 Parse response method  . . . . . > IFS           IFS, DBF, USR
```

- If we want to write incoming response to an IFS file, specify the “IFS” and provide the complete IFS path where we want to write the incoming response, if we want to write response to MDRJSONF file so we need to supply “DBF” and also provide the library name where MDRJSONF file exist.

```
IFS File for Response  . . . . .   /home/MDRest4i/Testjson.json
URL for Consumer Service . . . .   http://yourhost:yourport/yourlib/clients
IFS Path to store the Output . .     /MDRest4i/logs/IFSCNS.txt
File name on the IFS Directory
```

- Above we need to supply complete IFS path for writing the incoming response. Provide the complete URL with http or https. After that supply the IFS path where we want to write hold incoming request.

In the target library we will specify the library name where member should be generated.

In the target source file we will specify the source file name. In the target source member we will specify the source member name. In this example, we are going to use a POST method. This is because the Provider/Service accepting the request and sending back responds to an HTTP POST method request. So the parameters below should be set “S”.

```
HTTP Method T=PATCH D=DEL  . . . S            G=GET, P=PUT, S=POST
```

In this example we want process the request body through the IFS file so below parameter should be set “IFS” and also need to specify the complete IFS path including file name as per below:

```
Build request body method  . . . > IFS          IFS, DBF, USR
IFS File for Requestbody . . . .   /home/MDRest4i/Testjson.json
```

In this example we want write incoming response to an IFS file so below parameter should be set “IFS” and also need to specify the complete IFS path including the file name as per below:

```
Parse response method  . . . . . > IFS          IFS, DBF, USR
IFS File for ReSponse .  . . . .   /home/MDRest4i/Testjson.json
```

In the “URL for Consumer Service” we will have to put the url path of the Provider service.

```
URL for Consumer Service . . . . > http://yourserver:yourport/yourlib/clients'
```

In the “IFS Path to store the Output” we enter the IFS path location where the response output log file will be written.

```
IFS Path to store the Output . . > '/MDRest4i/youruser/Clientsdtl'
```

Press Enter

The generated code is below. The relevant code for IFS processing is highlighted.

```
0001.00 h dftactgrp(*no) actgrp(*new)
0002.00 h bnddir('LXRGLOBAL': 'BNDZIP')
0003.00  ************************************************************
0004.00  *  MDRest4i SSL Consumer Template GET JSON
0005.00  ************************************************************
0006.00
0007.00  /copy qrpglesrc,mdrestdfn
0008.00
0009.00 d Initialize      pr
0010.00
0011.00 d BuildRequest    pr
0012.00 d CloseDown       pr
0013.00
0014.00 d w_data          s        5242880a
0015.00  /free
0016.00
0017.00    InitVariant();
0018.00    tg_InitializePointer = %paddr(Initialize);
0019.00    tg_BuildReqPointer = %paddr(BuildRequest);
0020.00    tg_ClosedownPointer  = %paddr(Closedown);
0021.00    ng_cleanup = *Off;
0022.00    GskConsume();
0023.00
0024.00    if wg_httpStatus = '200';
0025.00
```

```
0026.00       // Change above condition to appropriate success status codes as
0027.00       // applicable to your REST service and add the processing logic here
0028.00
0029.00       // You may also call "GetErrorWarnings" procedure with one parameter
0030.00       // of 1024a attribute to get any errors/warnings reported in execution
0031.00
0032.00       // Write response to an IFS file
0033.00       GetBody(w_data);
0034.00       writeIFS('/home/MDRest4i/Testjson.json':w_data:%len(%trim(w_data)));
0035.00       Memcleanup();
0036.00     Endif;
0037.00
0038.00     // Cleanup the memory allocated dynamically from heap
0039.00     cleantree();
0040.00
0041.00     *Inlr = *On;
0042.00
0043.00   /End-Free
0044.00   *-------------------------------------------------------------
0045.00   * End of Mainline Section
0046.00   *-------------------------------------------------------------
0047.00
0048.00   * Procedure Interfaces
0049.00   *-------------------------------------------------------------
0050.00
0051.00 p CloseDown       b                   export
0052.00 d CloseDown       pi
0053.00   /Free
0054.00
0055.00     //Enter service closedown instructions here
0056.00   /End-Free
0057.00 p CloseDown       e
0058.00
0059.00 p BuildRequest    b                   export
0060.00 d BuildRequest    pi
0061.00 d wl_file         s            200a
0062.00 d wl_filedata     s        5242880a
0063.00 d wl_fileId       s             10i 0
0064.00 d wl_datalen      s             10i 0
0065.00   /Free
0066.00
0067.00     //Enter the request body  here
0068.00     wl_file = '/home/MDRest4i/Testjson.json';
0069.00     readIFS(wl_file:wl_filedata:%size(wl_filedata));
0070.00     wl_datalen = %len(%trim(wl_filedata));
0071.00     wlong(wl_filedata:wl_datalen);
0072.00
0073.00   /End-Free
0074.00 p BuildRequest    e
0075.00
0076.00 p Initialize      b                   export
0077.00 d Initialize      pi
0078.00   /Free
0079.00
0080.00     // Set Import Values
0081.00     wg_contentType = 'application/json; charset=UTF-8';
0082.00     wg_httpsMethod = 'POST';
0083.00
0084.00     wg_maxAttempt = 25;
0085.00     wg_mSecDelay = 500000;
0086.00
0087.00     wg_hostName = 'dev.mdcms.ch';
0088.00     wg_serverIP= ' ';
0089.00     wg_portNumber = 2525;
0090.00
0091.00     ng_parse = *Off;
```

```
0092.00     wg_servicePath = '/tut/ifsprd';
0093.00     wg_urlParm = '      ';
0094.00
0095.00     // Set below indicator to *Off if you don't want to save logs in IFS
0096.00     ng_saveRestSwitch = *Off;
0097.00     wg_saveRESTpath ='/MDRest4i/logs/IFSCNS.txt';
0098.00
0099.00
0100.00     // The setting "ng_ssl=*On" enables SSL request. Otherwise, it's non SSL
0101.00     ng_ssl = *Off;
0102.00
0103.00     // If the REST service requires authentication, set ng_authsend = *On
0104.00     // and then use one of the three subsequent variables to send auth info.
0105.00     // In order to send the basic authentication via userId and pwd, set the
0106.00     // variables wg_username and wg_password. If however, it's the "Bearer"
0107.00     // token or some other auth string, set the value in "wg_authstring".
0108.00     // e.g. wg_authstring = Bearer <token value>
0109.00
0110.00     ng_authsend = *off;
0111.00     wg_username = *blanks;
0112.00     wg_password = *blanks;
0113.00     wg_authstring = *blanks;
0114.00
0115.00     // If the REST service is accessible via proxy, set ng_proxy = *on
0116.00     // and set the proxy details in subsequent variables. If the proxy
0117.00     // host is in IP address format, set the value in "wg_proxyIP" and if
0118.00     // it's the host name format, set the hostname in the "wg_proxyhost"
0119.00     // variable without any http/https prefix and there shouldn't be any
0120.00     // slash before or after the host name or IP address. The port number
0121.00     // of the proxy host should be set in "wg_proxyport" variable below.
0122.00     // If the proxy is on SSL channel, set "ng_proxySSL" to *On.
0123.00     ng_proxy = *Off;
0124.00     ng_proxyssl = *Off;
0125.00     wg_proxyhost = *blanks;
0126.00     wg_proxyIp = *blanks;
0127.00     wg_proxyport = *zeros;
0128.00     wg_proxyusr = *blanks;
0129.00     wg_proxypwd = *blanks;
0130.00
0131.00   // If you want to load config via MRCNSDTLF file, please uncomment
0132.00   // the below line. In this case you need to make sure you have loaded
0133.00   // (Host, Port, path etc.) in the MRCNSDTLF file with the key of This
0134.00   // Program name. Add below file in the 'F' spec in this program
0135.00   // mrcnsdtlf if   e           k disk    Usropn
0136.00   // /copy qrpglesrc,mrsetcon
0137.00
0138.00   /End-Free
0139.00 p Initialize      e
0140.00
0141.00   * ------------------------------------------------------------------------
0142.00   *  Procedure - FixReprocess
0143.00   * ------------------------------------------------------------------------
0144.00 p FixReprocess    b                   export
0145.00 d FixReprocess    pi
0146.00   /free
0147.00   /end-free
0148.00 p FixReProcess    e
0149.00   * ------------------------------------------------------------------------
0150.00   *  Procedure - InitVariant
0151.00   * ------------------------------------------------------------------------
0152.00 p InitVariant     b
0153.00 d InitVariant     pi
0154.00   /free
0155.00     w_OpenCurly = %char(c_OpenCurly);
0156.00     w_CloseCurly = %char(c_CloseCurly);
0157.00     w_OpenSquare = %char(c_OpenSquare);
```

```
0158.00     w_CloseSquare = %char(c_CloseSquare);
0159.00     w_EscapeChar = %char(c_EscapeChar);
0160.00     w_Power = %char(c_Power);
0161.00     w_Tilde1 = %char(c_Tilde1);
0162.00     w_Exclaim = %char(c_Exclaim);
0163.00     w_Hash = %char(c_Hash);
0164.00     w_Pipe1 = %char(c_Pipe1);
0165.00     w_Accent = %char(c_Accent);
0166.00     w_Doller = %char(c_Doller);
0167.00     w_AtSign = %char(c_AtSign);
0168.00
0169.00   /end-free
0170.00 p InitVariant     e
```

I have highlighted the code of IFS processing with the bright green colour.

For a detailed explanation of this please see the Reference-Guide's section 3.2

4.3.3 Consumer POST Method Program using SSL

Now we will create a consumer program, which will make REST request to the Provider service for the program over SSL channel

First of all prompt the command MDRST/MDRGENCNS from the command line and use below parameters to run the command:

```
                    MDRest4i Generator - Consumer (MDRGENCNS)

 Type choices, press Enter.

 Target Library . . . . . .   MDRTUTLIB     Name
 Target Source File . . . . .   QRPGLESRC     Name
 Target Source Member . . . . .   CLTSSLCNS     Name
 Member Text  . . . . . .   'REST Pgm Generated by MDRest4i'

 SSL/Non SSL? . . . . . > Y             Y=SSL , N-Non SSL
 Single or Multiple Service . . .   S             M=Multiple, S=Single
```

- If we want to create SSL consumer, supply "Y" in "SSL/Non SSL" parm. Otherwise 'N" set as default for creation the NON SSL consumer. After that you can create single or multiple consumer program using this. So if you want to create multiple consumer program select "M" for multiple. You can create 10 consumer program using this. S is default set for creating the single consumer program then press enter to see next parameters.

```
 HTTP Method T=PATCH D=DEL  . . . > S             G=GET, P=PUT, S=POST
```

- Using this command we can create consumer program using HTTP method GET, PUT, POST, PATCH and DELETE. So if we can supply these values for creating the consumer programs (G= GET, P= PUT, S=POST, T=PATCH and D=DELETE. G is set as default for creating GET consumer.

```
 Build request body method  . . .   USR           IFS, DBF, USR
```

- Here are three provisions for sending request body to the API. If we want to send request body via IFS file, need to specify "IFS" in "Build request body method" in this case we need to supply the complete IFS path name where JSON available. If we want send request body via MDRJSONF DB file, need to specify "DBF" in this case we need to supply library name also where MDRJSONF file exist with the loaded JSON request.

```
 Parse response method  . . . . .   USR           IFS, DBF, USR
```

- If we want to write incoming response to an IFS file, specify the "IFS" and provide the complete IFS path where we want to write the incoming response, if we want to write response to MDRJSONF file so we need to supply "DBF" and also provide the library name where MDRJSONF file exist.

```
URL for Consumer Service . . . . > 'http://yourserver:yourport/yourlib/mdrecho'
```

- Above we need to supply the URL path of the Provider service. As per above

```
IFS Path to store the Output . . > '/MDRest4i/youruser/Clientssldtl'
File name on the IFS Directory     TESTSSLCNS.txt
```

- Above we need to specify the path and IFS file name.

In the target library we will specify the library name where member should be generated.

In the target source file we will specify the source file name. In the target source member we will specify the source member name. In this example, we are going to use a POST method. This is because the Provider/Service we are sending the request to, only responds to an HTTP POST method request. So the parameters below should be set "S".

```
HTTP Method T=PATCH D=DEL  . . . S             G=GET, P=PUT, S=POST
```

In the "URL for Consumer Service" we will have to put the url path of the Provider service.

```
URL for Consumer Service . . . . > http://yourserver:yourport/yourlib/mdrecho'
```

In the "IFS Path to store the Output" we enter the IFS path location where the response output log file will be written.

```
IFS Path to store the Output . . > '/MDRest4i/youruser/Clientssldtl'=
```

Once the command completes the execution, the source member below is generated under the specified source file and library combinations and it gets compiled as well. The name of the program is CLTSSLCNS. If you refer the statements highlighted in green colour under "Initialize" procedure, the variable "wg_dcmapplication" is set to *blanks by default. This is the application created in digital certificate manager and it either has some valid certificates assigned (if the target REST service requires those certificates) or it doesn't have any certificates assigned. Also, the indicator "ng_ssl" is set to *On to enable communicating via SSL channel. After the member generation, we have added some logic in "BuildRequest" procedure as highlighted in red color. This is the request body to the POST service called from the consumer.

```
0001.00 h dftactgrp(*no) actgrp(*new)
0003.00 h bnddir('LXRGLOBAL': 'BNDZIP')
0004.00  ************************************************************
0005.00  *  MDRest4i SSL Consumer Template GET JSON
0006.00  ************************************************************
0007.00
0008.00  /copy qrpglesrc,mdrestdfn
0013.00
0021.00 d Initialize      pr
0022.00
0023.00 d BuildRequest    pr
0024.00 d CloseDown       pr
0025.00
0026.00  /free
```

```
0027.00
0028.00     InitVariant();
0030.00     tg_InitializePointer = %paddr(Initialize);
0031.00     tg_BuildReqPointer = %paddr(BuildRequest);
0032.00     tg_ClosedownPointer  = %paddr(Closedown);
0033.00     GskConsume();
0034.00
0035.00     if wg_httpStatus = '200';
0036.00
0037.00       // Change above condition to appropriate success status codes as
0038.00       // applicable to your REST service and add the processing logic here
0039.00
0040.00       // You may also call "GetErrorWarnings" procedure with one parameter
0041.00       // of 1024a attribute to get any errors/warnings reported in execution
0042.00
0043.00     Endif;
0044.00
0045.00     // Cleanup the memory allocated dynamically from heap
0046.00     cleantree();
0047.00
0048.00     *Inlr = *On;
0049.00
0050.00   /End-Free
0051.00   *-------------------------------------------------------------
0052.00   * End of Mainline Section
0053.00   *-------------------------------------------------------------
0054.00
0055.00   * Procedure Interfaces
0056.00   *-------------------------------------------------------------
0057.00
0093.00 p CloseDown       b                   export
0094.00 d CloseDown       pi
0095.00   /Free
0096.00
0097.00     //Enter service closedown instructions here
0098.00   /End-Free
0099.00 p CloseDown       e
0100.00
0101.00 p BuildRequest    b                   export
0102.00 d BuildRequest    pi
0103.00   /Free
0104.00
0105.00     //Enter the request body  here
0105.01     beginObject(' ');
0105.02     addChar('message':'Hello World!');
0105.03     endObject();
0106.00   /End-Free
0107.00 p BuildRequest    e
0108.00
0109.00 p Initialize      b                   export
0110.00 d Initialize      pi
0111.00   /Free
0112.00
0113.00     // Set Import Values
0114.00     wg_dcmApplication = *blanks;
0115.00     wg_userAgent = 'LXRSSL 0.1';
0116.00     wg_contentType = 'application/json; charset=UTF-8';
0117.00     wg_httpsMethod = 'POST';
0118.00
0119.00     wg_maxAttempt = 25;
0120.00     wg_mSecDelay = 500000;
0121.00
0122.00     wg_hostName = 'dev.mdcms.ch';
0123.00     wg_serverIP= ' ';
0124.00     wg_portNumber = 443;
0125.00
```

```
0126.00     wg_servicePath = '/mdrdemod/mdrecho';
0127.00     wg_urlParm = '      ';
0128.00
0129.00     // Set below indicator to *Off if you don't want to save logs in IFS
0130.00     ng_saveRestSwitch = *On;
0131.00     wg_saveRESTpath ='/MDRest4i/youruser/Clientssldtl/sslcns.txt';
0132.00
0133.00     // The setting "ng_ssl=*On" enables SSL request. Otherwise, it's non SSL
0134.00     ng_ssl = *On;
0135.00
0136.00     // If the REST service requires authentication, set ng_authsend = *On
0137.00     // and then use one of the three subsequent variables to send auth info.
0138.00     // In order to send the basic authentication via userId and pwd, set the
0139.00     // variables wg_username and wg_password. If however, it's the "Bearer"
0140.00     // token or some other auth string, set the value in "wg_authstring".
0141.00     // e.g. wg_authstring = Bearer <token value>
0142.00
0143.00     ng_authsend = *off;
0144.00     wg_username = *blanks;
0145.00     wg_password = *blanks;
0146.00     wg_authstring = *blanks;
0147.00
0148.00     // If the REST service is accessible via proxy, set ng_proxy = *on
0149.00     // and set the proxy details in subsequent variables. If the proxy
0150.00     // host is in IP address format, set the value in "wg_proxyIP" and if
0151.00     // it's the host name format, set the hostname in the "wg_proxyhost"
0152.00     // variable without any http/https prefix and there shouldn't be any
0153.00     // slash before or after the host name or IP address. The port number
0154.00     // of the proxy host should be set in "wg_proxyport" variable below.
0155.00     // If the proxy is on SSL channel, set "ng_proxySSL" to *On.
0156.00     ng_proxy = *Off;
0157.00     ng_proxyssl = *Off;
0158.00     wg_proxyhost = *blanks;
0159.00     wg_proxyIp = *blanks;
0160.00     wg_proxyport = *zeros;
0161.00     wg_proxyusr = *blanks;
0162.00     wg_proxypwd = *blanks;
0163.00
0116.00   // If you want to load config via MRCNSDTLF file, please uncomment
0117.00   // the below line. In this case you need to make sure you have loaded
0118.00   // (Host, Port, path etc.) in the MRCNSDTLF file with the key of This
0119.00   // Program name. Add below file in the 'F' spec in this program
0120.00   // mrcnsdtlf if   e           k disk    Usropn
0121.00   // /copy qrpglesrc,mrsetcon
0122.00
0123.00   /End-Free
0124.00 p Initialize      e
0166.00
0167.00   * ------------------------------------------------------------------------
0168.00   *  Procedure - FixReprocess
0169.00   * ------------------------------------------------------------------------
0170.00 p FixReprocess    b                   export
0171.00 d FixReprocess    pi
0172.00   /free
0173.00   /end-free
0174.00 p FixReProcess    e
0175.00   * ------------------------------------------------------------------------
0176.00   *  Procedure - InitVariant
0177.00   * ------------------------------------------------------------------------
0178.00 p InitVariant     b
0179.00 d InitVariant     pi
0180.00   /free
0181.00     w_OpenCurly = %char(c_OpenCurly);
0182.00     w_CloseCurly = %char(c_CloseCurly);
0183.00     w_OpenSquare = %char(c_OpenSquare);
0184.00     w_CloseSquare = %char(c_CloseSquare);
```

```
0185.00     w_EscapeChar = %char(c_EscapeChar);
0186.00     w_Power = %char(c_Power);
0187.00     w_Tilde1 = %char(c_Tilde1);
0188.00     w_Exclaim = %char(c_Exclaim);
0189.00     w_Hash = %char(c_Hash);
0190.00     w_Pipe1 = %char(c_Pipe1);
0191.00     w_Accent = %char(c_Accent);
0192.00     w_Doller = %char(c_Doller);
0193.00     w_AtSign = %char(c_AtSign);
0194.00
0195.00   /end-free
0196.00 p InitVariant     e
```

Below is the response:

```
{
"receivedHeaders":{
                  "CONTENT_TYPE" : "application/json; charset=UTF-8" ,
                  "CONTENT_LENGTH" : "34" ,
                  "HTTP_HOST" : "dev.mdcms.ch" ,
                  "HTTP_USER_AGENT" : "LXRSSL 0.1" ,
                  "HTTP_CONNECTION" : "Keep-Alive"
                  },
"receivedPayload":{
                  "message": "Hello World!"
                  }
}
```

4.4 Generating a Service/Consumer from SWAGGER

4.4.1 Overview

MDRest4i SDK exposes a REST API on the IBM i that generates RPGLE consumer and Provider programs. The generator API is built with iCore and exposed on the IBM i as a REST service.

The specifications format used by the MDRest4i SDK generator is Open API (a.k.a SWAGGER). Requirements are added to a standard SWAGGER template specification in JSON format. This SWAGGER specification is then sent as the requestBody to the MDRest4i SDK REST API using a POST method.

MDRest4i SDK can also use a SWAGGER specification for a REST API to create a Consumer program. In this way it is easy to generate consumer programs from Swagger provided by the API host.

In some cases documentation for an API or requirement for a consumer may only have a sample JSON payload for the request or response bodies.

The MDRest4i SDK GUI can use API and Consumer requirements in multiple formats to build the swagger specifications used by the code generator. This includes converting a payload sample into a Swagger specification for generation of an RPGLE service or consumer

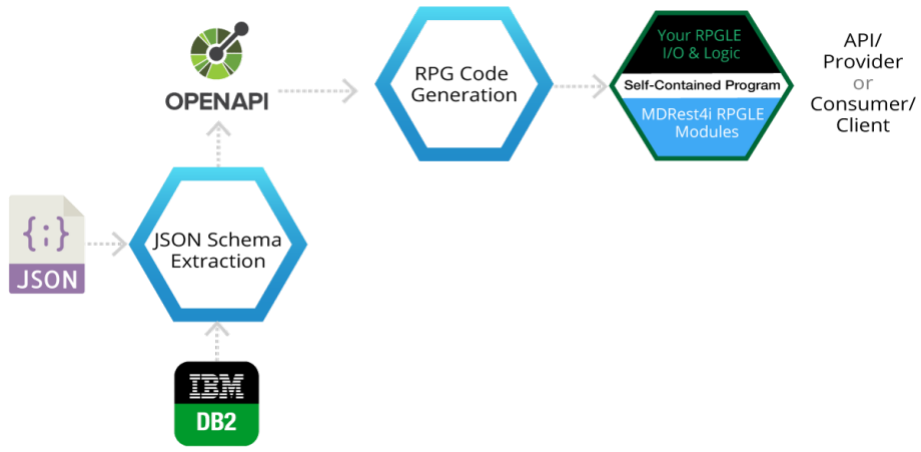


Figure 7 - Generate from any requirement

4.4.2 Creating and Editing SWAGGER Specifications

SWAGGER (now known as Open API Specification or sometimes abbreviated to OAS3) allows you to describe the structure of your APIs so that machines and humans can read them. The ability of APIs to describe their own structure is the root of all awesomeness in Swagger. Why is it so great? Well, by reading your API's structure, you can automatically build beautiful and interactive API documentation with SWAGGER UI. MDRest4i SDK can also automatically generate client/server libraries/programs for your API in many languages (now including RPGLE using rest4i), and explore other possibilities like automated testing.

API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines. The complete OpenAPI Specification can be found on GitHub: OpenAPI 3.0 Specification

SWAGGER extensions (JSON objects prefixed with an "x-") are added to each SWAGGER specification by MDREst4i SDK to facilitate some of the RPGLE generation process. For example to tell the generator where to build the source and compile the object the x-lxrgen JSON object is added:

```
"x-lxrgen": {
      "library": "MDRTUTLIB",
      "srcfile": "QRPGLESRC",
      "srclibrary": " MDRTUTLIB ",
```

```
      "reqType": "consumer",
      "object": "getclndtlc",
"lxrpath": http://yourserver:yourport/mdrst/getclndtl
}
```

All of the SWAGGER extensions used by the rest4i MDRest4i SDK generators can be reviewed in Appendix A below

Editing the Swagger specifications is greatly simplified with the MDRest4i SDK Web GUI, which automatically adds all extension fields used by the generators, and simplifies the SWAGGER editing process with a form based GUI.

4.4.3 Calling The Generator API – MDRGENXAPI

MDRGENXAPI is REST API that accepts the POST method.

The MDRest4i SDK REST API (MDRGENXAPI) can be called directly by any client that is able to make POST requests. For example other popular clients capable of handling POST requests include:

- POSTMAN - ***https://www.getpostman.com***,
- SOAP-UI ***https://www.soapui.org/***,
- ARC - ***Advanced REST Client for Chrome Browser***

The request payload should be in application/json format and a valid SWAGGER 3.0 format, plus have all of the swagger extensions described in the ***MDRest4i_11.0_User-and-Reference-Guide.***

For example

4.4.4 Using the MDRest4i SDK Web GUI

Once the SDK Console web application has been installed and started, browse to the appropriate URL e.g.: http://yourlinuxserver:portnumber/ in your web browser (Chrome, Edge or Firefox)

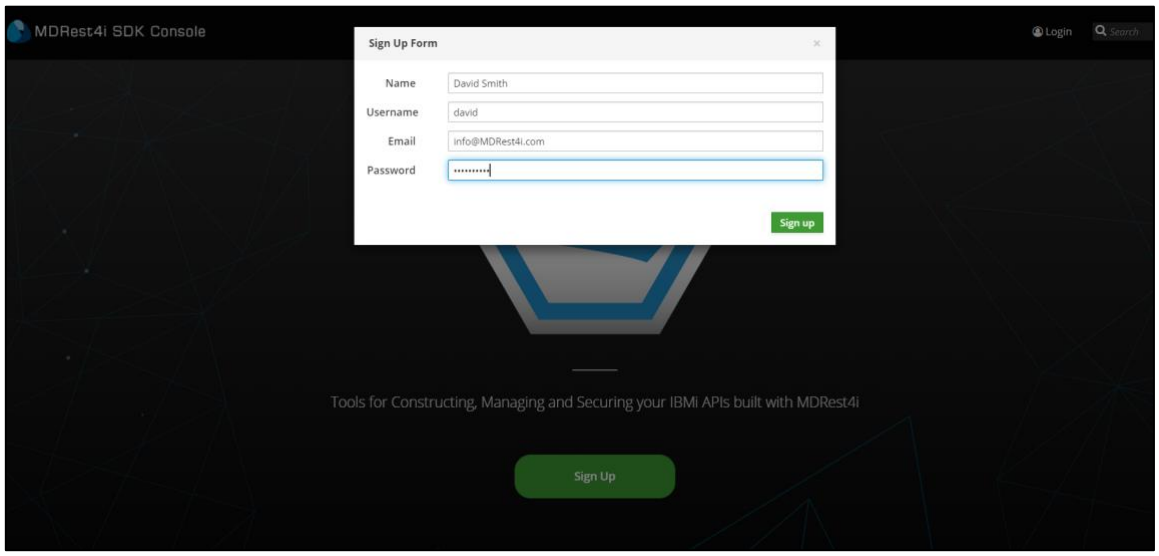


Figure 8 - MDRest4i SDK Sign-up Page

Login using user: lxradmin and pwd: lxradmin

or use “Sign Up” to create your own profile

If using “Sign Up”, verify using the activation email sent to the email supplied during signup. (requires smtp server to be setup while installing MDRest4i SDK)

Login using the details provided during sign up and Figure 7 will appear.

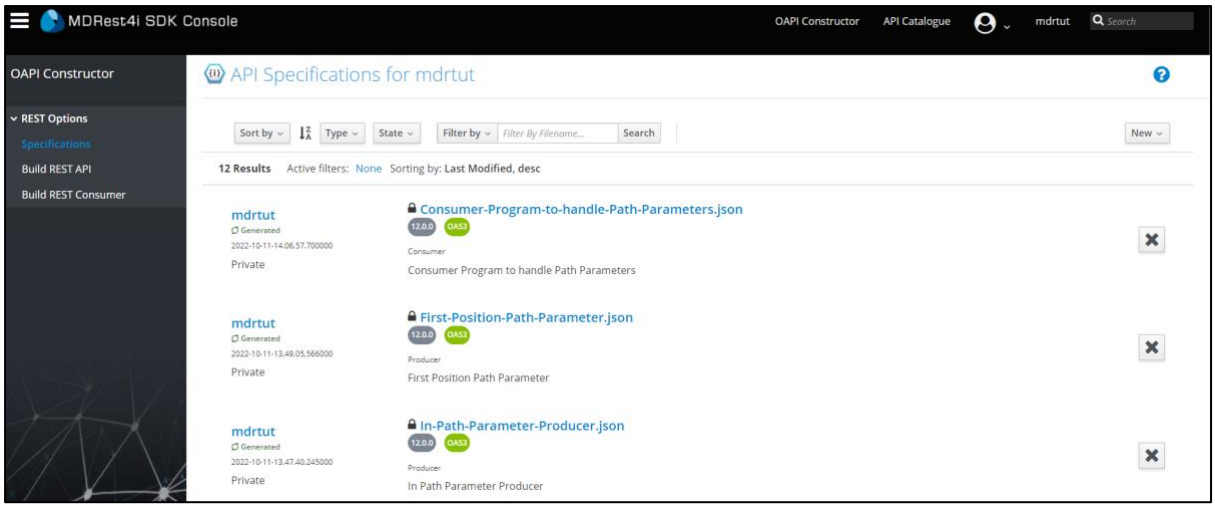


Figure 9 - Specifications List

Figure 7 shows a list of your specifications for both consumers and Providers. For a new user, the specifications list is empty. The buttons across the top provide additional functionality for filtering the list.

Next, you need to edit your profile selecting the “Edit Profile” option from the top menu.

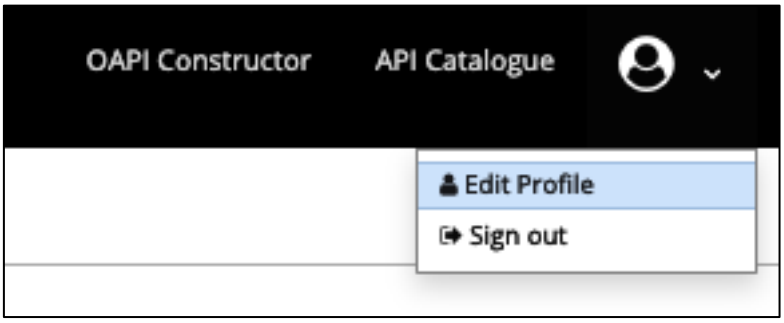


Figure 10(a) – Edit Profile

You need to edit the default values for library, Source Library, Source File and the server URL. These default values would be used while creating or importing a new Provider/consumer spec.

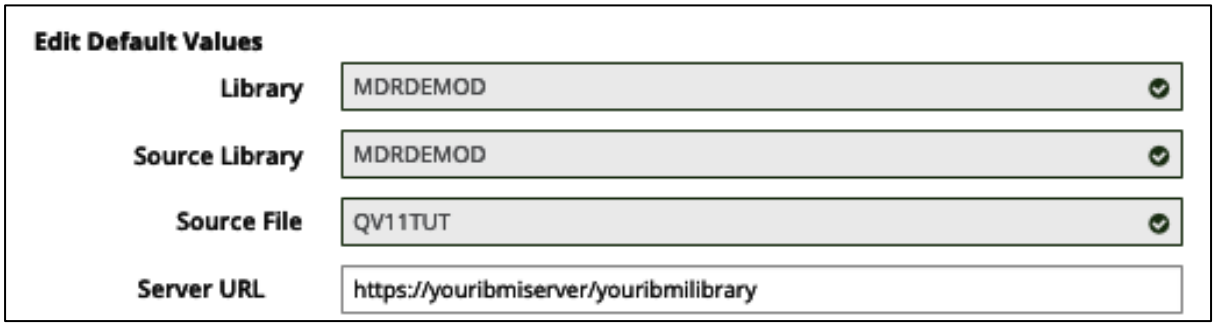


Figure 10(b) – Edit default values

Note: To navigate back to the Specifications List, click MDRest4i Constructor in the menu at the top of the page, then select REST Options, and API Specifications

4.5 SWAGGER MDRGENXAPI REST API Examples

This tutorial covers 10 examples of the generation from Swagger.

All of the required swagger specifications, rpgle and json examples are available in the MDRest4i-v12.0-Tutorial-swagger-rpgle-json-examples.zip, downloaded with the software. These can be found in the unzipped file above in the "MDRest4i-v12.0-Tutorial-swagger-rpgle-json-examples/Section-4.5-Swagger-Example-Code" folder.

4.5.1 Provider-API-GET-Clients-V12

4.5.1.1 Objective and Design

Demonstrate how to use MDRest4i SDK to create a REST API (Provider/service) that reads a DB2 PF/Table and returns the record/s as JSON, allowing specific records need to be requested if necessary(not mandatory).

A REST Provider using the GET method will be created, that uses the PF fields as output. The key of the PF/Table may be supplied as a query string parameter.

4.5.1.2 Create New Provider

From the API Specifications list in the MDRest4i SDK Web GUI, select the "New" button on the right-hand side and select "Provider" as displayed in Figure 12 below:

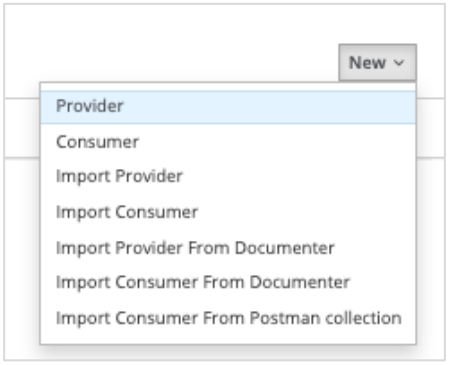


Figure 11 - New>Provider

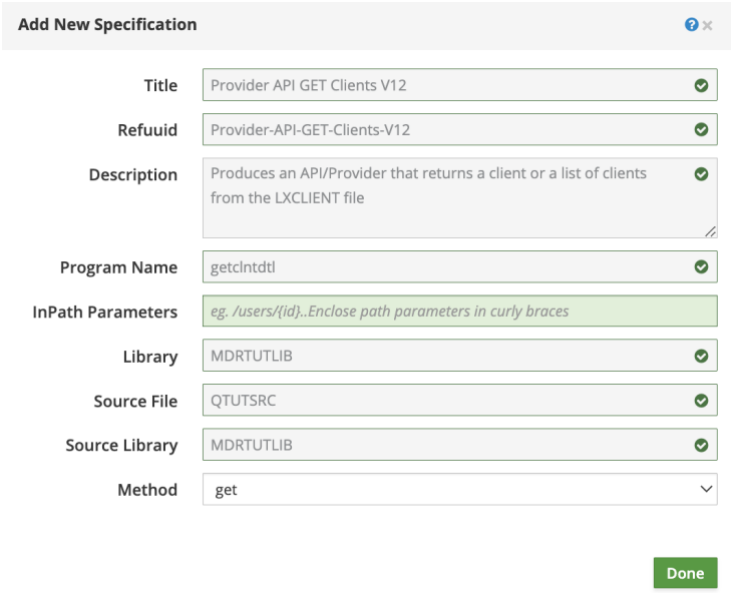


Figure 12 - New API Popup

The default values for the Library, Source Library and Source File can be set in the user profile using the "**Edit Profile**" option (refer to Figure 10(a) and 10(b)). The values would be automatically picked up from the user profile.

If the default values are not provided, you need to enter them manually every time while creating a spec.

Complete the fields in the popup that appears as per Figure 9 above. Select the "get" method in the drop down, before selecting "Done".

This has now copied the SWAGGER template for API's, added a few additional SWAGGER extension fields with the data provided by you in the popup, and created a new SWAGGER definition in JSON format in memory.

4.5.1.3 Adding General Info

The General tab seen in Figure 14 below now appears.

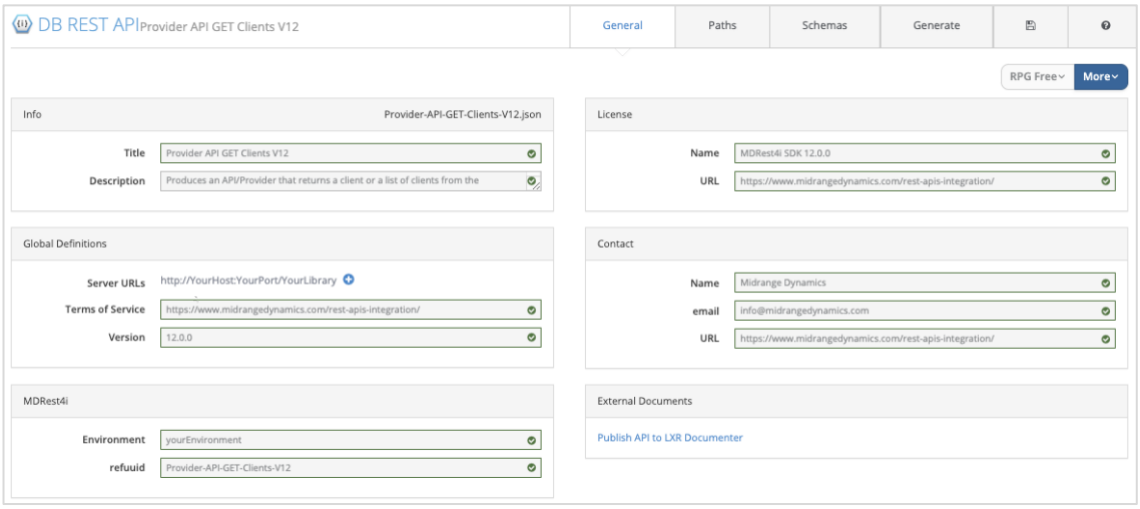


Figure 13 - General Tab API

The only thing that needs updating is the server URL's. Add the host name, Port number and library name used when setting up the MDRest4i HTTP server during installation.

This can again be set up as the default in the "**Edit Profile**"option (Figure 10(b)). If this is not available, it can be added later. The rest is general information and can be left as is for this tutorial.

Now select the "More" button and "Save" from the drop down menu.

4.5.1.4 Create a Schema

Now select the "Schemas" Tab.

Select the "+" button to add a new schema, de-select the "work fields only" checkbox, and provide a model name, library, and Physical File name. Then click the "Next" button. The X-Library name and PF name are validated on the IBM i using an MDRest4i SDK REST API written using iCore. Figure 15 below lists the fields in the PF file LXCLIENT in Library MDRST.

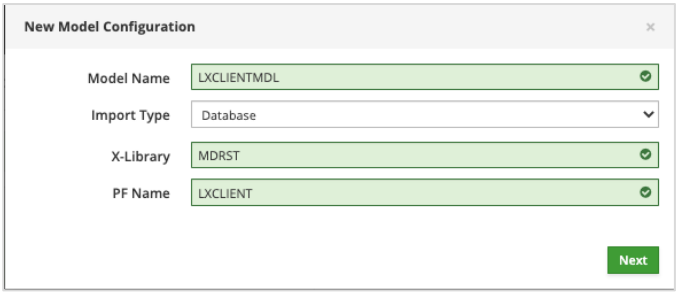


Figure 14 - Add a Schema

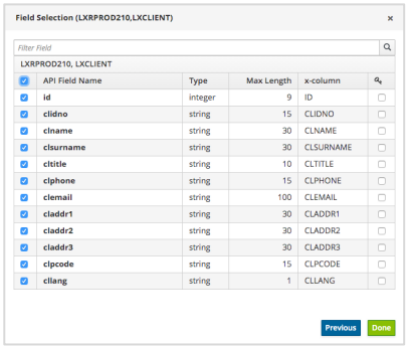


Figure 15 - Select PF Fields for API

Select all fields by clicking on the check box in the top left corner. This information is extracted in real-time by another iCore REST API on the IBM i. The API field name is taken from the field text and the x-column from the PF field name itself. This generates a schema in the SWAGGER specification that maps the API field name to the PF Field name. Figure 17 below is a snippet of that schema.

