MDRest4i User Manual

MDRest4i REST on IBM i Simplified

Midrange Dynamics

1. MDRest4i v14

1.1 MDRest4i Documentation

About This Guide

This manual is comprised of these sections:

- Installation
- Product Overview
- SDK Web UI Help
- Coding Guide
- MDRest4i Commands

You can download a **PDF version of this guide here**.

Getting Started

Here are a few steps to get you started and up and running with MDRest4i :-

- Download the Product.
- Read the Product Overviews.
- Install the Product Library on IBM i.
- Install the SDK Web UI
- Login to the SDK - http://youribmiserver:yourSDKport/cons/
- Create a Provider or consumer program

Useful Shortcuts

:octicons-browser-24: **SDK Web UI**

:octicons-file-code-24: **Coding**

:octicons-share-android-24: **Provider-Overview**

:octicons-git-pull-request-24: **Consumer-Overview**

:octicons-tracked-by-closed-completed-24: **Quick Install**

:octicons-bug-24: **Troubleshooting**

:octicons-download-24: **Downloads**

:octicons-versions-24: **Previous Versions**

1.2 Installation

1.2.1 MDRest4i 14.0 Installation Instructions

This installation guide will take you through the installation of the V14 MDRest4i MDRFRAME framework component, and the MDRest4i SDK web application on the IBM i.

> **Please Note**
>
> Please get license keys from Midrange Dynamics before continuing!

BRIEF DESCRIPTION OF INSTALLATION

The installation consists of installing the MDRST library using command from the MDRST.savf save file on the IBM i.

Apply the license key using command MDRST/MRLICKEY.

The MDRST library contains the **MDRDK** save file, which contains the MDRest4i SDK web UI web application.

If the SDK is required on that LPAR, the next step is to run the MDRST/MDRSDKINS command. This restores, configures, and adds an HTTP server instance for the MDRest4i SDK web UI, creates a default Provider and Consumer program library, and creates a default HTTP server instance to expose Providers(API's)

If additional REST API servers are required on this LPAR or to setup an LPAR for running REST Providers and Consumers only(not developing), create an API server instance using command **MDRHTTPAPI**

> **Please Note**
>
> This installation is for V14 upwards, and cannot be used to automatically upgrade from V12 and earlier.

> **Which Product to Install?**
>
> MDRest4i **MDRFRAME** is required on **all** LPAR's where any MDRest4i API or consumer runs, including the development LPAR where the SDK will be installed.
> MDRest4i **SDK** is only required on the LPAR where API or Consumer development takes place.

> **Updating Products**
>
> Command MDRST/MDREST4INS is used for both **installing** AND **updating** MDRest4i. Command MDRST/MDRSDKUPD is used for **updating** the SDK (after MDRST/MDREST4INS has completed)

SIMPLIFIED INSTALLATION STEPS

The following is a simplified version of the installation steps, assumes you have already installed the MDREST4iNS command, and you have your MDRest4i License keys.

Step	Comments
Login to your IBM i with a user that has QSECOFR Authority	
Download zip and upload MDRST.SAVF onto IBM i	MDREST4INS command should still be in QGPL so you can skip that
Stop the MDRST SDK HTTP server and any HTTP API server you have created :	`ENDTCPSVR SERVER(*HTTP) HTTPSVR(MDRST) ENDTCPSVR` `ENDTCPSVR SERVER(*HTTP) HTTPSVR(MDRSTAPI) ENDTCPSVR`
Make sure lib MDRST not locked and in use	`WRKOBJLCK OBJ(MDRST) OBJTYPE(*LIB)`
Prompt Command `MDREST4INS`	Prompt command **MDREST4INS** to specify a different backup lib or different instance name. **MDREST4INS** is also used to update the server applications.
`ADDLIBLE MDRST`	This lib will now be at the lasted build
Enter command `MRLICKEY`	Use command **MRLICKEY**Review the license status and the correct build and version number at the bottom. Press enter when complete
Prompt command `MDRSDKINS`	**MDRSDKINS** is only required if SDK is being installed. This will install the SDK web application server **MDRSDKUPD** is only required if SDK is being updated. This will update the SDK web application server code and copy back the existing settings.
Restart your SDK and API servers	`STRTCPSVR SERVER(*HTTP) HTTPSVR(MDRST)`
Test the server is running	`http://[your serverip or host]:[yourport]/mdrapi/mdrhello?firstName=Mike&lastName=Smith&Title=Mr`

DETAILED STEP-BY-STEP INSTALLATION

For first time installations, kindly select the following installation pages in sequence:

1. Pre-Requisites
2. Objects Created by Installation
3. MDRest4i MDRFRAME Installation
4. MDRest4i SDK Installation
5. Create API Server Instance

1.2.2 Step by Step Installation

Pre-Requisites

Operating System

IBM i (AS/400) System with OS/400 Release V7R2M0 or higher

The current version can be checked using command WRKLICINF

PTFs

- Latest cumulative PTF's for TCPIP/IP
- Latest cumulative PTF's for IBM HTTP Server for i

MRLICKEY License key command

In order to use the products, a valid License Key is required for the core product MDRest4i, which is based on the Serial Number of the system, and the version of the Product.

MDREST4i uses a license key system independent from MDCMS. So anylicense keys or installations for MDREST4i prior to May 21, 2020 will need to be reentered into MDRST4i using prompted command `MDRST/MRLICKEY LTYP(\*INT)` after the new version is installed.

Please provide Midrange Dynamics or a reseller with the serial number (obtained with command WRKLICINF or DSPSYSVAL QSRLNBR) and product version and continue with the installation once the key or keys have been provided.

The following screen will appear when using `MDRST/MRLICKEY LTYP(\*INT)`:

```
                        Modify MDRest4i License Keys


Enter the new License Key(s) and press Enter.

 Product       License Key          Status
 MDRest4i      ____ ____ ____ ____

                           System Serial Number: 78656EX
                                    Partition#: 007
                                    Version ID: 14

F3=Exit
```

Paste in the key provided in the email received from Midrange Dynamics support, and press enter

If the keys are provided in file MDLICENSE.savf, include this file in the same IFS path or library as the product save files. When MDLICENSE.savf is included, the keys for the given serial number will be automatically applied to the product instance, and the keys for all other included systems will be stored in the product instance so that if a switch is made to a backup system, the backup keys will automatically be applied to the product, thus avoiding any delays.

Note: License keys obtained as a save file (SAVF) from the mdlicense.mdcms.ch website will sometimes have additional data in the save file name. for example: “MDLICENSE_20220829_160955_411724.savf” Before running MRLICKEY to install this save file, rename it so the name of the save file is “MDLICENSE.savf”

User Authorization

The user performing the installation must have *SECADM and *ALLOBJ authority on the system.

Access to the Digital Certificate Manager (DCM) is required for using SSL for REST APIs or Consumers and SOAP Services or Consumers

Adopted Authority

The system value **QALWOBJRST** must include the choice of ***ALL** or ***ALWPGMADP** so that the product programs with the adopt attribute can be restored.

Disk Space

The product initially requires about 205 MB of space at installation time.

MDREST4i history typically requires an additional 5 MB per year of use. This is subject to the logging details selected in each program generated.

Product Library Locks when Upgrading

When upgrading an existing version of the MD Products to a new build, the existing library instances for MDRST, may not be in use.