```
"components": {
  "schemas": {
   "LXCLIENTMDL": {
    "x-library": "MDRST",
    "x-pfname": "LXCLIENT",
    "properties": {
     "id": {
      "type": "integer",
      "description": "Description of id",
      "maxLength": 9,
      "x-column": "id"
     },
     "clidno": {
      "type": "string",
      "description": "Description of clidno",
      "maxLength": 15,
      "x-column": "clidno"
     },
```

Figure 16 – SWAGGER Schema Snippet

Click “Done” to continue.

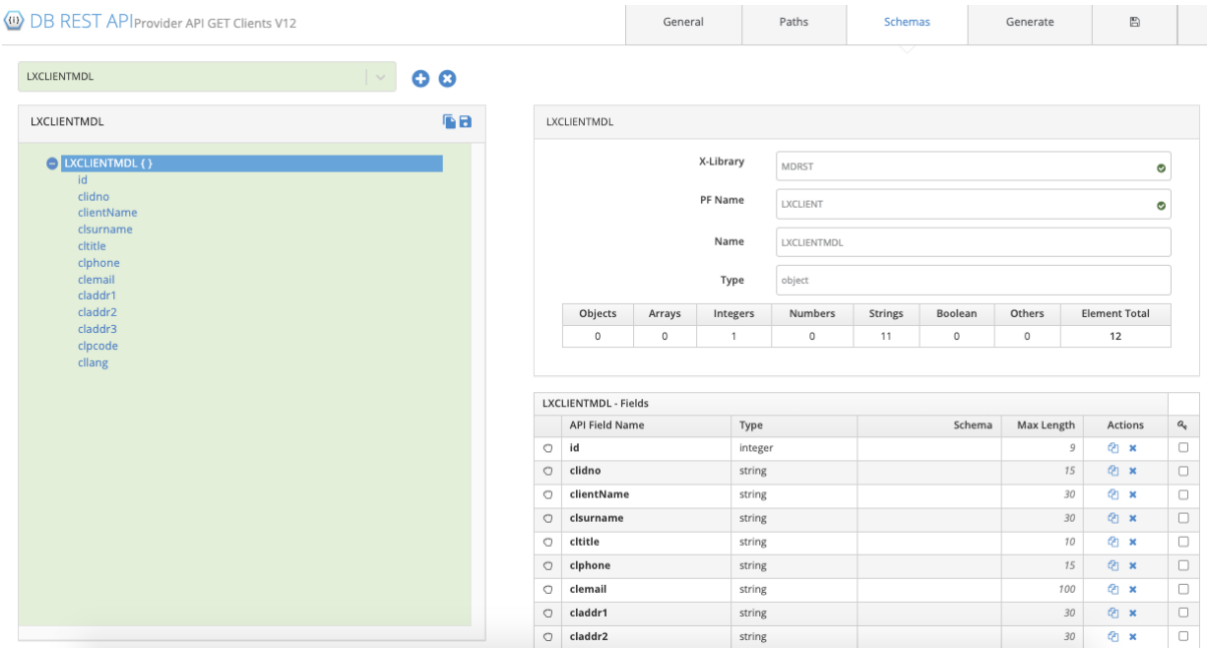


Figure 18(a)- API Schema Tab (Model Level Detail)

LXCLIENTMDL - Fields

	API Field Name	Type	Schema	Max Length	Actions	
	id	integer		*9*		
	clidno	string		*15*		
	clname	string		*30*		
	clsurname	string		*30*		
	cltitle	string		*10*		
	clphone	string		*15*		
	clemail	string		*100*		
	claddr1	string		*30*		
	claddr2	string		*30*		
	claddr3	string		*30*		
	clpcode	string		*15*		
	cllang	string		*1*		
	new api field name			*maxLength*		

Figure 18(b) - API Schema Tab (Field Level Details)

The left side brings up the schema in the tree form with all the fields listed in hierarchical order. Clicking

on the model name or the elements of the tree brings up the details on the right hand side (Figure 18(a) and Figure 18(b)).

The top right window displays the high level overview/editable details of the object selected.

The bottom right window brings up the children (if the selected element is an object or an array of objects) and their editable details.

Fields can be added/deleted for the selected element if it is of the type object or an array of objects.

4.5.1.5 Edit API Path Info

Now select the “Paths” tab and click on the Parameters button.

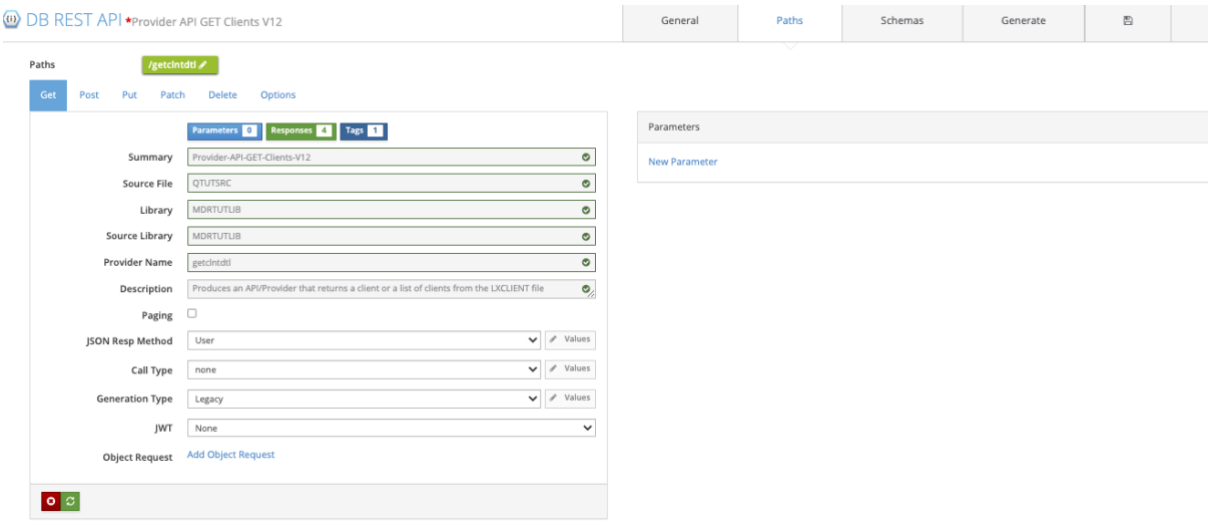


Figure 19 - Paths Tab with Parameters

In Figure 19, the Paths are shown as buttons across the top on the left hand side. Below these are the HTTP methods. For each method the details of where to build the generated code and compile it can be updated. This automatically updates the underlying swagger specification in the x-lxrgen object described above.

4.5.1.6 Edit API Parameters

Now select the "Parameters" button on the left hand side, then the "New Parameter" link on the right hand side.

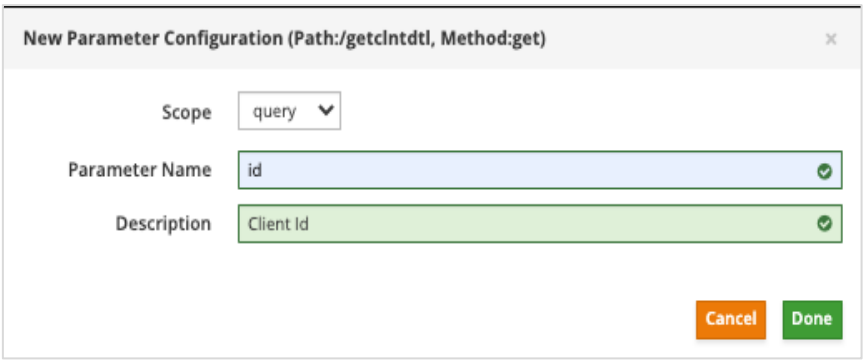


Figure 20 – Create Parameter

Select the scope as "query" from the drop-down. This means that the Provider program will expect this parameter in the query string of the request e.g. http://yourserver:yourport /yourlib/getclntdtl?`id=3`

Add the parameter name as "id" – this will be the name of the parameter supplied in the query string of the request URL. Add a description for the parameter (optional).

Click "Done"

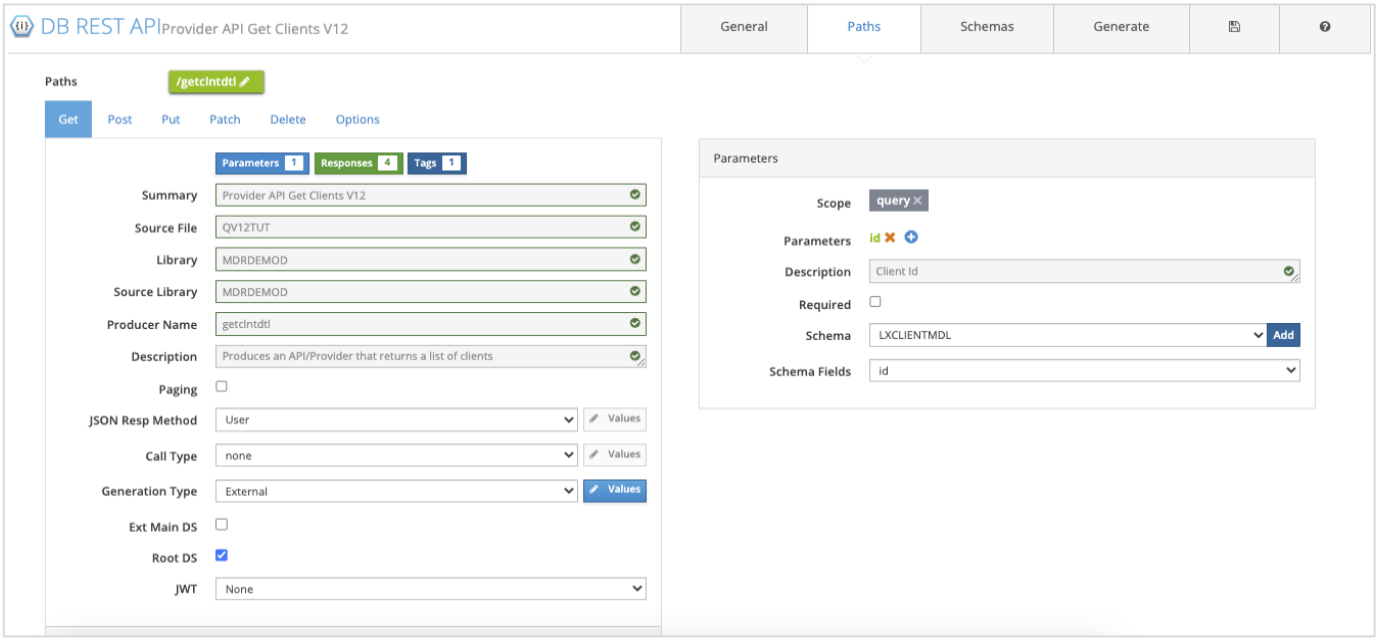


Figure 21 - Parameters and Schema Reference

The RPGLE generator can assign values from the query string parameters to the where clause in the SQL it generates. To do so, it needs to know which schema field (and therefore associated PF Field/Column) to assign the incoming query string parameter value to. So, the parameter must be associated first.

Select a Schema and then a Schema Field from this schema in the drop downs provided as displayed in Figure 18a. The parameter object in the underlying SWAGGER specification is updated to Figure 21 above.

```
"parameters": [
        {
          "in": "query",
          "name": "id",
          "description": "client ID",
          "required": false,
          "schema": {
            "x-schema": "LXCLIENTMDL",
            "x-schemaFld": "id"
          }
        }
      ]
```

Figure 22 - SWAGGER Parameter Snippet

Now select the Responses button on the left hand side of the paths tab.

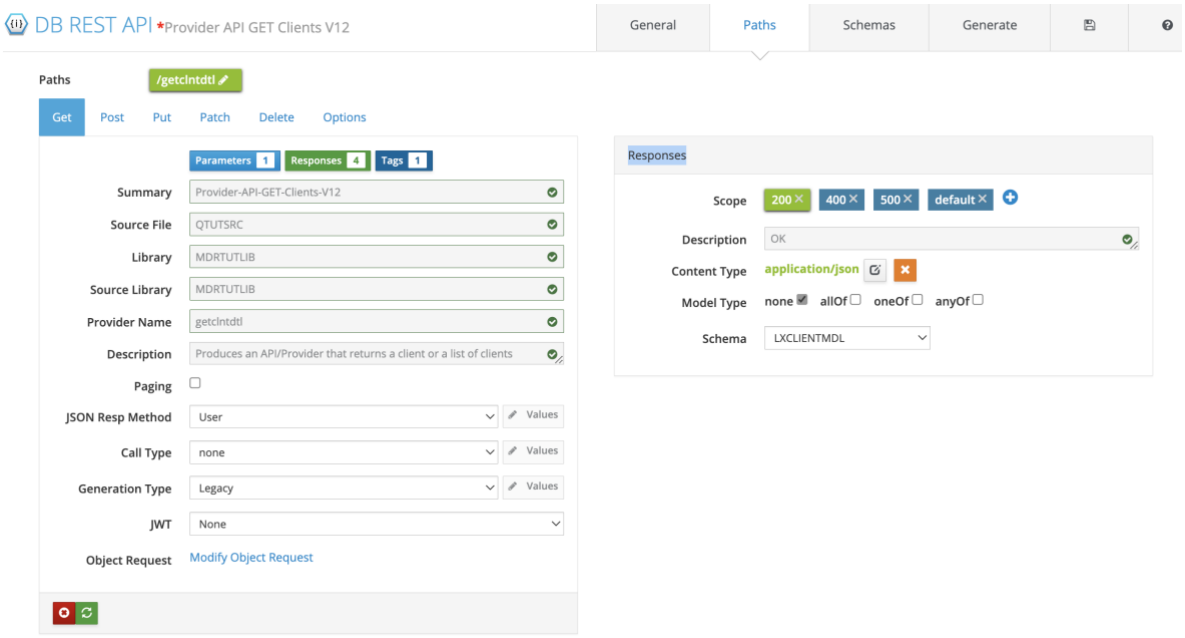


Figure 23 - Edit API Response

By default, four HTTP responses are added to each API. Response 200 is the default status header field and value, which tells the requesting browser or consumer, that everything is OK. To generate RPGLE logic in the Provider that sends back JSON in the response, the content type and schema used to define the response must be added.

This will tell the RPGLE generator to add logic that sends back the correct status HTTP header field. The generator will also automatically build embedded SQL in the RPGLE (to get the data from the PF named in the schema earlier), plus generate logic to assign data from the embedded SQL result set, while building the JSON response body, which is then sent back to the requesting browser or consumer.

From the “Responses” section on the right hand side, select the content type, “application/json” from the drop down, and then select the “LXCLIENTMDL” schema created earlier from the Schema drop down.

In this example we are using a schema that has PF information included in it. This uses the PF FFD to create the necessary variables in the generated RPG logic along with building the SQL statement/s. If only work fields were in the schema, the RPGLE code generated would also contain these work field definitions.

4.5.1.7 Select the generation type

Now select the generation type.

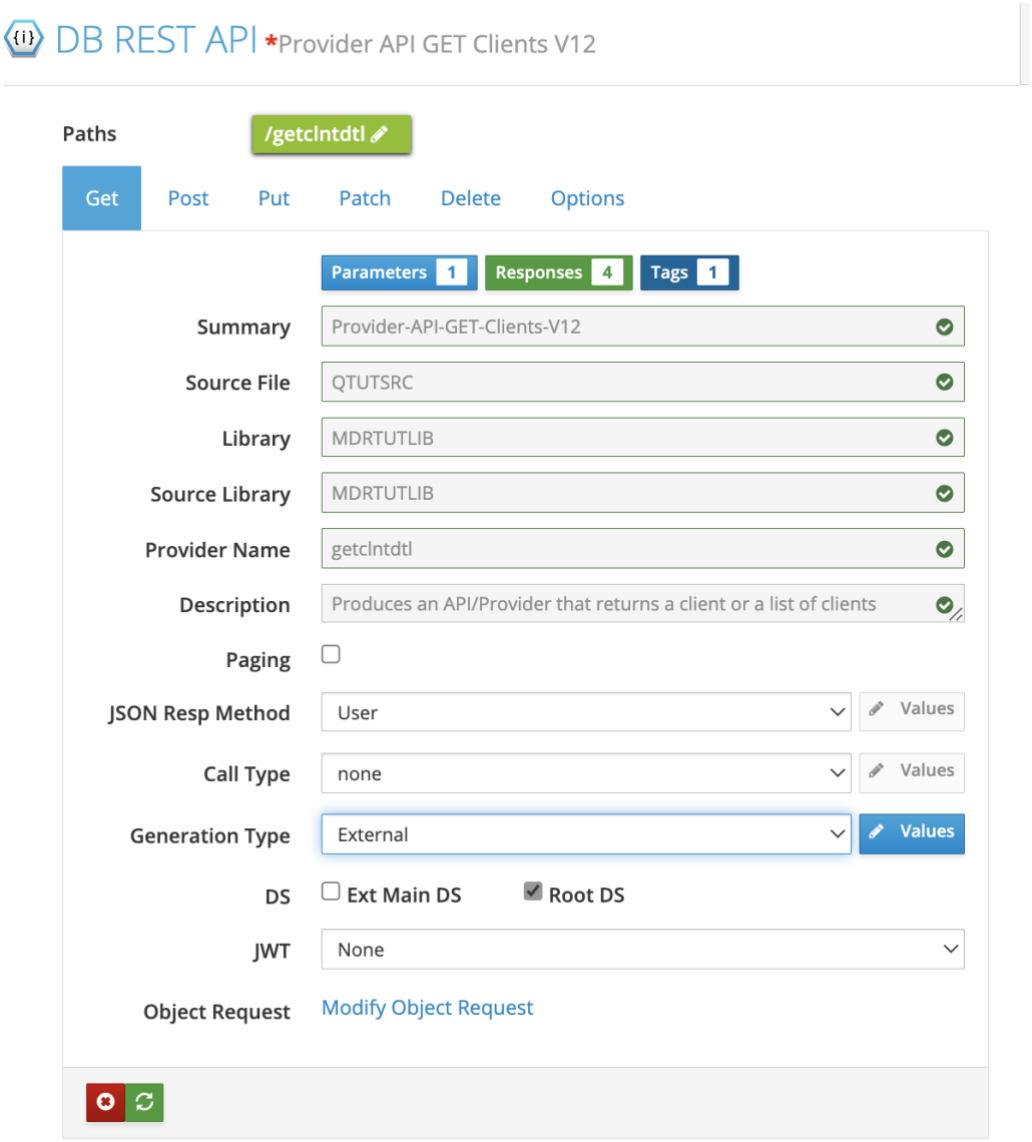


Figure 24 – Set Generation type

In the **Legacy** it will generate the API via MDRGENXAPI, in **Inline** it will generate the REST API via MDRGNAPI with all in one pgm and in **External** it will generate REST API using MDRGNAPI with procedures externalized.

Selecting **External** or **Inline** will require

4.5.1.8 Create API Service

Now select the “Generate” tab.

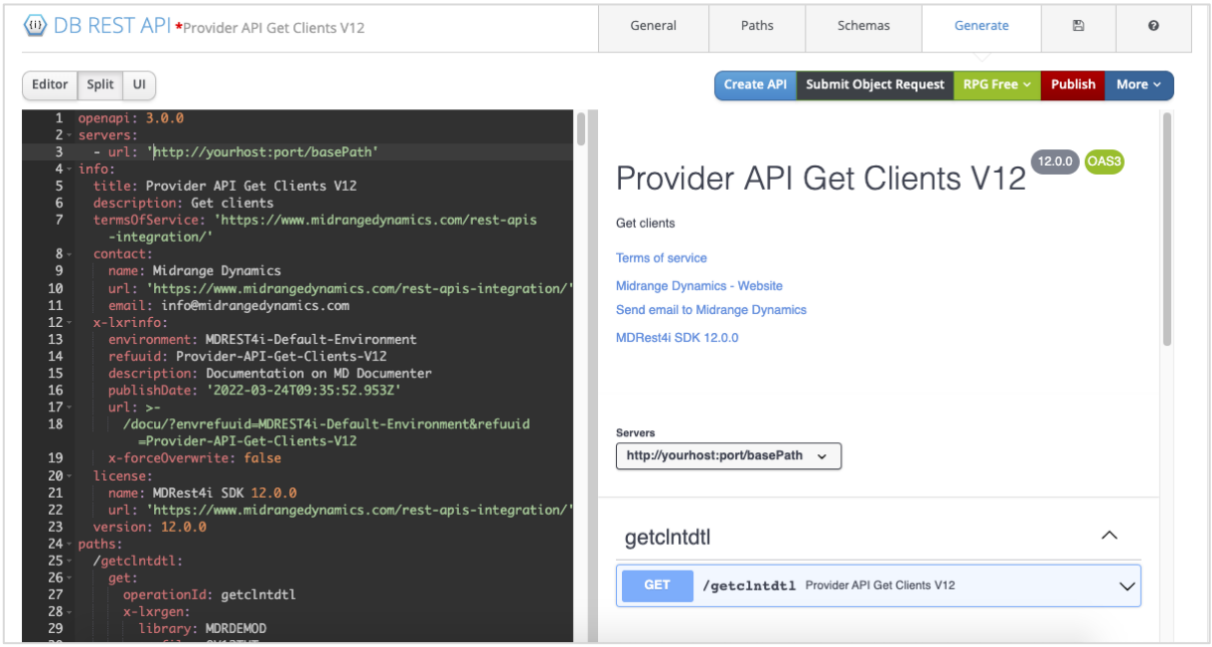


Figure 25 - API Generate Tab

Note: The Swagger Specification is updated automatically in memory with each change made on the screen but this does **NOT** save those changes to the specifications database.

From the display seen in Figure 25, select the “more” button on the right-hand side and save the specification. This updates the details added since the last save in the specifications database.

The “Generate” tab is a custom version of a full SWAGGER Editor. The left hand window enables editing of the SWAGGER specification directly, and the right-hand side shows the documentation and allows testing of the Provider once it is generated and completed.

Select the “Create Service” button.

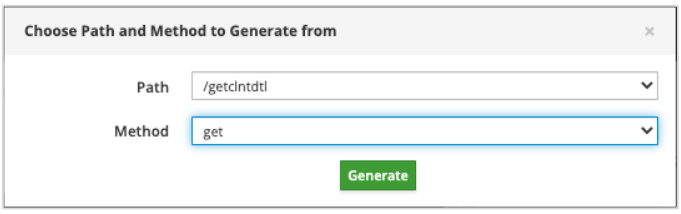


Figure 26 - Create Service

Select the “get” method from the drop down and click the “Generate” button.

Upon completion of the generation process on the IBM i, Figure 27 will appear:

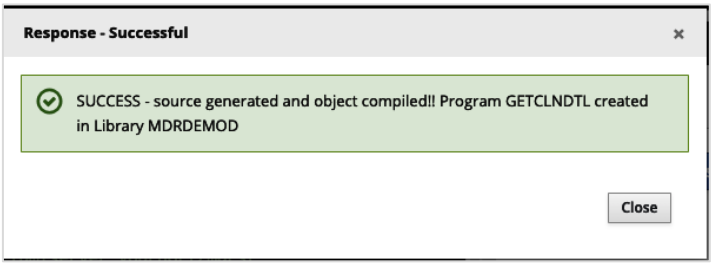


Figure 27 - Response – Successful

4.5.1.9 Generated Code Description

The code generated is almost identical to the code described in the command-generated example CLIENTS above.

The only difference in this specific case is that the MDRGENXAPI Rest Service uses the API field names in the response. The commands use the column names from the table by default. To test this, edit one of the API field names (see Figure 28 below for an example) in the schema tab and regenerate the service.

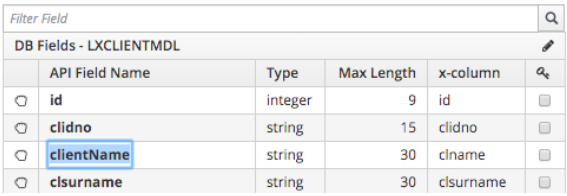


Figure 28 - Edit API Field Name

4.5.1.10 Test the Generated Service

In the "Generate" tab edit the "- URL: 'http://yourhost:port/basePath' " value (in the left hand SWAGGER window as per Figure 29) so that yourhost= your IBM i http server where MDRest4i server was setup, port is the port number used, and basePath is the name of the library where you generated the service into. Please ensure this service was generated into a library that was mapped in the http server in the first instance during installation. For example http://yourserver:yourport/yourlib where yourlib was added as a Library Name during the installation process.

Note: If you wish to have a separate HTTP instance for running your generated API's please refer to the section "Creating the default MDRest4i HTTP Server Instance" in the "MDRest4i_12.0_Installation_Instructions" guide

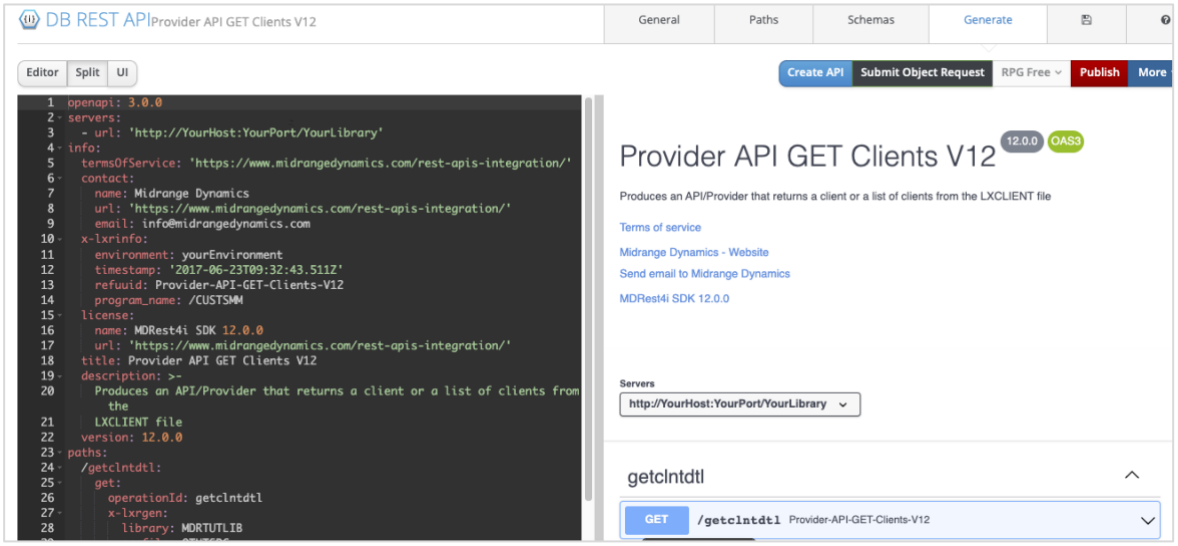


Figure 29(a) - URL for API end point

Now on the right hand side of the "Generate" tab, scroll down until the server and "GET" button appears. Click on the "GET" Button seen in Figure 29(b) to expand it.

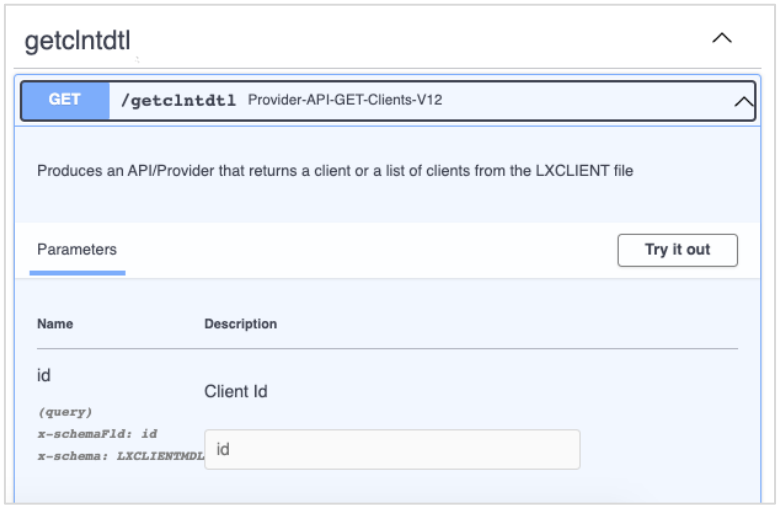


Figure 29(b) - GET Button for Testing

Now scroll down to the "Parameters" section. In this section enter "2" in the Client ID parameter as seen below.

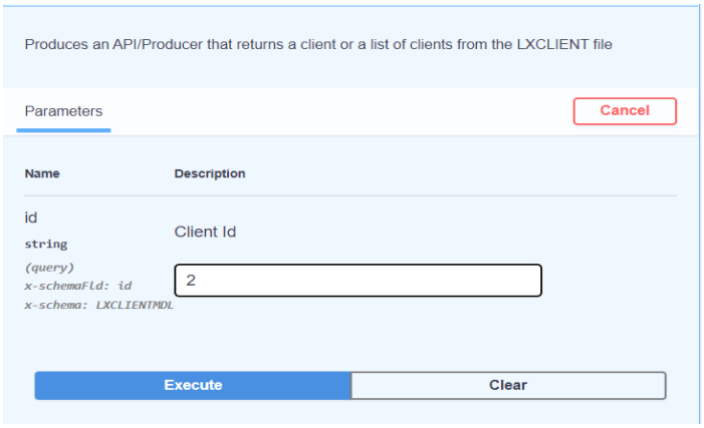


Figure 29(c) - Execute API Test

Scroll down to the "Server response" section.

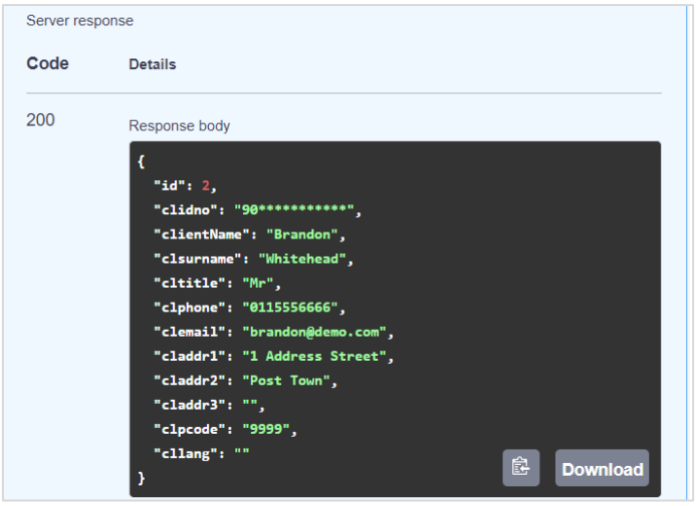


Figure 29(d) - API Test Response

The response body can be seen as JSON in this window. This was created in the RPGLE by the beginObject, addChar, addDeci, addIntr, addBool, etc. functions in the z_procGet subroutine.

Note: Scroll up to the "Request URL" above the request body. Copy the code in the Request URL and paste into the address bar or a new window in your web browser. The same JSON as above will be returned.

TopTip: To see formatted JSON from response in your browser, there are a number of extensions or add-ons available for your browser. For example, Chrome has two useful ones: JSON Viewer for simple viewing and Advanced REST client (referred to as ARC) for a full-blown testing tool.

4.5.2 Provider_Bike_Post-V12

In this we will create a POST API via importing the JSON.

4.5.2.1 Objective and Design

Demonstrate how to use MDRest4i SDK to create a REST API (Provider/service) that we create with the importing of JSON and create a POST type REST API.

4.5.2.2 Create New Provider

From the API Specifications list in the MDRest4i SDK Web GUI, select the “New” button on the right-hand side and select “Provider” as displayed in Figure 30 below.

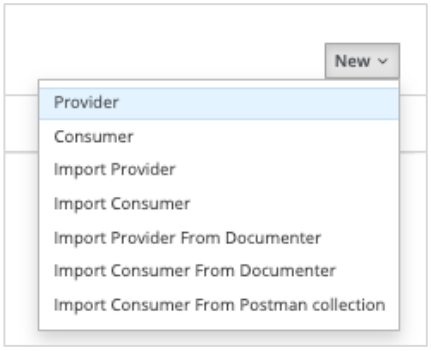


Figure 30 - New>Provider

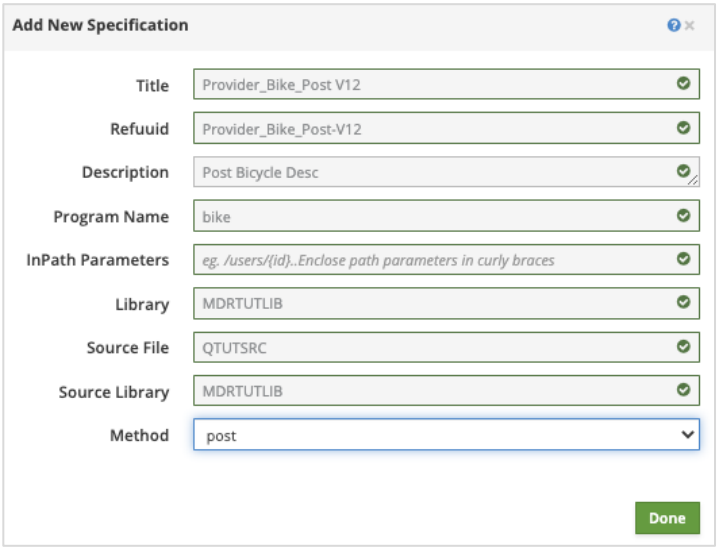


Figure 31 - New API Popup

The default values for the Library, Source Library and Source File can be set in the user profile using the “**Edit Profile**” option (refer to Figure 10(a) and 10(b)). The values would be automatically picked up from the user profile.

If the default values are not provided, you need to enter them manually every time while creating a spec.

Complete the fields in the popup that appears as per Figure 31 above. Select the “post” method in the drop down, before selecting “Done”.

This has now copied the SWAGGER template for API’s, added a few additional SWAGGER extension fields with the data provided by you in the popup, and created a new SWAGGER definition in JSON format in memory.

4.5.2.3 Adding General Info

The General tab seen in Figure 32 below now appears.

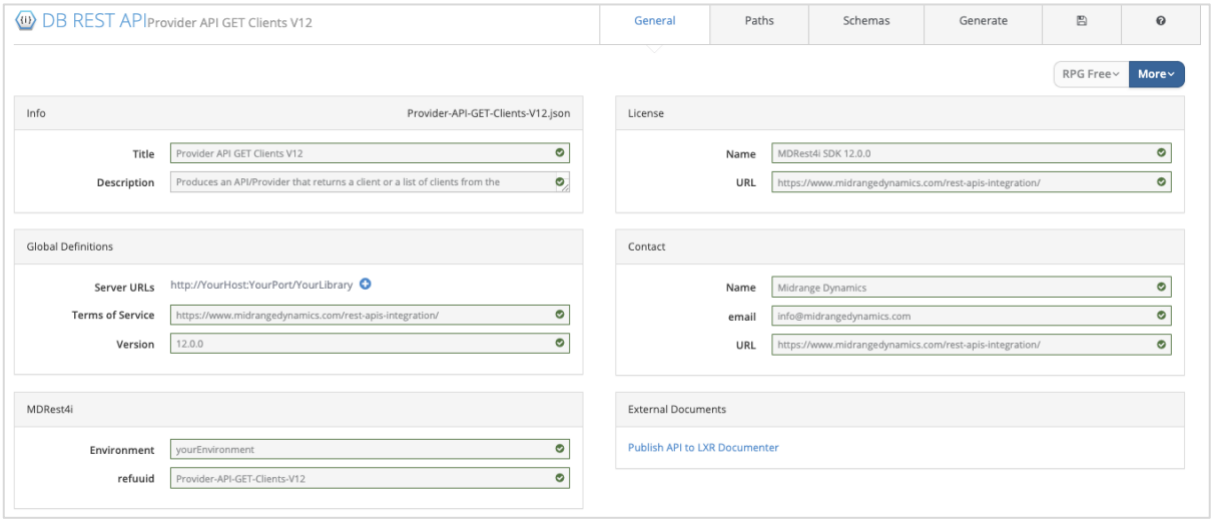


Figure 32: The general tab

The only thing that needs updating is the server URL's. Add the host name, Port number and library name used when setting up the MDRest4i HTTP server during installation.

This can again be set up as the default in the "**Edit Profile**"option (Figure 10(b)). If this is not available, it can be added later. The rest is general information and can be left as is for this tutorial.

Now select the "More" button and "Save" from the drop down menu.

4.5.2.4 Create a Schema

Now select the "Schemas" Tab.

Select the "+" button to add a new schema for using request body and response.

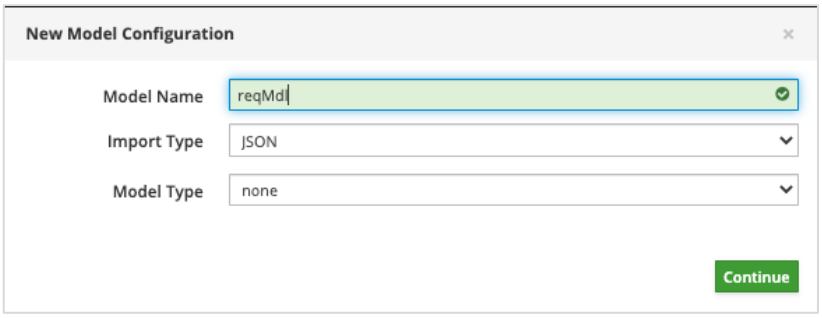


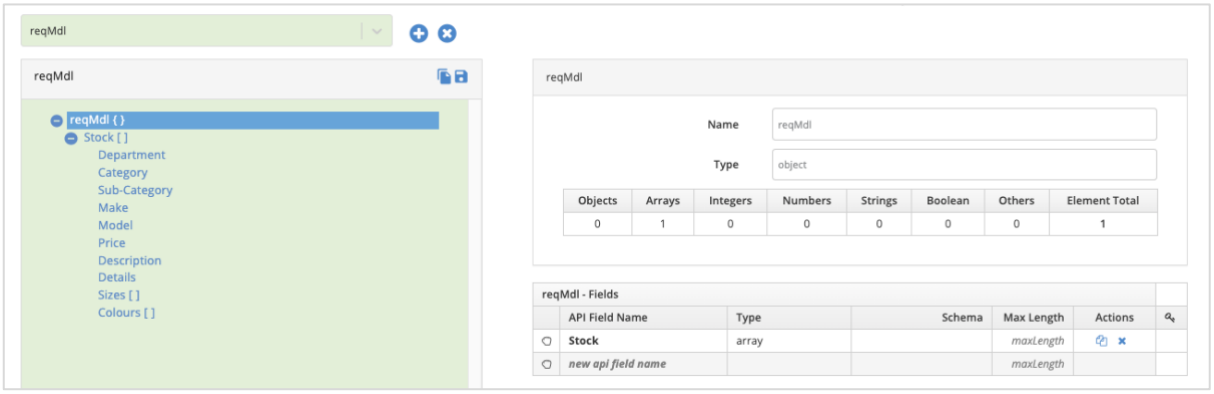
Figure 33 - Add a Schema

4.5.2.5 Import the JSON

In this Option we can import the JSON for making the swagger. We can import the JSON using external file or paste on the editor directly then click on  button.

Figure 34 – Json for import

Then schema will be added in the created schema name “reqMDL”



4.5.2.6 Edit API Path Info

Select “Paths” and click on Body button and assign the request body and content type ‘application/json’ After that select the model which you have created at a time of importing the JSON for request body and select the I/O Action assign. As per shown below figure35:

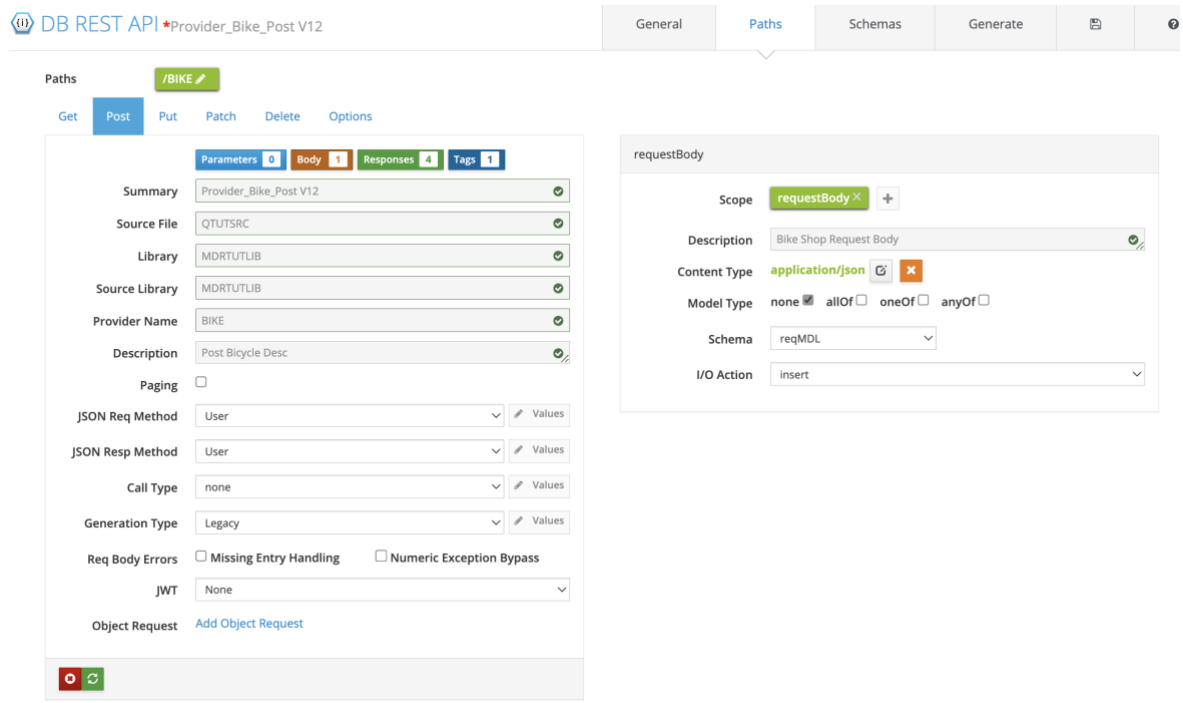


Figure 35 – Set request body

Now select the Responses button on the left hand side of the paths tab.

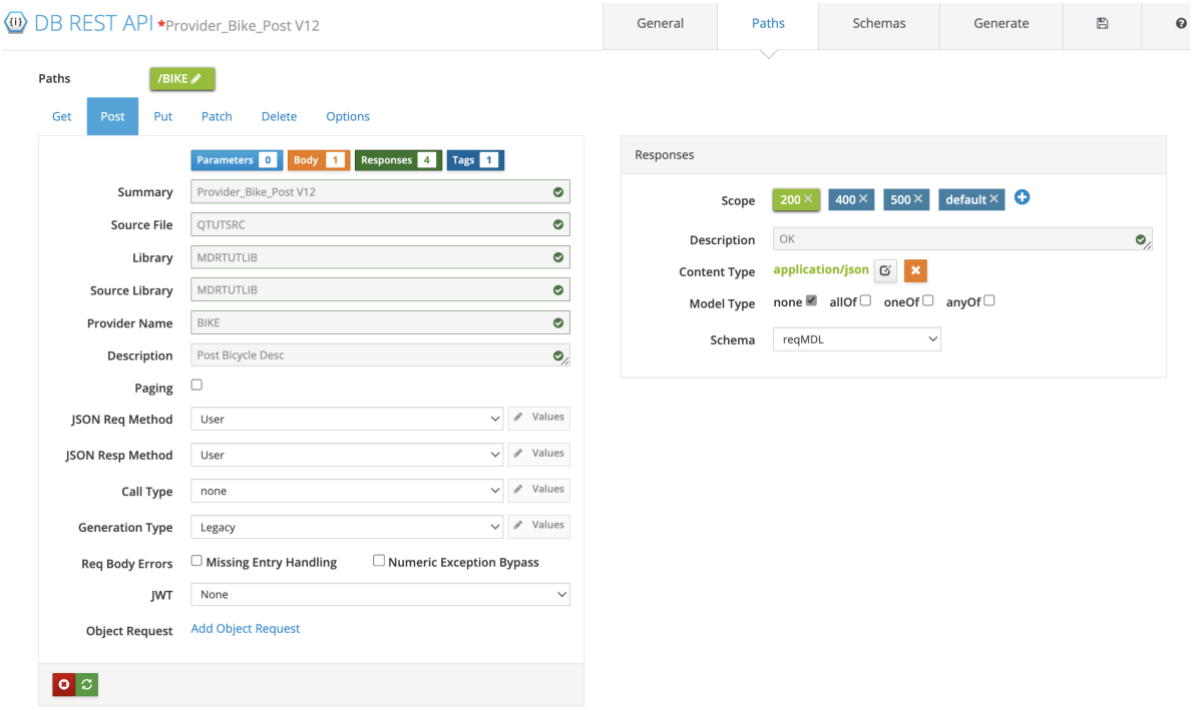


Figure 36 – Set response

4.5.2.7 Select the generation type

Now select the generation type.

Here we will be shown three values like below:

Legacy – In this way code will be generated via MDRGENXAPI.

Inline – In this way Code will be generated via MDRGNAPI with all in one pgm.

External – In this way Code will be generated via MDRGNAPI with procedures externalized

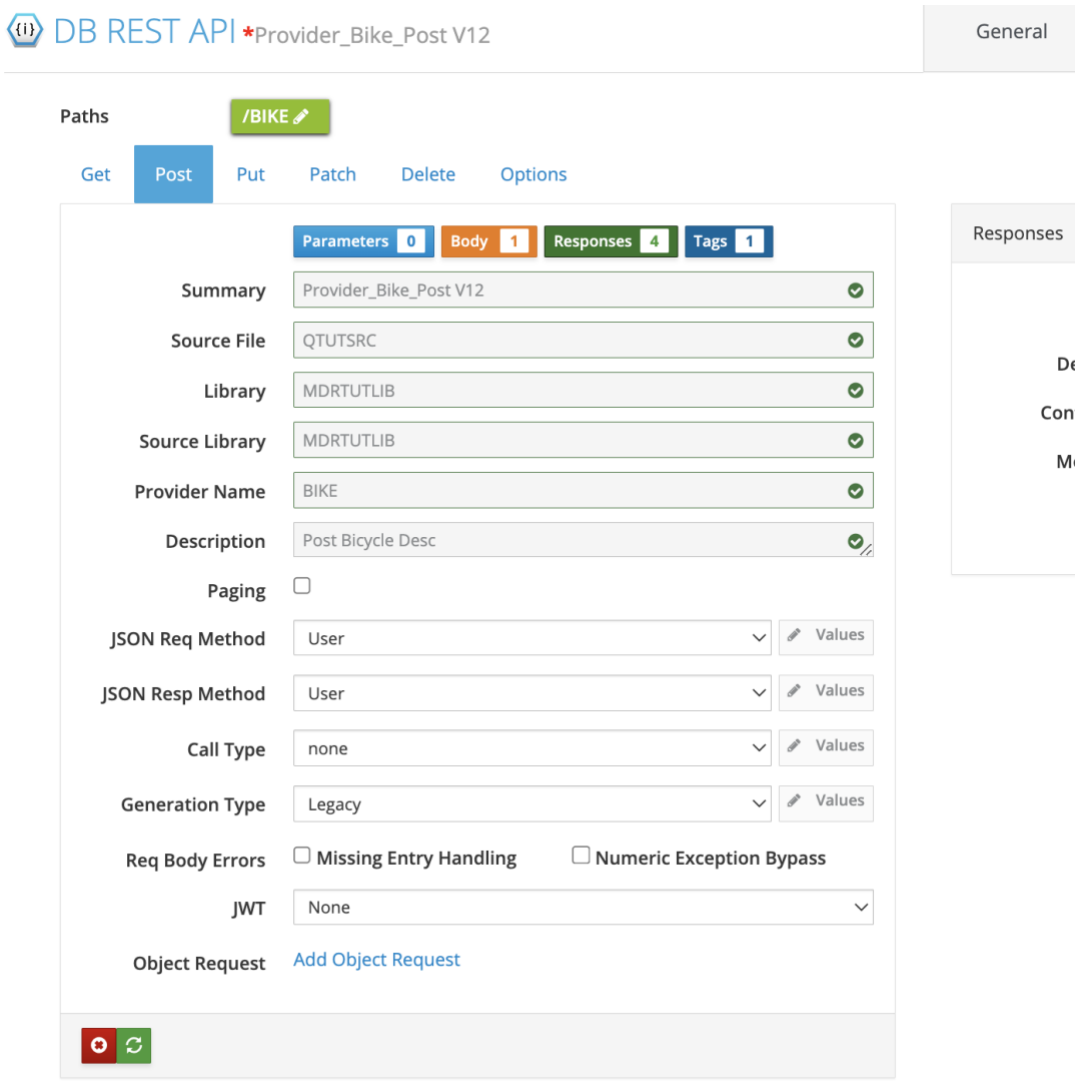


Figure 37 – Select the generation type

4.5.2.8 Create API Service

Select the "Generate" tab and then the "Create Service" button. The service RPGLE code is generated and compiled, and responds with a success message.

4.5.2.9 Generated Code – Parsing requestBody

When the request uses a PUT method, the subroutine "**z_ProcPost**" gets the control after the standard JSONSAX parsing (after the incoming request body has been fully parsed). In this subroutine, the data structure subfields are first loaded from the information received in the request body using the **JPathV** function built into MDRest4i.

Below is a sample pf a payload for the incoming requestBody

```
{
  "Stock": [
   {"Department": "Cycles",
    "Category": "Bikes",
    "Sub-Category": "RTB",
    "Make": "Elbiko",
    "Model": "Alta 5",
    "Price": 12500,
    "Description": "Elbiko Alta 5",
    "Details": "Super light, fast mountain bike.",
    "Sizes": ["51", "55", "58", "60"],
    "Colours": ["White", "Black", "Green", "Red", "Blue"]
    },
   {"Department": "Mountain",
    "Category": "Motors",
    "Sub-Category": "MTB",
    "Make": "ElbikoM",
    "Model": "Alta 6",
    "Price": 14000,
    "Description": "Elbiko Alta 6",
    "Details": "Slightly Heavier than the Alta 5, but much more durable.",
    "Sizes": ["51", "55", "58", "60"],
    "Colours": ["White", "Black", "Green", "Red", "Blue"]
    }
  ]

}
```

The generator creates parsing logic for this automatically at the beginning of the **z_ProcPost** subroutine.