Object locks can be checked by using command:

```
WRKOBJLCK OBJ(MDRSTxxxx) OBJTYPE(*LIB).
```

Where “xxx” is the instance extension – blank if the default MDRST was used.

Please also check any http instances that use MDRST instance libraries in the ScriptAlias, ScriptAliasMatch, **SetEnv QIBM_CGI_LIBRARY_LIST** or job descriptions (mdrst/mdrst) setting in the HTTP server instance configuration. These server instances should be stopped until after the installation.

If locks exist, you can cleanly end the jobs ahead of time, or specify parameter END(***YES**) on the MDREST4INS command to automatically end all jobs locking the product libraries.

Installed Objects

OBJECTS CREATED BY INSTALLATION PROCESS

QGPL Objects

The following commands are placed by default into library QGPL:

MDREST4INS – The MDREST4i Product Installer

For each command, a corresponding program and panel group are also placed in QGPL.

Libraries

Some or all of the following libraries will be created, depending on the product:

Library	Description	Initial Size
MDRST	MDRest4i objects	205 MB

MDRST contain some site-specific data, so they should be regularly backed up.

By default, the libraries are named as stated in the table. At installation time,
a 1-4 character Instance ID (ENV) can be defined which will be used as the suffix for the library names. For example, suffix X would mean that product MDRST would be stored in library MDRSTX.

This way, several instances of the MDREST4i products can reside on the same system.

User Profiles

A user profile is created to own the objects in the product libraries. The profile is created without the ability for users to sign on or otherwise make use of the profile. By default, the name of this user profile is MDOWNER, but can be changed at installation time by specifying a different value for MDREST4INS parameter OWN.

If the product owner profile already exists, it is left as is.

MDRST4i programs use adopted authority from the owner profile, which has *ALLOBJ authority, so that the actual users authorized to perform functions in MDREST4i do not need to have any special system authorities to accomplish the task of making changes to your business applications.

Any programs providing access to a command line do not have adopted (*OWNER) authority.

None of the programs have parameter “Use adopted authority” set to *YES, ensuring that authority won’t be inherited from your internal calling programs.

MDRFRAME Installation

> **Tip: Determine your installed version of MDRest4i**
>
> To determine the version you have installed use the following command: `DSPDTAARA MDRSTxxxx/MRVERSION` where xxxx is the instance name. Leave as MDRST if default instance is installed

> **Warning**
>
> This installation is for V14 upwards, and cannot be used to automatically upgrade from V12 and earlier.

> **Please Note**
>
> The value of MRVERSION will be displayed as 8.2.8 for v11 of MDREST4i.

THE INSTALLATION STEPS

Downloading and extracting the Save Files

1. Sign into the Midrange Dynamics Service Desk portal at https://support.mdcms.ch/plugins/servlet/desk

   You will need to be registered to use the portal. If not yet registered, request registration from: https://www.midrangedynamics.com/request-access-to-md-service-desk/
2. Proceed to the Downloads section. From here click on MDRest4i V14
3. Select a mirror and download the zip file
4. Save the zip file to a local directory on your PC.
5. Extract the save files to a local directory on your PC.

Option 1 - Installing from Save Files in IFS

1. Copy the save files to an IFS folder on your IBM i system.

> **Tip: Copying files to IBM i IFS**
>
> One of the easiest ways to copy the save files to an IFS folder on your IBM i is to drag and drop them using IBM's **System i Navigator**.
>
> You can also use FTP. A recommended FTP client is **FileZilla**, which is available for free from the internet. Or you can run FTP in a command or PowerShell/Terminal window on PC or MAC

1. If command **MDREST4INS** already exists in an IBM i library on your system, and it was created since version 8.2.5, build May 20, 2020, skip to section Invoke the MDREST4INS Command
2. Enter command:

   ```
   CRTSAVF QTEMP/MDREST4INS
   ```
3. Enter command:

   ```
   CPYFRMSTMF FROMSTMF('/x/mdrest4ins.savf') TOMBR('/qsys.lib/qtemp.lib/ mdrest4ins.file') MBROPT(\*REPLACE)
   ```

   Where **x** is the name of the IFS folder containing the save files
4. Enter command:

   ```
   RSTOBJ OBJ(\*ALL) SAVLIB(QGPL) DEV(\*SAVF) SAVF(QTEMP/MDREST4INS)
   ```

   The objects may be restored to a different library than QGPL, if desired. However, the CHGCMD command will need to be used on the **MDREST4INS** command to change the library for the program and panel group.

Option 2 - Installing from Save Files in a Library

1. Copy the save files to a Library on your IBM i system.
2. If command MDREST4INS already exists in an IBM i library on your system, and it was created since version 8.2.5, build May 20, 2020, skip to section Invoke the MDREST4INS Command
3. Enter command:

```
RSTOBJ OBJ(\*ALL) SAVLIB(QGPL) DEV(\*SAVF) SAVF(x/MDREST4INS)
```

Where **x** is the name of the library containing the save files

The objects may be restored to a different library than QGPL, if desired.

Invoke the MDREST4INS Command

The MDREST4INS command is used for both first time installs, and upgrades of MDRest4i. it also installs/updates the MDRDK save file into the MDRST library which contains the MDRest4i SDK software.

Enter command **MDREST4INS** and press F4 to review command parameters and make any necessary changes (press F1 or use the parameter table on next page for more information).

> **Note**
>
> Installations may occur in an interactive or batch job. Interactive is recommended for new installations for improved monitoring and prompting of the process.

MDREST4INS Command Screen:

```
                      Install MDRest4i (MDREST4INS)

Type choices, press Enter.

Save File Location Type  . . . .   *IFS          *IFS, *LIB
IFS-Path containing Save Files     '/MDREST4i'

Library containing Save Files  .                 Character value
Product Instance . . . . . . . .   *DFT          *DFT, Product Library Instanc
Copy Data from Instance  . . . .   *SAME         *SAME, *DFT, *NONE, Instance
Backup Library Suffix  . . . . .   '_BU'         Suffix for Backup Libs
Replace existing Backup Libs . .   *YES          *YES, *NO
Install in ASP Device  . . . . .   *SYSBAS       *SYSBAS, Device Name
End Jobs locking Product Lib . .   *NO           *NO, *YES
Product Object Owner . . . . . .   MDOWNER       User Profile

                                                                      Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

The default values typically used to install MDREST4i standalone are as follows:

MDREST4INS Parameter Table | Parameter | Label | Description | | :--- | :--- | :--- | | LTYP | Save File Location Type | Specifies if the product save files are in an IFS folder or in a library.
***IFS** – The save files are located in an IFS folder. Specify the full path of the folder in parameter PATH
***LIB** – The save files are located in a library. Specify the name of the library in parameter LIB | | PATH | IFS-Path containing Save Files | Specifies the IFS path containing the save files used to install the Midrange Dynamics products. The contents of the path must contain the MDRST.savf save file. | | LIB | Library containing Save Files | Specifies the Library containing the save files used to install the Midrange Dynamics products. The contents of the library must contain the MDRST.savf save file. | | ENV | Product Instance | Specifies the instance ID of the product. The instance value is appended to the MDRST product library in order to allow multiple instances of MDRest4i on the same partition.

***DFT** – A suffix is not appended to the library name. The product library will be named **MDRST**.

The suffix to be used. Up to 4 characters are allowed and each character must be accepted as part of a library name. Example **ENV=T11** so the name of the installed product library becomes **MDRSTT11** | | FENV | Copy Data from Instance | Specifies the location of an instance of the products, if it
exists, that should be used for the copy of the data to the new

version.

***SAME** – The same instance, or library suffix, that was defined in parameter ENV.

***DFT** – Copy the data from the default instance, which doesn't contain a suffix.

***NONE** – Do not copy any data to the new version. The new version will be a clean installation.

Specify the product instance, or library suffix, containing the data to be copied to the new version. | | BSFX | Backup Library Suffix | Specifies the suffix to be appended to the libraries containing the version to be replaced by this installation, if a prior version exists for the instance defined in parameter ENV. This way, the prior version isn't lost and can be reactivated by renaming the libraries from the backup suffix to the instance suffix. | | BREP | Replace existing Backup Libs | Specifies if existing backup libraries using the same suffix as defined in parameter BSFX should be automatically replaced.

***YES** – Any existing libraries with the same name as the new Backup libraries will be automatically replaced.

***NO** – Libraries with the same names as the new Backup libraries will not be replaced, and the installation will not occur if any exist. | | ASPD | Install in ASP Device | Specifies the auxiliary storage pool (ASP) device to which the product libraries should be restored.

***SYSBAS** – The product libraries will be restored to the base system iASP.

character-value – the product libraries will be restored to the indicated iASP device. | | END | End Jobs locking Product Libs | Specifies if all jobs that have a lock on one of the product libraries should automatically be ended immediately.

***NO** – If a lock exists for one of the libraries, the installation process will be cancelled.

***YES** – Each job that has a lock on one of the product libraries will be ended immediately. If ending fails for a job, the installation process will be cancelled. | | OWN | Product Object Owner | Specifies the user profile to be used to own the objects in the MD product libraries and IFS folders. If the profile doesn't exist yet, the installer will create it. The profile should have ***ALLOBJ**, ***JOBCTL**, and ***SPLCTL** special authorities to ensure that deployments function correctly. |

Using MDCMS to Install MDRest4i

If using MDREST4INS to upgrade an existing product to a newer version, the distribution and installation can occur as part of an RFP using MDCMS. In this case ensure the following in MDCMS:

- MDREST4INS should be a Post-Installation (3) attribute command attached to the *IFS or *FILE attribute defined with the IFS path or save file library as the target object library.
- Run for Modifications = Y
- Keep MD Libs in Libl = N
- Frequency = R
- MDREST4INS command should begin with SBMJOB so that it runs separately from the RFP
- END(*YES) to terminate any locks
- DLY(30) to allow RFP time to finish before starting with installation
- USER for SBMJOB should be a profile with *SECADM authorit00
- INLLIBL for SBMJOB should include only QTEMP and QGPL (or library where MDREST4INS command exists)

Otherwise continue as per the following example command definition:

```
Appl.....: MD Run for Modifications: Y Y/N
Lvl......: 50 Recompiles...: N Y/N
Attribute: INSTALL Attribute, \*RFP Deletes......: N Y/N
Updates......: N Y/N
Type.....: 3 Post-Installation Ignore Errors........: Y Y/N
```

```
Sequence.: 1 Keep MD Libs in Libl.: N Y/N
Frequency: R O=Object, R=RFP
```

Command:

```
SBMJOB CMD(MDREST4INS PATH('##OBJLIB##') END(\*YES) DLY(30)) JOB(MRINSTALL) USER(QSO) INLLIBL(QTEMP QGPL)
```

Default API and Consumer logs folder

The examples provided by MDREST4i and the default generator options, store logs and uploaded files in folders on the IFS. These folders is created automatically by the installer. Each instance installed will add instance sub-folders to these folders.

The folders created are:

/MDREST4i/**logs**/<instances>
/MDREST4i/**attachments**/<instances>
/MDREST4i/**json**/<instances>
/MDREST4i/**sdk**/<instances>
/MDREST4i/**sdklogs**/<instances>

Installing or Updating the MDRest4i SDK

The MDRest4i SDK web interface runs as a web application served by an HTTP server on IBM i. The MDREST4i SDK web application uses REST APIs (Built using MDREST4i) which are exposed via the **same instance** of this Apache HTTP server.

To **install** SDK web UI application, run the MDRSDKINS command

To **update** an existing installation, first stop the MDRST server instance and any server instance created over MDRST for running your API's, and then run the MDRSDKUPD command.

Invoking the MDRSDKINS command

- `ADDLIBLE MDRST`
- Prompt `MDRSDKINS`

The following screen will appear:

```
                     Install MDRest4i SDK (MDRSDKINS)

Type choices, press Enter.

SDK HTTP PORT  . . . . . . . . . PORT           4545
Start SDK HTTP Instance  . . . . STRSRV         Y
Default SDK Administrator  . . . ADMPRF       > DFTPROFILE
Default Generator Library  . . . DFTLIB         MDRAPIDFT
Default Hostname or IP . . . . . DFTHOST        'ibmi.yourdomain.com'

Create Default API Server  . . . CRTSRVA        Y
Default API Server . . . . . . . APISRV         MDRAPISRV
Default API Server PORT  . . . . PORTA          4546