```
0106.00     Begsr z_ProcPost;
0107.00
0108.00       // Start Loading the Request body info
0109.00       w_tidx1 = JGetArrayIdx('Stock');
0110.00       for w_idx2 = 1 to w_tidx1;
0111.00         w_str2 = JPathV('Stock$'+ %editc(w_idx2:'X') + '!.Department');
0112.00         If w_str2 <> '*notFound';
0113.00           d_stock(w_idx2+1).s_Department = w_str2;
0114.00         Endif;
0115.00
0116.00         w_str2 = JPathV('Stock$'+ %editc(w_idx2:'X') + '!.Category');
0117.00         If w_str2 <> '*notFound';
0118.00           d_stock(w_idx2+1).s_Category = w_str2;
0119.00         Endif;
0120.00
```

```
0121.00         w_str2 = JPathV('Stock¢'+ %editc(w_idx2:'X') + '!.Sub-Category');
0122.00         If w_str2 <> '*notFound';
0123.00           d_stock(w_idx2+1).s_SubCategory = w_str2;
0124.00         Endif;
0125.00
0126.00         w_str2 = JPathV('Stock¢'+ %editc(w_idx2:'X') + '!.Make');
0127.00         If w_str2 <> '*notFound';
0128.00           d_stock(w_idx2+1).s_Make = w_str2;
0129.00         Endif;
0130.00
0131.00         w_str2 = JPathV('Stock¢'+ %editc(w_idx2:'X') + '!.Model');
0132.00         If w_str2 <> '*notFound';
0133.00           d_stock(w_idx2+1).s_Model = w_str2;
0134.00         Endif;
0135.00
0136.00         w_str2 = JPathV('Stock¢'+ %editc(w_idx2:'X') + '!.Price');
0137.00         If w_str2 <> '*notFound';
0138.00           monitor;
0139.00             d_stock(w_idx2+1).s_Price = %dec(w_str2:10:0);
0140.00           on-error;
0141.00             clear d_stock(w_idx2+1).s_Price;
0142.00           endmon;
0143.00         Endif;
0144.00
0145.00         w_str2 = JPathV('Stock¢'+ %editc(w_idx2:'X') + '!.Description');
0146.00         If w_str2 <> '*notFound';
0147.00           d_stock(w_idx2+1).s_Description = w_str2;
0148.00         Endif;
0149.00
0150.00         w_str2 = JPathV('Stock¢'+ %editc(w_idx2:'X') + '!.Details');
0151.00         If w_str2 <> '*notFound';
0152.00           d_stock(w_idx2+1).s_Details = w_str2;
0153.00         Endif;
0154.00
0155.00         w_tidx2=JGetArrayIdx('Stock'+ w_opensquare + %char(w_idx2) +
0156.00         w_closesquare + '.Sizes');
0157.00         for w_idx3 = 1 to w_tidx2;
0158.00           w_str2 = JGetElementV('Stock¢'+ %editc(w_idx2:'X') + '!.Sizes':
0159.00           w_idx3);
0160.00           If w_str2 <> '*notFound';
0161.00             d_stock(w_idx2+1).s_Sizes(w_idx3+1) = w_str2;
0162.00           Endif;
0163.00
0164.00         endfor;
0165.00
0166.00         w_tidx3=JGetArrayIdx('Stock'+ w_opensquare + %char(w_idx2) +
0167.00         w_closesquare + '.Colours');
0168.00         for w_idx3 = 1 to w_tidx3;
0169.00           w_str2 = JGetElementV('Stock¢'+ %editc(w_idx2:'X') +
0170.00           '!.Colours': w_idx3);
0171.00           If w_str2 <> '*notFound';
0172.00             d_stock(w_idx2+1).s_Colours(w_idx3+1) = w_str2;
0173.00           Endif;
0174.00
0175.00         endfor;
0176.00
0177.00       endfor;
0178.00
0179.00
0180.00       // Set Appropriate HttpStatus
0181.00       // e.g - SetHttpStatus(202:'Success')
0182.00
0183.00       //  Process Response Data for HTTP Status 200
0184.00       if w_Status = '200';
0185.00         // Start Writing the Response info
0186.00         beginObject(' ');
```

```
0187.00         addChar('rspMessage':%trim(w_rspMessage):'*LAST');
0188.00
0189.00
0190.00       // Process Response Data for HTTP Status 400
0191.00       elseif w_Status = '400';
0192.00
0193.00         // No schema in swagger. Write the logic manually in this section
0194.00
0195.00
0196.00       // Process Response Data for HTTP Status 500
0197.00       elseif w_Status = '500';
0198.00
0199.00         // No schema in swagger. Write the logic manually in this section
0200.00
0201.00       endif;
0202.00         endObject(' ');
0203.00
0204.00       // Logic from custom copybook
0205.00
0206.00
0207.00     Endsr;
0208.00   /End-Fre
e
```

4.5.3 Provider-API-PUT-Client-V12.json

4.5.3.1 Objective and Design

Demonstrate how to specify a requestBody in an update (PUT Method) type API/Provider, and how the MDRest4i generator can build code to parse the incoming requestBody, and also conditional SQL insert code, using a key defined in the schema/model.

4.5.3.2 Import the example SWAGGER to MDRest4i SDK Web UI

This can be found in the specs stored in the user mdrtut.

Select “New>Import Provider” on the right-hand side. Browse to the folder where you saved the downloaded JSON above and import “Provider-API-PUT-Client-V12-swagger.json”.

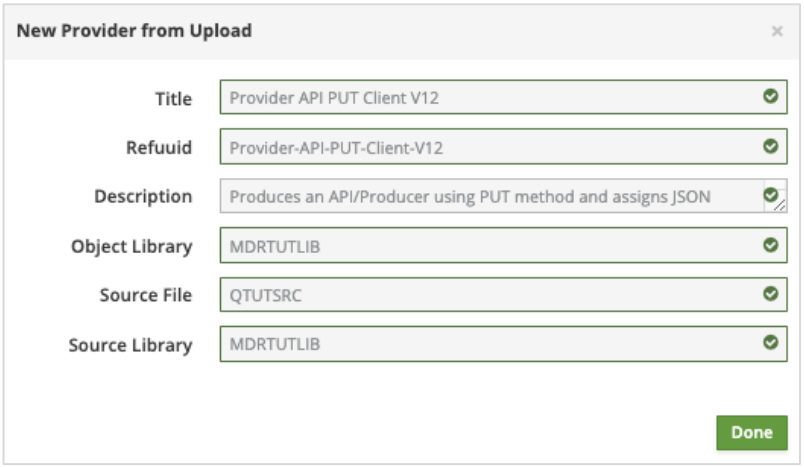


Figure 38 - Import Provider(PUT request example)

Select “Done” from the wizard popup window.

Select the “General” tab, and then “More>save” from the “General” tab to save the specification in your list of specifications.

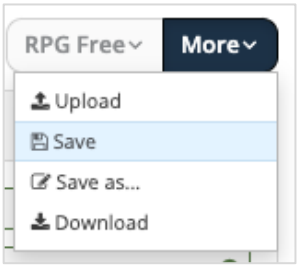


Figure 39 – Save from General Tab

Or by selecting the  button

4.5.3.3 Schemas

Navigate to the schemas tab. Here there are two schemas that have been pre-loaded.

LXCLIENTMDL is for the requestBody – the JSON the API expects will be sent with the request. Select top-most node in the schema tree. This will display the details on the right hand side.

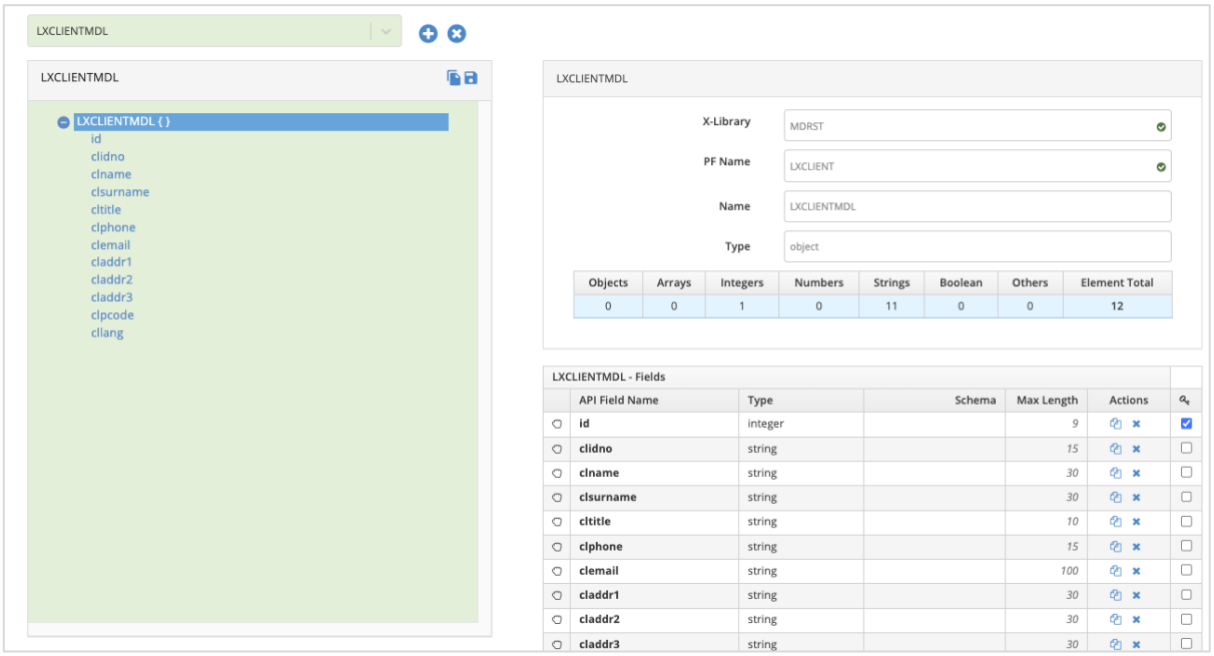


Figure 40- requestBody Schema

and the other is for the response. This JSON will be sent as the response to the client making the request.

Take note of :

The **x-library** and **PFName** values specified. This will tell the generator where to get the field definitions from plus where to insert data if “insert” is selected in the “paths” tab (see below for the description of this)

The selected checkbox against the clidno API field name. This tells the generator which field to use to create RPG code to check if a record exists before inserting.

Select the “id” in the schema tree:

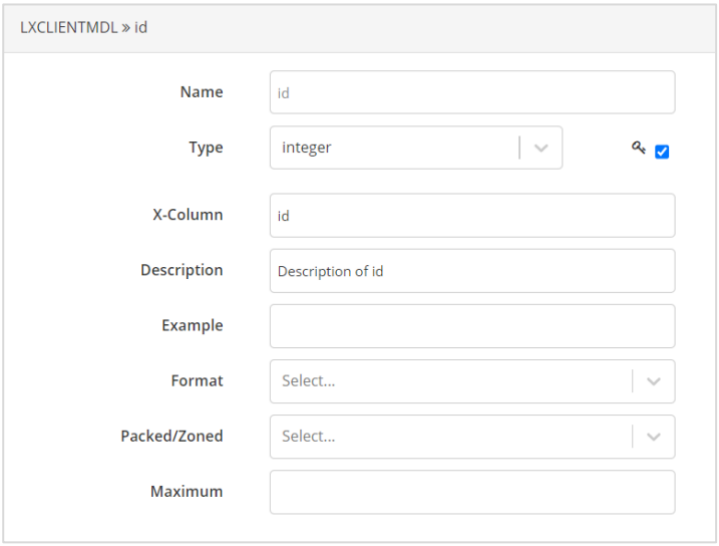


Figure 41 - API Field Details for key field

Take note of :

The selected key checkbox next to the type drop down. This is an alternative way to specify the to generator which field to use to create RPG code to check if a record exists before inserting.

The **x-column** value maps this to appropriate field in the table defined in x-library and x-pfname. The underlying swagger is updated as below:

```
"components": {
 "schemas": {
  "LXCLIENTMDL": {
   "x-library": "MDRST",
   "x-pfname": "LXCLIENT",
   "properties": {
    "id": {
     "type": "integer",
     "description": "Description of id",
     "maxLength": 9,
     "x-column": "id",
     "x-key": "true"
    },
```

4.5.3.4 Paths

Now select the main “Paths” tab, and the select the “Put” method button on the left hand side. This UI is very similar to the GET method, except for PUT, PATCH, and POST there is an additional button “Body” below the source information section on the left hand side.

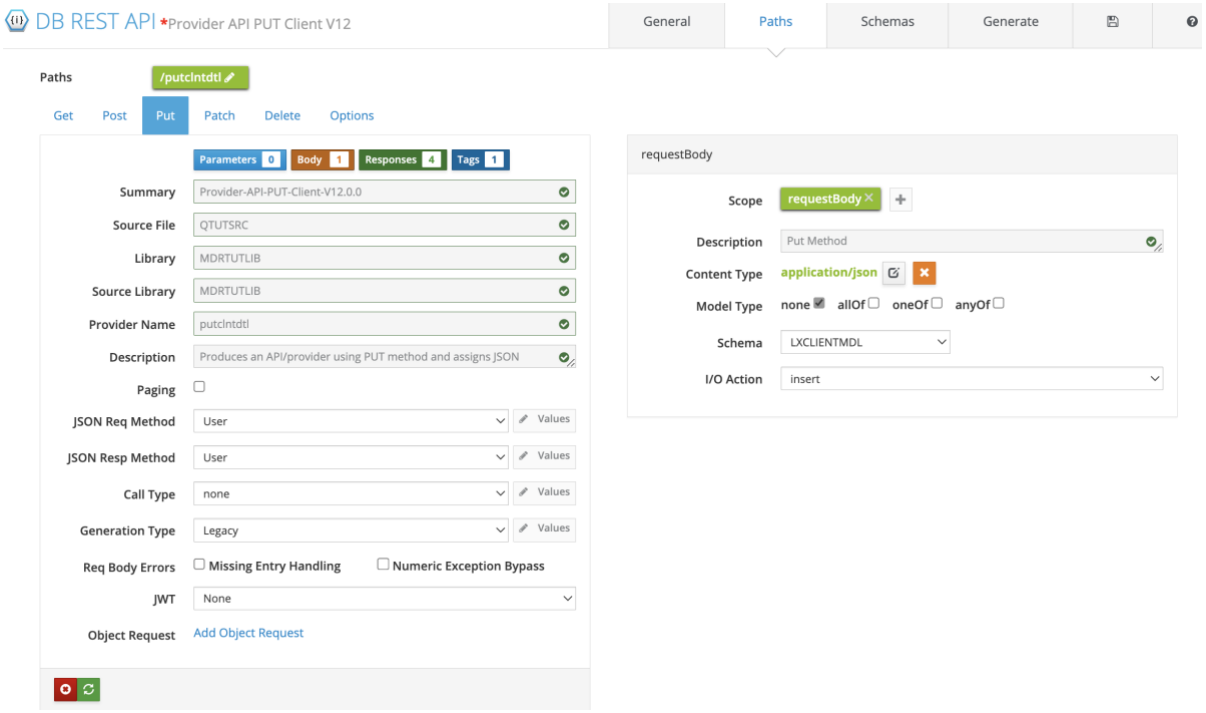


Figure 42 - PUT method requestBody

When this button is selected the "requestBody" section appears. Identical to the response section, it allows the request body schema and format to be selected. There are no parameters required for POST, PATCH, or PUT methods in REST style, although it is not forbidden and sometimes parameters are used with these methods.

4.5.3.5 Create the API Service

Select the "Generate" tab and then the "Create Service" button. The service RPGLE code is generated and compiled, and responds with a success message.

4.5.3.6 Generated Code – Parsing requestBody

When the request uses a PUT method, the subroutine "**z_ProcPut**" gets the control after the standard JSONSAX parsing (after the incoming request body has been fully parsed). In this subroutine, the data structure subfields are first loaded from the information received in the request body using the **JPathV** function built into MDRest4i.

Below is a sample pf a payload for the incoming requestBody

```
{
  "id": "29",
  "clidno": "35792053390",
  "clname": "Richard1",
  "clsurname": "Gail123NEW4",
  "cltitle": "Mr",
  "clphone": "11111111",
  "clemail": "test@test.com",
  "claddr1": "aaaaaaaa",
  "claddr2": "aaaaaaa",
  "claddr3": "aaaaaa",
  "clpcode": "11111111",
  "cllang": "E"
}
```

The generator creates parsing logic for this automatically at the beginning of the **z_ProcPut** subroutine.

```
0013.60     Begsr z_ProcPut;
0014.60
0014.70       // Start Loading the Request body info
0015.70       w_str2 = JPathV('id');
```

```
0016.70       If w_str2 <> '*notFound';
0017.70         monitor;
0018.70           d_lxclientmdl.s_id = %dec(w_str2:9:0);
0019.70         on-error;
0020.70           clear d_lxclientmdl.s_id;
0021.70         endmon;
0022.70       Endif;
0023.70
0024.70       w_str2 = JPathV('clidno');
0025.70       If w_str2 <> '*notFound';
0026.70         d_lxclientmdl.s_clidno = w_str2;
0027.70       Endif;
                      ..........................................
```

It is possible to parse directly into the qualified RPG ds like this:

```
0018.70           d_lxclientmdl.s_id = %dec(JPathV('id'):9:0);
```

However, if the "id" JSON element was not present in the payload, the program would crash. To avoid this, the generator adds additional logic to allow for instances where JSON elements may not be present in the incoming payload.

4.5.3.7 Generated Code – Inserting data records

After generating parsing logic, the MDRGENXAPI has added SQL logic to check the record existence first and if the record is found, it performs update operation, otherwise, it inserts a record. The data loaded in the data structure is used to check the record existence and update/insert operation. The schema field in the swagger with an "x-key" identification is used as the key to check record existence. Below is the excerpt from the swagger showing "x-key" and "x-column" mapping:

Below is the relevant code generated in "z_ProcPut" subroutine for update/insert operations:

```
0121.00       //  Check if record already exists with/without specified keys
0122.00       exec sql select count(*) into :w_idx1 from  LXCLIENT where ID = :
0123.00       d_LXCLIENTMDL.s_id;
0124.00
0125.00       if w_idx1 = *zeros;
0126.00
0127.00         //  Record does not exist, proceed to insert
0128.00         exec sql insert into lxclient (clidno , clname , clsurname ,
0129.00         cltitle , clphone , clemail , claddr1 , claddr2 , claddr3 ,
0130.00         clpcode , cllang) values(:d_lxclientmdl.
0131.00         s_clidno , :d_lxclientmdl.s_clname , :d_lxclientmdl.s_clsurname ,
0132.00         :d_lxclientmdl.s_cltitle , :d_lxclientmdl.s_clphone , :
0133.00         d_lxclientmdl.s_clemail , :d_lxclientmdl.s_claddr1 , :
0134.00         d_lxclientmdl.s_claddr2 , :d_lxclientmdl.s_claddr3 , :
0135.00         d_lxclientmdl.s_clpcode , :d_lxclientmdl.s_cllang);
0136.00
```

```
0137.00       if sqlstt <> c_SqlSuccess;
0138.00         ErrorJSON('Error occurred during insert operation');
0139.00         Leavesr;
0140.00       Endif;
0141.00     else;
0142.00
0143.00       //  Record Exist, update the record
0144.00       exec sql update LXCLIENT set CLIDNO = :
0145.00       d_LXCLIENTMDL.s_clidno , CLNAME = :d_LXCLIENTMDL.s_clname ,
0146.00       CLSURNAME = :d_LXCLIENTMDL.s_clsurname , CLTITLE = :d_LXCLIENTMDL.
0147.00       s_cltitle , CLPHONE = :d_LXCLIENTMDL.s_clphone , CLEMAIL = :
0148.00       d_LXCLIENTMDL.s_clemail , CLADDR1 = :d_LXCLIENTMDL.s_claddr1 ,
0149.00       CLADDR2 = :d_LXCLIENTMDL.s_claddr2 , CLADDR3 = :d_LXCLIENTMDL.
0150.00       s_claddr3 , CLPCODE = :d_LXCLIENTMDL.s_clpcode , CLLANG = :
0151.00       d_LXCLIENTMDL.s_cllang where ID = :d_LXCLIENTMDL.s_id;
0152.00
0153.00       if sqlstt <> c_SqlSuccess;
0154.00         ErrorJSON('Error occurred during update operation');
0155.00         Leavesr;
0156.00       Endif;
0157.00     endif;
```

4.5.4 Consumer-POST-Client-12.json

4.5.4.1 Using API/Provider SWAGGER to generate a REST Consumer in RPGLE

REST architectural style is Client-Server. A RESTful consumer program reverses the logic sequence we used in the API/Provider code in the previous sections. So the output of the requesting consumer becomes the input of the Provider and therefore the requests, payloads and responses must be synchronized. MDRest4i SDK exploits this fact to use a SWAGGER API/Provider specification to generate consumer code that can call the API and process the response.

4.5.4.2 Import the example SWAGGER

This can be found in the downloaded “MDRest4i-v12.0-Tutorial-swagger-rpgle-json-examples/Section-4.5-Swagger-Example-Code” folder.

Select “New>Import Consumer” on the right-hand side. Browse to the folder above and import “4.5.3-Consumer-POST-Client-V12-swagger.json”.

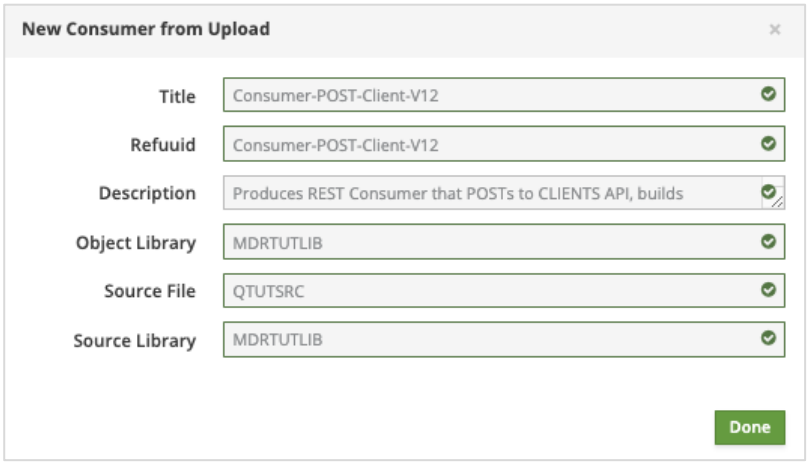


Figure 43 - Import Consumer SWAGGER

Select “Done” to close the wizard popup window. Select the “More>save” button/option on the right-hand side, in the “General” tab to save the specification in your list of specifications.

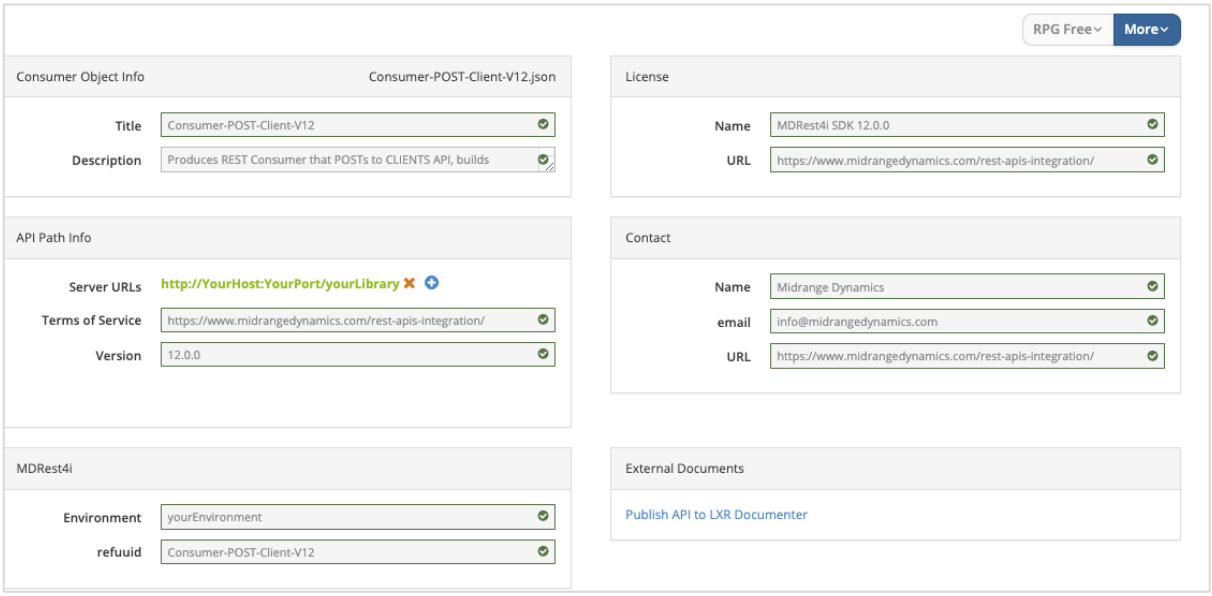


Figure 44 – General tab API

Navigate to the “Paths” tab. Change the consumer name to “postclntac” . This would be the name of the source member and the object name.

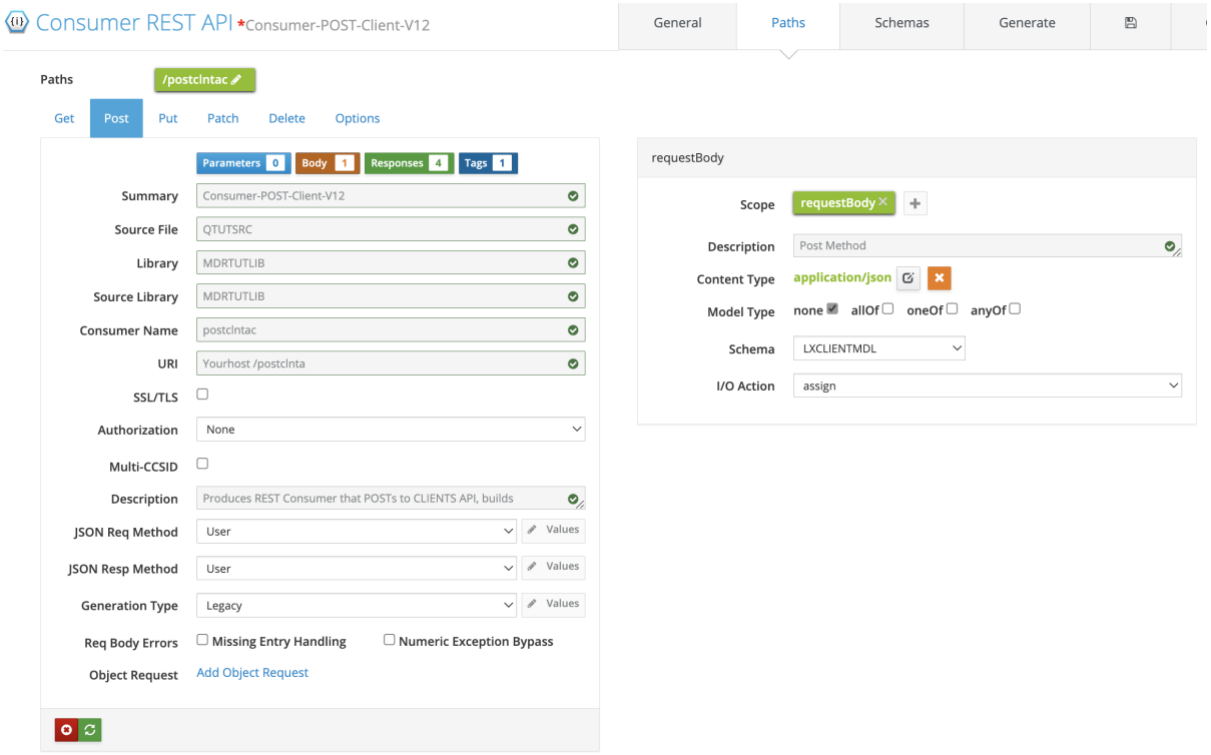


Figure 45 – Paths tab API

4.5.4.3 Create the Consumer Program

Select the “Generate” tab and then the “Create Consumer” button. The consumer RPGLE code is generated and compiled, and a success message is displayed.

4.5.4.4 Generated Code Description

In consumer for POST method, the API generates the logic in BuildRequest procedure for writing the JSON to be sent as part of the request body. This information is written from the data structure based on the request body schema specification in the swagger. This data structure should have the values set already and it is the developer's responsibility to load the data structure from the database file or hard-code the values (if the fix values are to be sent in this request). This data can be loaded either in mainline section of the program or at the top of "BuildRequest" procedure. Below is the sample code generated in request body of POST service:

```
0142.00 p BuildRequest    b                   export
0143.00 d BuildRequest    pi
0144.00  /Free
0145.00
0146.00    //Enter the request body  here
0147.00
0148.00     // Start Writing the RequestBody info
0149.00     beginObject(' ');
0150.00     addIntr('id':d_LXCLIENTMDL.s_id);
0151.00
0152.00     addChar('clidno':%trim(d_LXCLIENTMDL.s_clidno));
0153.00
0154.00     addChar('clname':%trim(d_LXCLIENTMDL.s_clname));
0155.00
0156.00     addChar('clsurname':%trim(d_LXCLIENTMDL.s_clsurname));
0157.00
0158.00     addChar('cltitle':%trim(d_LXCLIENTMDL.s_cltitle));
0159.00
0160.00     addChar('clphone':%trim(d_LXCLIENTMDL.s_clphone));
0161.00
0162.00     addChar('clemail':%trim(d_LXCLIENTMDL.s_clemail));
0163.00
0164.00     addChar('claddr1':%trim(d_LXCLIENTMDL.s_claddr1));
0165.00
0166.00     addChar('claddr2':%trim(d_LXCLIENTMDL.s_claddr2));
0167.00
0168.00     addChar('claddr3':%trim(d_LXCLIENTMDL.s_claddr3));
0169.00
0170.00     addChar('clpcode':%trim(d_LXCLIENTMDL.s_clpcode));
0171.00
0172.00     addChar('cllang':%trim(d_LXCLIENTMDL.s_cllang):' ');
0173.00
0174.00     endObject(' ');
0175.00  /End-Free
0176.00 p BuildRequest    e
```

Below is the relevant section of the swagger related to the object level information. The "library" is used to generate the compiled object, "srcfile" and "srclibrary" are used to generate the source member at that location and "object" is the name used for the source member as well as the compiled object. The specification "reqtype" tells the generator to create the REST consumer, otherwise, it will generate the REST Provider.