                                                                        Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel
```

MDRSDKINS Parameter Table

PORT	HTTP Instance PORT Number	Port number the HTTP server instance for the SDK web UI runs under. 4545 is a default value and should be set as per your own port preferences on the IBM i server
STRSRV	Start HTTP Instance	Select **Y** to start the SDK HTTP server instance.
ADMPRF	Default SDK Administrator	MDRest4i SDK uses hierarchal authority classes for users. An administrator user must be created when installing MDRest4i SDK, so additional users can be authorized. The value supplied in this parameter MUST be a valid, existing IBM i user, that can login to the IBM i. No special IBM i authority is required for this user. This default Admin user has authority to Manage SDK Users, including authorizing new users.
DFTLIB	Default Generator Library	A default library to contain generated Provider and Consumer programs and their source. MDRSDKINS creates this library if it does not exist. DFTLIB is used when creating the **Default API Server** below as one of the libraries in the JOBD for this default API Server. DFTLIB is also used for setting default values, for default the SDK user profile. See Default User MDRDFTUSR below for more details.
DFTHOST	Default Hostname or IP	This is the host name or IP address of the IBM i server, where MDRest4i Providers will run from. It is used to to add the DFT Server value for new users. See Default User MDRDFTUSR below for more details.
CRTSRVA	Create Default API Server	Specify **Y** to create an HTTP instance to host REST Providers created by MDRest4i SDK. MDRSDKINS calls the MDRHTTPAPI command to create this HTTP server instance. MDRSDKINS will start the Default API Server after installation completes.
APISRV	Default API Server	The name of the HTTP instance created as a default Provider(API) server. A job description of the same name will also be created in MDRST, and used by this HTTP instance.
PORTA	Default API Server PORT	Port number that the Default API Server created above will listen on.

Default User MDRDFTUSR

A dummy user: **MDRDFTUSR** is created by MDRSDKINS.

When a new SDK user signs up using the Signup link in the SDK Login page, this MDRDFTUSR record is copied to create the new user profile record, in file MDRST/MDRDUSER.

Default values are set in this dummy profile by MDRSDKINS. Individual users default values can be edited later from the Edit User screen in the SDK UI. The default values set by MDRSDKINS are:

- The default value for Object Library is set with the **DFTLIB** value.
- The default Source File for code generation is set as MDRDFTSRC using the library specified in **DFTLIB** above. For example: DFTLIB = MYAPILIB. The default source file value will be set as `/QSYS.LIB/MYAPILIB.LIB/MDRDFTSRC.FILE`
- The default Server URL value added to the OAPI/SWAGGER definition when a new Provider or Consumer is created in the SDK UI.

See the SDK Edit Profile for editing the default values set for a user during Signup.

The default values for MDRDFTUSR can be edited at a later date, using the MDRSDKUSR command.

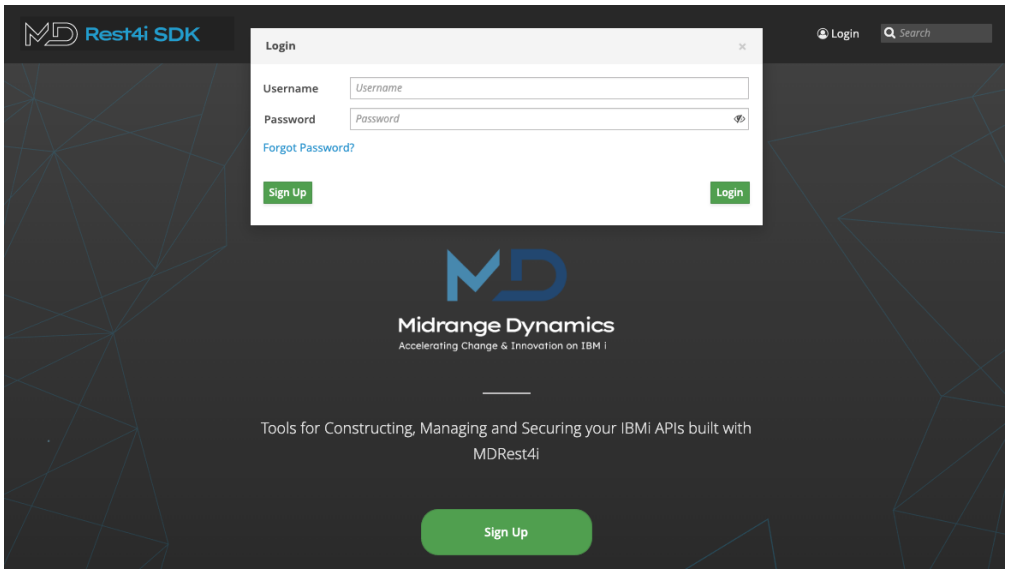
> **Note**
>
> The Owner of the SDK web UI objects in the IFS is set the same owner used to install the product via the MDREST4INS command.

Running MDRest4i SDK Web UI

Use the following url to connect to the web UI

HTTP://youribmiserver:yourport/cons/

This will bring up the following interface:



Login with user profile provided in the MDRSDKIN.ADMPRF command parameter above

Once logged in, click on any “?” in the UI to bring up the online help

Updating the SDK Web Application

The **MDRSDKUPD** command has no parameters. It only executes if the SDK has already been installed. It backs up the existing config files for the MDRest4i SDK web UI, installs the new code base, and copies back the SDK web UI config files.

> **Warning**
>
> If unsure if the SDK has already been installed, check the value of data area **MDRST/MDRSDKROOT**. If this is blank, then command MDRSDKINS MUST be run, not MDRSDKUPD

MDRSDKUPD backs up the SDK web UI application to IFS folder: /mdrest4i/sdkbu-mdrst

> **Note**
>
> The backup path is based uon the instance name for MDRest4i. If the MDRest4i instance is 'V14' for example, the backup folder for te SDK update will be: /mdrest4i/sdkbu-mdrst

To run command MDRSDKUPD:

- `ADDLIBLE MDRST`
- `MDRSDKUPD` command and enter

API Server Setup

CREATING AN HTTP SERVER FOR YOUR API'S

REST APIs are exposed via an IBM i Apache server instance. By default command MDRSDKINS installs a default server for your Provider and Consumer programs.

> **Note**
>
> The default API server created by the SDK installation can be used as necessary. Addiotnal API servers can easily be installed for production, or other testing environments. There is virtually no limit to the number of API servers that can be added in the same MDRest4i instance. The job description and configuration required in these instances, can be created on the IBM i using the command **MDRHTTPAPI**

1.2.3 Troubleshooting

Troubleshooting the HTTP Servers

Please Note: In the sections below, if you are using any instance name other than default for MDRest4i, MDRST should be replaced with MDRSTxxxx where, xxxx is the instance name

If the SDK web UI wont load, or an API wont respond, it may be caused by a server configuration problem.

Please check the following areas.

IS THE SERVER RUNNING

Use the following command to see if the server jobs are started:

```
WRKACTJOB SBS(QHTTPSVR) JOB(MDRST)
```

where "MDRST" is the library where MDrest4i MDRFRAME is installed.

If it isn't running, use the following command to start the server

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(MDRST)
```

Then check to see if the jobs are active. If they are test the web ui again

SDK SERVER CONFIG - HTTPD.CONF

If the server wont start of you cannot connect it may be the server configuration.

View the HTTP config file here: /www/mdrst/conf/httpd.conf file (either from the HTTPAdmin console or the IFS in 5250 screens/ VSCode). It should look like this:

```
# MDRest4i: Created by MDRHTTPSDK on 2024-02-28-16.43.16
Listen *:2519
DocumentRoot /www/mdrst/htdocs
TraceEnable Off
LogFormat "%h %T %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%{Cookie}n \"%r\" %t" cookie
LogFormat "%{User-agent}i" agent
LogFormat "%{Referer}i -> %U" referer
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log combined
LogMaint logs/access_log 30 0
LogMaint logs/error_log 30 0
SetEnvIf "User-Agent" "Mozilla/2" nokeepalive
SetEnvIf "User-Agent" "JDK/1\.0" force-response-1.0
SetEnvIf "User-Agent" "Java/1\.0" force-response-1.0
SetEnvIf "User-Agent" "RealPlayer 4\.0" force-response-1.0
SetEnvIf "User-Agent" "MSIE 4\.0b2;" nokeepalive
SetEnvIf "User-Agent" "MSIE 4\.0b2;" force-response-1.0
<Directory />
 Require all denied
</Directory>
<Directory /www/mdrst/htdocs>
  Require all denied
</Directory>

### MDRest4i: Config Start
HTTPStartJobDesc mdrst/mdrst
SetEnv QIBM_CGI_CHANGE_CURDIR N
SetEnv QIBM_CGI_CHANGE_CURLIB N
## MDREST4i: The line below allows authorization tokens to be passed through to the CGI program
SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1

## MDRest4i: CORS header settings
Header Set Access-Control-Allow-Origin "*"
Header Set Access-Control-Allow-Headers "AUTHORIZATION,ORIGIN,CONTENT-TYPE,ACCEPT"
Header set Access-Control-Allow-Methods "POST,GET,OPTIONS,DELETE,PUT"

## MDRest4i: API Script Alias Settings
ScriptAliasMatch mdrsdk /QSYS.LIB/mdrst.LIB/MDRSDK.PGM
ScriptAliasMatch mdrapi /QSYS.LIB/mdrst.LIB/MDRAPI.PGM

## MDRest4i: SDK WEB UI Alias Settings
AliasMatch /wiki/(.*) /www/mdrst/wiki/$1
AliasMatch /cons/(.*) /www/mdrst/cons/$1
AliasMatch /docu/(.*) /www/mdrst/docu/$1

## MDRest4i: REST API Library
<Directory /QSYS.LIB/mdrst.LIB>
```

```
 Require all granted
 SetEnv MDREST4I_RESOURCE_COMPONENT 3
 SetEnv MDREST4I_SDK_CONFIG_PATH "/www/mdrst/cons/mdrsdk_Templates.json"
 SetEnv MDREST4I_SDK_INITIALIZE_DEFAULTS *NO
 DefaultNetCCSID 1208
 CgiConvMode binary
 ServerUserID mdowner
</Directory>

## MDRest4i: Console Config
<Directory /www/mdrst/cons>
 <FilesMatch "index.html|startup.json|build.json">
  FileEtag None
  Header Unset ETag
  Header Set Cache-Control "max-age=0, no-store, no-cache, must-revalidate"
  Header Set Pragma "no-cache"
  Header Set Expires "Thu, 1 Jan 1970 00:00:00 GMT"
 </FilesMatch>
 Header set Access-Control-Allow-Origin "*"
 Options +FollowSymLinks -MultiViews
 DirectoryIndex index.html
 RewriteEngine On
 RewriteRule mdrconsoleapp_(.*)\.js mdrconsoleapp.js
 RewriteCond %{REQUEST_FILENAME} !-f
 RewriteRule artefacts/TokenManager/tokenapp_(.*)\.js artefacts/TokenManager/tokenapp.js
 RewriteCond %{REQUEST_FILENAME} !-f
 RewriteRule ^ index.html [QSA,L]
 Require all granted
 ServerUserID mdowner
</Directory>

## MDRest4i: Documenter Config
<Directory /www/mdrst/docu>
 <FilesMatch "index.html|startup.json|build.json">
  FileEtag None
  Header Unset ETag
  Header Set Cache-Control "max-age=0, no-store, no-cache, must-revalidate"
  Header Set Pragma "no-cache"
  Header Set Expires "Thu, 1 Jan 1970 00:00:00 GMT"
 </FilesMatch>
 Header set Access-Control-Allow-Origin "*"
 Options FollowSymLinks MultiViews
 DirectoryIndex index.html
 RewriteEngine On
 RewriteRule mdrdocumenterapp_(.*)\.js mdrdocumenterapp.js
 RewriteCond %{REQUEST_FILENAME} !-f
 RewriteRule ^ index.html [QSA,L]
 Require all granted
 ServerUserID mdowner
</Directory>

## MDRest4i: Wiki Docs
<Directory /www/mdrst/wiki>
 Require all granted
 ServerUserID mdowner
</Directory>
### MDRest4i: Config End
```

HTTP SERVER LOGS

Navigate to /www/mdrst/logs and view the contents of the error and/or access logs.

SPOOL FILES FOR THE SERVER JOB

If there is a crash or server startup problem, it may generate a spool file which can be found here:

`WRKSPLF SELECT(QTMHHTTP *ALL *ALL MDRSTT14)`

HTTP SERVER JOB DESCRIPTION

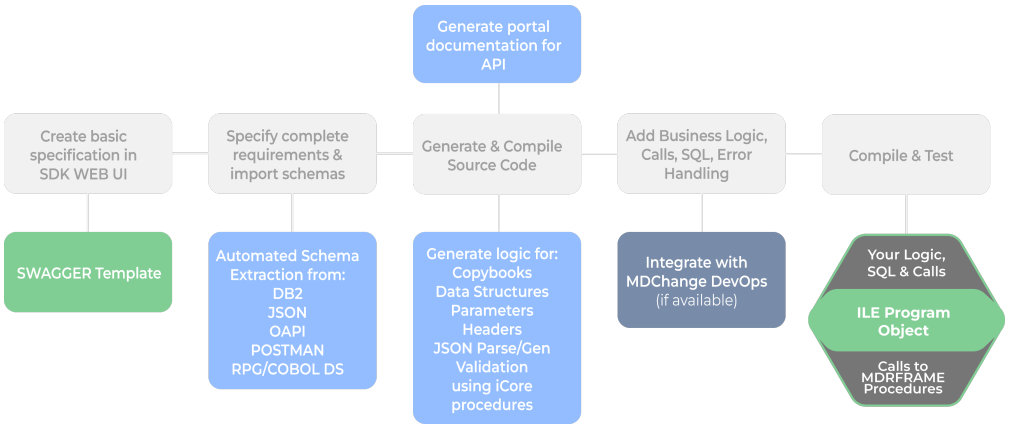
In the httpd.conf configuration file there is a directive: `HTTPStartJobDesc mdrst/mdrst`

Look at the library list part of this job description, to ensure that **MDRST** (or the library name **MDRSTxxx** consistent with the instance installed) is in the library list of this JOBD. `DSPJOBD JOBD(MDRST/MDRST)`

1.3 Concepts & Overviews

1.3.1 MDRest4i 14 Development Overview

THE MDREST4I DEVELOPMENT PROCESS



The MDRest4i SDK uses templates to create a basic OAPI (SWAGGER) specification.

The user then adds details according to requirements in the form based SDK GUI.

The SDK extracts OAPI schemas for payloads from many different types of source information(DDS).

The user then assigns these schemas to request or response payloads, and generates the RPG or COBOL code.

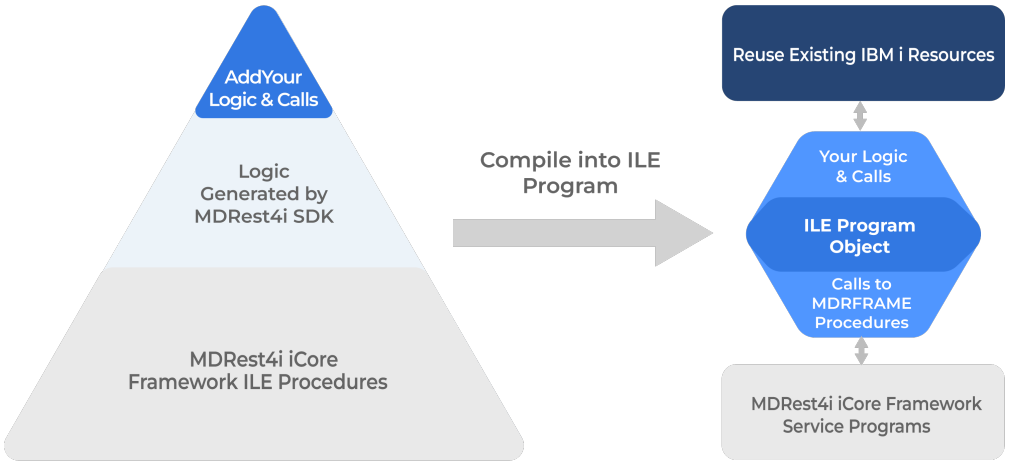
MDChange DevOPs requests can be triggered at this stage if available.

The user then manually adds their own logic to the generated programs to handle getting data to and from payloads, calls to existing assets, business logic and any additional messages. The MDRFRAME ILE framework functions can be called to customize, or add to the data and flow of the generated code.

Compile and test as normal.

THE MDREST4I CONCEPT

MDRest4i is a combination of ILE Framework and automation tools, that simplify REST API and REST client development on IBM i.