```
29    "paths": {
30        "/postclnta": {
31            "post": {
32                "operationId": "postclntac",
33                "x-lxrgen": {
34                    "library": "LXRDEMOD",
35                    "srcfile": "qexamples",
36                    "srclibrary": "LXRDEMOD",
37                    "reqType": "consumer",
38                    "object": "postclntac",
39                    "lxrpath": "http://iseries.rest4i.com:2510/lxr/postclnta"
40                },
```

Below is the logic generated in "Initialize" procedure of the REST consumer. As we can see, the setting of "lxrpath" as shown in above screenshot is used to set the variable "wg_urlpath", "wg_urlport" and "wg_urlhost". If the port number is not specified in "lxrpath" entry, default port 80 will be assigned to "wg_urlport" variable:

```
0075.40 p Initialize      b                   export
0075.50 d Initialize      pi
0075.60  /Free
0075.70
0075.80    // Set Import Values
0075.90    wg_contentType = 'application/json; charset=UTF-8';
0076.00    wg_httpsMethod = 'POST';
0076.10
0076.20    wg_maxAttempt = 25;
0076.30    wg_mSecDelay = 500000;
0076.40
0076.50    wg_hostName = 'your hostname';
0076.60    wg_serverIP= ' ';
0076.70    wg_portNumber = 80;
0076.80
0077.80    wg_servicePath = '/libraryname/apiname';
0078.80    wg_urlparm = *Blanks;
0078.90
0079.00    // Set below indicator to *Off if you don't want to save logs in IFS
0079.10    ng_saveRestSwitch = *On;
0079.20    wg_saveRESTpath ='/mdrest4i/logs/postclntac.txt';
0079.30
0079.40    // The setting "ng_ssl=*On" enables SSL request. Otherwise, it's non SSL
0079.50    ng_ssl = *Off;
0079.60
0079.70    // If the REST service requires authentication, set ng_authsend = *On
0079.80    // and then use one of the three subsequent variables to send auth info.
0079.90    // In order to send the basic authentication via userId and pwd, set the
0080.00    // variables wg_username and wg_password. If however, it's the "Bearer"
0080.10    // token or some other auth string, set the value in "wg_authstring".
0080.20    // e.g. wg_authstring = Bearer <token value>
0080.30
0080.40    ng_authsend = *off;
0080.50    wg_username = *blanks;
0080.60    wg_password = *blanks;
0080.70    wg_authstring = *blanks;
0080.80
0080.90    // If the REST service is accessible via proxy, set ng_proxy = *on
0081.00    // and set the proxy details in subsequent variables. If the proxy
0081.10    // host is in IP address format, set the value in "wg_proxyIP" and if
0081.20    // it's the host name format, set the hostname in the "wg_proxyhost"
0081.30    // variable without any http/https prefix and there shouldn't be any
0081.40    // slash before or after the host name or IP address. The port number
```

```
0081.50    // of the proxy host should be set in "wg_proxyport" variable below.
0081.60    // If the proxy is on SSL channel, set "ng_proxySSL" to *On.
0081.70    ng_proxy = *Off;
0081.80    ng_proxyssl = *Off;
0081.90    wg_proxyhost = *blanks;
0082.00    wg_proxyIp = *blanks;
0082.10    wg_proxyport = *zeros;
0082.20    wg_proxyusr = *blanks;
0082.30    wg_proxypwd = *blanks;
0082.40
0082.50   // If you want to load config via MRCNSDTLF file, please uncomment
0082.60   // the below line. In this case you need to make sure you have loaded
0082.70   // (Host, Port, path etc.) in the MRCNSDTLF file with the key of This
0082.80   // Program name. Add below file in the 'F' spec in this program
0082.90   // mrcnsdtlf if   e           k disk    Usropn
0083.00   // /copy qrpglesrc,mrsetcon
0083.10
0083.20   /End-Free
0083.30 p Initialize      e
```

4.5.5 Consumer-TelephoneNo-Validate-12.json

In this example, a consumer is built that uses a web based REST API from a website called www.numverify.com, for telephone number validation. The API accepts a GET method, receives four parameters, and returns JSON if the number is valid or not.

Note: The purpose of this example is to demonstrate how to build consumers that use remote REST services, and also how to build schemas(from JSON code) and generate the necessary RPGLE definitions and logic to handle any JSON sample supplied for either an API or a Consumer.

4.5.5.1 Self-service REST API documentation

Here is the documentation for the external API:

https://numverify.com/documentation

4.5.5.2 Creating SWAGGER using Sample JSON

From this we created this SWAGGER definition using the MDRest4i SDK Web GUI.

Let's explain how we did this:

In the numverify.com APi documentation page, scroll down to the "Make an API Request" section.

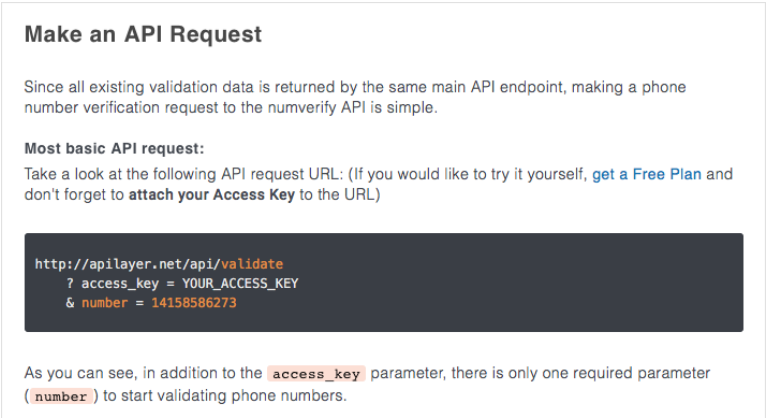


Figure 176 - Numverify.com Parameters

Now in the MDRest4i SDK GUI Specifications List window, select “New>Consumer” from the Specifications window,

The Actual API URI is: http://apilayer.net/api/validate . This is edited in the lxrPath field on the left hand side of the “General” tab, and can also be found in the x-lxrgen extension object under lxrpath.

Now select the Paths Tab and Parameters, and add two Query parameters: one for access_key, and one for number, no schema ref is required, as we are not reading data from a PF/Table with SQL.

Now go back to the numverify.com APi documentation page and scroll down to the API Response section. This is a sample of the JSON sent back by the API.

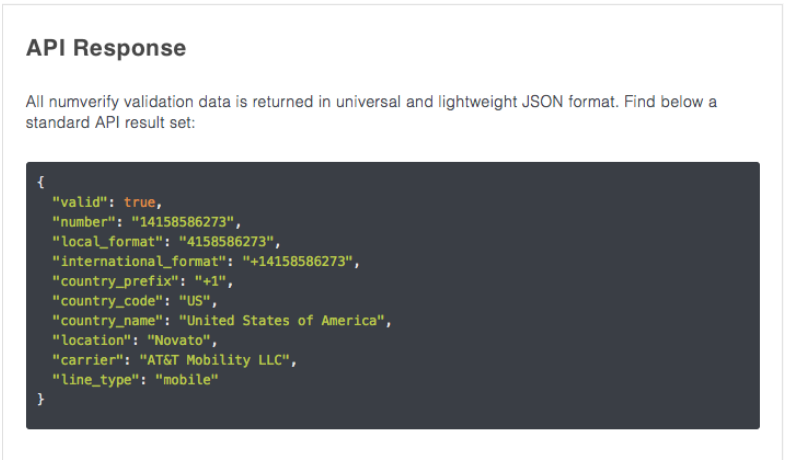
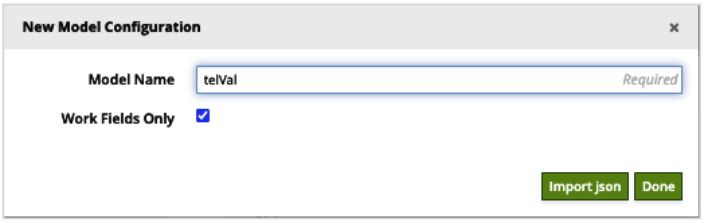


Figure 47 - Numverify.com API Response

Copy the contents of the example response JSON, returned to MDRest4i SDK,

In the schema tab, Select the + button to create a new schema and select the Import json button.



Paste the API Response JSON into the popup window.

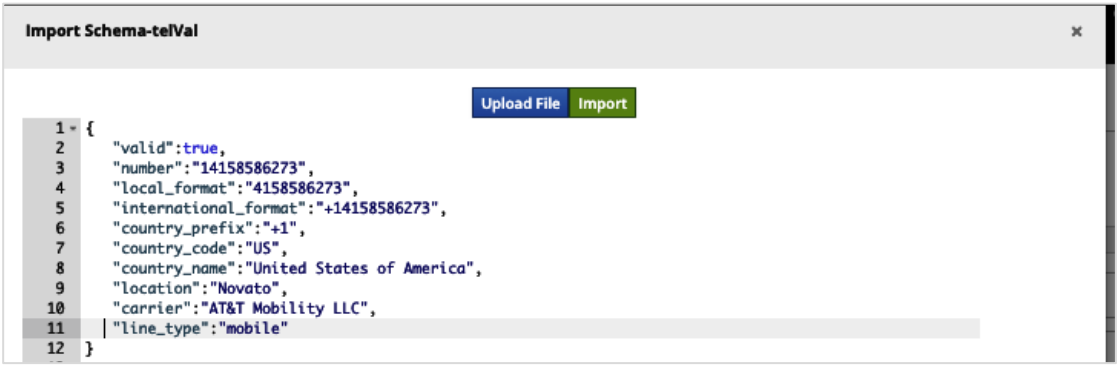


Figure 48 - Import JSON Sample to Schema

Now click the “Import” button.

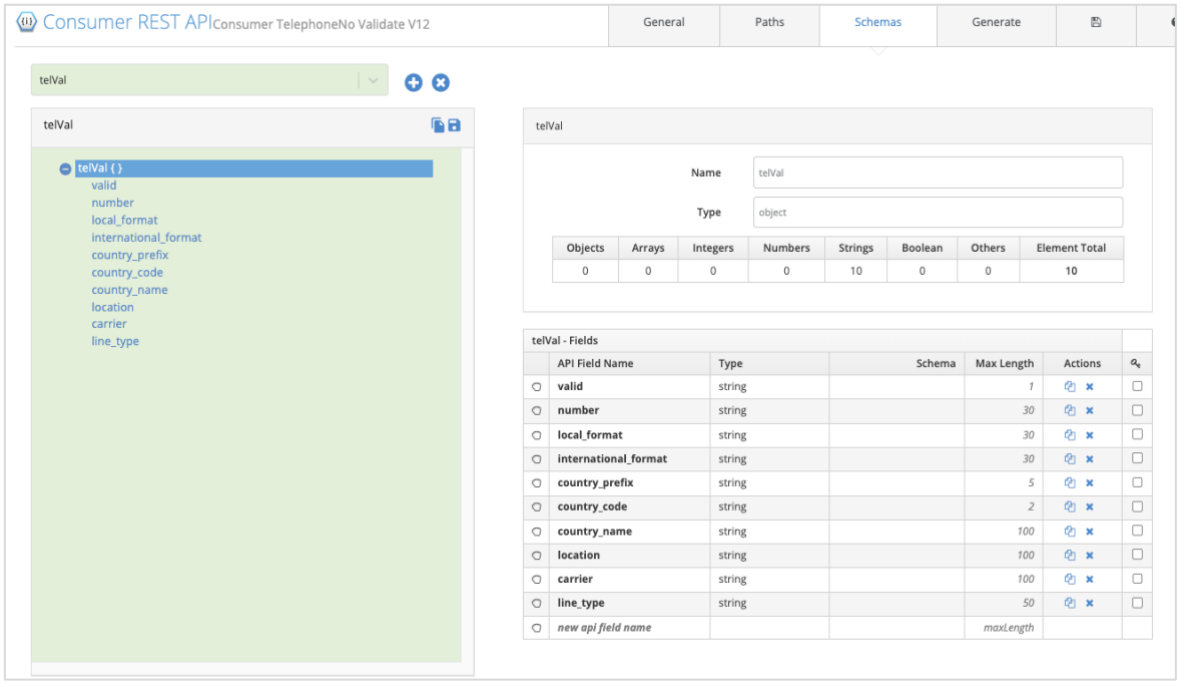


Figure 49 - Imported Schema as workfields

n the Figure above the imported schema is defined as work fields. This is because they have no x-PFname reference to a database file as per the API GET Clients example above. These work fields will then be defined as variables in the generated RPG code below. Logic will also be automatically added to the generated RPG

code that processes the JSON response returned by numverify.net, and assigns these JSON values to the defined work fields, which can be seen here.

4.5.5.3 Generated Code Description

Below are the parameters in swagger (they can be edited in the SWAGGER directly under the “General” tab. As we can see, the data type is specified for the first and fourth parameters but that for second and third is being picked from “x-schema” and “x-schemaFld”. We can also see that the first two parameters are mandatory and the last two are optional since “required” is marked as “false” for them.

```
"parameters": [
     {
      "in": "query",
      "name": "access_key",
      "description": "access key",
      "required": true,
      "schema": {
       "type": "string"
      }
     },
     {
      "in": "query",
      "name": "number",
      "description": "tel number",
      "required": true,
      "schema": {
       "type": "string"
      }
     }
    ]
```

Below is the generated main procedure prototype and procedure interface.

```
0011.10      d w_numparm       s              5s 0
0011.20       * Write the main program prototype and PI
0011.30      d main            pr                  Extpgm('CNSVALTEL')
0011.40      d p_access_key                 50a
0011.50      d p_number                     50a
0012.70
0012.80      d main            pi
0012.90      d p_access_key                 50a
0013.00      d p_number                     50a
```

Below is the relevant part of mainline section. We can see that “w_numparm” variable stores the number of parameters supplied to this program through entry parameter list.

```
0009.20
0009.30 dw_str2           s           2048a
0009.40
0009.50  /free
0009.60
0009.70    InitVariant();
0009.80    tg_InitializePointer = %paddr(Initialize);
0009.90    tg_BuildReqPointer = %paddr(BuildRequest);
0010.00    tg_ClosedownPointer  = %paddr(Closedown);
0010.10    w_numparm = %parms();
0010.20    GskConsume();
0011.20
0011.30    // You may also call "GetErrorWarnings" procedure with one parameter
0011.40    // of 1024a attribute to get any errors/warnings reported in execution
```

```
0011.50
0011.60    // Process Response Data for HTTP Status 200
0011.70    if wg_httpStatus = '200';
0012.70      w_str2 = JPathV('valid');
0013.70      If w_str2 <> '*notFound';
0014.70        d_telval.s_valid = w_str2;
0015.70      Endif;
```

The next relevant code section is in “Initialize” procedure about the mandatory and optional parameters. The parameters were marked as mandatory in swagger and therefore their value is directly appended in “wg_urlparm” variable. “wg_urlparm” is the global variable declared in LXRRESTCM and imported in consumer program to set the query parameter value before consumer makes the call to the Provider.

```
0042.90          wg_httpsMethod = 'GET';
0043.90          wg_urlparm = 'access_key=' + %trim(p_access_key) + '&number=' +
0044.90          %trim(p_number);
0052.00          wg_ContentType = 'application/json; charset=UTF-8';
*****************************************************************
```

We will now make little modifications to this program and use it in a new interactive program for validating the telephone number. We make below changes to the REST consumer created by the generator:

1. Change the definition of “s_valid” from 1A to 10A (to accommodate the true/false value returned by the API).
2. Add another parameter and define it as likeds on “d_telval” which is the response structure available in the consumer

After making above changes, below is the final source of CNSVALTEL program:

```
0000.10 h dftactgrp(*no) actgrp(*new)
0000.20 h bnddir('LXRGLOBAL': 'BNDZIP')
0000.30  ************************************************************
0000.40  *  MDRest4i SSL Consumer Template GET JSON
0000.50  ************************************************************
0000.60
0000.70  /copy qrpglesrc,mdrestdfn
0000.80
0000.90 d Initialize      pr
0001.00
0001.10 d BuildRequest    pr
0001.20 d CloseDown       pr
0002.20
0002.30  * Definitions based on Swagger content
0002.40 d d_telval        ds                  qualified inz
0002.50 d s_valid                      10A
0002.60 d s_number                     30A
0002.70 d s_local_format               30A
```

```
0002.80 d s_international_format...
0002.90 d                               30A
0003.00 d s_country_prefix...
0003.10 d                                5A
0003.20 d s_country_code                 2A
0003.30 d s_country_name               100A
0003.40 d s_location                   100A
0003.50 d s_carrier                    100A
0003.60 d s_line_type                   50A
0004.60
0004.70  * Work variable for SQL processing
0004.80 dw_sqlstmt        s           2048a
0004.90 dw_stmtcnt        s           2048a
0005.00 dw_count          s              5s 0
0006.00
0006.10 dw_numparm        s              5s 0
0006.20  * Write the main program prototype and PI
0006.30 d main            pr                  Extpgm('CNSVALTEL')
0006.40 d p_access_key                  50A
0006.50 d p_number                      50A
0006.60 d d p_telval                            likeds(d_telval)
0007.70
0007.80 d main            pi
0007.90 d p_access_key                  50A
0008.00 d p_number                      50A
0008.10 d d p_telval                            likeds(d_telval)
0009.20
0009.30 dw_str2           s           2048a
0009.40
0009.50  /free
0009.60
0009.70    InitVariant();
0009.80     tg_InitializePointer = %paddr(Initialize);
0009.90     tg_BuildReqPointer = %paddr(BuildRequest);
0010.00     tg_ClosedownPointer  = %paddr(Closedown);
0010.10     w_numparm = %parms();
0010.20     GskConsume();
```

Note: This code is generated from the workfields section of the schema rferenced in the Response part of the swagger

```
0011.30    // You may also call "GetErrorWarnings" procedure with one parameter
```

```
0011.40    // of 1024a attribute to get any errors/warnings reported in execution
0011.50
0011.60    // Process Response Data for HTTP Status 200
0011.70    if wg_httpStatus = '200';
0012.70      w_str2 = JPathV('valid');
0013.70      If w_str2 <> '*notFound';
0014.70        d_telval.s_valid = w_str2;
0015.70      Endif;
0016.70
0017.70      w_str2 = JPathV('number');
0018.70      If w_str2 <> '*notFound';
0019.70        d_telval.s_number = w_str2;
0020.70      Endif;
0021.70
0022.70      w_str2 = JPathV('local_format');
0023.70      If w_str2 <> '*notFound';
0024.70        d_telval.s_local_format = w_str2;
0025.70      Endif;
0026.70
0027.70      w_str2 = JPathV('international_format');
0028.70      If w_str2 <> '*notFound';
0029.70        d_telval.s_international_format = w_str2;
0030.70      Endif;
0031.70
0032.70      w_str2 = JPathV('country_prefix');
0033.70      If w_str2 <> '*notFound';
0034.70        d_telval.s_country_prefix = w_str2;
0035.70      Endif;
0036.70
0037.70      w_str2 = JPathV('country_code');
0038.70      If w_str2 <> '*notFound';
0039.70        d_telval.s_country_code = w_str2;
0040.70      Endif;
0041.70
0042.70      w_str2 = JPathV('country_name');
0043.70      If w_str2 <> '*notFound';
0044.70        d_telval.s_country_name = w_str2;
0045.70      Endif;
0046.70
0047.70      w_str2 = JPathV('location');
0048.70      If w_str2 <> '*notFound';
0049.70        d_telval.s_location = w_str2;
```

```
0050.70      Endif;
0051.70
0052.70      w_str2 = JPathV('carrier');
0053.70      If w_str2 <> '*notFound';
0054.70        d_telval.s_carrier = w_str2;
0055.70      Endif;
0056.70
0057.70      w_str2 = JPathV('line_type');
0058.70      If w_str2 <> '*notFound';
0059.70        d_telval.s_line_type = w_str2;
0060.70      Endif;
0061.70      p_telval = d_telval;
0061.80    Endif;
0062.80
0063.80
0063.90    // Process Response Data for HTTP Status 400
0064.00    if wg_httpStatus = '400';
0065.00
0065.10      // No schema in swagger. Write the logic manually in this section
0066.10
0066.20    Endif;
0067.20
0067.30    // Process Response Data for HTTP Status 500
0067.40    if wg_httpStatus = '500';
0068.40
0068.50      // No schema in swagger. Write the logic manually in this section
0069.50
0069.60    Endif;
0069.70
0069.80    // Cleanup the memory allocated dynamically from heap
0069.90    cleantree();
0070.00
0070.10    *Inlr = *On;
0070.20
0070.30  /End-Free
0070.40  *---------------------------------------------------------
0070.50  * End of Mainline Section
0070.60  *---------------------------------------------------------
0070.70
0070.80  * Procedure Interfaces
0070.90  *---------------------------------------------------------
0071.00
```

```
0071.10 p CloseDown       b                   export
0071.20 d CloseDown       pi
0071.30  /Free
0071.40
0071.50    //Enter service closedown instructions here
0071.60  /End-Free
0071.70 p CloseDown       e
0071.80
0071.90 p BuildRequest    b                   export
0072.00 d BuildRequest    pi
0072.10  /Free
0072.20
0072.30    //Enter the request body  here
0072.40  /End-Free
0072.50 p BuildRequest    e
0072.60
0072.70 p Initialize      b                   export
0072.80 d Initialize      pi
0072.90  /Free
0073.00
0073.10    // Set Import Values
0073.20    wg_contentType = 'application/json; charset=UTF-8';
0073.30    wg_httpsMethod = 'GET';
0073.40
0073.50    wg_maxAttempt = 25;
0073.60    wg_mSecDelay = 500000;
0073.70
0073.80    wg_hostName = 'apilayer.net';
0073.90    wg_serverIP= ' ';
0074.00    wg_portNumber = 80;
0074.10
0075.10    wg_servicePath = '/api/validate';
0076.10    wg_urlparm = 'access_key=' + %trim(p_access_key)+ '&number=' + %trim(
0077.10    p_number);
0084.20
0084.30    // Set below indicator to *Off if you don't want to save logs in IFS
0084.40    ng_saveRestSwitch = *On;
0084.50    wg_saveRESTpath ='/mdrest4i/logs/cnsvaltel.txt';
0084.60
0084.70    // The setting "ng_ssl=*On" enables SSL request. Otherwise, it's non SSL
0084.80    ng_ssl = *Off;
0084.90
```

```
0085.00    // If the REST service requires authentication, set ng_authsend = *On
0085.10    // and then use one of the three subsequent variables to send auth info.
0085.20    // In order to send the basic authentication via userId and pwd, set the
0085.30    // variables wg_username and wg_password. If however, it's the "Bearer"
0085.40    // token or some other auth string, set the value in "wg_authstring".
0085.50    // e.g. wg_authstring = Bearer <token value>
0085.60
0085.70    ng_authsend = *off;
0085.80    wg_username = *blanks;
0085.90    wg_password = *blanks;
0086.00    wg_authstring = *blanks;
0086.10
0086.20    // If the REST service is accessible via proxy, set ng_proxy = *on
0086.30    // and set the proxy details in subsequent variables. If the proxy
0086.40    // host is in IP address format, set the value in "wg_proxyIP" and if
0086.50    // it's the host name format, set the hostname in the "wg_proxyhost"
0086.60    // variable without any http/https prefix and there shouldn't be any
0086.70    // slash before or after the host name or IP address. The port number
0086.80    // of the proxy host should be set in "wg_proxyport" variable below.
0086.90    // If the proxy is on SSL channel, set "ng_proxySSL" to *On.
0087.00    ng_proxy = *Off;
0087.10    ng_proxyssl = *Off;
0087.20    wg_proxyhost = *blanks;
0087.30    wg_proxyIp = *blanks;
0087.40    wg_proxyport = *zeros;
0087.50    wg_proxyusr = *blanks;
0087.60    wg_proxypwd = *blanks;
0087.70
0087.80  // If you want to load config via MRCNSDTLF file, please uncomment
0087.90  // the below line. In this case you need to make sure you have loaded
0088.00  // (Host, Port, path etc.) in the MRCNSDTLF file with the key of This
0088.10  // Program name. Add below file in the 'F' spec in this program
0088.20  // mrcnsdtlf if   e           k disk    Usropn
0088.30  // /copy qrpglesrc,mrsetcon
0088.40
0088.50  /End-Free
0088.60 p Initialize      e
0088.70
0088.80  * ------------------------------------------------------------------------
0088.90  *  Procedure - FixReprocess
0089.00  * ------------------------------------------------------------------------
0089.10 p FixReprocess    b                   export
```

```
0089.20 d FixReprocess    pi
0089.30  /free
0089.40  /end-free
0089.50 p FixReProcess    e
0089.60  * ------------------------------------------------------------------------
0089.70  *  Procedure - InitVariant
0089.80  * ------------------------------------------------------------------------
0089.90 p InitVariant     b
0090.00 d InitVariant     pi
0090.10  /free
0090.20    w_OpenCurly = %char(c_OpenCurly);
0090.30    w_CloseCurly = %char(c_CloseCurly);
0090.40    w_OpenSquare = %char(c_OpenSquare);
0090.50    w_CloseSquare = %char(c_CloseSquare);
0090.60    w_EscapeChar = %char(c_EscapeChar);
0090.70    w_Power = %char(c_Power);
0090.80    w_Tilde1 = %char(c_Tilde1);
0090.90    w_Exclaim = %char(c_Exclaim);
0091.00    w_Hash = %char(c_Hash);
0091.10    w_Pipe1 = %char(c_Pipe1);
0091.20    w_Accent = %char(c_Accent);
0091.30    w_Doller = %char(c_Doller);
0091.40    w_AtSign = %char(c_AtSign);
0091.50
0091.60  /end-free
0091.70 p InitVariant     e
```

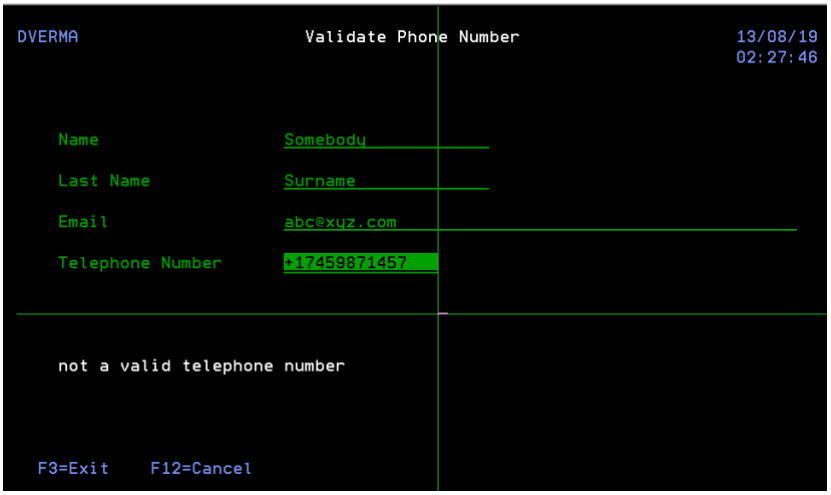
4.5.5.4 Integrating a REST Consumer into an RPG Display Program

We then write another program with display file where we use this consumer program to get the information about the telephone number. We take the telephone number as the input from the screen and supply the fixed access key and supply these two values to the consumer program. We supply the third parameter which is a data structure of the response returned by the telephone number validation REST service in REST consumer. We use this information to display on screen. Below is the complete source of “NUMVERFPGM”.

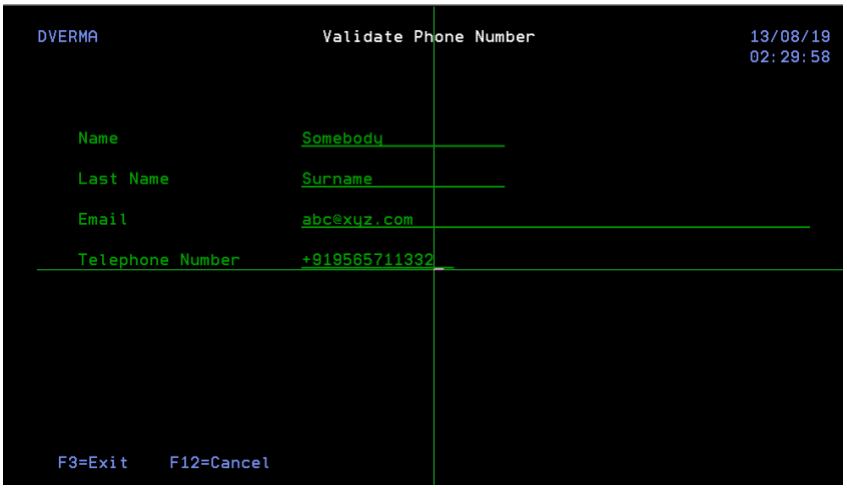
```
     FNumVerfDspCF   E             Workstn
     D w_AccessKey     S             50A
     D w_TelNum        S             30A

     D D_s             DS                   Qualified
     D Lx_Valid                      10a
     D Lx_Number                     30a
     D Lx_Local_Format...
     D                               30A
     D Lx_International_Format...
     D                               30A
     D Lx_Country_Prefix...
     D                                5A
     D Lx_Country_Code...
     D                                2A
     D Lx_Country_Name...
```

```
D                                100A
D Lx_Location                    100a
D Lx_Carrier                     100a
D Lx_Line_Type                    50a

D CnsValTel       Pr                   ExtPgm('CNSVALTEL')
d p_access_key                    50A
d p_number                        30A
d p_ds                                 likeds(D_s)

 /Free
  w_AccessKey = '78c65fdbcca02f6cb94b7e27ea3041a0';

  Write(E) Header;
  Write(E) Footer;
  ExFmt RcdFmt1;
  dow *In03 = *Off And *In12 = *Off;

    w_TelNum = TelNum;
    CnsValTel(w_AccessKey:w_TelNum:D_s);
    If d_s.lx_valid = 'false';

      Error = 'not a valid telephone number';
      *In45 = *On;
      Write(E) Header;
      Write(E) Footer;
      ExFmt RcdFmt1;
      *In45 = *Off;
      Error = *blanks;

    Else;

      Msg1 = 'message : valid number';
      Valid = D_S.Lx_Valid;
      LFormat = D_S.Lx_Local_Format;
      Number = D_S.Lx_Number;
      IFormat = D_S.Lx_International_Format;
      CPrefix = D_S.Lx_Country_Prefix;
      CCode = D_S.Lx_Country_Code;
      CName =  D_S.Lx_Country_Name;
      Location = D_S.Lx_Location;
      Carrier = D_S.Lx_Carrier;
      LineType = D_S.Lx_Line_Type;
      Write(E) Header;
      Write(E) Footer;
      ExFmt RcdFmt2;
      Msg1 = *Blanks;

      If *In12 = *On;
        *In12 = *Off;
        Write(E) Header;
        Write(E) Footer;
        ExFmt RcdFmt1;
      EndIf;

    EndIf;

  EndDo;
  // Terminate The Program
  *InLr = *On;
 /End-Free
```

We enter any number on screen and if its not valid, it will give the response appropriately:



Likewise, if the entered number is valid:



We press enter, below information is displayed:

```
DVERMA                      Validate Phone Number                  13/08/19
                                                                   02:30:37

         message : valid number

         Valid                          true
         Number                         919566711332
         Local_Format                   09566711332
         International_Format           +919566711332
         Country_Prefix                 +91
         Country_Code                   IN
         Country_Name                   India (Republic of)
         Location                       Uttar Pradesh (East)
         Carrier                        Vodafone Idea Ltd (formerly Vo
         Line_Type                      mobile




  F3=Exit     F12=Cancel
```

4.5.6 Provider-API-GET-Clients-SQL-LocalV12

4.5.6.1 Objective and Design

Demonstrate how to use MDRest4i SDK to create a REST API (Provider/service) that reads a DB2 PF/Table and returns the records as JSON

A REST Provider using the GET method will be created, that uses the PF fields as output. Create New Provider

From the API Specifications list in the MDRest4i SDK Web GUI, select the “New” button on the right-hand side and select “Provider” as displayed in Figure below:

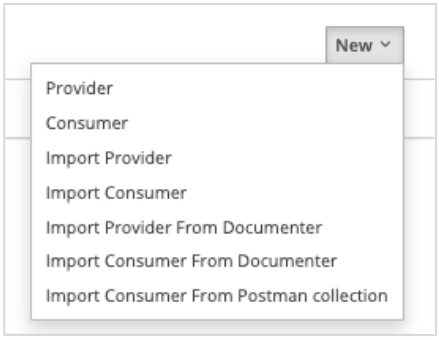


Figure: New > Provider

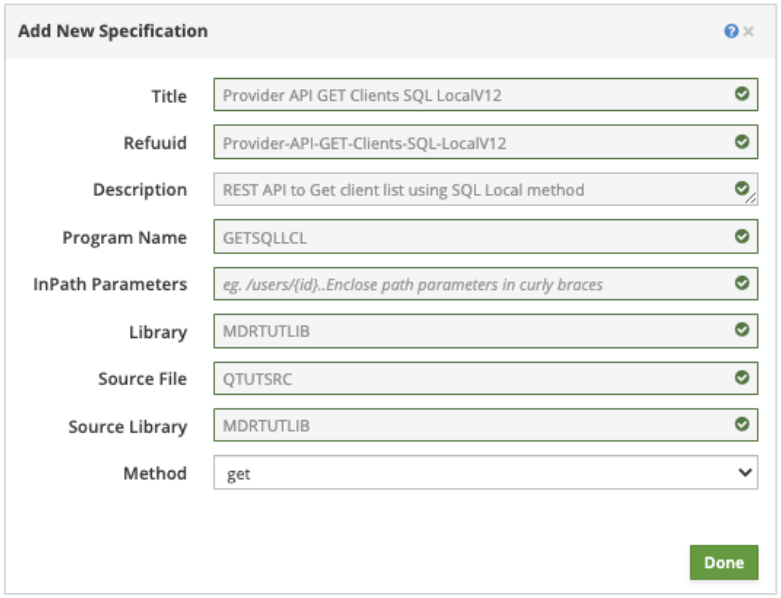


Figure - New API Popup

The default values for the Library, Source Library and Source File can be set in the user profile using the "**Edit Profile**" option (refer to Figure 10(a) and 10(b)). The values would be automatically picked up from the user profile.

If the default values are not provided, you need to enter them manually every time while creating a spec.

Complete the fields in the popup that appears as per the figure above. Select the "get" method in the drop down, before selecting "Done".

This has now copied the SWAGGER template for API's, added a few additional SWAGGER extension fields with the data provided by you in the popup, and created a new SWAGGER definition in JSON format in memory.

4.5.6.2 Adding General Info

The General tab as seen below now appears.

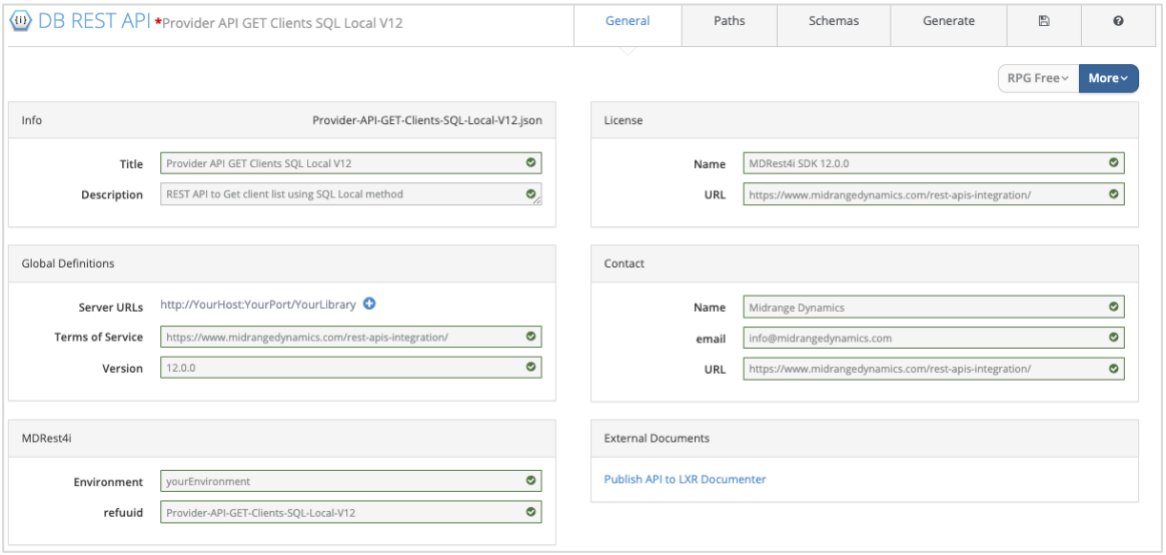


Figure - General Tab API

The only thing that needs updating is the server URL's. Add the host name, Port number and library name used when setting up the MDRest4i HTTP server during installation.

This can again be set up as the default in the "**Edit Profile**"option (Figure 10(b)). If this is not available, it can be added later. The rest is general information and can be left as is for this tutorial.

Now select the "More" button and "Save" from the drop down menu or using the "Save" button".

4.5.6.3 Create a Schema

Now select the "Schemas" Tab.

Click the button to add a new schema.

1. Enter the Model name.
2. Select **SQL** from the **Import Type** dropdown menu.
3. Select **Local/Manual** from the **From** dropdown menu.
4. Click the "Continue" button.

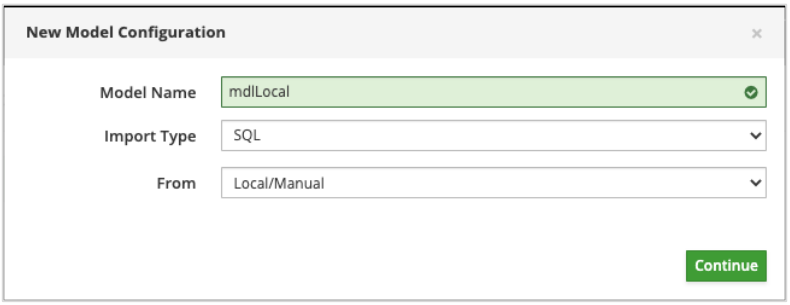


Figure: Add New Schema

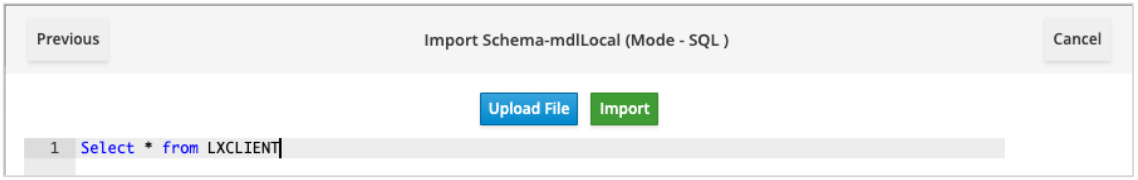
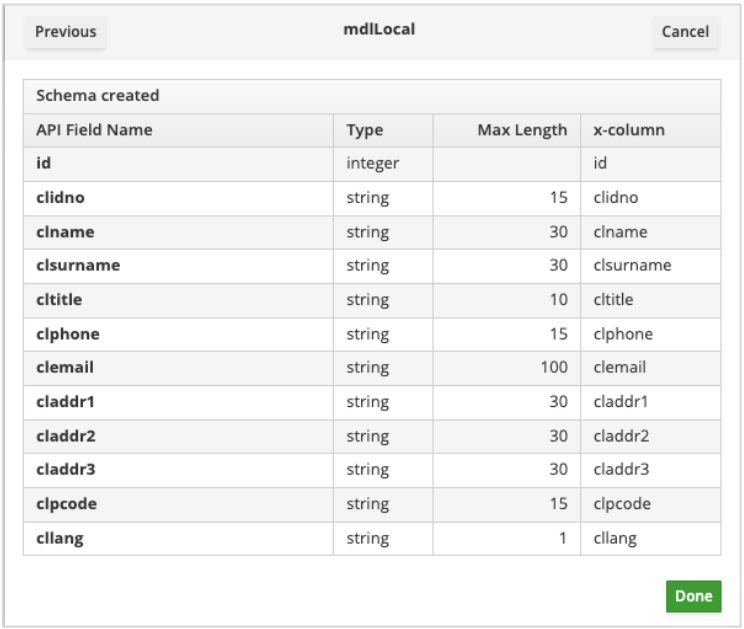


Figure: Schema Editor



API Field Name	Type	Max Length	x-column
id	integer		id
clidno	string	15	clidno
clname	string	30	clname
clsurname	string	30	clsurname
cltitle	string	10	cltitle
clphone	string	15	clphone
clemail	string	100	clemail
claddr1	string	30	claddr1
claddr2	string	30	claddr2
claddr3	string	30	claddr3
clpcode	string	15	clpcode
cllang	string	1	cllang

Figure: API Fields

This information is extracted in real-time by another iCore REST API on the IBM i. The API field name is taken from the field text and the x-column from the PF field name itself. This generates a schema in the SWAGGER specification that maps the API field name to the PF Field name. Figure below is a snippet of that schema.

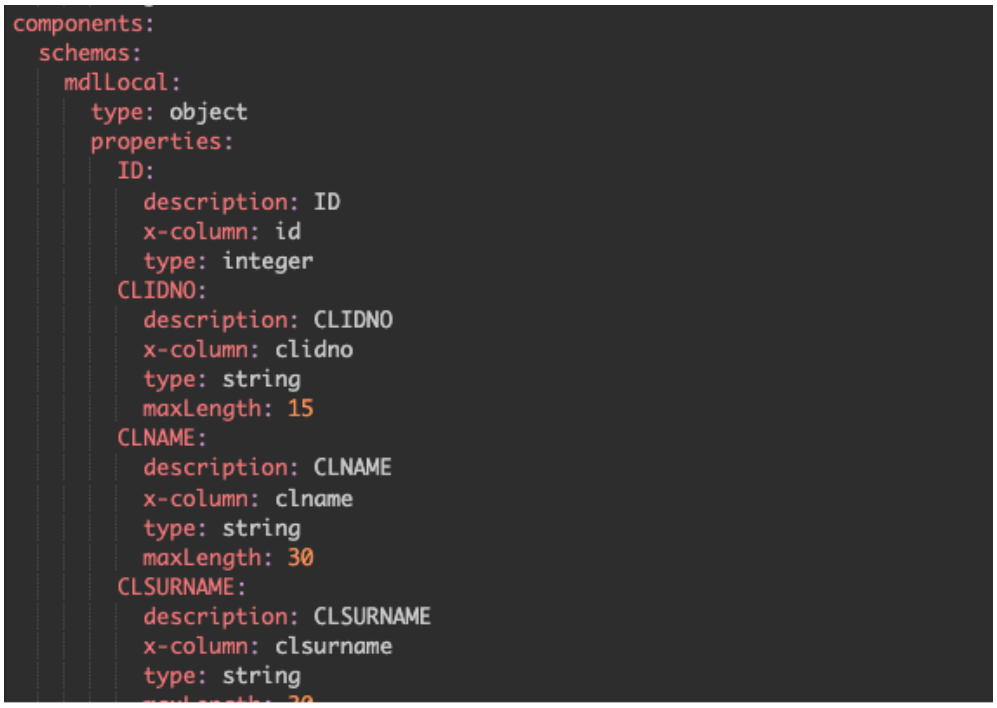


Figure – SWAGGER Schema Snippet

Some external fields also get added to the schema object like so:

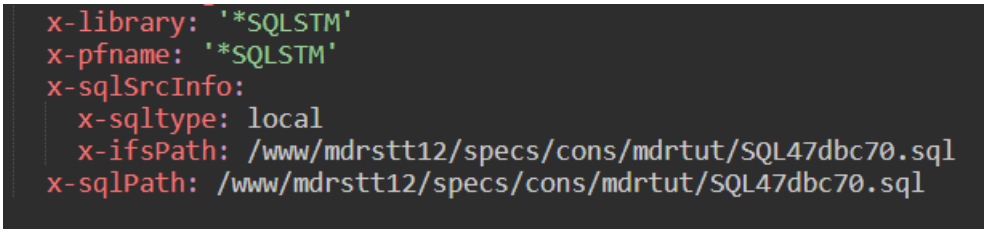


Figure – SWAGGER Schema Snippet (External Fields)

4.5.6.4 Edit API Path Info

Now select the Responses button on the left hand side of the paths tab.

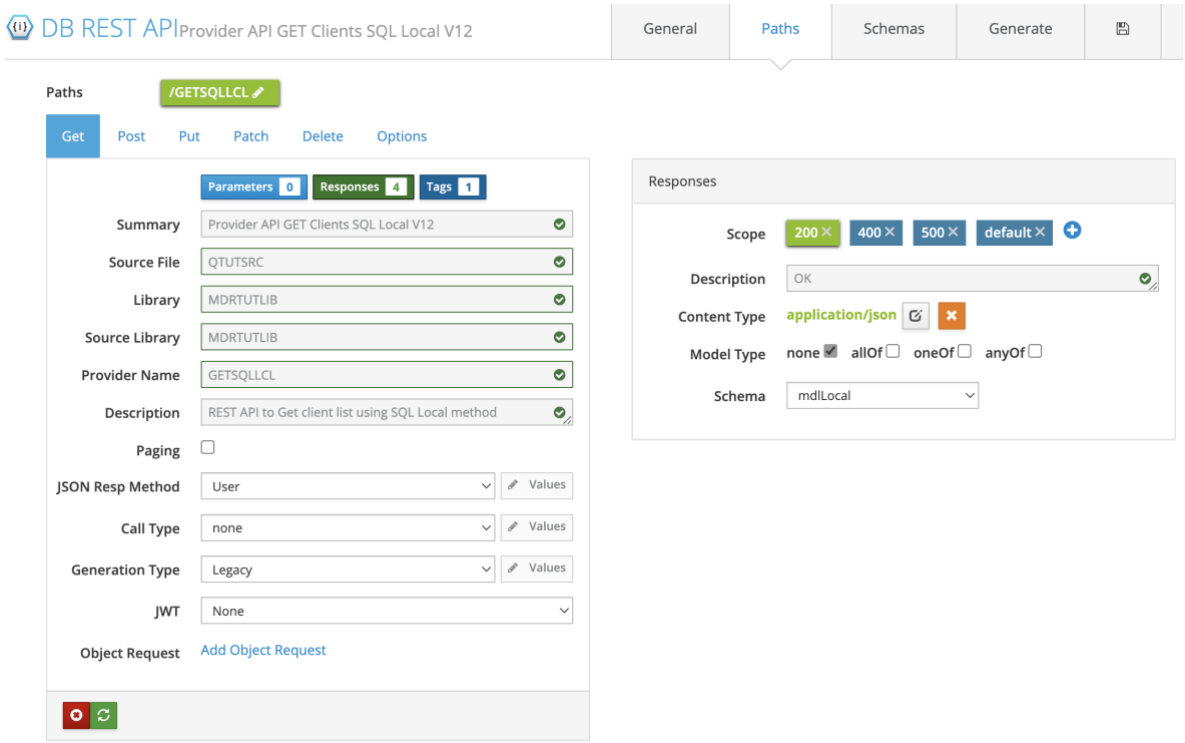


Figure - Edit API Response

By default, four HTTP responses are added to each API. Response 200 is the default status header field and value, which tells the requesting browser or consumer, that everything is OK. To generate RPGLE logic in the Provider that sends back JSON in the response, the content type and schema used to define the response must be added.

This will tell the RPGLE generator to add logic that sends back the correct status HTTP header field. The generator will also automatically build embedded SQL in the RPGLE (to get the data from the PF named in the schema earlier), plus generate logic to assign data from the embedded SQL result set, while building the JSON response body, which is then sent back to the requesting browser or consumer.

From the “Responses” section on the right hand side, select the content type, “application/json” from the drop down, and then select the “LXCLIENTMDL” schema created earlier from the Schema drop down.

In this example we are using a schema that has PF information included in it. This uses the PF FFD to create the necessary variables in the generated RPG logic along with building the SQL statement/s. If only work fields were in the schema, the RPGLE code generated would also contain these work field definitions.

4.5.6.5 Create API Service

Now select the “Generate” tab.

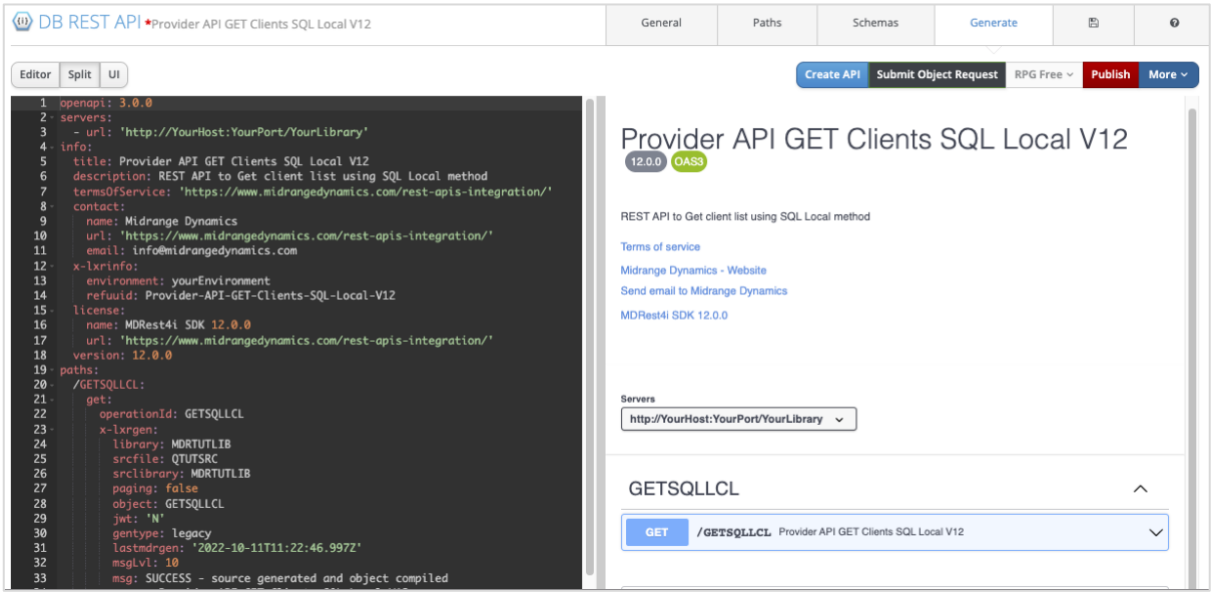


Figure – Generate tab

Note: The Swagger Specification is updated automatically in memory with each change made on the screen but this does **NOT** save those changes to the specifications database.

From the display seen in above Figure, select the "more" button on the right-hand side and save the specification or you can use save button also. This updates the details added since the last save in the specifications database.

The "Generate" tab is a custom version of a full SWAGGER Editor. The left hand window enables editing of the SWAGGER specification directly, and the right-hand side shows the documentation and allows testing of the Provider once it is generated and completed.

Select the "Create Service" button.

Figure - Create Service

Select the "get" method from the drop down and click the "Generate" button.

Upon completion of the generation process on the IBM i, Figure will appear as below:

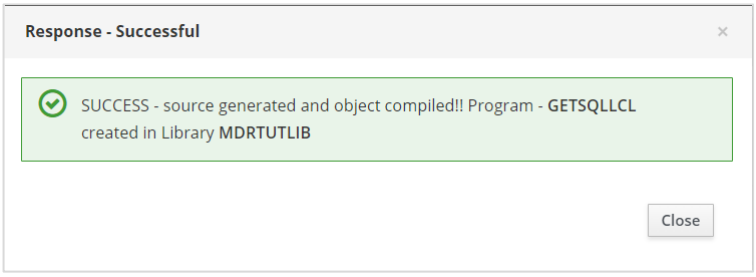


Figure Response – Successful

4.5.6.6 Generated Code Description

The code generated is almost identical to the code described in the command-generated example CLIENTS above.

4.5.6.7 Test the Generated Service

In the "Generate" tab edit the "- URL: 'http://yourhost:port/basePath' " value (in the left hand SWAGGER window as per above Figure of Generate tab) so that yourhost= your IBM i http server where MDRest4i server was setup, port is the port number used, and basePath is the name of the library where you generated the service into. Please ensure this service was generated into a library that was mapped in the http server in the first instance during installation. For example http://yourserver:yourport/yourlib where yourlib was added as a Library Name during the installation process.

Note: If you wish to have a separate HTTP instance for running your generated API's please refer to the section "Creating the default MDRest4i HTTP Server Instance" in the "MDRest4i_12.0_Installation_Instructions" guide

Now on the right hand side of the "Generate" tab, scroll down until the server and "GET" button appears. Click on the "GET" Button seen in Figure to expand it.

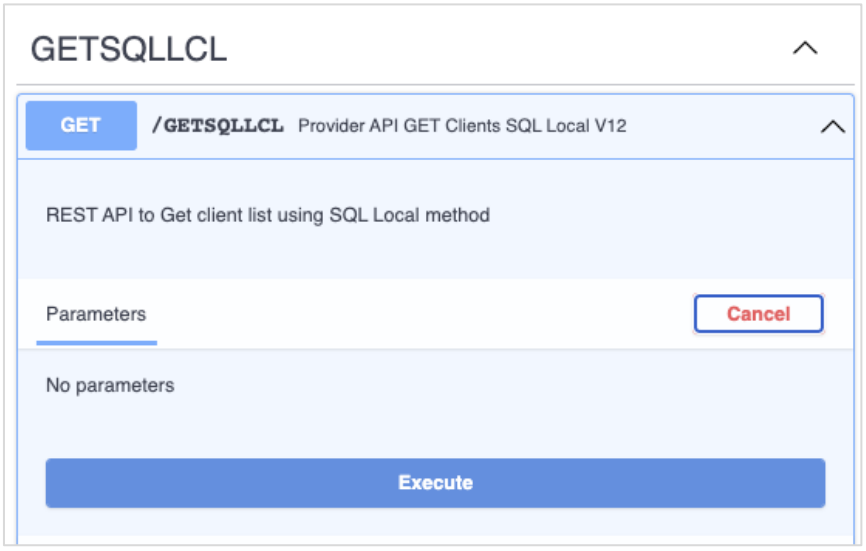


Figure - Execute API Test

Scroll down to the "Server response" section.

Figure - Execute API Test

The response body can be seen as JSON in this window. This was created in the RPGLE by the beginObject, addChar, addDeci, addIntr, addBool, etc. functions in the z_procGet subroutine.

Note: Scroll up to the "Request URL" above the request body. Copy the code in the Request URL and paste into the address bar or a new window in your web browser. The same JSON as above will be returned.

TopTip: To see formatted JSON from response in your browser, there are a number of extensions or add-ons available for your browser. For example, Chrome has two useful ones: JSON Viewer for simple viewing and Advanced REST client (referred to as ARC) for a full-blown testing tool.

4.5.7 Provider-API-GET-Clients-V12-SQL-IFS

4.5.7.1 Objective and Design

Demonstrate how to use MDRest4i SDK to create a REST API that reads a DB2 PF/Table and returns the record/s as JSON, allowing specific records need to be requested if necessary(not mandatory).

A REST Provider using the GET method will be created, that uses the PF fields as output. The key of the PF/Table may be supplied as a query string parameter.

4.5.7.2 Create New Provider

From the API Specifications list in the MDRest4i SDK Web GUI, select the "New" button on the right-hand side and select "Provider" as displayed in the figure below:

The default values for the Library, Source Library and Source File can be set in the user profile using the "**Edit Profile**" option (refer to Figure 10(a) and 10(b)). The values would be automatically picked up from the user profile.

If the default values are not provided, you need to enter them manually every time while creating a spec.

Complete the fields in the popup that appears as per Figure 9 above. Select the "get" method in the drop down, before selecting "Done".

This has now copied the SWAGGER template for API's, added a few additional SWAGGER extension fields with the data provided by you in the popup, and created a new SWAGGER definition in JSON format in memory.

4.5.7.3 Adding General Info

The General tab seen in Figure below now appears.

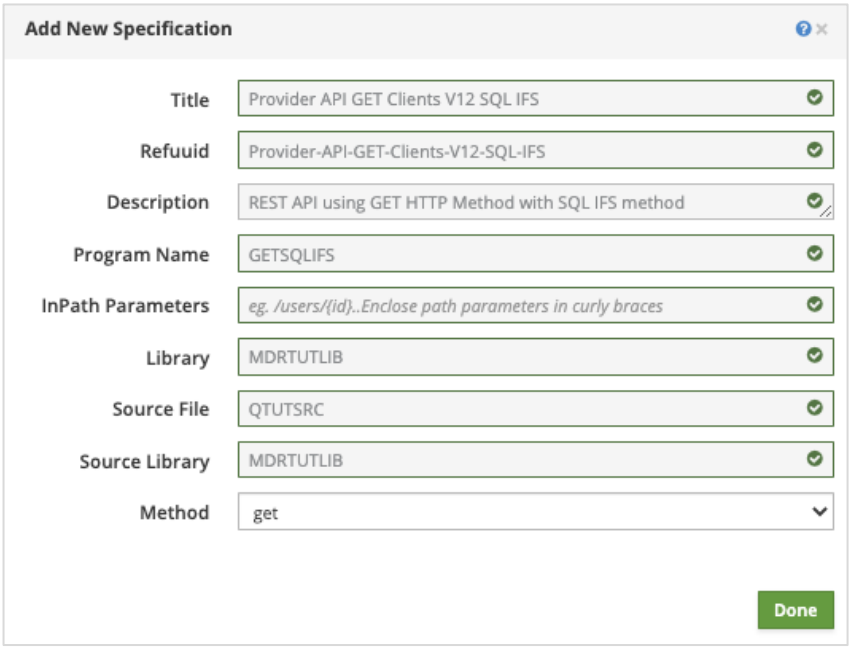


Figure – Add New Specification

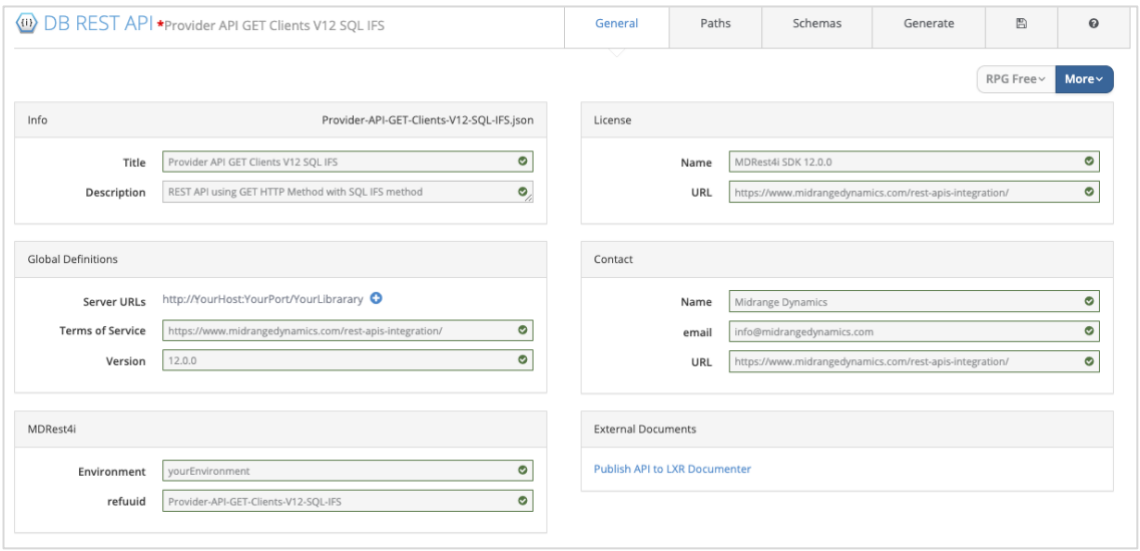


Figure - General Tab API

The only thing that needs updating is the server URL's. Add the host name, Port number and library name used when setting up the MDRest4i HTTP server during installation.

This can again be set up as the default in the "**Edit Profile**"option (Figure 10(b)). If this is not available, it can be added later. The rest is general information and can be left as is for this tutorial.

Now select the "More" button and "Save" from the drop down menu.

4.5.7.4 Create a Schema

Now select the "Schemas" Tab. Click the  button to add a new schema .

The following popup appears.

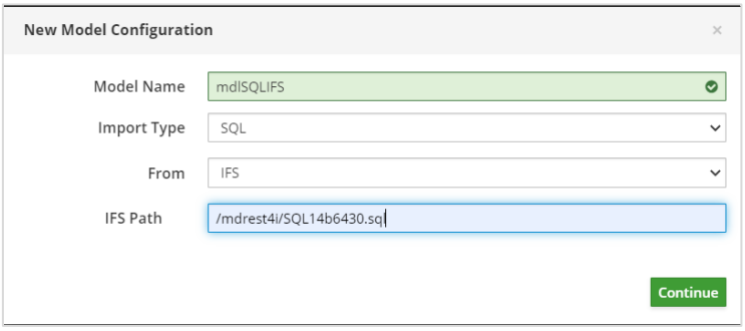


Figure - Add a Schema

The schema editor window appears where the imported SQL statement can be seen and edited.
The SQL statement edit does not change the original sql file. The user is issued a warning in case any changes are made in the SQL statement.

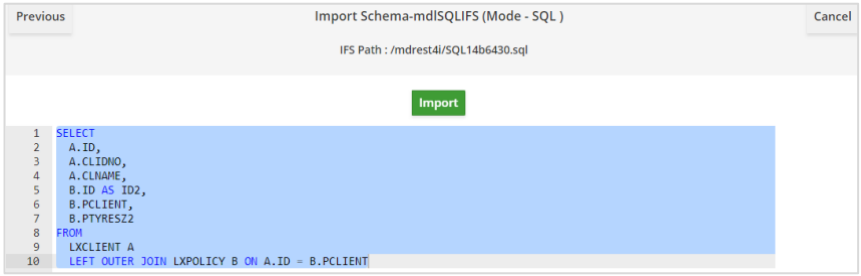


Figure – Schema Editor

Click the “Import” button. The list of fields comes up.

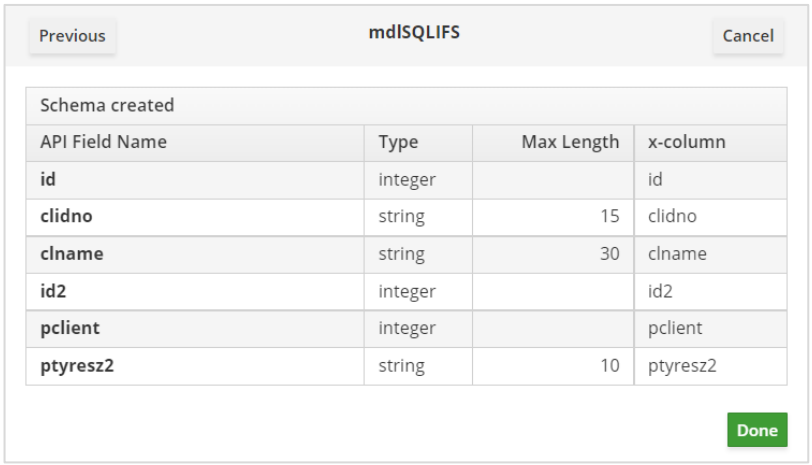


Figure – API Fields List

The user is allowed to edit the API Fields by clicking on the fields.

Click “Done” to save your changes.

The swagger schema gets updated like so:

```yaml
components:
  schemas:
    mdlClientSQL:
      type: object
      properties:
        ID:
          description: ID
          x-column: id
          type: integer
        CLIDNO:
          description: CLIDNO
          x-column: clidno
          type: string
          maxLength: 15
        CLNAME:
          description: CLNAME
          x-column: clname
          type: string
          maxLength: 30
        ID2:
          description: ID2
          x-column: id2
          type: integer
          x-reffld: ID
```

Figure – SWAGGER Schema Snippet

Some external fields also get added to the schema object like so:

```yaml
x-library: '*SQLSTM'
x-pfname: '*SQLSTM'
x-sqlSrcInfo:
  x-sqltype: IFS
  x-ifsPath: /mdrest4i/SQL14b6430.sql
x-sqlPath: /www/mdrstt12/specs/cons/mdrtut/SQLa31e390.sql
```

Figure – SWAGGER Schema Snippet (External Fields)

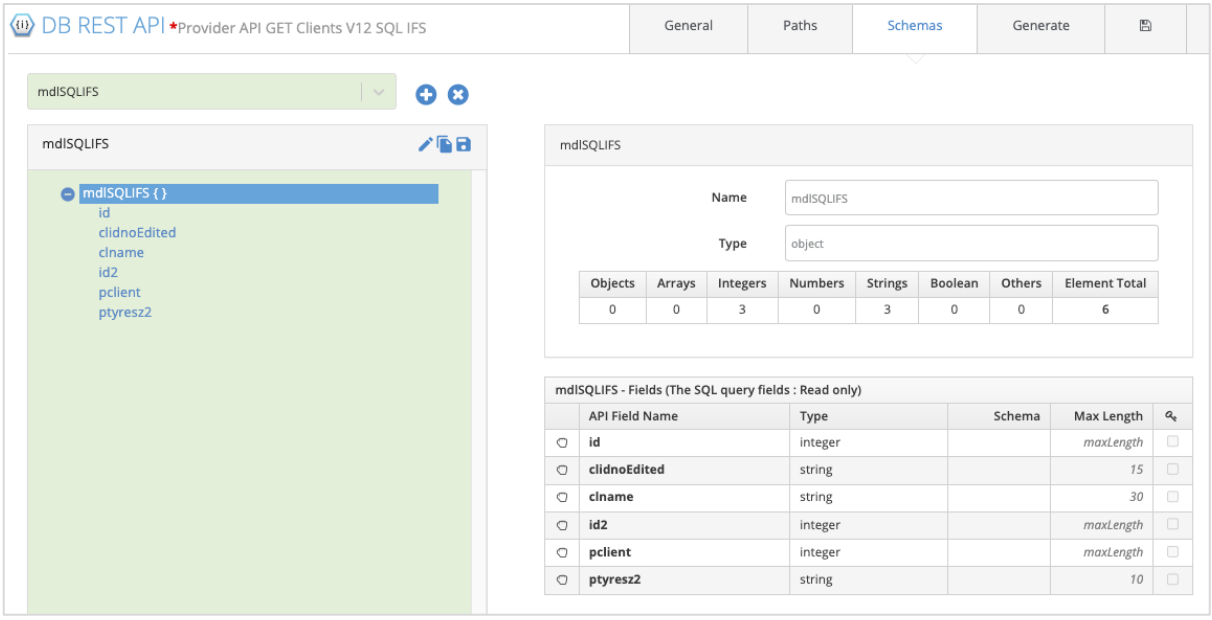


Figure - API Schema Tab (Model Level Details)

mdlSQLIFS - Fields (The SQL query fields : Read only)

	API Field Name	Type	Schema	Max Length	
	id	integer		*maxLength*	
	clidno	string		*15*	
	clname	string		*30*	
	id2	integer		*maxLength*	
	pclient	integer		*maxLength*	
	ptyresz2	string		*10*	

Figure - API Schema Tab (Field Level Details)

The left side brings up the schema in the tree form with all the fields listed in hierarchical order. Clicking on the model name or the elements of the tree brings up the details on the right hand side

The top right window displays the high level overview of the object selected.

The bottom right window brings up the children (if the selected element is an object or an array of objects) and their editable details.

4.5.7.5 Edit API Path Info

Now select the “Paths” tab and click on the Responses button.

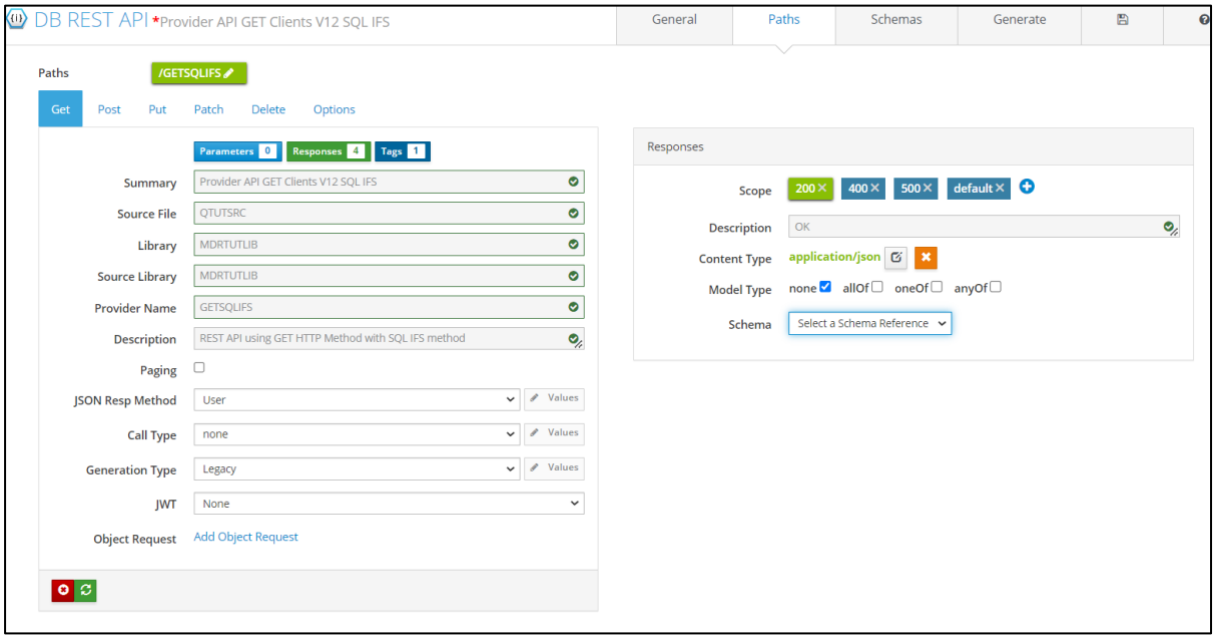
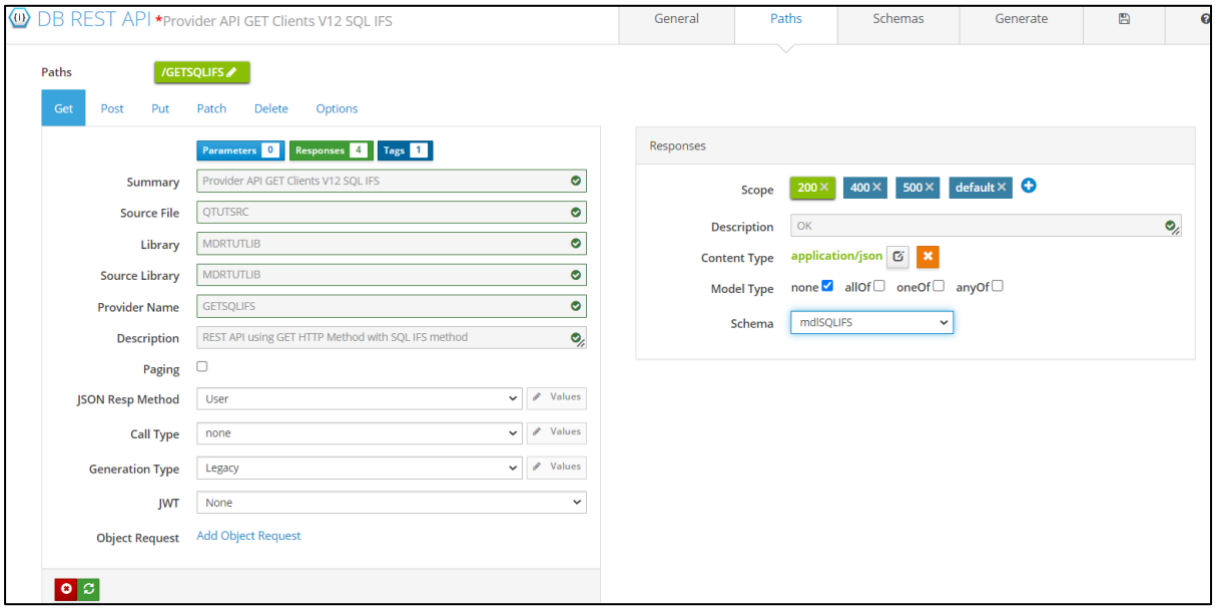


Figure - Paths Tab with Responses 1

In the above figure, the Paths are shown as buttons across the top on the left hand side. Below this are the HTTP methods. For each method the details of where to build the generated code and compile it can be updated. This automatically updates the underlying swagger specification in the x-lxrgen object described above.

Now select the Responses button on the left hand side of the paths tab.



By default, four HTTP responses are added to each API. Response 200 is the default status header field and value, which tells the requesting browser or consumer, that everything is OK. To generate RPGLE logic in the Provider that sends back JSON in the response, the content type and schema used to define the response must be added.

This will tell the RPGLE generator to add logic that sends back the correct status HTTP header field. The generator will also automatically build embedded SQL in the RPGLE (to get the data from the PF named in the schema earlier), plus generate logic to assign data from the embedded SQL result set, while building the JSON response body, which is then sent back to the requesting browser or consumer.

From the “Responses” section on the right hand side, select the content type, “application/json” from the drop down, and then select the “mdlClientSQL” schema created earlier from the Schema drop down.

In this example we are using a schema that has PF information included in it. This uses the PF FFD to create the necessary variables in the generated RPG logic along with building the SQL statement/s. If only work fields were in the schema, the RPGLE code generated would also contain these work field definitions.

4.5.7.6 Create API Service

Now select the “Generate” tab.

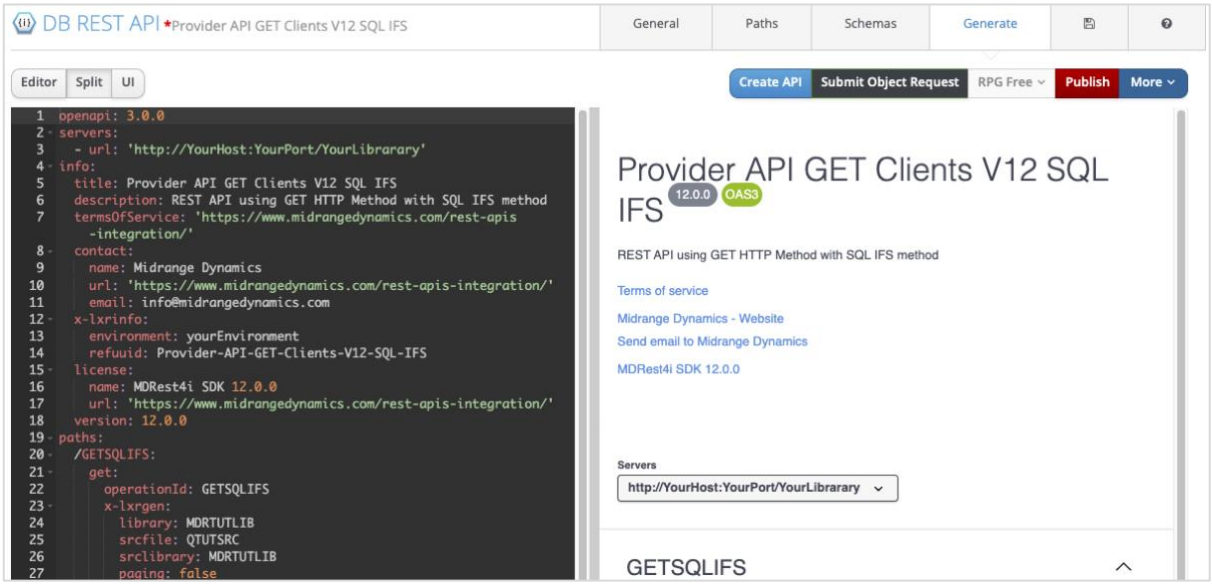


Figure - API Generate Tab

Note: The Swagger Specification is updated automatically in memory with each change made on the screen but this does **NOT** save those changes to the specifications database.

Save the specification using the Save button next to the “Generate” button. This updates the details added since the last save in the specifications database.

The “Generate” tab is a custom version of a full SWAGGER Editor. The left hand window enables editing of the SWAGGER specification directly, and the right-hand side shows the documentation and allows testing of the Provider once it is generated and completed.

Select the “Create Service” button.

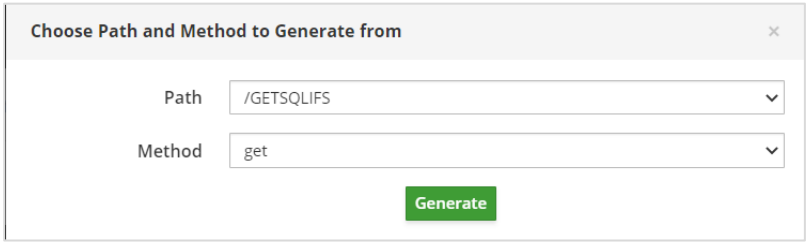


Figure - Create Service

Select the “get” method from the drop down and click the “Generate” button.

Upon completion of the generation process on the IBM i, the following figure will appear:

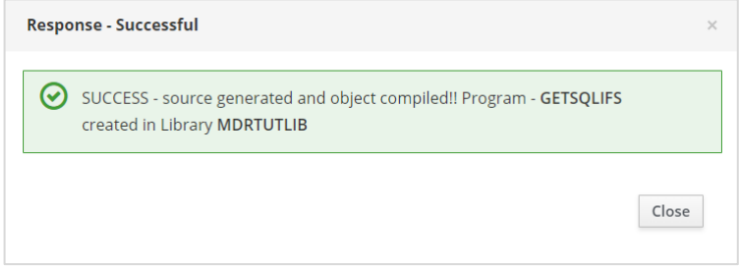


Figure - Response – Successful

4.5.7.7 Generated Code Description

The code generated is almost identical to the code described in the command-generated example CLIENTS above.

The only difference in this specific case is that the MDRGENXAPI Rest Service uses the API field names in the response. The commands use the column names from the table by default. To test this, edit one of the API field names (see Figure below for an example) in the schema tab and regenerate the service.

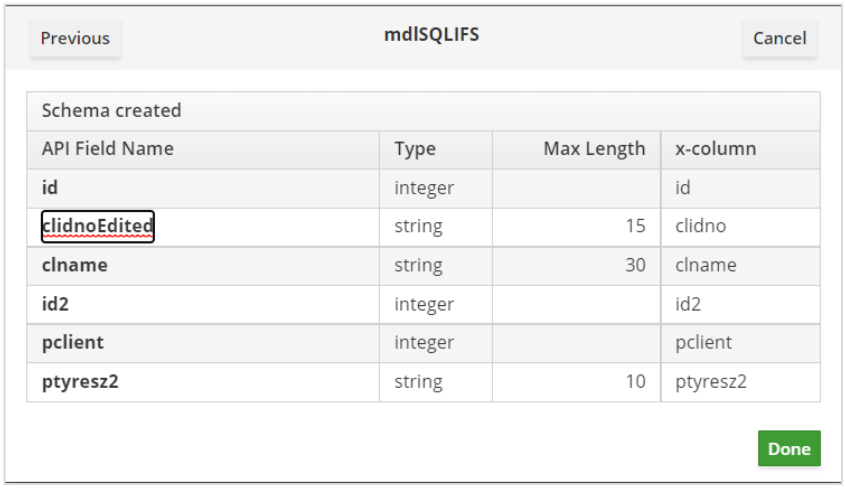


Figure: Edit API Field Name

4.5.7.8 Test the Generated Service

In the “Generate” tab edit the “- URL: 'http://yourhost:port/basePath' ” value (in the left hand SWAGGER window as per the above “API Generate Tab “ Figure) so that yourhost= your IBM i http server where MDRest4i server was setup, port is the port number used, and basePath is the name of the library where you generated the service into. Please ensure this service was generated into a library that was mapped in the http server in the first instance during installation. For example http://yourserver:yourport/yourlib where yourlib was added as a Library Name during the installation process.

Note: If you wish to have a separate HTTP instance for running your generated API’s please refer to the section “Creating the default MDRest4i HTTP Server Instance” in the “MDRest4i_12.0_Installation_Instructions” guide

Now on the right hand side of the “Generate” tab, scroll down until the server and “GET” button appears. Click on the “GET” Button seen in the figure to expand it.

Figure : GET Button for Testing

Scroll down to the “Server response” section.

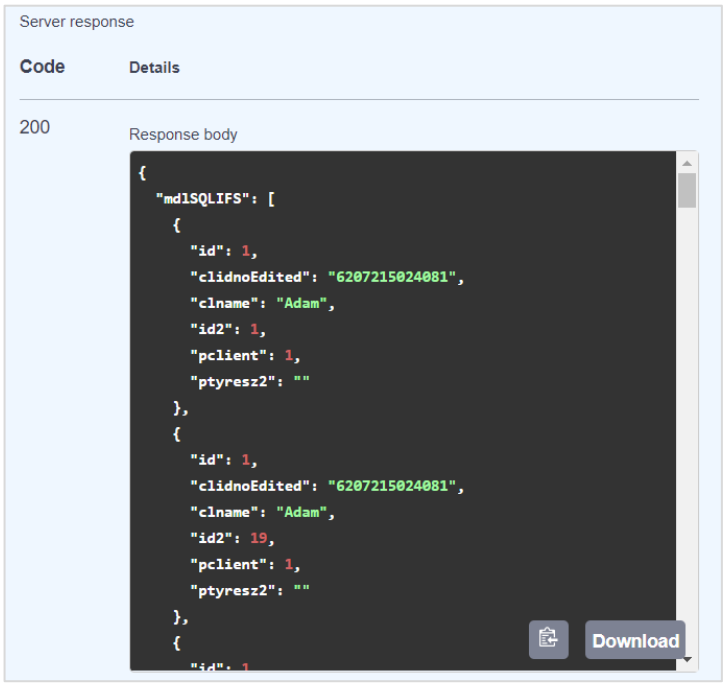


Figure - API Test Response

The response body can be seen as JSON in this window. This was created in the RPGLE by the beginObject, addChar, addDeci, addIntr, addBool, etc. functions in the z_procGet subroutine.

Note: Scroll up to the “Request URL” above the request body. Copy the code in the Request URL and paste into the address bar or a new window in your web browser. The same JSON as above will be returned.

TopTip: To see formatted JSON from response in your browser, there are a number of extensions or add-ons available for your browser. For example, Chrome has two useful ones: JSON Viewer for simple viewing and Advanced REST client (referred to as ARC) for a full-blown testing tool.

4.5.8 Provider-API-GET-Clients-V12-SQL-SF

4.5.8.1 Objective and Design

Demonstrate how to use MDRest4i SDK to create a REST API (Provider/service) that reads a DB2 PF/Table using a schema created from a source file and returns the record/s as JSON, allowing specific records need to be requested if necessary(not mandatory).

A REST Provider using the GET method will be created, that uses the PF fields as output. The key of the PF/Table may be supplied as a query string parameter.

4.5.8.2 Create New Provider

From the API Specifications list in the MDRest4i SDK Web GUI, select the "New" button on the right-hand side and select "Provider" as displayed in Figure 12 below:

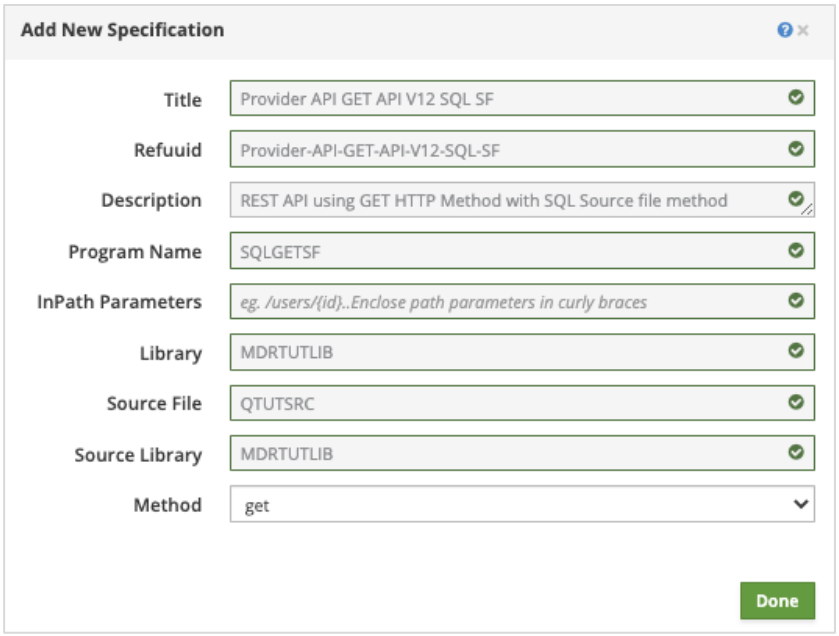


Figure – Add New Specification

The default values for the Library, Source Library and Source File can be set in the user profile using the "**Edit Profile**" option (refer to Figure 10(a) and 10(b)). The values would be automatically picked up from the user profile.

If the default values are not provided, you need to enter them manually every time while creating a spec.

Complete the fields in the popup that appears as per Figure 9 above. Select the "get" method in the drop down, before selecting "Done".

This has now copied the SWAGGER template for API's, added a few additional SWAGGER extension fields with the data provided by you in the popup, and created a new SWAGGER definition in JSON format in memory.

4.5.8.3 Adding General Info

The General tab seen in the figure below now appears.

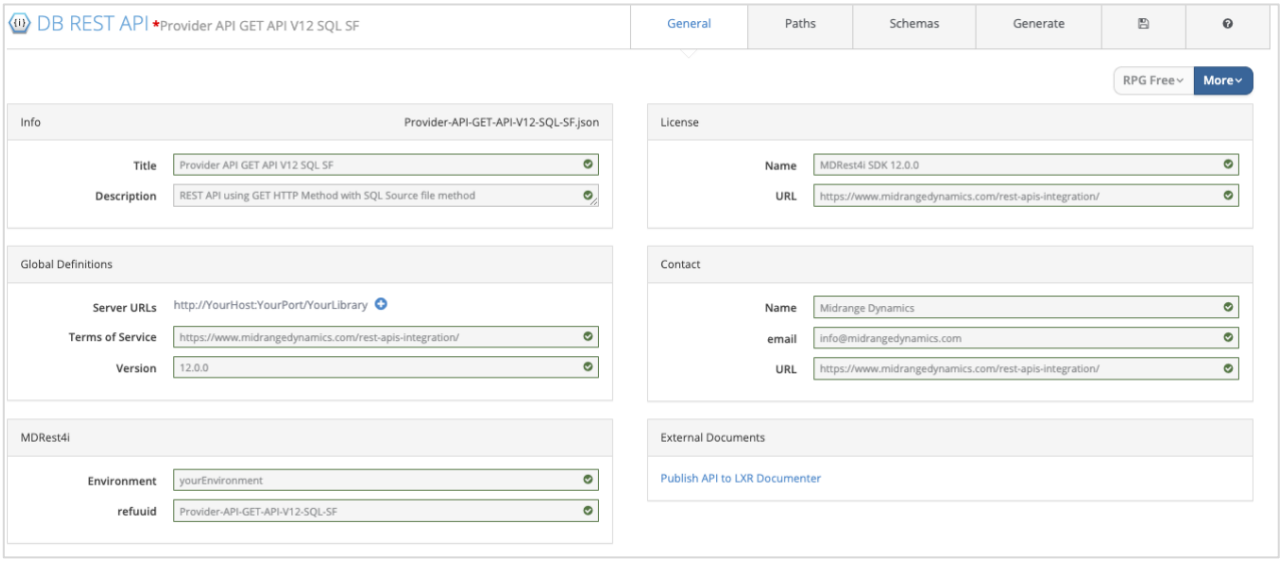


Figure - General Tab information

The only thing that needs updating is the server URL(s). Add the host name, Port number and library name used when setting up the MDRest4i HTTP server during installation.

This can again be set up as the default in the "**Edit Profile**"option (Figure 10(b)). If this is not available, it can be added later. The rest is general information and can be left as is for this tutorial.

Now select the "More" button and "Save" from the drop down menu.

4.5.8.4 Create a Schema

Now select the "Schemas" Tab. Click the  button to add a new schema .

The following popup appears.

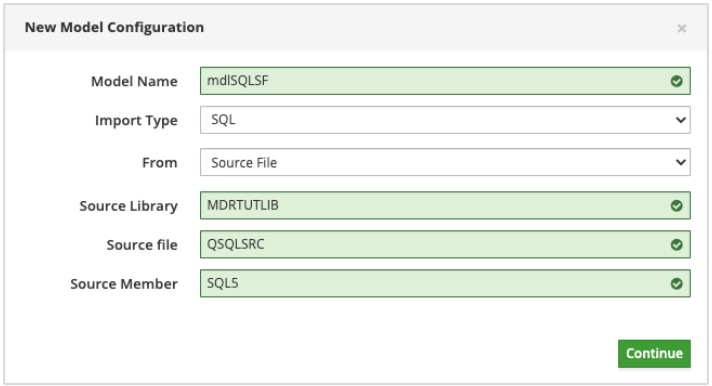


Figure - Add a Schema

The schema editor window appears where the imported SQL statement can be seen and edited.
The SQL statement edit does not change the original sql file. The user is issued a warning in case any changes are made in the SQL statement.

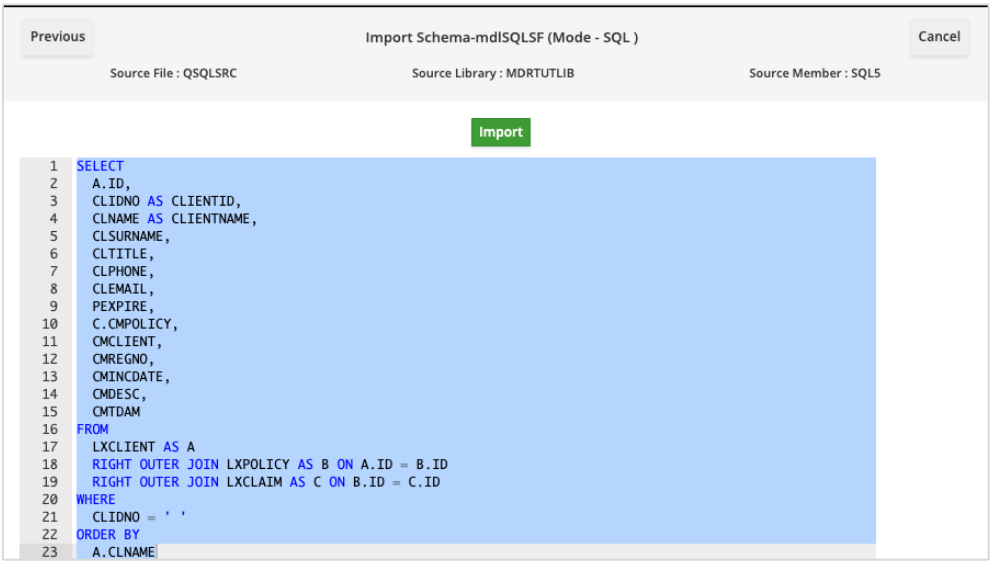
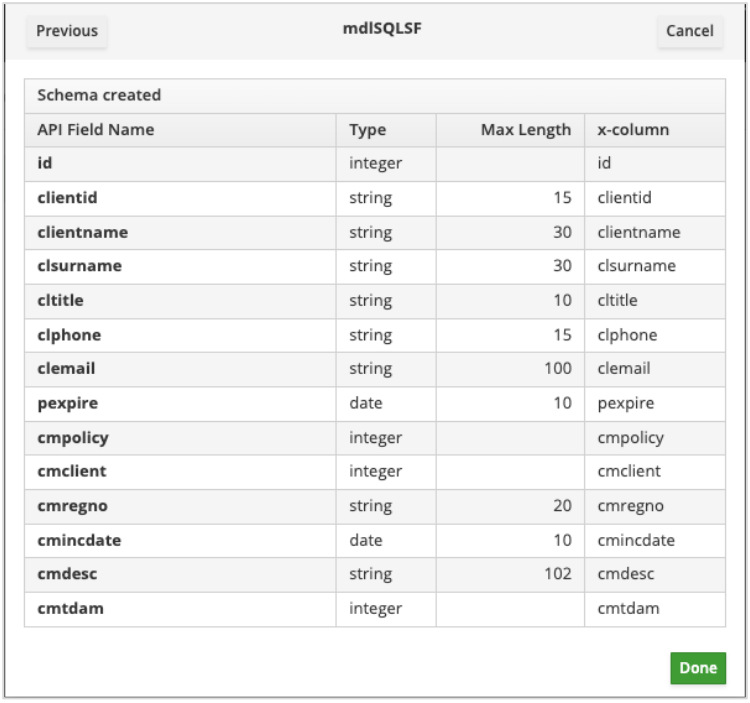


Figure – Schema Editor

Click the “Import” button. The list of fields comes up.



API Field Name	Type	Max Length	x-column
id	integer		id
clientid	string	15	clientid
clientname	string	30	clientname
clsurname	string	30	clsurname
cltitle	string	10	cltitle
clphone	string	15	clphone
clemail	string	100	clemail
pexpire	date	10	pexpire
cmpolicy	integer		cmpolicy
cmclient	integer		cmclient
cmregno	string	20	cmregno
cmincdate	date	10	cmincdate
cmdesc	string	102	cmdesc
cmtdam	integer		cmtdam

Figure – API Fields List

Let us, now edit the API field cltitle to clientTitle. Clicking on the field changes it to the edit mode.

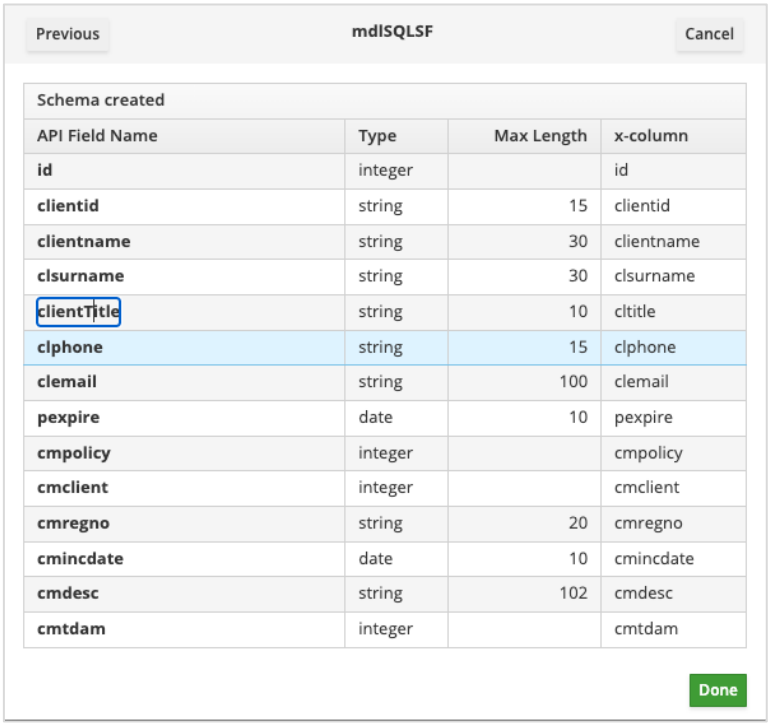


Figure – Edit API Field

Click “Done” to save your changes.

The swagger schema gets updated like so:

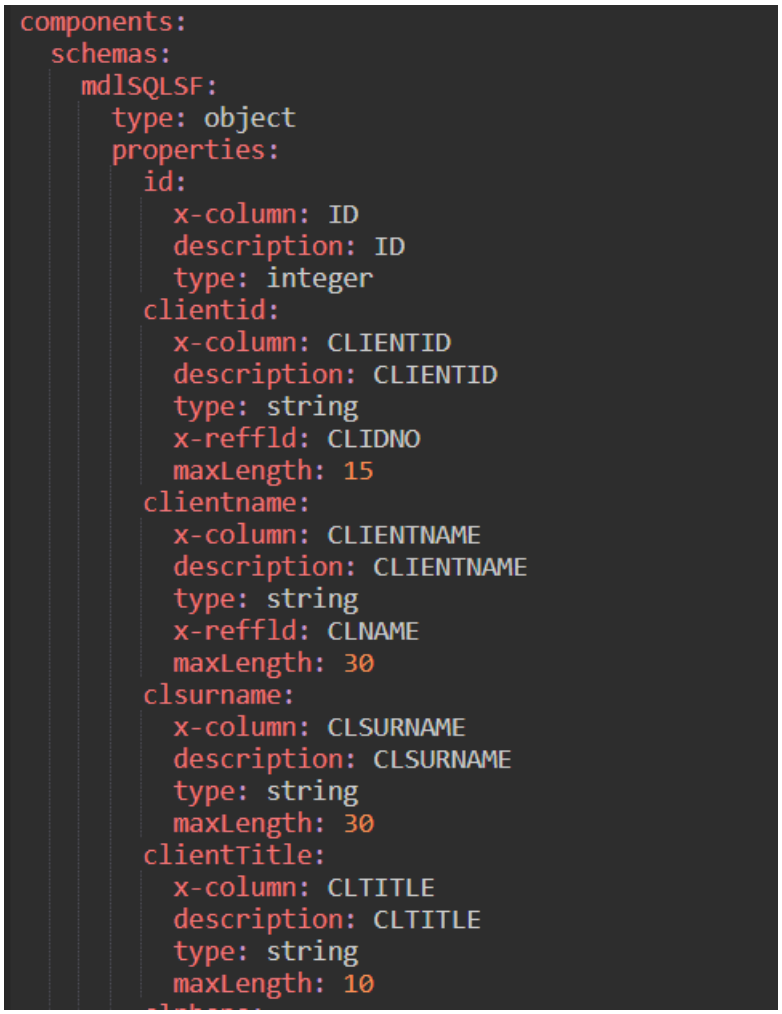


Figure – SWAGGER Schema Snippet

Some external fields also get added to the schema object.

Figure – SWAGGER Schema Snippet (External Fields)

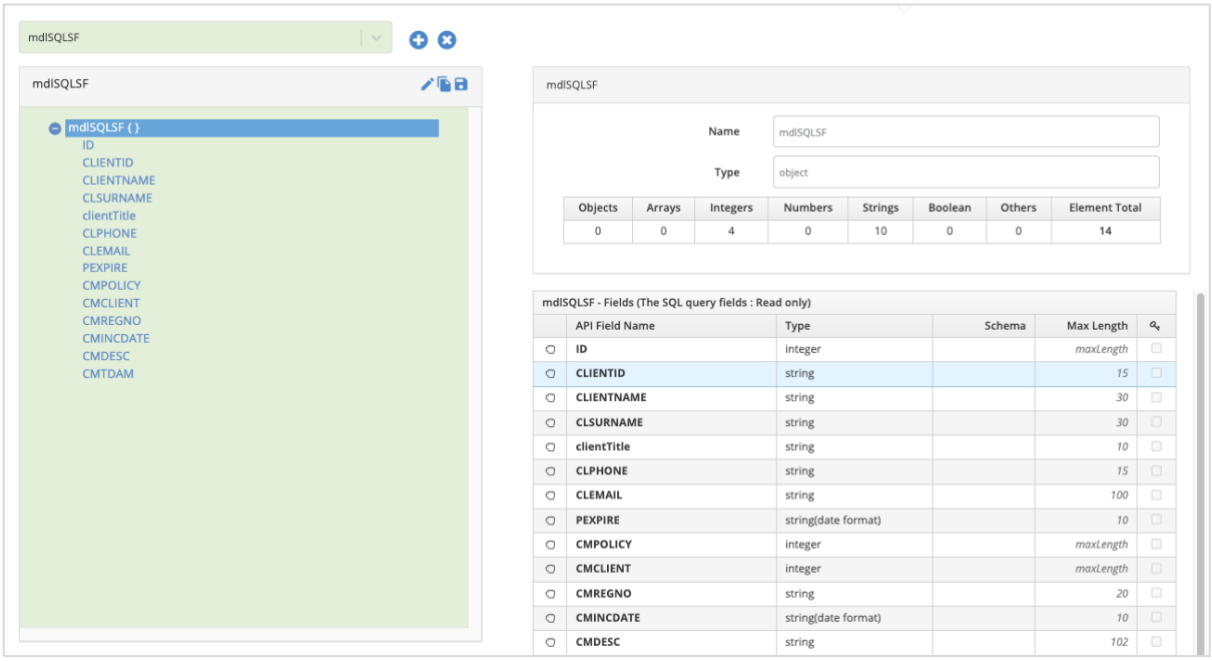


Figure - API Schema Tab (Model Level Details)

The left side brings up the schema in the tree form with all the fields listed in hierarchical order. Clicking on the model name or the elements of the tree brings up the details on the right hand side

The top right window displays the high level overview of the object selected.

The bottom right window brings up the children (if the selected element is an object or an array of objects) and their editable details.

4.5.8.5 Edit API Path Info

Now select the “Paths” tab and click on the Parameters button.

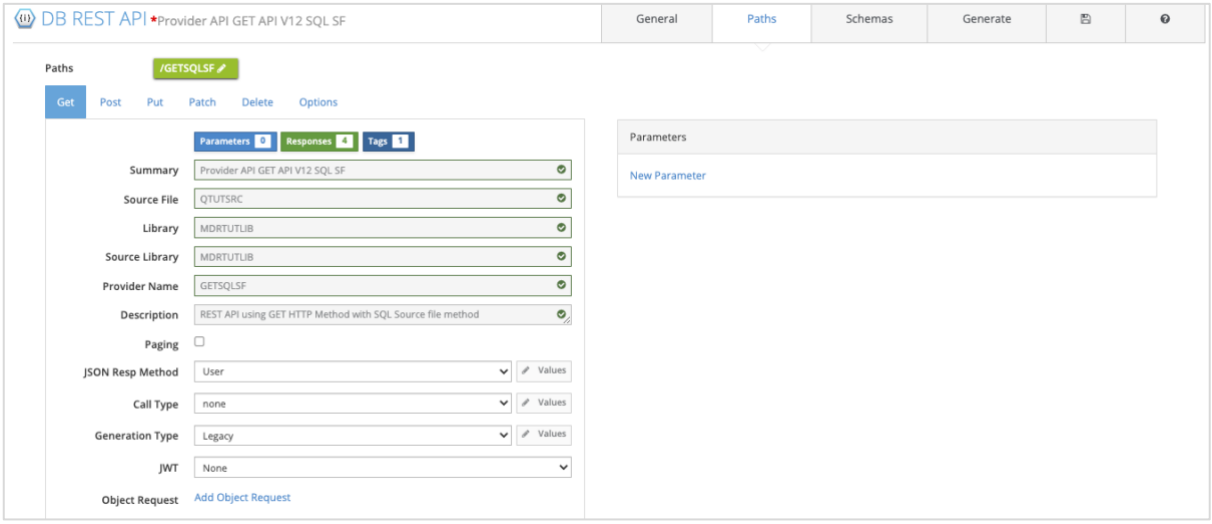


Figure - Paths Tab with Parameters

In the above figure, the Paths are shown as buttons across the top on the left hand side. Below this are the HTTP methods. For each method the details of where to build the generated code and compile it can be updated. This automatically updates the underlying swagger specification in the x-lxrgen object described above.

4.5.8.6 Edit API Parameters

Now select the “Parameters” button on the left hand side, then the “New Parameter” link on the right hand side.

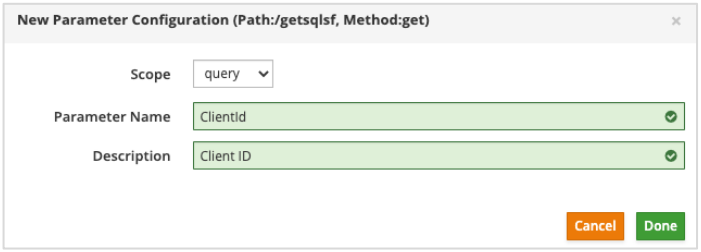


Figure - New Parameter Popup

Select the scope as “query” from the drop-down. This means that the Provider program will expect this parameter in the query string of the request

e.g. http://yourserver:yourport /yourlib/getclntdtl?**ClientId=3**

Add the parameter name as “ClientId” – this will be the name of the parameter supplied in the query string of the request URL. Add a description for the parameter (optional).

Click “Done”

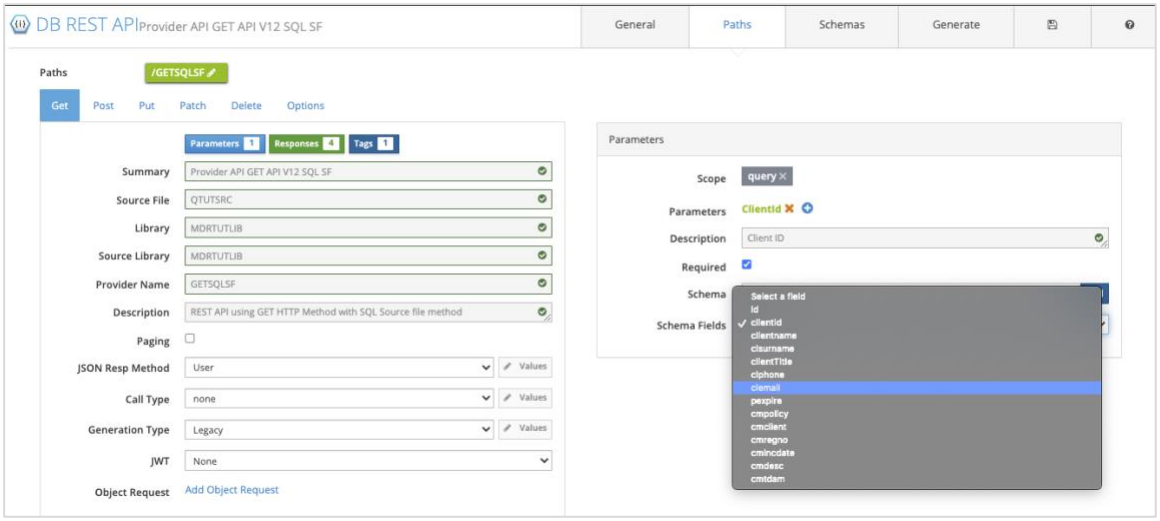


Figure - Parameters and Schema Reference

The RPGLE generator can assign values from the query string parameters to the where clause in the SQL it generates. To do so, it needs to know which schema field (and therefore associated PF Field/Column) to assign the incoming query string parameter value to. So, the parameter must be associated first.

Select a Schema and then a Schema Field from this schema in the drop downs provided as displayed in the following figure. The parameter object in the underlying SWAGGER specification is updated to Figure below.