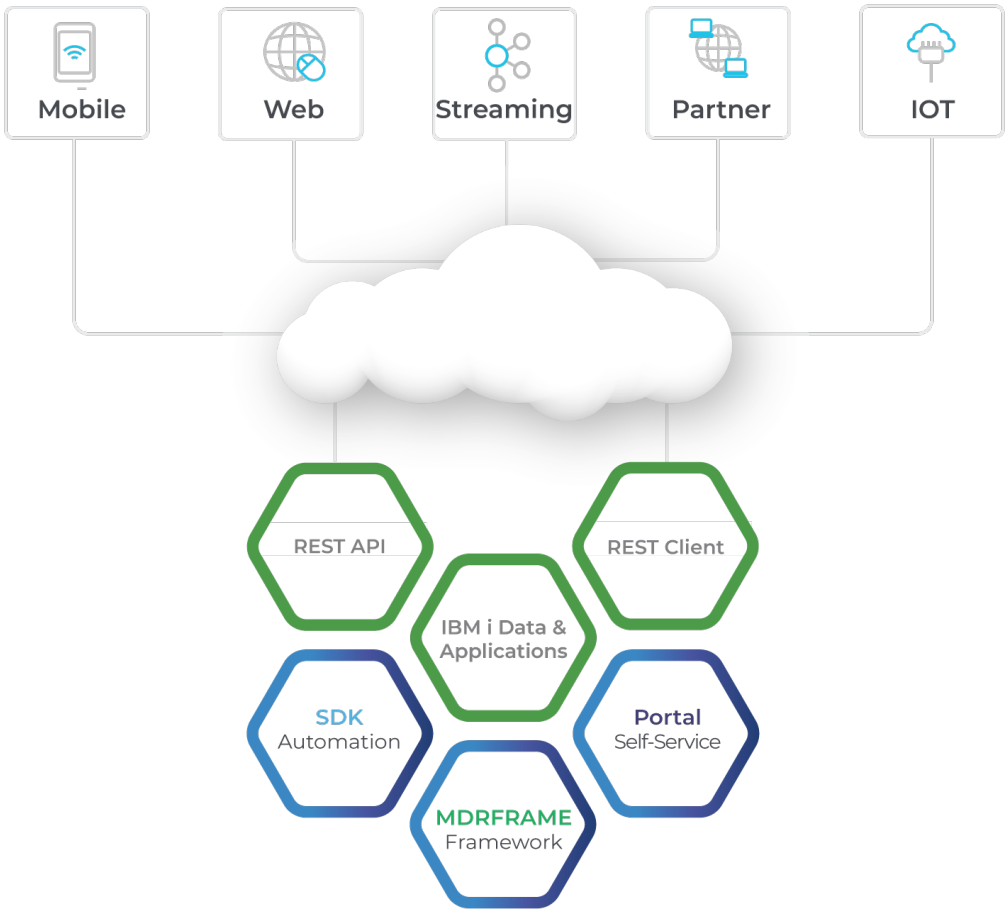


MDREST4I COMPONENTS

MDRFRAME Framework - a native IBM i ILE framework for building REST API's and REST Clients in any ILE language

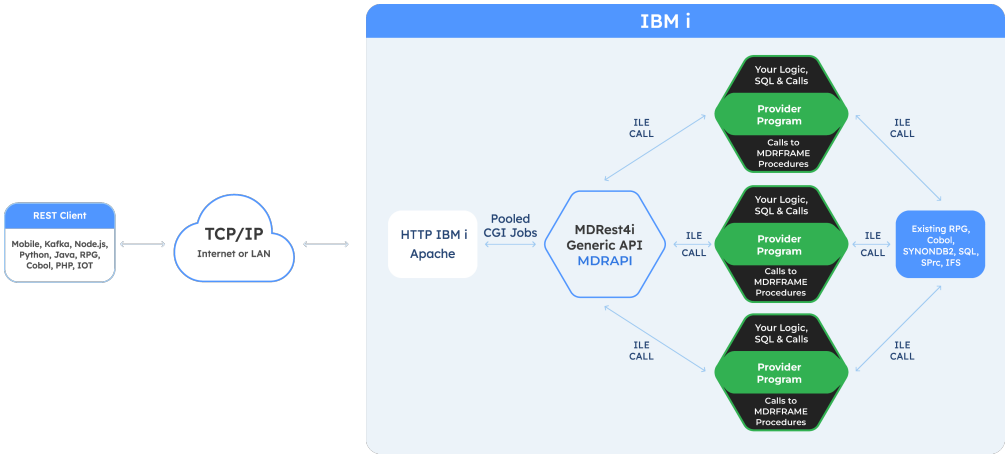
SDK - a Software Development Kit, that automates the generation of REST API and Rest Client Programs that use the MDRFRAME framework

Portal - a generator, editor and web portal for REST API documentation



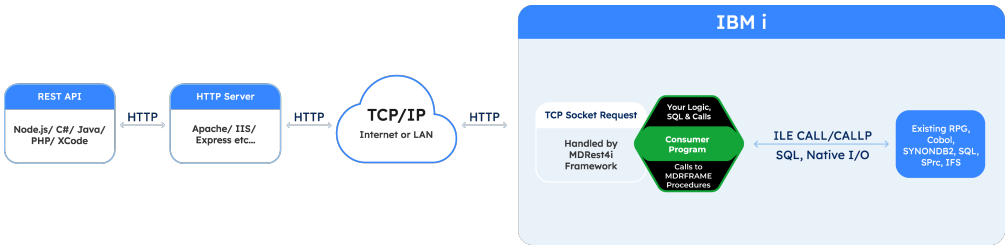
REST API RUNTIME ARCHITECTURE

REST API's are also referred to as Providers. More details available in the REST API-Provider section.



REST CLIENT RUNTIME ARCHITECTURE

REST Clients are also referred to as Consumers. More details available in the REST CLIENT-Consumer section.



USEFUL SHORTCUTS:

Provider Overview

Consumer Overview

Copybooks

Coding in COBOL

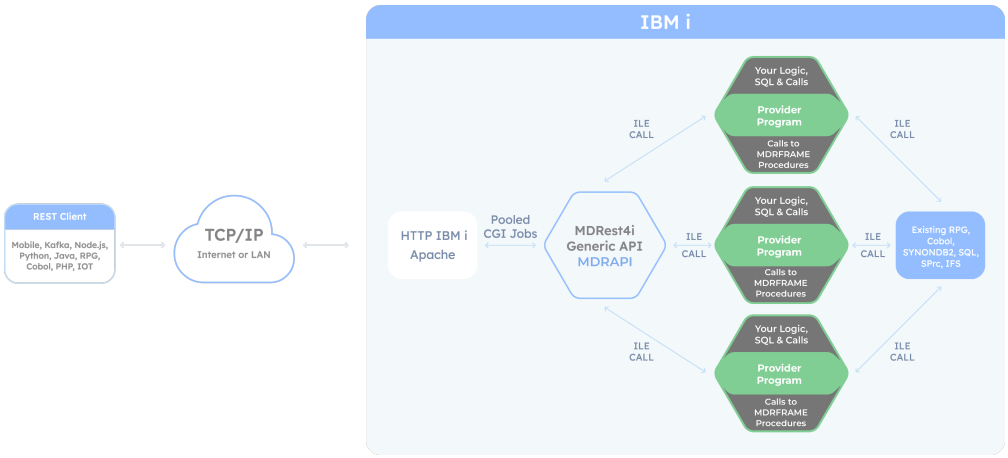
Create HTTP API Server

Troubleshooting

1.3.2 REST API - Provider

The common used term for a REST API is a Provider. A REST Client is referred to as a Consumer.

RUNTIME ARCHITECTURE



The runtime architecture a REST API/Provider is self contained on the IBM i using native components.

An HTTP server is created using the MDRST/MDRHTTPAPI command. This server instance is configured to receive requests from a REST client and call the MDRAPI controller program.

The Provider program is the program generated by the SDK and is called by MDRAPI.

MDRAPI

MDRAPI is an ILE program. It is the only program called as a CGi program by the HTTP server. Its roll is to control the flow of data between the HTTP server and the REST API program.

MDRAPI allocates a unique "handle", gathers all the inbound message data from a request, and sets this up in memory using pointers keyed by the "handle" for each request.

It then calls the REST API program(written by the developer), passing some of the inbound data(handle, method, body) in the initial call. The REST API program, then manipulates the data in memory using calls to the MDRFRAME service program procedures, passing the "handle" passed in by MDRAPI each time.

The REST API program then passes control back to MDRAPI, and MDRAPI returns the response to the REST Client request, via the HTTP server, releasing any CGi JOB resource allocation created with the initial "handle".

ScriptAliasMatch Directive

Because MDRAPI must **always** be called as a CGi script from the HTTP server instance, the minimum directive required in the httpd.conf is:

```
ScriptAliasMatch mdrapi /QSYS.LIB/MDRSTT14.LIB/MDRAPI.PGM
```

Using the above ScriptAliasMatch , these two different URIs would call the MDRSTAPI/MDRAPI CGi program: `http://yourapiserver.com/mdrapi/callme` or `http://yourapiserver.com/mdrapi/group/callmetoo`

The command MDRHTTPAPI automatically adds this directive when creating an API HTTP instance.

Custom ScriptAliasMatch

It is important therefore to either explicitly specify "mdrapi" in the uri to invoke a REST API in V14, or use your own **ScriptAliasMatch** directive in the httpd.conf file if a different value is preferred. In both cases an httpd.conf server directive must be set to ensure MDRAPI is called. Here are some custom examples:

HTTPD.conf Directive	Request URI
`ScriptAliasMatch v1 /QSYS.LIB/MDRSTV14T.LIB/MDRAPI.PGM`	`http://yourapiserver.com/v1/group/callmetoo`
`ScriptAliasMatch ^/MDCMST86/AZURE/RELEASE/CALLBACK /QSYS.LIB/MDCMST86.LIB/MDRAPI.PGM`	`http://yourapiserver.com//MDCMST86/AZURE/RELEASE/CALLBACK`

URI TO API/PROVIDER PROGRAM MAPPING

MDRAPI determines which API program to call by reading the MDRDCFG file, located in the library list of the CGi job for the HTTP server. It maps the request URI and method to the MDRDCFG two primary keys. MDRAPI first checks for the specific path and method provided in the request, and then if it doesn't find it, it looks for a the path and a blank method.

MDRDCFG Table

Column	Description
Root Resource **MDRRSC**	This is user friendly resource name used to invoke the API. MDRAPI will resolve this and call the **Program** below
Method **METHOD**	The HTTP request METHOD
Program	Name of the API program called by MDRAPI
Library	Library name used in the call. No value will cause MDRAPI to use *LIBL to make the call

Maintaining MDRDCFG Records

The command MDRST/MDRSTCFG is used to maintain records in the MDRST/MDRDCFG table.

The MDRest4i SDK Provider generator, creates the MDRDCFG records when it generates the Provider source.

MDRAPI uses two IBM i environment variables to establish what pattern to use when reading the MDRDCFG file with SQL. Both can be set globally for the system, or in the httpd.conf for the HTTP instance used to handle the providers/apis.

Example on how to set these in httpd.conf. These can be set for the entire IBM i system, site, location, directory or filesmatch contexts.:

Example for HTTPD.CONF

```
SetEnv MDREST4I_RESOURCE_COMPONENT 2
SetEnv MDREST4I_RESOURCE_MATCH 5
SetEnv MDREST4I_RESOURCE_MATCH_ONLY 1
```

MDREST4I_RESOURCE_COMPONENT

This variable tells MDRAPI which part of the URI PATH (the part after the host and port number), contains the key to use when reading the MDRDCFG file to determine which program to call.

If no value is set for MDREST4I_RESOURCE_COMPONENT, MDRAPI just uses the value of 2.

MDREST4I_RESOURCE_MATCH

This variable tells MDRAPI to behave in a different way when trying to find a record in MDRDCFG. It specifies how many of the parts of the URI PATH to use in trying to find a match in the MDRDCFG.MDRRSC column

If no value is set for MDREST4I_RESOURCE_MATCH it uses only the MDREST4I_RESOURCE_COMPONENT described above.

Provider URI Mapping Example

Lets say MDREST4I_RESOURCE_COMPONENT is 2, and MDREST4I_RESOURCE_MATCH is 5 and you have a path like this:

```
/mdrapi/two/three/four/five
```

MDRAPI will look look for the first entry in MDRDCFG by checking for the following keys (in the sequence displayed below):

```
two/three/four/five
two/three/four
two/three
two
```

If one of these has a program name found, it will stop searching and use that program (and library). If none do, it will not try to call a program named 'two' because MDREST4I_RESOURCE_MATCH is higher than MDREST4I_RESOURCE_COMPONENT.

MDREST4I_RESOURCE_MATCH_ONLY

This variable tells MDRAPI to stop further searches in MDRDCCFG if no match for the resource as dictated by MDREST4I_RESOURCE_MATCH above is found. This avoids exposing programs being called in the LIBL, that are mistakenly found by MDRAPI when the resource does NOT actually match any resource value in MDRDCFG.MDRRSC

There are two possible values 1 or on and 0 for off. By default it is switched off.

PROVIDER PROGRAM

MDRAPI calls the provider program passing three parameters:

Parameter	Description
handle	unique threadsafe memory key used by MDRFRAME to allocate and deallocate memory structures
method	the HTTP method of the request
body	the HTTP message request body (if applicable)

The general flow of the provider program is:

- Handle the extraction of inbound of REST headers, parameters, attachments, parsing payloads etc.
- Call or run your business logic, DB/IFS IO, and calls to IBM i applications/Stored procedures etc.
- Set any error messages.
- Build REST headers and payloads.
- Set the HTTP status.
- Return control to MDRAPI which sends the response via the HTTP server to the REST Client.

This data extraction of the request, and writing of the response is executed by calling the appropriate MDRFRAME procedures (found in the MDRFRAME service program).

EXAMPLES

Below is an example of a simple REST Provider program that uses a POST method. The source of these examples can be found in MDRST/EXAMPLES.

Info

In the COBOL example below the MDR-DATAGEN SECTION is used to generate JSON. This function is unique to MDRFRAME, and overcomes the limitation that IBM's DATA-GEN procedure is only available in RPGLE or from V7R2 upwards. The Copybook "MAPIC001C" below shows how the MDR_DATAGEN and MDR_DATAINTO functions use a special 'schema' to parse or generate JSON via the MDRFRAME framework. The MDR_DATAINTO-MDR_DATAGEN section has the details about how to use this.