```
    "parameters": [
        {
            "in": "query",
            "name": "ClientId",
            "description": "Client ID",
            "required": true,
            "schema": {
                "x-schemaFld": "CLIENTID",
```

```
                "x-schema": "#/components/schemas/mdlSQLSF"
            }
        }
    ]
```

Now select the “Paths” tab and click on the Responses button.

Now select the Responses button on the left hand side of the paths tab.

Select the Content Type as “application/json” and the schema “mdlSQLSF" from the dropdown.

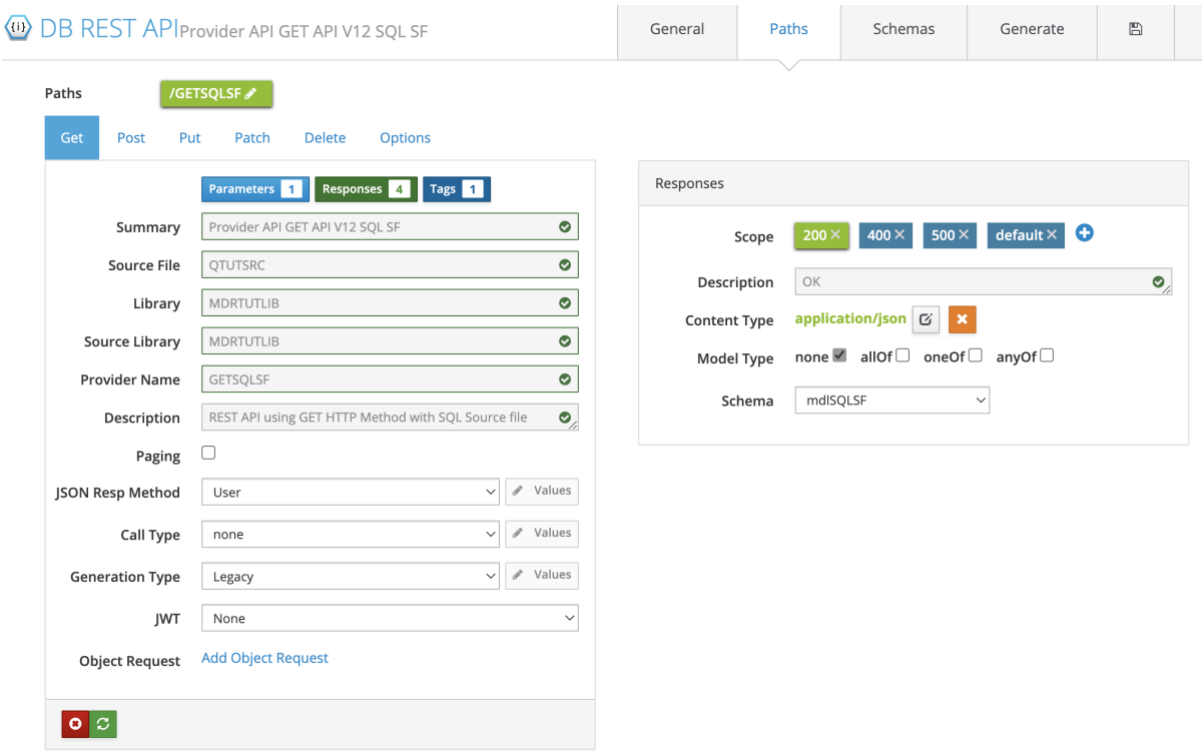


Figure - Edit API Response

In the above figure, the Paths are shown as buttons across the top on the left hand side. Below this are the HTTP methods. For each method the details of where to build the generated code and compile it can be updated. This automatically updates the underlying swagger specification in the x-lxrgen object described above.

By default, four HTTP responses are added to each API. Response 200 is the default status header field and value, which tells the requesting browser or consumer, that everything is OK. To generate RPGLE logic in the Provider that sends back JSON in the response, the content type and schema used to define the response must be added.

This will tell the RPGLE generator to add logic that sends back the correct status HTTP header field. The generator will also automatically build embedded SQL in the RPGLE (to get the data from the PF named in the schema earlier), plus generate logic to assign data from the embedded SQL result set, while building the JSON response body, which is then sent back to the requesting browser or consumer.

In this example we are using a schema that has PF information included in it. This uses the PF FFD to create the necessary variables in the generated RPG logic along with building the SQL statement/s. If only work fields were in the schema, the RPGLE code generated would also contain these work field definitions.

4.5.8.7 Create API Service

Now select the “Generate” tab.

Note: The Swagger Specification is updated automatically in memory with each change made on the screen but this does **NOT** save those changes to the specifications database.

From the display seen in below Figure, select the “more” button on the right-hand side and save the specification or click on the save. This updates the details added since the last save in the specifications database.

The “Generate” tab is a custom version of a full SWAGGER Editor. The left hand window enables editing of the SWAGGER specification directly, and the right-hand side shows the documentation and allows testing of the Provider once it is generated and completed.

Select the “Create API” button.

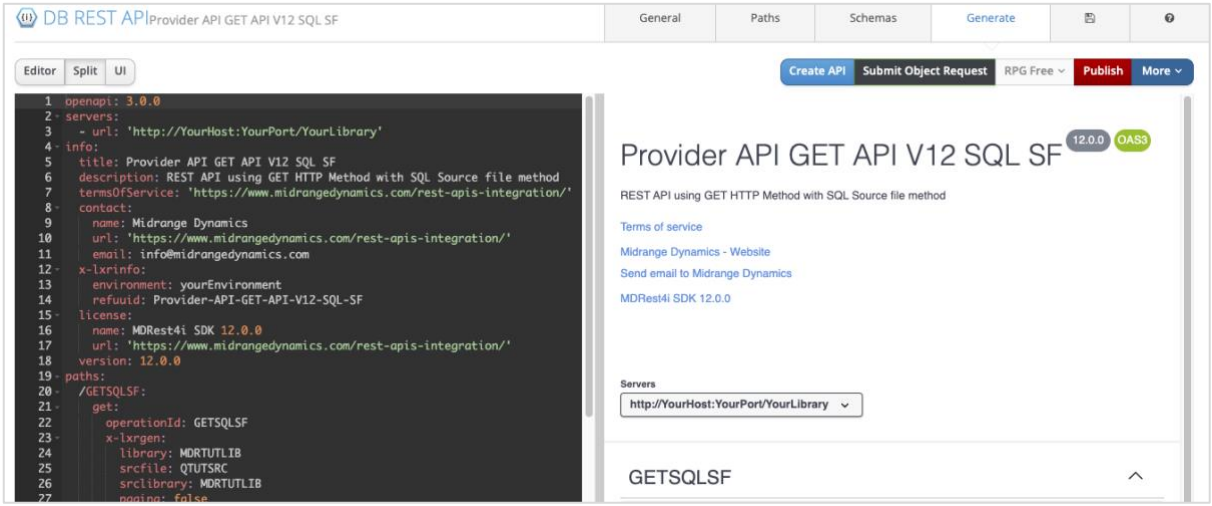


Figure - Generate Tab information

Figure - Create Service

Select the “get” method from the drop down and click the “Generate” button.

Upon completion of the generation process on the IBM i, the following popup appears:

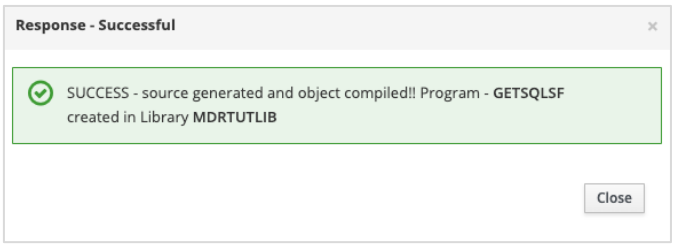


Figure - Response – Successful

4.5.8.8 Test the Generated Service

In the “Generate” tab edit the “- URL: 'http://yourhost:port/basePath’” value (in the left hand SWAGGER window) so that yourhost= your IBM i http server where MDRest4i server was setup, port is the port number used, and basePath is the name of the library where you generated the service into. Please ensure this service was generated into a library that was mapped in the http server in the first instance during installation. For example http://yourserver:yourport/yourlib where yourlib was added as a Library Name during the installation process.

Note: If you wish to have a separate HTTP instance for running your generated API’s please refer to the section “Creating the default MDRest4i HTTP Server Instance” in the “MDRest4i_12.0_Installation_Instructions” guide

Now on the right hand side of the “Generate” tab, scroll down until the server and “GET” button appears. Click on the “GET” Button seen in the figure to expand it.

Click on the Parameters and enter the value of the mandatory parameter “ClientId”

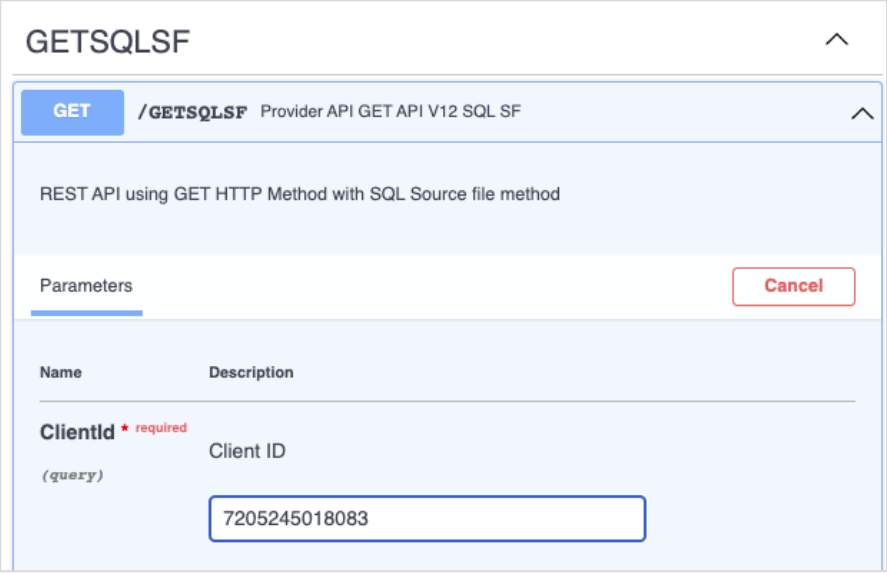


Figure : GET Button for Testing

Scroll down to the “Server response” section.

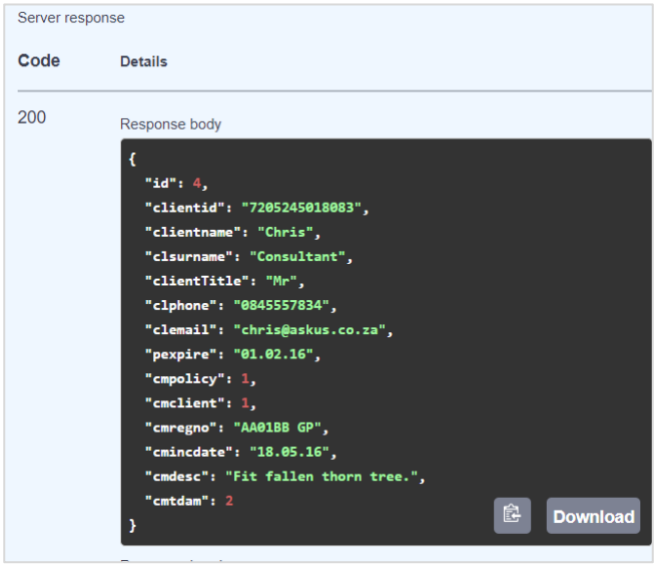


Figure - API Test Response

The response body can be seen as JSON in this window. This was created in the RPGLE by the beginObject, addChar, addDeci, addIntr, addBool, etc. functions in the z_procGet subroutine.

Note: Scroll up to the "Request URL" above the request body. Copy the code in the Request URL and paste into the address bar or a new window in your web browser. The same JSON as above will be returned.

TopTip: To see formatted JSON from response in your browser, there are a number of extensions or add-ons available for your browser. For example, Chrome has two useful ones: JSON Viewer for simple viewing and Advanced REST client (referred to as ARC) for a full-blown testing tool.

4.5.9 Consumer-Postman-import-V12.json

4.5.9.1 Using API/Provider SWAGGER to generate a REST Consumer in RPGLE

REST architectural style is Client-Server. A RESTful consumer program reverses the logic sequence we used in the API/Provider code in the previous sections. So the output of the requesting consumer becomes the input of the Provider and therefore the requests, payloads and responses must be synchronized. MDRest4i SDK exploits this fact to use a SWAGGER API/Provider specification to generate consumer code that can call the API and process the response.

4.5.9.2 Export the postman collection

Inside the postman, select the collection you want to export.

Let us try to import the collection we use to save the tutorial examples - **MDRest4i-V12.0-Tutorial-Public**

Click on the ellipses and select the option "Export" from the menu as shown in the figure below.

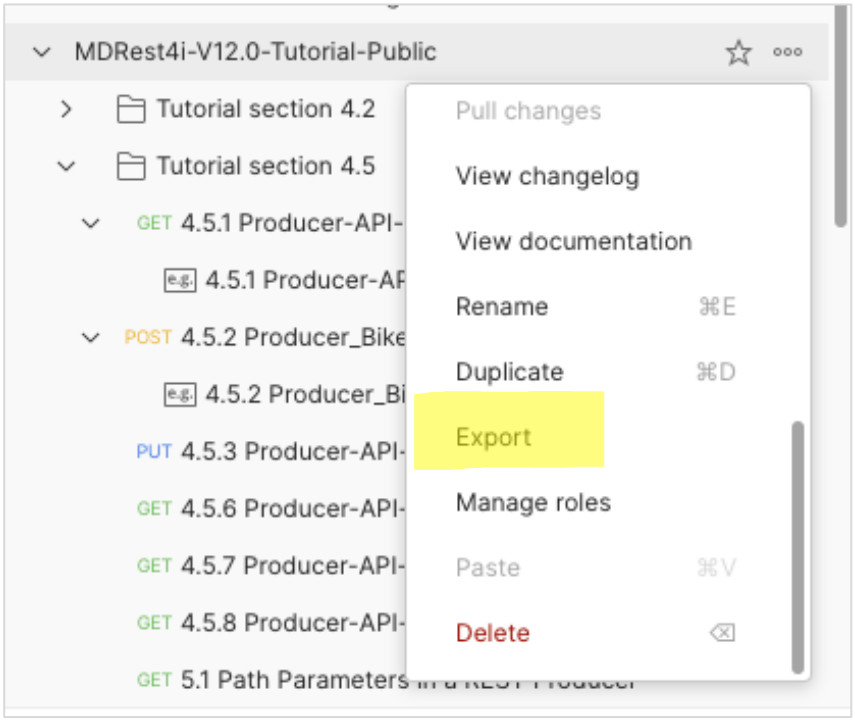


Figure – Export postman collection

Click the **Export** button in the next dialog box and the collection gets exported in json format.

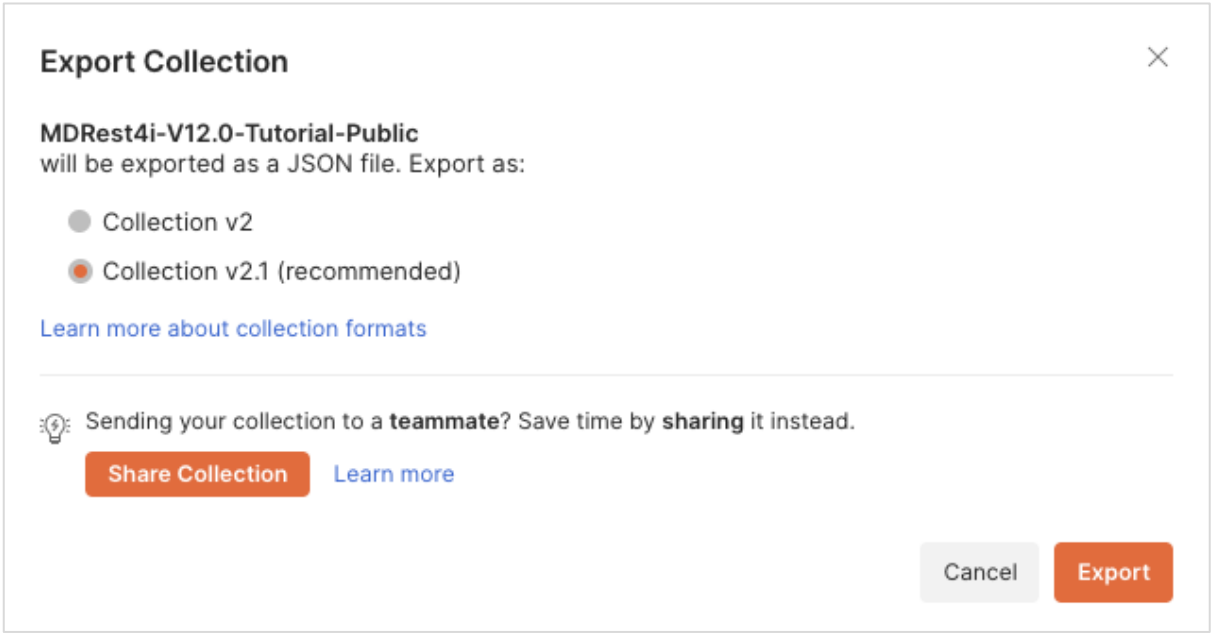
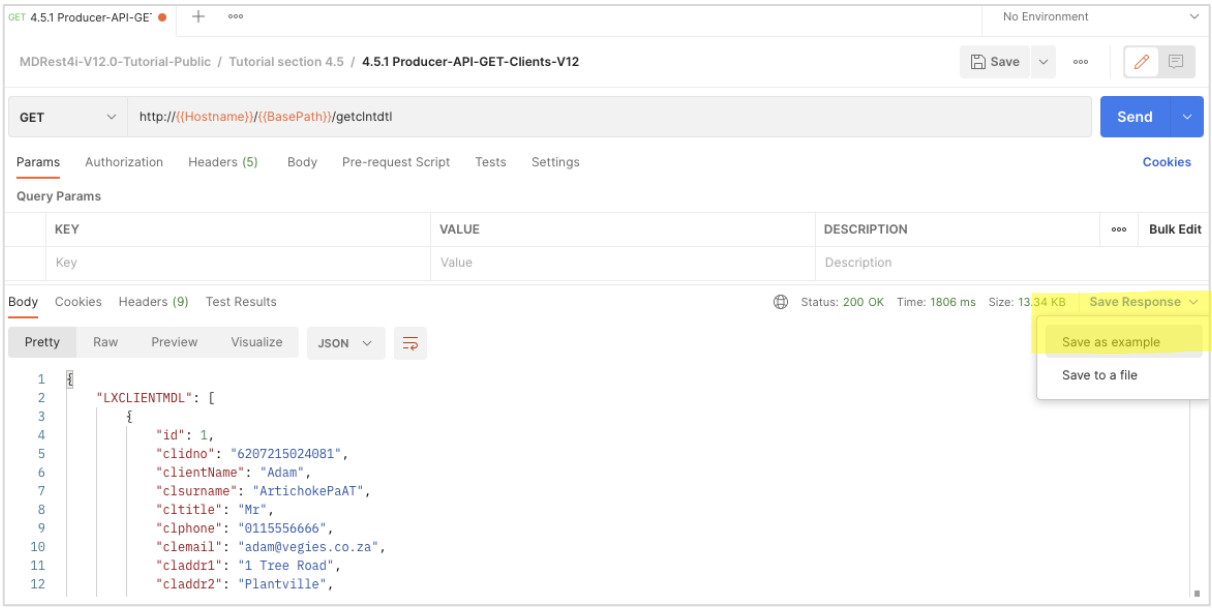


Figure: Export Dialog box

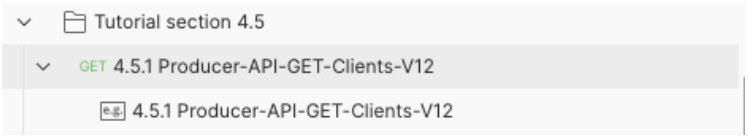
Note: If you want to include the response object in the export, please make sure you save run the APIs and save the response as example.

Please refer to the example below to see how to save the response as example.

On the right hand side of the response section, choose “Save as example” from the “Save response” dropdown.



This will add the example underneath your API



4.5.9.3 Import the exported postman collection

Select **"New>Import Consumer From Postman Collection"** on the right-hand side. Import the file "MDRest4i-V11.0-Tutorial-Public.postman_collection.json" exported from your postman collection.

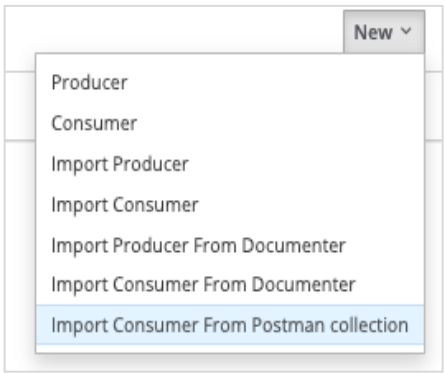


Figure: Import Consumer from Postman Collection

The list of APIs in the collection comes up. Select the API you wish to import.

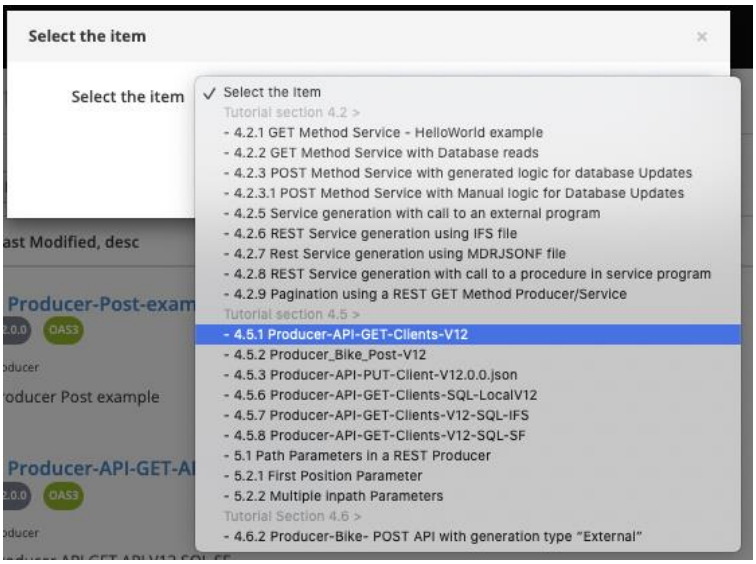


Figure – Select

The following popup appears

Figure – New Consumer From Upload

Change the Refuuid by removing the invalid characters and by giving it a name of your choice.

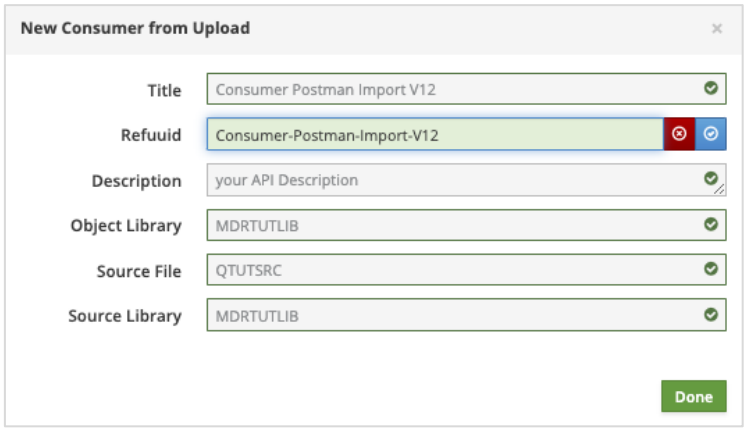


Figure – New Consumer From Upload (Edited)

Select "Done" to close the wizard popup window. Click the "Save" button to save the specification in your list of specifications.

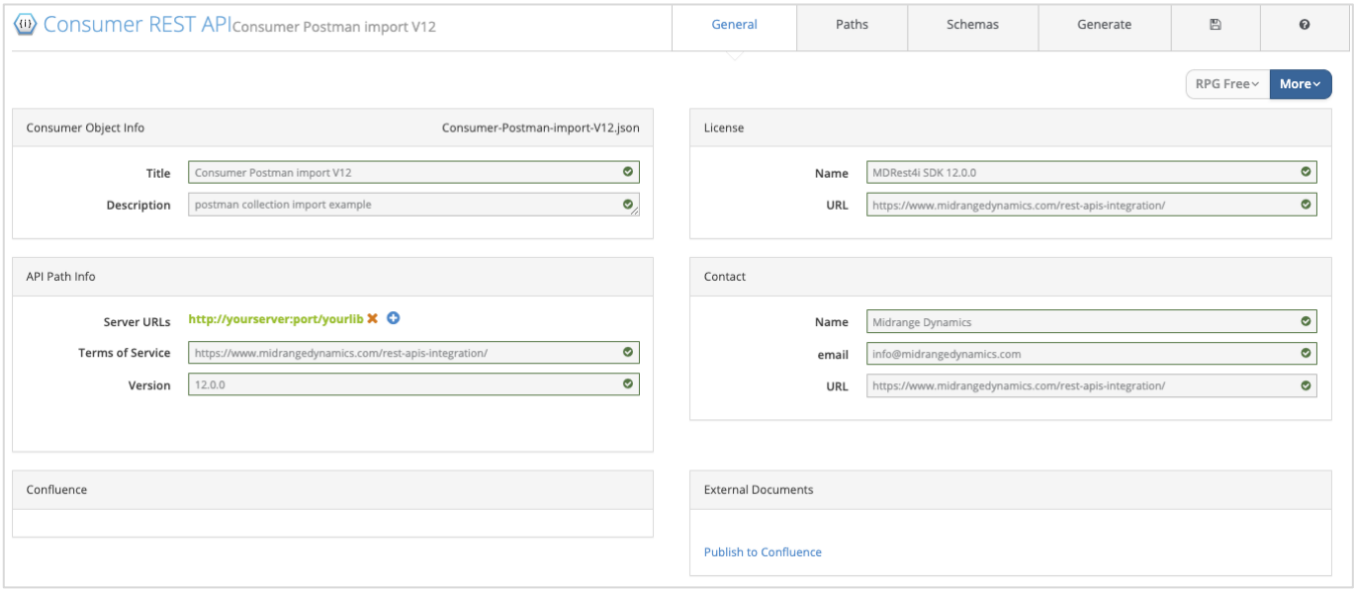


Figure – General tab API

Navigate to the “Paths” tab. Change the consumer name to “postclnta” . This would be the name of the source member and the object name.

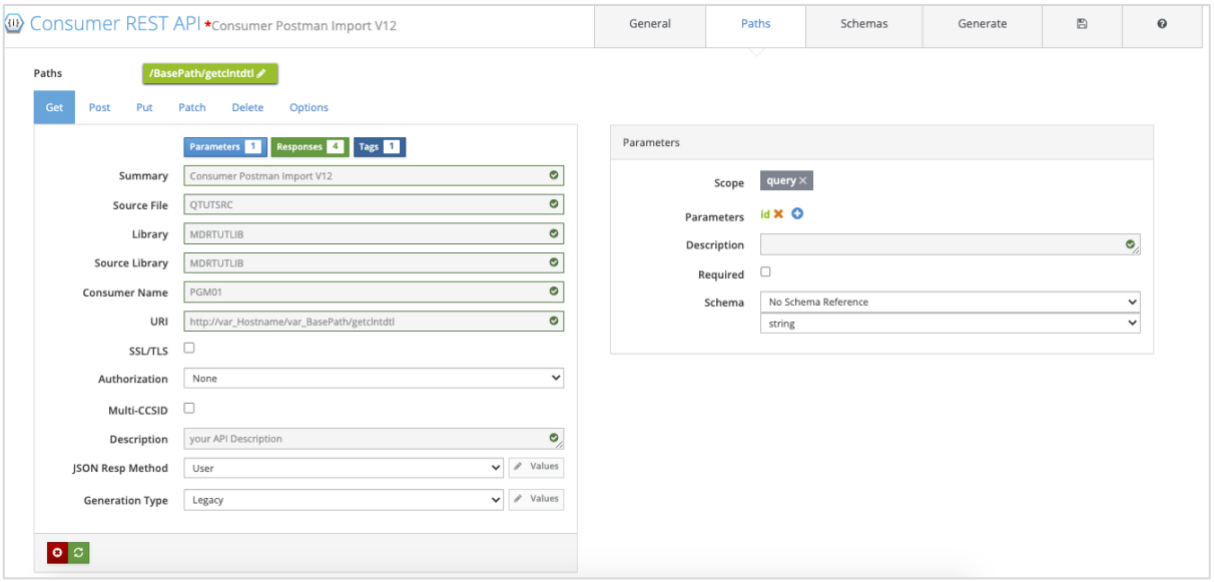


Figure – Paths tab API

4.5.9.4 Create the Consumer Program

Select the “Generate” tab and then the “Create Consumer” button. The consumer RPGLE code is generated and compiled, and a success message is displayed.

4.5.10 Provider-POST-example.json (Saving and Importing Schemas)

4.5.10.1 Objective and Design

Demonstrate how to use MDRest4i SDK to create a REST API (Provider/service) that imports a schema saved on the server and returns the record/s as JSON

A REST Provider using the POST method will be created.

4.5.10.2 Create New Provider

Click New -> Provider

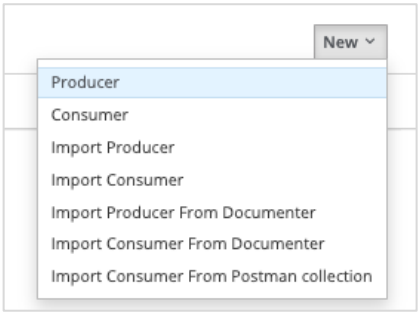


Figure – New Provider

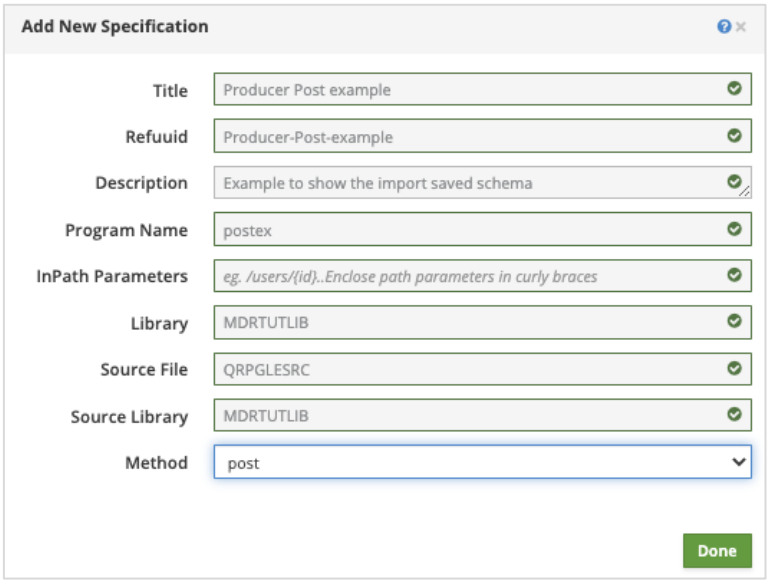


Figure – New Provider

Select “Done” to close the wizard popup window. Select the “More>save” button/option on the right-hand side, in the “General” tab to save the specification in your list of specifications.

4.5.10.3 Adding General Info

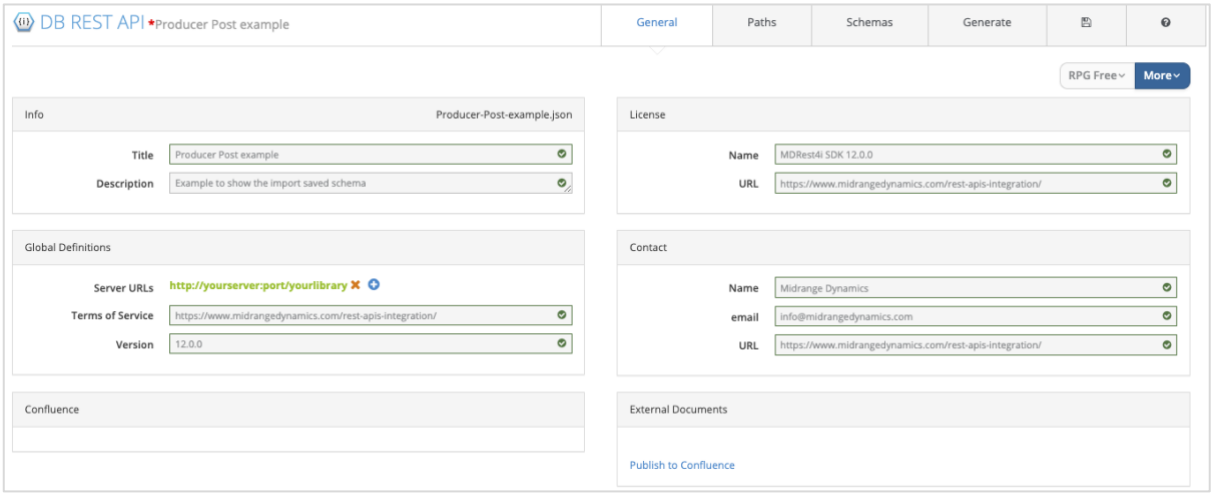


Figure – General Tab

4.5.10.4 Create a Schema

Now select the “Schemas” Tab.

The following popup appears.

Click the schema tab. Click the  button to add a new schema .

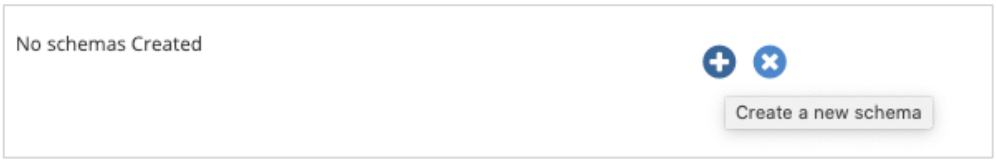


Figure – Create new Schema

Figure – Create new json schema

Click “Continue”. Enter the json in the schema edior and click “Import”.

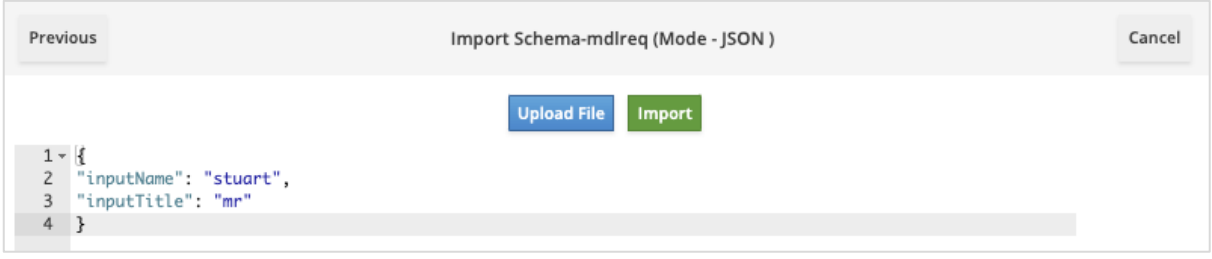


Figure – Schema Editor

The schema gets created!

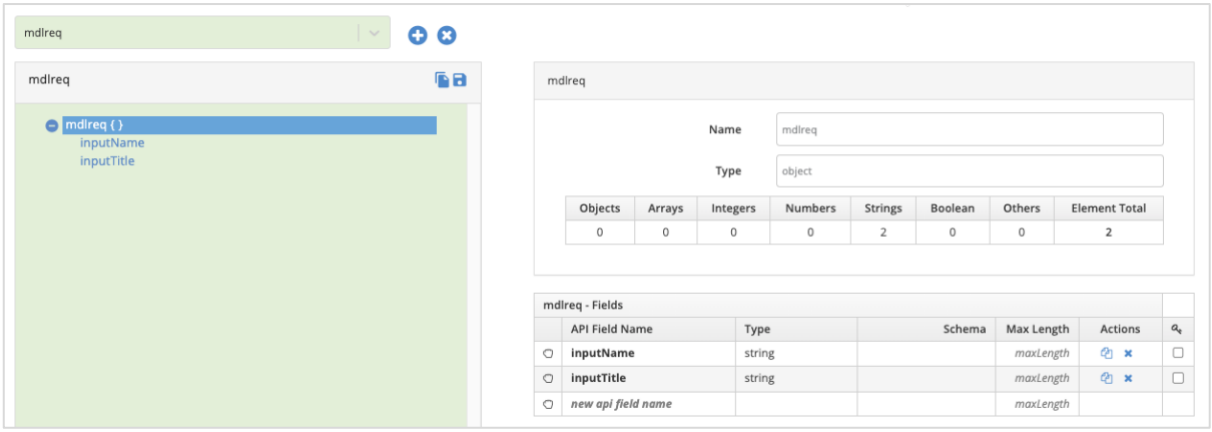


Figure – The created Schema

4.5.10.5 Save the Schema

If you wish to save this schema on the server for future imports, you may do so using the button. The following popup appears which lets you enter the name and the description for the future reference while importing.

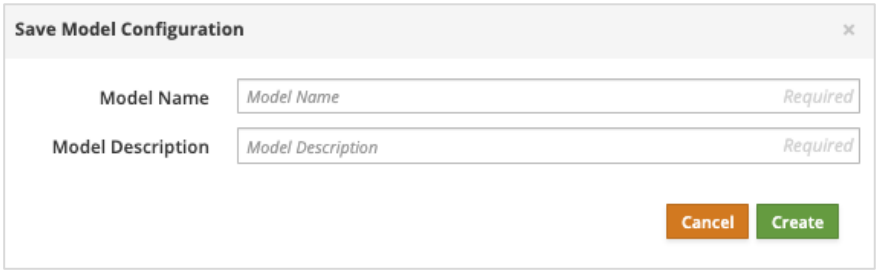


Figure – Save model configuration

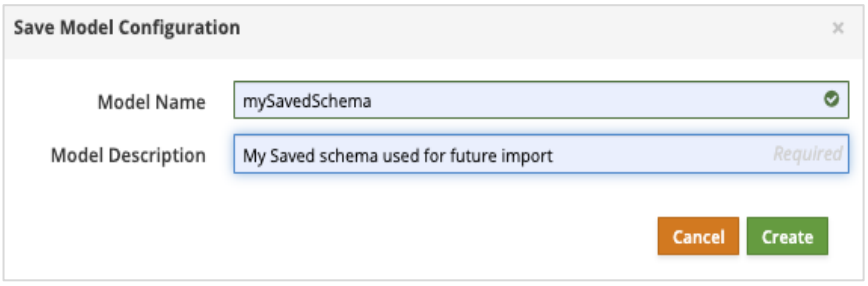
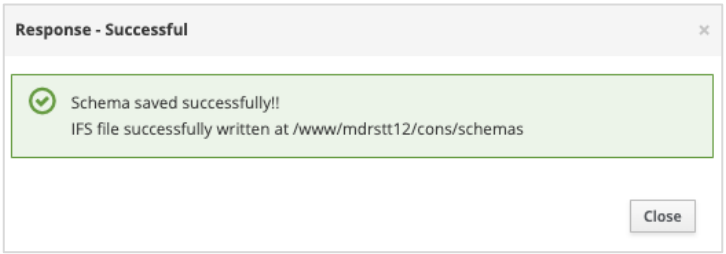


Figure – Save model configuration (filled up)

Click on the Create button to save the schema.



Now, let us create another schema importing the schema we have created previously.

Click the button to add a new schema. We give the model name as mdlResp and the import Type is selected as “Saved Schema”.

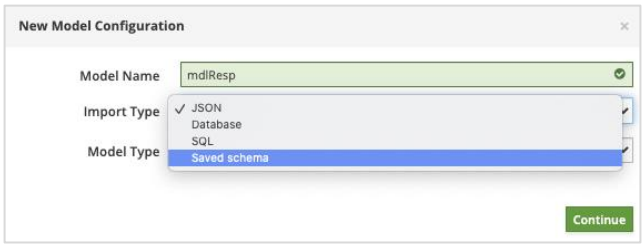
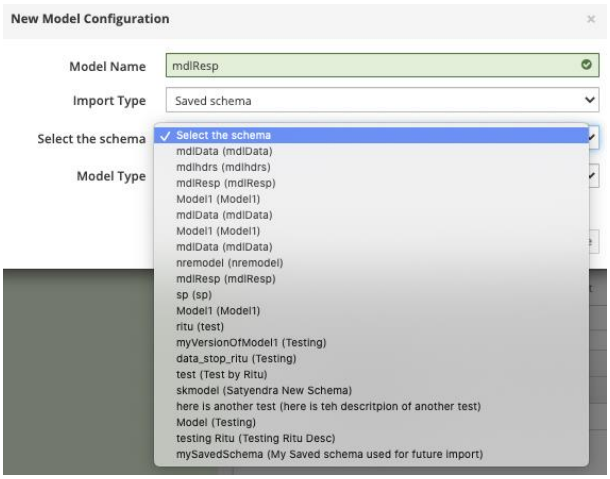


Figure : Create new Schema using import saved schema

On selecting the Import Type as Saved Schema, a dropdown list of saved schemas and their descriptions shows up and the user can import the one he selects.



We choose the one we created earlier with the name mySavedSchema and the description- My Saved Schema used for future import.

Click Done

The schema gets imported!

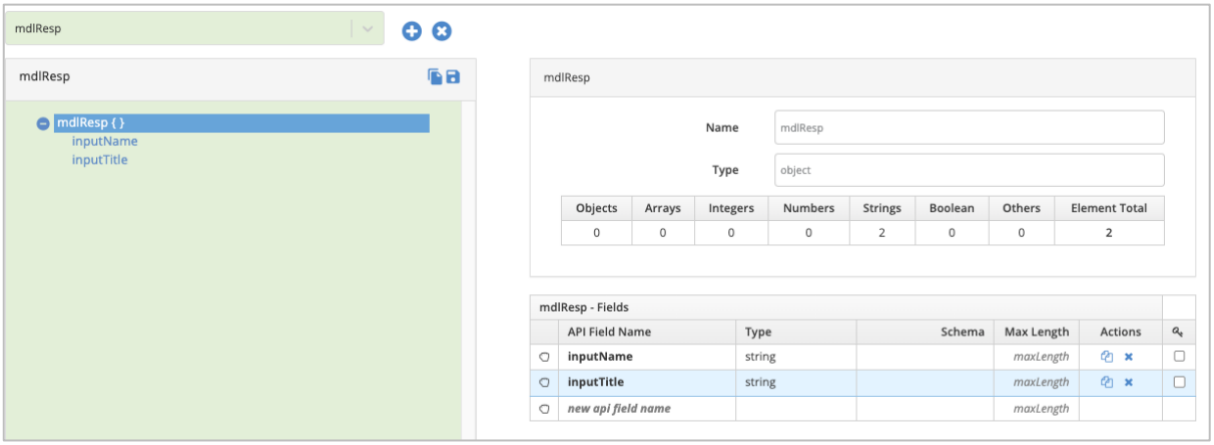


Figure – The imported schema

Let us now edit the field names by changing the inputName to “respName” and inputTitle to “respTitle”

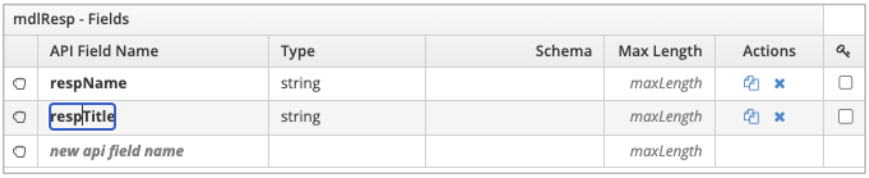


Figure : Edit Field Names

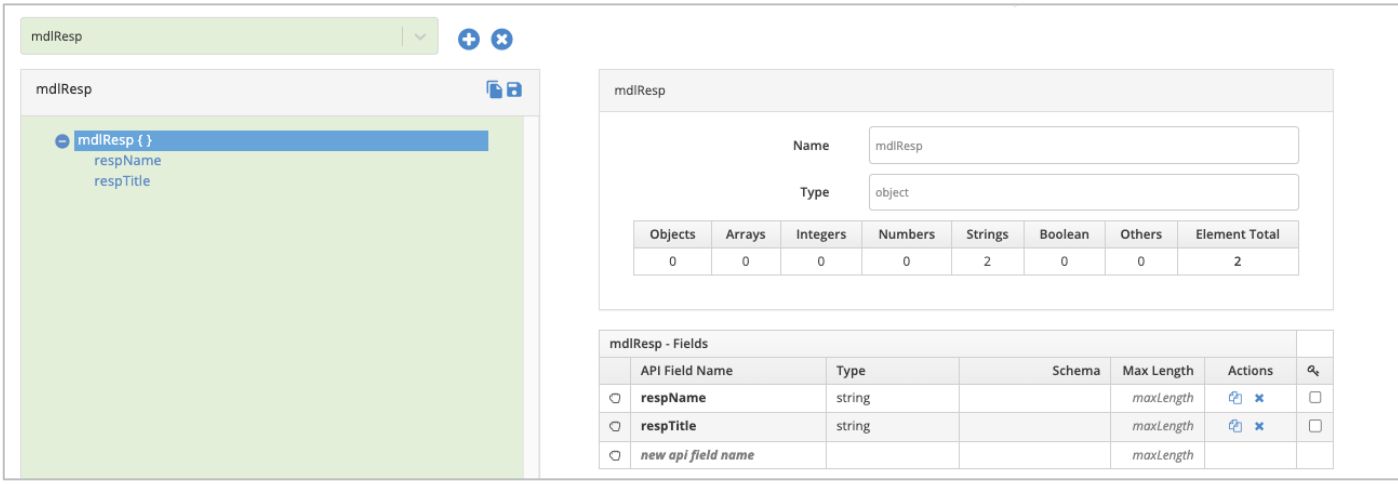
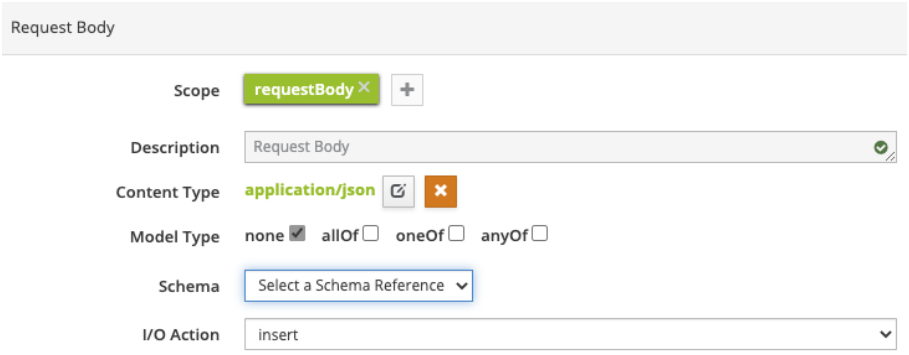


Figure : Edited Field Names

Click the save button to save your changes.

4.5.10.6 Edit API Path Info

Click on the "Paths" tab and select the Body 1 button



4.5.10.6.1 Creating Request Body

The request body gets created by default in case of post, put and patch methods when a new spec is created or a new method is enabled.

However, to create a request body (if accidentally deleted), click on the + button

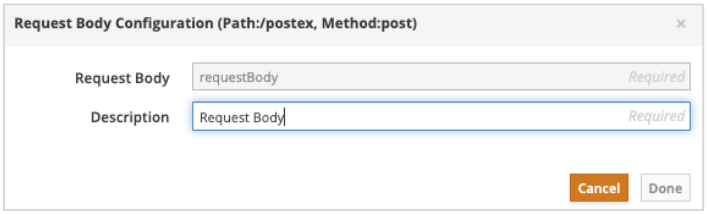


Figure : Create Request Body

Click Done after filling in the description.

Select application/json as the Content Type from the drop downand mdlReq as the schema.

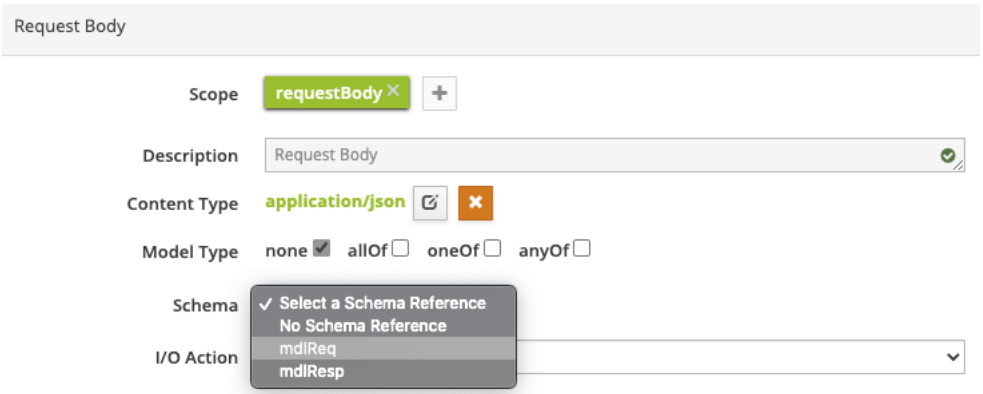


Figure : Request Body

4.5.10.6.2 Edit the Response

Select the scope 200 tab

Figure : Responses

Select “application/json” as the Content Type and select the schema mdlResp from the dropdown

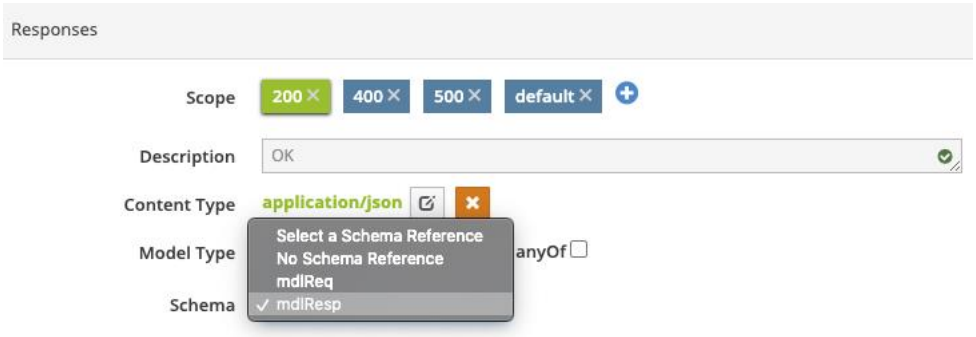


Figure : Response:200

4.5.10.7 Create API Service

Now select the "Generate" tab. Create the API like in other examples.

5 REST "Path Parameter" Handling

Where the parameter value is actually part of the API's URL. This does not include the host or base path of the API. For example, in /MDRTUTLIB/{bikeno}/bikepath2, the path parameter is "bikeno".

While the RPG functions used for handling in-path parameters can be added manually to any API. MDRest4i SDK can create this RPG logic automatically during API generation.

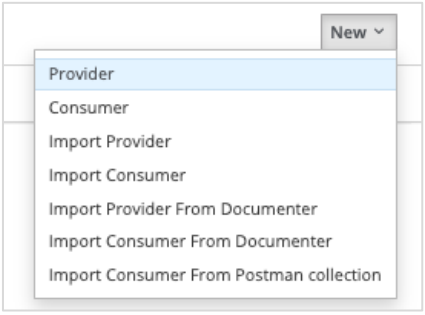
Below are some examples of in path parameters in different positions of the request URL.

5.1 Handling Path Parameters in a REST Provider

In this Provider example we will use the SDK to create the SWAGGER definition, and generate the RPG code from the SWAGGER.

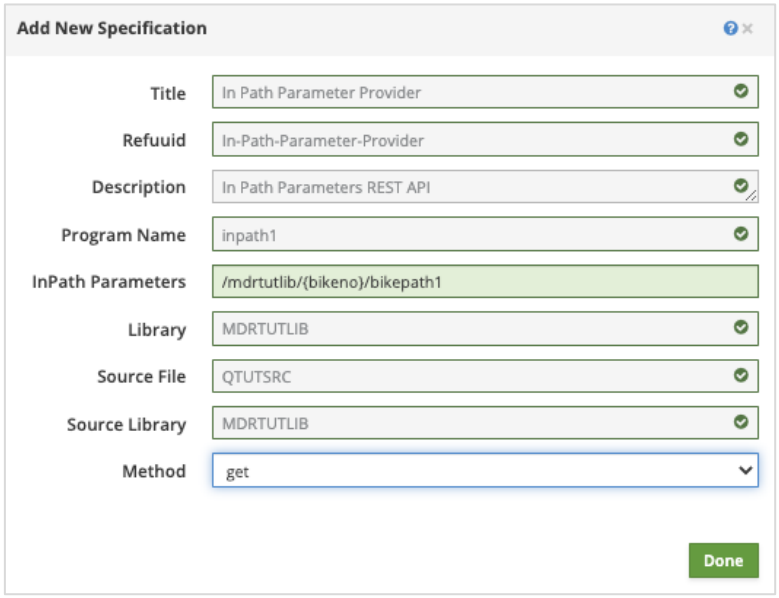
5.1.1 SWAGGER Specifications for in path parameters

- From the Specifications list in the SDK UI select New>Provider



- Add the Title, Description & Program name. Paste the following value into the Path field:

```
/MDRTUTLIB/{bikeno}/bikepath2
```



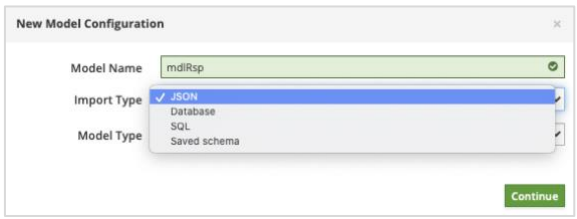
- Select “Done”
- Navigate to the “Generate” tab.
- Select “More” and “Save”
- In the left hand SWAGGER code window, scroll down to the “parameters” section

```
  - inpath1
parameters:
  - in: path
    name: bikeno
    description: bikeno description
    required: true
    schema:
      type: string
```

Using the value added to the “path” field in the new Provider popup, MDRESt4i SDK has added a parameter “bike” specifying the “path” position.

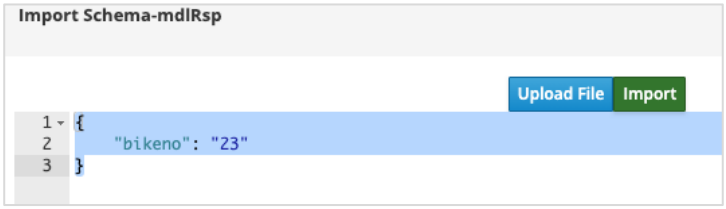
Note that in-path parameters are ALWAYS mandatory

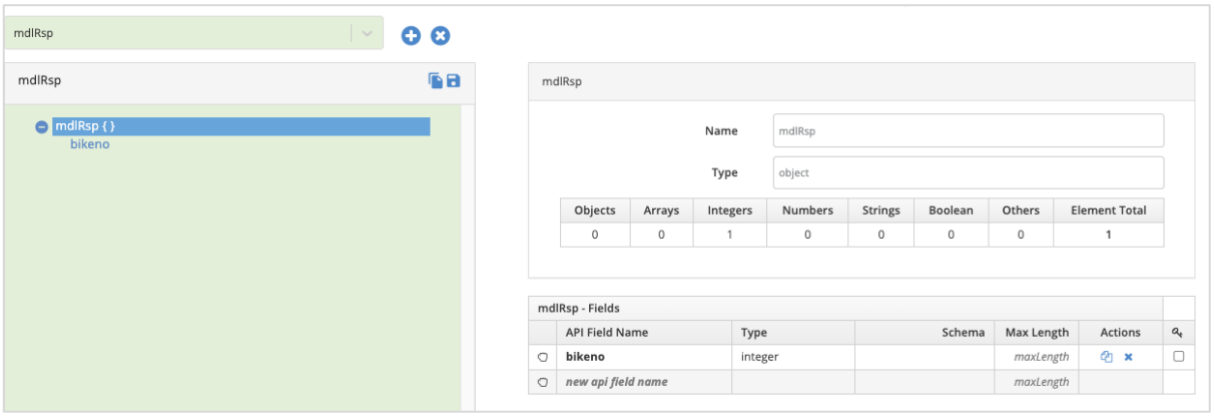
- Now select the “Schemas” tab
- Add a new schema and select “Import JSON”



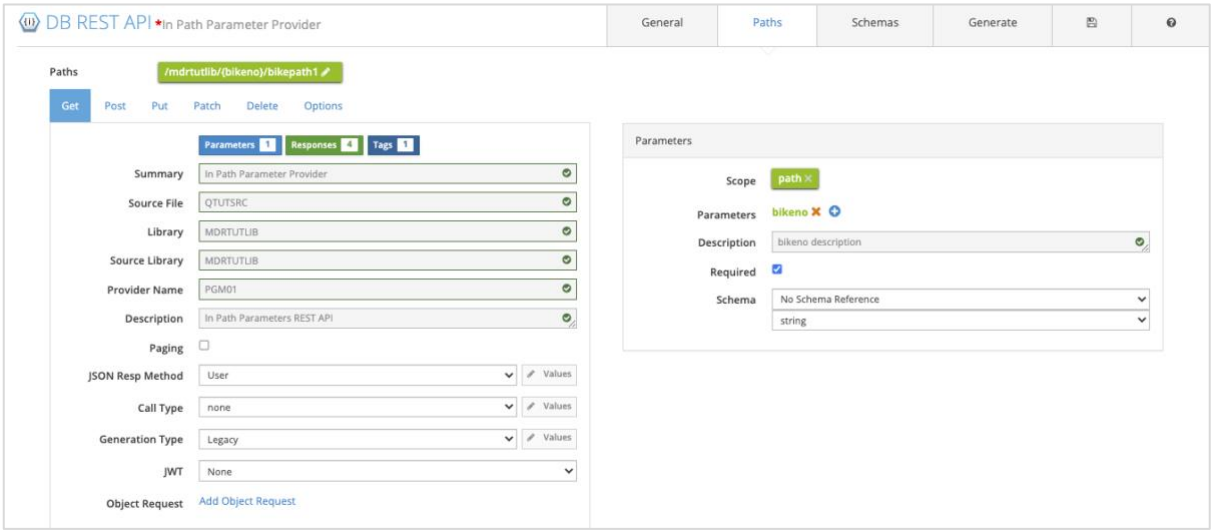
Paste in the following JSON and select “Import”

```
{
    "bikeno": "23"
}
```

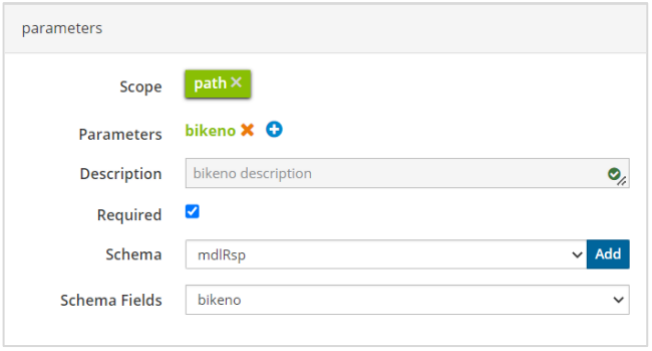




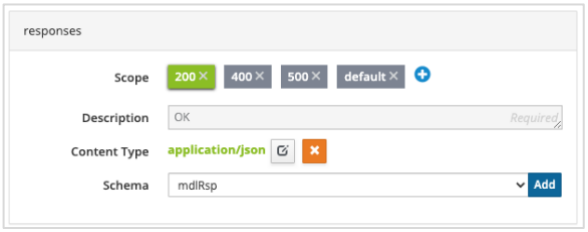
- From the schema screen change the “bikeno” field to type “integer”
- Select the “Paths” tab and then the “Parameters” button for the Get method on the bottom left



- On the right-hand side, select the “Schema” drop-down and select the schema created above.
- The Schema field drop down then appears. Select the “bikeno” field from this list.



- Select the “Responses” button in the Paths tab, and set the 200 response content type to Application/json, and select the schema created above.



- Select the “Generate” tab, and in the SWAGGER window on the left, scroll down to the “parameters” section.

```
    parameters:
      - in: path
        name: bikeno
        description: bikeno description
        required: true
        schema:
          x-schema: '#/components/schemas/mdlRsp'
          x-schemaFld: bikeno
```

Note the update to the schema information. The x-schema and x-schema-Fld extensions tell the MDRest4i SDK generator how to define the variables used in the parameter handling logic.

- Save the specification, then generate using the “Create API” button

5.1.2 RPG Logic to handle Path Parameters

MDRest4i SDK generates an RPG subroutine (z_GetPathParm).

The call to the z_GetPathParm subroutine is placed at the top of the method subroutine. In the example this will be the z_ProcGET subroutine (which is called directly after the MDRest4i iCore components have parsed the request URL, headers and body).

```
// ================================================================
//    Override Get Method
// ================================================================
Begsr z_ProcGET;

  Exsr z_GetPathParm;

  if n_bikeno = *Off;
    Leavesr;
  endif;
```

The z_GetPathParm subroutine extracts the path parameters from the request into work fields, and processes any errors in the path parameters, such missing parameters or type mismatch.

```
Begsr z_GetPathParm;

  w_str2 = GetPathParm('MDRTUTLIB':1);
  if w_str2 <> '*notFound';
    n_bikeno = *On;
    monitor;
      p_bikeno = %dec(%trim(w_str2):10:0);
    on-error;
      n_bikeno = *off;
      SetHttpStatus(499:'Invalid Path Parameter');
      ErrorJson('Invalid Path Parameter');
    Endmon;
  Endif;
Endsr;
```

If there any errors in extracting the parameters, the HTTP status is set to “499 Invalid Path Parameter” and the JSON response below is sent back to the request.

```
{
  "error": "Invalid Path Parameters"
}
```

5.1.3 HTTP server ScriptAliasMatch for inpath parameters

To use in-path parameters the HTTP server config must be manually added to ensure the correct API Program is reached by the request. So the default wildcard regex ScriptAliasMatch :

```
ScriptAliasMatch ^/MDRTUTLIB/(.*) /QSYS.LIB/MDRTUTLIB.LIB/$1.PGM
```

Has the following ScriptAliasMatch to be used for the example above add ABOVE it:

```
ScriptAliasMatch bikepath1 /QSYS.LIB/MDRTUTLIB.LIB/inpath1.PGM
ScriptAliasMatch ^/MDRTUTLIB/(.*) /QSYS.LIB/MDRTUTLIB.LIB/$1.PGM
```

NOTE: The names of the program and the path have been deliberately differed in these examples to demonstrate the different configuration elements used in the HTTP config file

So the following url:

```
http://yourserver.com/MDRTUTLIB/23/bikepath1
```

will actually call program “inpath1” in library “MDRTUTLIB” passing inpath parameter “bikeno” 23

5.2 In-path Parameter other examples

5.2.1 First Position Parameter

URL/Path:

```
http://yourserver.com/{bikeno}/MDRTUTLIB/bikepath2
```

Program Name:

```
inpath2
```

HTTP config ScriptAliasMatch:

```
ScriptAliasMatch bikepath2 /QSYS.LIB/MDRTUTLIB.LIB/inpath2.PGM
```

Generated RPG Code:

```
        Begsr z_GetPathParm;

          w_str2 = GetPathParm('MDRTUTLIB':-1);
          if w_str2 <> '*notFound';
            n_bikeno = *On;
            p_bikeno = %trim(w_str2);
          Else;
            n_bikeno = *off;
            SetHttpStatus(499:'Missing Path Parameter');
            ErrorJson('Missing Path Parameter - bikeno');
          Endif;
        Endsr;
```

5.2.2 Multiple inpath Parameters

URL/Path:

```
http://yourserver.com/MDRTUTLIB/{deptno}/bikepath3/{bikeno}
```

Program Name:

```
Inpath3
```

HTTP config ScriptAliasMatch:

```
ScriptAliasMatch bikepath3 /QSYS.LIB/MDRTUTLIB.LIB/inpath3.PGM
```

Generated RPG Code:

```
        Begsr z_GetPathParm;

          w_str2 = GetPathParm('MDRTUTLIB':1);
          if w_str2 <> '*notFound';
            n_bikeno = *On;
            p_bikeno = %trim(w_str2);
          Else;
            n_bikeno = *off;
            SetHttpStatus(499:'Missing Path Parameter');
            ErrorJson('Missing Path Parameter - bikeno');
          Endif;
          w_str2 = GetPathParm('MDRTUTLIB':3);
          if w_str2 <> '*notFound';
            n_deptno = *On;
            p_deptno = %trim(w_str2);
          Else;
            n_deptno = *off;
            SetHttpStatus(499:'Missing Path Parameter');
            ErrorJson('Missing Path Parameter - deptno');
          Endif;
        Endsr;
```

5.3 Handling “path” parameters in a REST consumer

You can build REST consumer that call a REST service supplying path parameters to locate the target resource. If you want to specify path parameter(s), you need to have the relevant section in swagger at the same path level where query parameters are specified like below:

```
"parameters": [
     {
      "in": "path",
      "name": "bikeno",
      "description": "department",
      "required": true,
      "schema": {
       "x-schema": "#/components/schemas/mdlResp",
       "x-schemaFld": "model"
      }
     },
     {
      "in": "query",
      "name": "dept",
      "description": "department",
      "required": false,
      "schema": {
       "x-schema": "#/components/schemas/mdlResp",
       "x-schemaFld": "Department"
      }
     }
    ]
```

This specification would add the path parameter as the entry parameter to the generated consumer program is the same way as the query parameters. The relevant code snipped is shown below.

```
0010.60   * Write the main program prototype and PI
```

```
0010.70 d main            pr                  Extpgm('INPATHCNS')
0010.80 d p_bikeno                      50A
0010.90 d p_dept                        50A
0011.90
0012.00 d main            pi
0012.10 d p_bikeno                      50A
0012.20 d p_dept                        50A
```

The path parameters are added in “wg_servicepath” variable in “Initialize” procedure. In below example, the value “{bikeno}” must be there swagger if the path parameter name is “bikeno”. As you can see in “Initialize” procedure, “wg_servicepath” gets this path parameter replaced with the relevant variable name.

```
  "version": "12.0.0"
 },
 "paths": {
  "/mdrtutlib/{bikeno}/path1": {
   "get": {
    "operationId": "INPATHCNS",
    "x-lxrgen": {
```

```
0104.20     wg_hostName = 'bikes.com';
0104.30     wg_serverIP= ' ';
0104.40     wg_portNumber = 80;
0104.50
0105.50     wg_servicePath = '/mdrtutlib/' + %trim(p_bikeno) + '/path1';
0106.50     if w_numparm > 0;
0107.50       wg_urlparm = 'dept=' + %trim(p_dept);
0108.50     Endif;
0108.60
0108.70     // Set below indicator to *Off if you don't want to save logs in IFS
0108.80     ng_saveRestSwitch = *On;
0108.90     wg_saveRESTpath ='/mdrest4i/logs/listbikec.txt';
```

6 Automated parsing and writing of JSON using MDRJSONF File

6.1 Introduction

MDRESt4i has two useful JSON functions:

- **JsonToDB** – will automatically parse incoming JSON and write the name value data into a generic, predefined database file/table called MDRJSONF
- **JsonFromDB** – will build a structured JSON payload from a predefined database file/table called MDRJSONF.

The purpose of these is to avoid having to write any specific JSON parsing or JSON writing logic at all in the API or Consumer, and rather use SQL/native I/O to read or write the data being transported in JSON format.

Here are some use cases:

- A single, generic consumer program that can make different requests to any endpoint. All payload data is handled via MDRJSONF in external program/s which call the generic consumer
- Enable non RPG developers (COBOL/SYNON etc) to make complex and structured REST requests from IBM i.

- A generic consumer that is used as part of a security (OAUTH2 for example) flow to obtain and send credentials for tokens etc

6.2 MDRJSONF File Definition and Description

The MDRJSONF file is a predefined database file that can be found in library MDRSTxxxx. The design allows up to 12 levels of structure and 10,000,000 elements within a specific level

It has the following structure:

Name	Description	Length	Type	
REQRSP	REQ, RSP	3	A	It contains the entry type e. g. “REQ”, “RSP”
L01IDX	Level 1 Idx	7(0)	P	Array index value if level# 1 element is an array
L01NAM	Level 1 Name	80	A	Element name at level# 1
L02IDX	Level 2 Idx	7(0)	P	Array index value if level# 2 element is an array
L02NAM	Level 2 Name	80	A	Element name at level# 2

With additional levels down to 12….

L12IDX	Level 12 Idx	7(0)	P	Array index value if level# 12 element is an array
L12NAM	Level 12 Name	80	A	Element name at level# 12
LDEPTH	Level Depth	3(0)	P	Depth level (i.e. 1 to 12) of nested elements
ENTSEQ	Entry Seq	9	P	Entry sequence (record number)
ISNUMB	is Numeric Y/N	1	A	Y/N – Whether the entry is numeric type
ISBOOLEAN	is Boolean Y/N	1	A	Y/N – Whether the entry is Boolean type
VALUEN	Value Idx	32(15)	P	Numeric value (if this entry is numeric type)
VALUEA	Value	8000	A	Character value (if this is non-numeric value)

6.3 Flows

6.3.1 Consumer Flow

Sending Request Body: If you have your JSON data loaded in “MDRJSONF” file, you can simply use “JsonFromDB” procedure specifying the entry type from the DB file to be processed and the library where the file exists. In below example, the procedure is getting called to process “RSP” type entries of “MDRJSONF” file in “MDRTUTLIB” library.

```
p BuildRequest    b                   export
d BuildRequest    pi
 /Free

   JsonFromDB('RSP':'MDRTUTLIB':w_success);

 /End-Free
p BuildRequest    e
```

Receiving Response: The indicator "ng_parse" decides whether the parsing of the JSON response received in consumer is to happen via automatic processing route or you want to manually control it. In order to manually control, the first step is to set the indicator "ng_parse" to "*Off" in "Initialize" procedure of the consumer program. The second step is to call the procedures to create/clear the JSON file and load the JSON to DB once "GskConsume" procedure returns the control back to the program. In below example, the "CreateJSONF" procedure is getting called to create the "MDRJSONF" file in "MDRTUTLIB" library. If the return code is "N", means the file was not created as it already existed. In that case, we have called "ClearJSONF" procedure to clear the records (with entry type = "RSP") from "MDRJSONF" file in "MDRTUTLIB" library . Subsequently, we call "JsonToDB" procedure to parse the JSON response and load the entries in DB file in "MDRTUTLIB" library ad set entry type of those records as "RSP".

```
InitVariant();
tg_InitializePointer = %paddr(Initialize);
tg_BuildReqPointer = %paddr(BuildRequest);
tg_ClosedownPointer  = %paddr(Closedown);
GskConsume();

CreateJSONF('MDRTUTLIB':w_success);
if w_success = 'N';
  ClearJsonF('MDRTUTLIB':'RSP':w_success:w_dbfsts);
endif;

JsonToDB('MDRTUTLIB':'REX':w_success);
```

6.3.2 Provider Flow

6.3.2.1 Receiving Request Body:

If your REST service is receiving the JSON content as part of the request body, MDRest4i by default runs the JSON parser so that you can retrieve the values of required elements via JGet, JPathv etc functions. However, if you instead want to write the JSON entries in DB file, you can set the indicator "n_parse" to *On in "z_CustomInit" subroutine of your REST service to stop automatic JSON parsing. You can then control the JSON parsing by using "JsonToDB" procedure in the z_Procxxx procedure.

6.3.2.2 Sending Response:

If you have your JSON data loaded in "MDRJSONF" file, you can simply use "JsonFromDB" procedure specifying the entry type from the DB file to be processed and the library where the file exists. In below example, the procedure is getting called to process "RSP" type entries of "MDRJSONF" file in "MDRTUTLIB" library.

6.4 Writing Data to MDRJSONF to use with an API or Consumer

Let's assume we want to use MDRest4i functions and write data to get below JSON output:

```
{
    "Stock": [
      {"Department": "Cycles",
       "Category": {
                     "MainCategory": "Bikes",
                     "Sub-Category": "Racing"
                 },
       "Sizes": [51, 58, 60]
       },
      {"Department": "Mountain",
       "Category": {
                     "MainCategory": "E-Bikes",
                     "Sub-Category": "MTB"
                 },
```

```
        "Sizes": [51, 60],
        "Colours": ["White", "Black", "Green", "Red", "Blue"]
        }
    ]
}
```

One way is that, we use the standard logic below so that the data is directly written on the output variable/pointer.

```
Begsr z_PrcResponse;

   beginObject(' ');
   beginArray('Stock');
   beginObject(' ');
   addChar('Department':'Cycles');
   beginObject('Category');
   addChar('MainCategory':'Bikes');
   addChar('Sub-Category':'Racing':'*LAST');
   endObject();
   beginArray('Sizes');
   addIntr(' ':51);
   addIntr(' ':58);
   addIntr(' ':60:'*LAST');
   endArray('*LAST');
   endObject();
   beginObject(' ');
   addchar('Department':'Mountain');
   beginObject('Category');
   addChar('MainCategory':'E-Bikes');
   addChar('Sub-Category':'MTB':'*LAST');
   endObject();
   beginArray('Sizes');
   addIntr(' ':51);
   addIntr(' ':60:'*LAST');
   endArray();
   beginArray('Colours');
   addChar(' ':'White');
   addChar(' ':'Black');
   addChar(' ':'Green');
   addChar(' ':'Red');
   addChar(' ':'Blue':'*LAST');
   endArray('*LAST');
   endObject('*LAST');
   endArray('*LAST');
   endObject('*LAST');

Endsr;
```

However, if you want the output to be built in some other program (using SQL or native RPG) based on some different logic, lets try the second way which is the context of this example here. We are now going to write the entries in DB file via RPG native operation and then we will use this DB file to build the JSON:

```
0011.00 fmdrjsonf  uf a e           k disk    Usropn extfile(w_extfile)

0054.00 d w_extFile       s             21a
0055.00 d w_success       s              1a
0056.00 d w_dbfsts        s             50a

0086.00    Begsr z_ProcPost;
0087.00

0088.00    CreateJSONF('MDRTUTLIB':w_success);
0089.00    if w_success = 'N';
0092.00      ClearJsonF('MDRTUTLIB':'KY1':w_success:w_dbfsts);
0093.00    endif;
```

```
0098.00     w_extFile = 'MDRTUTLIB/MDRJSONF';
0099.00     open(e) mdrjsonf;
0100.00     REQRSP = 'KY1';   // Set the key value for the records being written
0101.00     L01IDX = 1;       // First entry "Stock" is an array, start with index=1
0102.00     L01NAM = 'Stock';
0103.00     L02IDX = 0;       // Second entry "Department" is normal, set index=0
0104.00     L02NAM = 'Department'; // This is the label for the entry being written
0105.00     VALUEA = 'Cycles';    // This is the value of "Department" label
0106.00     ENTSEQ = 1;      // Entry number - incremental value for each record
0107.00     LDEPTH = 2;      // How far to go in cascaded object/non-object entries
0108.00     Write Rdrjsonf;  // Write first record
0109.00     //Below is second entry. Set only elements where value changes
0110.00     L02NAM = 'Category';
0111.00     L03IDX = *zeros;   // Maincategory is non-array entry at depth level 3
0112.00     L03NAM = 'MainCategory';
0113.00     VALUEA = 'Bikes';
0114.00     ENTSEQ = 2;  //Second record, so entry sequence incremented to 2
0115.00     LDEPTH = 3;  //Depth level=3 (i.e. Stock[xxxxx].Category.MainCategory)
0116.00     Write Rdrjsonf;
0117.00
0118.00     L03NAM = 'Sub-Category';
0119.00     VALUEA = 'Racing';
0120.00     ENTSEQ = 3;  //third record
0121.00     LDEPTH = 3;
0122.00     Write Rdrjsonf;
0123.00
0124.00     Clear L03IDX;
0125.00     Clear L03NAM;
0126.00     Clear VALUEA;
0127.00     L02IDX = 1;
0128.00     L02NAM = 'Sizes';
0129.00     VALUEN = 51;
0130.00     ISNUMB = 'Y';
0131.00     ENTSEQ = 4;
0132.00     LDEPTH = 2;
0133.00     Write Rdrjsonf;
0134.00
0135.00     L02IDX = 2;
0136.00     VALUEN = 58;
0137.00     ENTSEQ = 5;
0138.00     Write Rdrjsonf;
0139.00
0140.00     L02IDX = 3;
0141.00     VALUEN = 60;
0142.00     ENTSEQ = 6;
0143.00     Write Rdrjsonf;
0144.00
0145.00     L01IDX = 2;
0146.00     L02IDX = *zeros;
0147.00     Clear Valuen;
0148.00     L02NAM = 'Department';
0149.00     VALUEA = 'Mountain';
0150.00     ISNUMB = 'N';
0151.00     ENTSEQ = 7;
0152.00     Write Rdrjsonf;
0153.00
0154.00     L02NAM = 'Category';
0155.00     L03IDX = *zeros;
0156.00     L03NAM = 'MainCategory';
0157.00     VALUEA = 'E-Bikes';
0158.00     ENTSEQ = 8;
0159.00     LDEPTH = 3;
0160.00     Write Rdrjsonf;
0161.00
0162.00     L03NAM = 'Sub-Category';
0163.00     VALUEA = 'MTB';
```

```
0164.00     ENTSEQ = 9;
0165.00     LDEPTH = 3;
0166.00     Write Rdrjsonf;
0167.00
0168.00     Clear L03IDX;
0169.00     Clear L03NAM;
0170.00     Clear VALUEA;
0171.00     L02IDX = 1;
0172.00     L02NAM = 'Sizes';
0173.00     VALUEN = 51;
0174.00     ISNUMB = 'Y';
0175.00     ENTSEQ = 10;
0176.00     LDEPTH = 2;
0177.00     Write Rdrjsonf;
0178.00
0179.00     L02IDX = 2;
0180.00     VALUEN = 60;
0181.00     ENTSEQ = 11;
0182.00     Write Rdrjsonf;
0183.00
0184.00     L02IDX = 1;
0185.00     L02NAM = 'Colours';
0186.00     Clear VALUEN;
0187.00     Clear ISNUMB;
0188.00     ENTSEQ = 12;
0189.00     LDEPTH = 2;
0190.00     VALUEA = 'White';
0191.00     Write Rdrjsonf;
0192.00
0193.00     L02IDX = 2;
0194.00     ENTSEQ = 13;
0195.00     VALUEA = 'Black';
0196.00     Write Rdrjsonf;
0197.00
0198.00     L02IDX = 3;
0199.00     ENTSEQ = 14;
0200.00     VALUEA = 'Green';
0201.00     Write Rdrjsonf;
0202.00
0203.00     L02IDX = 4;
0204.00     ENTSEQ = 15;
0205.00     VALUEA = 'Red';
0206.00     Write Rdrjsonf;
0207.00
0208.00     L02IDX = 5;
0209.00     ENTSEQ = 16;
0210.00     VALUEA = 'Blue';
0211.00     Write Rdrjsonf;
0212.00     close(e) mdrjsonf;
0213.00     // Below statement builds the JSON from the DB using same “beginObject”,
0214.00     // “endObject”, “addchar” etc functions internally.
0218.00     JsonFromDB('MDRTUTLIB':'KY1':w_success);
0219.00
0220.00     Endsr;
```

6.5 Reading Data from the MDRJSON file using RPG/SQL

The example given above to write the JSON structure in MDRJSONF file provides the data as you can see in excel sheet below. This table has columns from L01IDX/L01NAM to L12IDX/L12NAM for the possibility of up to 12 level depth. However, the columns L04IDX/L04NAM to L12IDX/L12NAM are excluded from below display to fit on A4 page. Suppose you have received the JSON data as request body in a POST request and you used JSONTODB function to load that JSON in DB file.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	REQRSP	L01IDX	L01NAM	L02IDX	L02NAM	L03IDX	L03NAM	LDEPTH	ENTSEQ	ISNUMB	ISBOOLEA	VALUEN	VALUEA
2	KY1	1	Stock	0	Department	0		2	1			0	Cycles
3	KY1	1	Stock	0	Category	0	MainCategory	3	2			0	Bikes
4	KY1	1	Stock	0	Category	0	Sub-Category	3	3			0	Racing
5	KY1	1	Stock	1	Sizes	0		2	4	Y		51	
6	KY1	1	Stock	2	Sizes	0		2	5	Y		58	
7	KY1	1	Stock	3	Sizes	0		2	6	Y		60	
8	KY1	2	Stock	0	Department	0		2	7	N		0	Mountain
9	KY1	2	Stock	0	Category	0	MainCategory	3	8	N		0	E-Bikes
10	KY1	2	Stock	0	Category	0	Sub-Category	3	9	N		0	MTB
11	KY1	2	Stock	1	Sizes	0		2	10	Y		51	
12	KY1	2	Stock	2	Sizes	0		2	11	Y		60	
13	KY1	2	Stock	1	Colours	0		2	12			0	White
14	KY1	2	Stock	2	Colours	0		2	13			0	Black
15	KY1	2	Stock	3	Colours	0		2	14			0	Green
16	KY1	2	Stock	4	Colours	0		2	15			0	Red
17	KY1	2	Stock	5	Colours	0		2	16			0	Blue
18													
19													

You now want to read the JSON data from DB file using RPG native/SQL operations for database reading. Suppose you have defined the data structure like below for this JSON structure. Subsequently, there is logic to load that DS from the DB file. This is just an example but you can build your own sophisticated examples using native DB I/O or SQL.

```
0036.02 d d_stock         Ds                  dim(10) qualified
0036.03 d   s_department               100a
0036.04 d   s_category                      likeds(d_category)
0036.05 d   s_sizes                   3p 0 dim(10)
0036.06 d   s_colours                 10a   dim(10)
0036.08
0036.09 d d_category      ds
0036.10 d   s_maincat                 50a
0036.11 d   s_subcat                  50a

0066.00    Begsr Z_ProcessData;
0067.00
0068.00      setll 1 mdrjsonf;
0069.00      read mdrjsonf;
0070.00      dow not %eof(mdrjsonf);
0071.00        if L02NAM = 'Department';
0072.00          d_stock(L01IDX).s_department = VALUEA;
0072.01        elseif L02NAM = 'Category' and L03NAM = 'MainCategory';
0072.02          d_stock(L01IDX).s_category.s_maincat = VALUEA;
0072.03        elseif L02NAM = 'Category' and L03NAM = 'Sub-Category';
0072.04          d_stock(L01IDX).s_category.s_subcat = VALUEA;
0072.05        elseif L02NAM = 'Sizes';
0072.06          d_stock(L01IDX).s_sizes(L02IDX) = VALUEN;
0072.07        elseif L02NAM = 'Colours';
0072.08          d_stock(L01IDX).s_colours(L02IDX) = VALUEN;
0073.00        endif;
0077.00
0078.00        read mdrjsonf;
0079.00      enddo;
0104.00
0105.00    Endsr;
```

6.6 REST Consumer Example using MDRJSONF database file

Here is an example of the MDRGENCNS command used to build a consumer template that uses this MDRJSONF concept.

```
MDRest4i Generator - Consumer (MDRGENCNS)

 Type choices, press Enter.

 Target Library . . . . . .   MDRTUTLIB     Name
```

```
 Target Source File . . . . . .   QRPGLESRC     Name
 Target Source Member . . . . .   DBFCNS        Name
 Member Text  . . . . . . . . .   'Test Member Generated by MDRest4i'

 SSL/Non SSL? . . . . . . . . .   N             S=SSL , N-Non SSL
 Single or Multiple Service . .   S             M=Multiple, S=Single
```

- *Press enter*

```
HTTP Method T=PATCH D=DEL  . . .   S             G=GET, P=PUT, S=POST
Build request body method  . . .   DBF           IFS, DBF, USR
Library Name . . . . . . . . . . > MDRTUTLIB     Character value
Parse response method  . . . . .   DBF           IFS, DBF, USR
Library Name . . . . . . . . . . > MDRTUTLIB     Character value
URL for Consumer Service . . . .   _____________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________
IFS Path to store the Output . .   _____________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________
File name on the IFS Directory     _____________________________________
_
```

- *Press enter*

The generated code is below. The relevant code for MDRJSONF processing is highlighted.

```
0001.00 h dftactgrp(*no) actgrp(*new)
0002.00 h bnddir('LXRGLOBAL': 'BNDZIP')
0003.00  ************************************************************
0004.00  *  MDRest4i SSL Consumer Template GET JSON
0005.00  ************************************************************
0006.00
0007.00  /copy qrpglesrc,mdrestdfn
0008.00
0009.00 d Initialize      pr
0010.00
0011.00 d BuildRequest    pr
0012.00 d CloseDown       pr
0013.00
0014.00 d w_success       s              1
0015.00 d w_dbfsts        s             50
0016.00  /free
0017.00
0018.00    InitVariant();
0019.00    tg_InitializePointer = %paddr(Initialize);
0020.00    tg_BuildReqPointer = %paddr(BuildRequest);
0021.00    tg_ClosedownPointer  = %paddr(Closedown);
0022.00    GskConsume();
0023.00
0024.00    if wg_httpStatus = '200';
0025.00
0026.00      // Change above condition to appropriate success status codes as
0027.00      // applicable to your REST service and add the processing logic here
0028.00
0029.00      // You may also call "GetErrorWarnings" procedure with one parameter
0030.00      // of 1024a attribute to get any errors/warnings reported in execution
0031.00
0032.00      // Create or Clear MDRJSONF
0033.00      CreateJSONF('MDRTUTLIB':w_success);
```

```
0034.00       // Write response in MDRJSONF
0035.00       JsonToDB('MDRTUTLIB':'CRS':w_success);
0036.00     Endif;
0037.00
0038.00     // Cleanup the memory allocated dynamically from heap
0039.00     cleantree();
0040.00
0041.00     *Inlr = *On;
0042.00
0043.00   /End-Free
0044.00   *------------------------------------------------------------
0045.00   * End of Mainline Section
0046.00   *------------------------------------------------------------
0047.00
0048.00   * Procedure Interfaces
0049.00   *------------------------------------------------------------
0050.00
0051.00 p CloseDown       b                   export
0052.00 d CloseDown       pi
0053.00   /Free
0054.00
0055.00     //Enter service closedown instructions here
0056.00   /End-Free
0057.00 p CloseDown       e
0058.00
0059.00 p BuildRequest    b                   export
0060.00 d BuildRequest    pi
0061.00   /Free
0062.00
0063.00     //Enter the request body  here
0064.00
0065.00     // Please write the logic to load request in MDRJSONF file
0066.00     // in MDRTUTLIB library with the  reqrsp = 'CRQ' (if not already
0067.00     // loaded before the calling JsonFromDB function below:
0068.00     JsonFromDB('MDRTUTLIB':'CRQ':w_success);
0069.00   /End-Free
0070.00 p BuildRequest    e
0071.00
0072.00 p Initialize      b                   export
0073.00 d Initialize      pi
0074.00   /Free
0075.00
0076.00     // Set Import Values
0077.00     wg_contentType = 'application/json; charset=UTF-8';
0078.00     wg_httpsMethod = 'POST';
0079.00
0080.00     wg_maxAttempt = 25;
0081.00     wg_mSecDelay = 500000;
0082.00
0083.00     wg_hostName = '';
0084.00     wg_serverIP= ' ';
0085.00     wg_portNumber = 80;
0086.00
0087.00     wg_servicePath = '';
0088.00     wg_urlParm = '      ';
0089.00
0090.00     // Set below indicator to *Off if you don't want to save logs in IFS
0091.00     ng_saveRestSwitch = *Off;
0092.00     wg_saveRESTpath = '/MDREST4I/log/T12';
0093.00
0094.00
0095.00     // The setting "ng_ssl=*On" enables SSL request. Otherwise, it's non SSL
0096.00     ng_ssl = *Off;
0097.00
0098.00     // If the REST service requires authentication, set ng_authsend = *On
0099.00     // and then use one of the three subsequent variables to send auth info.
```

```
0100.00    // In order to send the basic authentication via userId and pwd, set the
0101.00    // variables wg_username and wg_password. If however, it's the "Bearer"
0102.00    // token or some other auth string, set the value in "wg_authstring".
0103.00    // e.g. wg_authstring = Bearer <token value>
0104.00
0105.00    ng_authsend = *off;
0106.00    wg_username = *blanks;
0107.00    wg_password = *blanks;
0108.00    wg_authstring = *blanks;
0109.00
0110.00    // If the REST service is accessible via proxy, set ng_proxy = *on
0111.00    // and set the proxy details in subsequent variables. If the proxy
0112.00    // host is in IP address format, set the value in "wg_proxyIP" and if
0113.00    // it's the host name format, set the hostname in the "wg_proxyhost"
0114.00    // variable without any http/https prefix and there shouldn't be any
0115.00    // slash before or after the host name or IP address. The port number
0116.00    // of the proxy host should be set in "wg_proxyport" variable below.
0117.00    // If the proxy is on SSL channel, set "ng_proxySSL" to *On.
0118.00    ng_proxy = *Off;
0119.00    ng_proxyssl = *Off;
0120.00    wg_proxyhost = *blanks;
0121.00    wg_proxyIp = *blanks;
0122.00    wg_proxyport = *zeros;
0123.00    wg_proxyusr = *blanks;
0124.00    wg_proxypwd = *blanks;
0125.00
0126.00   // If you want to load config via MRCNSDTLF file, please uncomment
0127.00   // the below line. In this case you need to make sure you have loaded
0128.00   // (Host, Port, path etc.) in the MRCNSDTLF file with the key of This
0129.00   // Program name. Add below file in the 'F' spec in this program
0130.00   // mrcnsdtlf if   e           k disk    Usropn
0131.00   // /copy qrpglesrc,mrsetcon
0132.00
0133.00   /End-Free
0134.00 p Initialize      e
0135.00
0136.00   * ------------------------------------------------------------------------
0137.00   *  Procedure - FixReprocess
0138.00   * ------------------------------------------------------------------------
0139.00 p FixReprocess    b                   export
0140.00 d FixReprocess    pi
0141.00   /free
0142.00   /end-free
0143.00 p FixReProcess    e
0144.00   * ------------------------------------------------------------------------
0145.00   *  Procedure - InitVariant
0146.00   * ------------------------------------------------------------------------
0147.00 p InitVariant     b
0148.00 d InitVariant     pi
0149.00   /free
0150.00    w_OpenCurly = %char(c_OpenCurly);
0151.00    w_CloseCurly = %char(c_CloseCurly);
0152.00    w_OpenSquare = %char(c_OpenSquare);
0153.00    w_CloseSquare = %char(c_CloseSquare);
0154.00    w_EscapeChar = %char(c_EscapeChar);
0155.00    w_Power = %char(c_Power);
0156.00    w_Tilde1 = %char(c_Tilde1);
0157.00    w_Exclaim = %char(c_Exclaim);
0158.00    w_Hash = %char(c_Hash);
0159.00    w_Pipe1 = %char(c_Pipe1);
0160.00    w_Accent = %char(c_Accent);
0161.00    w_Doller = %char(c_Doller);
0162.00    w_AtSign = %char(c_AtSign);
0163.00
0164.00   /end-free
0165.00 p InitVariant     e
```

6.7 REST Service Example using MDRJSONF database file

Here is an example of the MDRGENPRD command used to build a Provider template that uses this MDRJSONF concept.

```
                  MDRest4i Generator - Provider (MDRGENPRD)

 Type choices, press Enter.

 Target Library . . . . . .   MDRTUTLIB     Name
 Target Source File . . . . .   QRPGLESRC     Name
 Target Source Member . . . . .   TSTDBFPRD     Name
 Member Text  . . . . . .   'Test Member Generated by MDR'

 Get Method Required? . . . . . .   N            Y=Yes , N=No
 Put Method Required? . . . . . .   N            Y=Yes , N=No
 Post Method Required?  . . . . .   Y            Y=Yes , N=No
 Patch Method Required? . . . . .   N            Y=Yes , N=No
 Delete Method Required?  . . . .   N            Y=Yes , N=No
 Options Method Required? . . . .   N            Y=YES , N=NO
 Data Format  . . . . . .   J            J=JSON, X-XML




                                                                  Bottom
 F3=Exit    F4=Prompt    F5=Refresh    F12=Cancel    F13=How to use this display
 F24=More key
s
```

```
Response Processing Method . . .   DBF           IFS, DBF, USR
Library Name for DBF Request . . > MDRTUTLIB     Name
```

```
Requestbody Parse Method . . . .   DBF          IFS, DBF, USR
Library Name for DBF Request . . > MDRTUTLIB    Name
```

- *Press enter*

The generated code is below. The relevant code for MDRJSONF processing is highlighted.

```
0001.00 h dftactgrp(*no) actgrp(*NEW) alloc(*teraspace)
0002.00 h bnddir('MDRUSERBND')
0003.00 h bnddir('LXRGLOBAL':'BNDZIP')
0004.00  * ©=2019 Midrange Dynamics=======================================
0005.00  * MDRest4i OVERRIDE - uncomment to allow comma, decimal seperator
0006.00  * ©=2019 Midrange Dynamics=======================================
0007.00  * h decedit(',')
0008.00  * ©=2019=Midrange Dynamics=======================================
0009.00  * ©=2019=Midrange Dynamics=======================================
0010.00  * Create Date :
0011.00  * Created By  : Midrange Dynamics
0012.00  * Description : This is the MDRest4i RESTfull Webservice
0013.00  * Service Type: RESTfull Webservice
0014.00  * Input       : GET Request - URL PARMS
0015.00  * Output      : Error/warning - JSON Component
0016.00  * ©=2019=Midrange Dynamics=======================================
0017.00
0018.00  // Standard MDRest4i Copybooks for REST Service
```

```
0019.00  /copy qrpglesrc,mdrusercpy
0020.00
0021.00  // Variable Definitions
0022.00 d w_string        s          20480
0023.00 d w_str2          s          20480
0024.00 d w_workstr       s        5242880
0025.00 d w_datalen       s             10i 0
0026.00 d w_success       s              1
0027.00 d w_dbfsts        s             50
0028.00
0029.00
0030.00  // Indicate subroutine overrides defined locally
0031.00  /define LXR_CustomInit
0032.00
0033.00  /define LXR_ProcPost
0034.00
0035.00  /copy qrpglesrc,mdrestdfn
0036.00  /copy qrpglesrc,lxrrestc
0037.00
0038.00  /undefine LXR_CustomInit
0039.00
0040.00  /undefine LXR_ProcPost
0041.00
0042.00  /Free
0043.00    // ===============================================================
0044.00    //   Customized initialization
0045.00    // ===============================================================
0046.00    Begsr Z_CustomInit;
0047.00
0048.00      // If the response (GET/POST/PUT/PATCH) is greater than 5Mb,
0049.00      // set w_maxrsplen to the expected maximum response length
0050.00      // uncomment the statement below and set the value as required
0051.00      // w_maxrsplen =15542880;
0052.00
0053.00      // If the requestBody (POST/PUT/PATCH) is greater than 5Mb,
0054.00      // set w_maxreqlen to the expected maximum request body length
0055.00      // w_maxreqlen =15542880;
0056.00
0057.00      // If request or response value is greater than 16MB,
0058.00      // declare the keyword "alloc(*teraspace)" in control spec of
0058.00      // this pro
0059.00
0060.00      // In order to determine the actual request/response size, set the
0061.00      // indicators "n_getreqsize"/"n_getrspsize" (as applicable) to *On.
0062.00      // The REST API will then only return the request/response
0062.00      // size. The indicator "n_getrspsize" will allocate 500MB
0062.00      // memory from heap
0064.00      // to process the reqeust and return the response size.
0065.00      // If the expected response size is much lesser, set the max
0065.00      // expected size in "w_debug_max_rsp_size" e.g. 10485760
0065.00      // (10MB) size in "w_debug_max_rsp_size" e.g. 10485760 (10MB)
0067.00
0068.00      // Use this IFS switch to log the API data in an IFS Folder
0069.00      // If you set n_saveIFSSwitch to *on; Then set the log
0069.00      // filepath e.g:
0070.00      // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/' +
0070.00      // 'LogFileddmmyyyy.txt'
0071.00      n_saveIFSSwitch = *off;
0072.00
0073.00      // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-
0073.00      // MYSERVERJOB-nnnnnn
0074.00      // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP
0073.00      // Instanc
0075.00      ng_mdrJobHdr = *off;
0076.00
0077.00      // Extracts request query parameters to d_Qparm data structure
```

```
0078.00       n_extractQryPrms = *off;
0079.00       n_parse = *off;
0080.00
0081.00     Endsr;
0082.00
0083.00     // ===============================================================
0084.00     //    Override Post Method
0085.00     // ===============================================================
0086.00     Begsr z_ProcPost;
0087.00
0086.00       CreateJSONF('MDRTUTLIB':w_success);
0087.00       if w_success = 'N';
0088.00         ClearJsonF('MDRTUTLIB':'REQ':w_success:w_dbfsts);
0089.00       endif;
0090.00       JsonToDB('MDRTUTLIB':'REQ':w_success);
0091.00
0092.00       // Response processing
0093.00
0094.00       // Below is the dummy update which you need to remove and add
0095.00       // appropriate logic to process the request body  and load the
0096.00       // DB file with the response data
0097.00       exec sql update MDRTUTLIB/mdrjsonf set reqrsp = 'RSP' where
0098.00       reqrsp = 'REQ';
0099.00
0100.00       JsonFromDB('MDRTUTLIB':'RSP':w_success);
0099.00
0100.00       // Logic from custom copybook
0101.00
0102.00
0103.00     Endsr;
0104.00   /End-Free
0105.00     // ===============================================================
0106.00     //   LXR Generic Procedures
0107.00     // ===============================================================
0108.00   /copy qrpglesrc,lxrrestp
```

Appendix A - OAPI Extensions used by MDRGENXAPI

1 Information Extensions

These are available for general information use and integration with other systems and parts of MDRest4i SDK such as SCM details or Developer portal references

Extension Name	Parent Object	Description
x-lxrinfo	info	General Information used for SCM and API Portal details
environment	x-lxrinfo	API environment
timestamp	x-lxrinfo	Last update
refuuid	x-lxrinfo	Unique identifier
program_name	x-lxrinfo	Program name generated – used by Open API Studio

1.1 Example - Information Extensions

```
"x-lxrinfo": {
   "environment": "yourEnvironment",
   "timestamp": "2017-06-23T09:32:43.511Z",
   "refuuid": "8f1066c8-9065-4d8a-9230-d2df8065dee0",
   "program_name": "/getclntdtl"
  }
```

2 Generator Directives

REST APi MDRGENXAPI uses these to save and generate the object into the correct library, and source file, along with a SSL and CCSSID settings and also record the response of the generation from MDRGENXAPI

Here the names of objects, libraries and source members are provided for the generator.

Extension Name	Parent Object	Description
x-lxrgen	method-get, post etc	Contains the generator directive details
library	x-lxrgen	Library where the object will be compiled to
srcfile	x-lxrgen	Last update
srclibrary	x-lxrgen	Library where the source file exists – srcFile will be created by the generator if it does not exist in this library
object	x-lxrgen	Program and source member name to be generated.
reqtype	x-lxrgen	This parameter MUST contain the value “consumer” if you want to generate a consumer for the swagger.

sslcns	x-lxrgen	For consumer generation request, this parameter guides whether the consumer is going to process SSL service or non-SSL. The values could be “Y” or “N”
lxrpath	x-lxrgen	For consumer generation request, this parameter should be specified the complete URL of the REST service.
multi-ccsid	x-lxrgen	This parameter can be used to set “Y” or “N”. It is basically needed when it’s possible to generate the service on one machine with some CCSID and is expected to run on a machine with different CCSID. In that case it basically generates the code with hexa-decimal constants for open/close square and curly braces.
jwt	x-lxrgen	“Y” or “N”. this sets on JWT handling and implements the ValidateJWT() procedure in the relevant z_proc[methodxxx] subroutine. It extracts the “authorization: Bearer …..” header value
jsonreqmethod	x-lxrgen	This parameter controls building JSON. For the consumer, the JSON is built for the request body and for the REST service, JSON is created for the response. If you want to automatically send the JSON from an IFS file, the value should be“IFS”. “DBF” means the JSON parsed content is stored in MDRJSONF file and JSON should be built from the data in this file. The default value “USR” means continue with the standard way of building JSON using addChar, beginObject etc functions.
reqdblib	x-lxrgen	This parameter is only used when “jsonreqmethod” is set to “DBF” and it tells from where to pick the MDRJSONF file.
reqifspath	x-lxrgen	When “jsonreqmethod” is set to “IFS”, this parameter is used to pick the IFS file where the final JSON (i.e. the request body received for the REST service) would be written. When used with “reqtype” set as “consumer”, the JSON content available in this IFS file will be read and written as the request body while sending the request.
jsonrespmethod	x-lxrgen	This parameter controls on how to parse the incoming JSON. For the consumer, the incoming JSON is the response from the REST service and that for the REST service is the request body. Use the value “IFS” to load the JSON in an IFS file, whereas, the value “DBF” can be used to load the json labels, values in the DB file named “MDRJSONF”. The default value

		"USR" means continue with the standard way of parsing JSON and load the required values using JGet, JPathV etc functions.
respdblib	x-lxrgen	This parameter is only used when "jsonrespmethod" is set to "DBF" and it tells from where to pick the MDRJSONF file.
rspifspath	x-lxrgen	When "jsonrespmethod" is set to "IFS", this parameter tells the IFS file where the final JSON (i.e. the response received from the REST service in consumer) would be written. When used with REST service, the JSON content available in this IFS file will be read and written as the response to the REST service.
allownotfound	x-lxrgen	The JSON values for the specific JSON label at the specific path are extracted using "JPathv" function. If the requested label/path is not found, the function returns "*notFound". However, if you expect "*notFound" to be something possibly coming in JSON, set this parameter to "true" and then "JpathE" function will be used to extract the value of JSON path which returns blank if the entry is not found.
Missingentryhandling	x-lxrgen	This parameter is used to apply condition before setting the value to avoid setting the variable when the value is not found. This is explained below using code example.
lastmdrgen	x-lxrgen	This key/value pair gets added to the x-lxrgen after the API is successfully generated. It creates/updates the timestamp of the last generated date/time.
msgLvl	x-lxrgen	This key/value pair gets added to the x-lxrgen after the API is successfully generated. It saves the value of the message level returned by the MDRGENXAPI (10 or 20 in case of successful generation or generation with warning respectively)
msg	x-lxrgen	This key/value pair gets added to the x-lxrgen after the API is successfully generated. It saves the value of the message returned by the MDRGENXAPI
gentype	x-lxrgen	This parameter accepted three values (legacy, Inline and external). If we want to use generate the REST APIS and consumer programs via MDRGNAPI and we will set this gentype as 'Inline' it will generate with all in one pgm., If we will set this as "gentype": "external", it will generate program with procedures externalised.

2.1 Provider Example - Generator Directives


```
{
   "get": {
    "operationId": "getclntdtl",
    "x-lxrgen": {
     "library": "MDRTUTLIB",
     "srcfile": "QRPGLESRC",
     "srclibrary": "MDRTUTLIB",
     "object": " getclntdtl ",
     "jwt": "Y",
     "lastmdrgen": "2021-05-31T13:28:44.333Z",
     "msgLvl": 10,
     "msg": "SUCCESS - source generated and object compiled",
     "gentype": "legacy"
    }
}
```


2.2 Consumer Example - Generator Directives


```
   "post": {
     "operationId": "postclntac",
     "x-lxrgen": {
       "library": "MDRTUTLIB",
       "srcfile": "QV11TUT",
       "srclibrary": "MDRTUTLIB",
       "reqType": "consumer",
       "object": "postclntac",
       "lxrpath": "http://yourserver:9999/yourlib/postclnta",
       "reqtype": "consumer",
       "sslcns": "N",
       "multi-ccsid": "N",
       "lastmdrgen": "2021-04-08T06:02:29.888Z",
       "msgLvl": 10,
       "msg": "SUCCESS - source generated and object compiled",
       "gentype": "legacy"
      }
```


2.3 Example using MDRGNAPI with generation type Inline

In this way MDRGNAPI will generate the all in one program.


```
{
   "get": {
    "operationId": "getclntdtl",
    "x-lxrgen": {
     "library": "MDRTUTLIB",
     "srcfile": "QRPGLESRC",
     "srclibrary": "MDRTUTLIB",
     "object": " getclntdtl ",
     "jwt": "Y",
     "lastmdrgen": "2021-05-31T13:28:44.333Z",
     "msgLvl": 10,
     "msg": "SUCCESS - source generated and object compiled",
     "gentype": "inline"
    }
}
```


2.4 Example using MDRGENAPI with the generation type external

In this MDRGNAPI will generate the pgm with procedures externalized. In the below example member will generate ‘getclntdtl’ and copybook name getclntdtc’ and module name will be getclntdtlm.


```
{
   "get": {
    "operationId": "getclntdtl",
    "x-lxrgen": {
     "library": "MDRTUTLIB",
     "srcfile": "QRPGLESRC",
     "srclibrary": "MDRTUTLIB",
     "object": "getclntdtl",
     "jwt": "N",
     "lastmdrgen": "2022-03-02T06:23:13.716Z",
     "msgLvl": 10,
     "msg": "SUCCESS - source generated and object compiled",
```

```
    "gentype": "external",
    "copylib": "MDRTUTLIB",
    "modslib": "MDRTUTLIB",
    "copymember": "getclntdtC",
    "prcmodule": "getclntdtM",
    "modsrcf": "QQRPGLESRC",
    "copysrcf": "QRPGLESRC"}
```

3 Query Parameter Association

The values of the REST API query parameters can be used as variables in the API and even as values in the SQL where clause of the generated API/Service. The name of a schema object and a corresponding object in that schema must be provided, for the generator to build the necessary assignment logic in the parameter subroutines in the RPGLE.

Extension Name	Parent Object	Description
x-schemaFld	Item in Parameter Array	Contains the name of the field in the schema defined in x-schema
x-schema	Item in Parameter Array	Contains the name of Schema Object Name

3.1 Example - Parameter Association

```
"parameters": [
        {
          "in": "query",
          "name": "id",
          "description": "client ID",
          "required": false,
          "schema": {
            "x-schema": "#/components/schemas/LXCLIENTMDL",
            "x-schemaFld": "id"
          }
        }
      ]
```

4 SQL IO & GET Provider

MDRest4i SDK automatically generates RPGLE logic that associates REST methods: GET, POST, PATCH, PUT, DELETE - and SQL actions: select, insert, set or delete over the DB2 data. The generator will build the necessary definition, assignment to and from API fields, and SQL IO statements in the generated RPGLE.

The common OAPI extensions for all REST Methods are the x-PFname and x-library and x-column extensions.

Extension Name	Parent Object	Description
x-pfname	Schema Object Name	Contains the name of the DB2 Table or Physical File Name
x-library	Schema Object Name	Library where the DB2 Table or Physical File exists

x-column	schema field name	Column name in the DB2 Table or Physical File
x-key	schema field name	Column name in the DB2 Table or Physical File used as a unique key - only required for PATCH and DELETE Method APIs

When a GET method API Provider is being generated, if the x-pfname, x-library and x-column extensions are specified in the schema associated with the response object, the generator will build the necessary SQL statements and associated assignment and definition code in the generated RPGLE. This facilitates the necessary SQL IO and assignments to API Response fields without any additional coding.

4.1 Example - SQL IO Extensions

```
"responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/LXCLIENTMDL"
                }
              }
            }
          },...

"components": {
    "schemas": {
      "LXCLIENTMDL": {
        "x-library": "MDRST",
        "x-pfname": "LXCLIENT",
        "properties": {
          "id": {
            "type": "integer",
            "description": "Description of id",
            "maxLength": 9,
            "x-column": "id"
          },.....
```

5 SQL IO & POST, PUT & PATCH Provider

Specify the DB IO action for the incoming requestBody of a POST & PUT Provider API.

When a swagger schema defined in the requestBody object has the x-pfname, x-library and x-column extensions added, the generator will build the necessary SQL statements and associated assignment and definition code in the generated RPGLE, to facilitate the necessary SQL IO - insert or set.

This is facilitated using the extension x-ioAction, which is added as an object to the requestBody object of the swagger def. The four options available are “none”, “read”, “insert” and “assign”.

5.1 Example - SQL POST, PUT

```
“requestBody": {
          "description": "Post Method",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/LXCLIENTMDL"
              }
            }
          },
          "x-ioAction": "assign"
```

```
        }},.....
```

“none” creates only the field and variable definitions in the RPGLE only **“read”** is used for a POST Provider that only reads the record from the DB file. It uses the requestBody fields as the parameters, making the assumption that they are all parameters. **“assign”** is used to make the generator build the data definitions and assignment statements only in RPGLE - no SQL/IO **“insert”** is used to build all the necessary code (refs/assignments/IO) to insert records specified in the x-pfname in the schema.

Specifying data record key fields for patch and put Provider using x-key: True A Provider that uses PUT or PATCH methods that also has x-pfname defined in the schema needs to know the key fields to use when checking if the record exists first. x-key has been added as an extension to the properties of a schema object (same level as x-column) ### Example - SQL PATCH, DELETE

```
"TICLIENTMDL": {
  "x-library": "MDRTUTLIB",
  "x-pfname": "LXCLIENT",
  "properties": {
   "id": {
    "type": "integer",
    "description": "Description of id",
    "maxLength": 9,
    "x-column": "id",
    "x-key": "true"
   },.....
```

6 SQL IO Consumer

Specify the DB IO action using x-rspAction for GET, POST, PATCH, PUT CONSUMER The response received by a consumer can be parsed and inserted to the database defined in x-pfname. to activate this add the extension object x-rspAction to an item in the response object (usually the 200 response)

6.1 Example - SQL SQL IO Consumer

```
"responses": {
        "200": {
          "description": "OK",
          "x-rspAction": "assign",
          "content": {
            "application/json"
```

Further Information & Support:

For information about support, training, education and customisation services, please contact us at:

www.midrangedynamics.com

support.midrangedynamics.com

An online Tutorial of the definition of the REST Architectural Style:
https://www.restapitutorial.com/index.html

Here is Roy Fielding's thesis that started it all. It's long but very interesting:
https://www.ics.uci.edu/~fielding/pubs/dissertation/software_arch.htm

Here is a short set of YouTube Videos by a young guy named Andy Balaam who makes it all make sense from a programmer's POV. He uses Python in his examples but keeps it simple enough to be understood by anyone with programming skills.
https://www.youtube.com/playlist?list=PLgyU3jNA6VjSKjhzhVcYH6MQeFGwptXfQ

Here is another YouTube playlist that includes a course that was recorded and published. I found this very useful as it covers a lot of the legacy architectures and their evolution to REST. It also has plenty of details about the relevant parts of REST and also its implementation technologies. Easy to follow and understand:
https://www.youtube.com/playlist?list=PL6pNVRirQTlRMvKU4tqHRVWTD0x9y1sjO