RPGLE **COBOL** **COBOL COPY MAPIC001C**

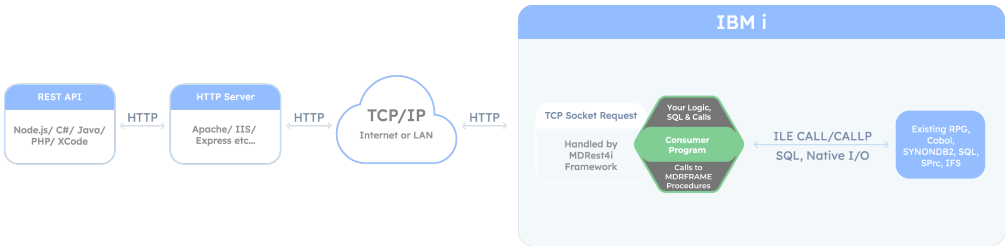
```
**free
// CRTBNDRPG PGM(&O/&ON) SRCFILE(&L/&F) OPTION(*EVENTF) DBGVIEW(*ALL) -
// USRPRF(*OWNER) TGTRLS(V7R2M0)

// RPGLE REST API Template Using DATA-GEN and DATA-INTO for JSON.
ctl-opt dftactgrp(*no) actgrp(*new);
ctl-opt BNDDIR('MDRFRAME');

/copy mdrframe

dcl-ds output qualified;
message char(50); // example only : change as required
end-ds;

dcl-ds input qualified;
message char(50); // example only : change as required
end-ds;

dcl-pi *n;
handle    like(MDR_Handle_t);
method    char(32) const;
body      varchar(500000); // MAxSize: 16000000
end-pi;

dcl-s result varchar(1000);

// Logic to process request body
MDR_genParseOptions(handle: 'document_name=input');
data-into input %data('':'')
               %parser('MDRFRAME(PARSER)':handle);

eval-corr output = input;
// Logic to process response
MDR_genParseOptions(handle: 'document_name=output');
data-gen output %data(result: '')
               %gen('MDRFRAME(GENERATOR)':handle);

*Inlr = *On;
```

```
PROCESS varchar
        apost
        nomonoprc
        nosync
        nostdtrunc

IDENTIFICATION DIVISION.
PROGRAM-ID.   MAPIC001.
AUTHOR.       Midrange Dynamics.

*>  CRTCBLMOD MODULE(&O/&ON) SRCFILE(&L/&F) DBGVIEW(&DV) -
*>            OPTION(*SOURCE *EVENTF *IMBEDERR)
*>  CRTPGM PGM(&O/&ON) MODULE(&O/&ON) BNDDIR(MDRFRAME) -
*>         ACTGRP(*NEW)
*****************************************************************
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-I.
OBJECT-COMPUTER. IBM-I.

SPECIAL-NAMES.
* MDRest4i Special Names for MDRFRAME Procedures
* Please insert any program specific SEPCIAL NAMES BEFORE this
* COPYBOOK, and the COPYBOOK delimits the divsion with a period
COPY MDRCBLSPC OF QLBLSRC.

*****************************************************************
DATA DIVISION.
*****************************************************************
WORKING-STORAGE SECTION.
* MDRest4i REQ/RSP Payload Schemas
COPY MAPIC001C OF QLBLSRC.
* MDRest4i Framework Variables
COPY MDRCBLWSC OF QLBLSRC.
* MDRest4i Large Framework Variables
COPY MDRCBLWSVS OF QLBLSRC.

* ===============================================================
LINKAGE SECTION.
COPY MDRCBLLSC OF QLBLSRC.

* MDR-HTTP-METHOD - To recive supplied HTTP method
*****************************************************************
01  MDR-HTTP-METHOD      PIC X(10).

* MDR-INPUT-DATA - To recive input JSON request.
*****************************************************************
01  MDR-INPUT-DATA.
    10 MDR-INPUT-DATA-LEN      PIC 9(9) USAGE BINARY.
    10 MDR-INPUT-DATA-BUFFER   PIC X(5000000).

* MDR-BODY-TYPE - To recive Body type ( *CAR or *VARCHAR)
*****************************************************************
01  MDR-BODY-TYPE              PIC X(10).

PROCEDURE DIVISION USING MDR-HANDLE
                         MDR-HTTP-METHOD
                         MDR-INPUT-DATA
                         MDR-BODY-TYPE.

0000-MAIN-CONTROL SECTION.

BEG.
```

1.3.3 REST Client - Consumer

A REST API is referred to as a PROVIDER. The common used term for a REST CLIENT is a CONSUMER.

RUNTIME ARCHITECTURE



CONSUMER PROGRAM

The Consumer program is the program generated by the SDK. It can be called directly or be embedded in any ILE program.

The general flow of a Consumer is:

- Create a REST Client instance in memory via the MDRFRAME framework. This "handle" key ensures threadsafe data when calling MDRFRAME procedures during the execution of the Consumer program.
- Set the request options: URL, logging, authorisation, PROXY, DCM, KDB etc see MDR_setClientOpt
- Call or run your business logic, DB/IFS IO, and calls to IBM i applications/Stored procedures etc.
- Build JSON, file, XML etc payloads.
- Set request headers and query parameters.
- Add attachments.
- Make the request.
- Handle the extraction of response headers and parsing payloads, or attachments
- Call or run your business logic, DB/IFS IO, and calls to IBM i applications/Stored procedures etc.
- Set any error messages
- Release the "handle" created by MDRFRAME and therefore release resource allocation.
- Return control to the calling program.

MDRFRAME takes care of all the socket communications, including SSL/TLS etc that occur when making the request, and receiving a response from the API/URI.

EXAMPLES

Info

In the COBOL example below the MDR-DATAGEN SECTION is used to generate JSON. This function is unique to MDRFRAME, and overcomes the limitation that IBM's DATA-GEN procedure is only available in RPGLE or from V7R2 upwards. The Copybook "MAPIC001C" below shows how the MDR_DATAGEN and MDR_DATAINTO functions use a special 'schema' to parse or generate JSON via the MDRFRAME framework. The MDR_DATAINTO-MDR_DATAGEN section has the details about how to use this.

RPGLE **COBOL** **COBOL COPYBOOK CNS001C**

```
**Free
// CRTBNDRPG PGM(&O/&ON) SRCFILE(&L/&F) OPTION(*EVENTF) DBGVIEW(*ALL) -
// USRPRF(*OWNER) TGTRLS(V7R2M0)

// RPGLE REST API Template Using DATA-GEN and DATA-INTO for JSON.
ctl-opt dftactgrp(*no) actgrp(*new);
ctl-opt BNDDIR('MDRFRAME');

/copy mdrframe

dcl-ds output qualified;
  message char(50); // example only : change as required
end-ds;

dcl-ds input qualified;
  message char(50); // example only : change as required
end-ds;

dcl-pi *n;
  handle    like(MDR_Handle_t);
  method    char(32) const;
  body      varchar(500000); // MAxSize: 16000000
end-pi;

dcl-s result varchar(1000);

// Logic to process request body
MDR_genParseOptions(handle: 'document_name=input');
data-into input %data('':'')
               %parser('MDRFRAME(PARSER)':handle);

eval-corr output = input;
// Logic to process response
MDR_genParseOptions(handle: 'document_name=output');
data-gen output %data(result: '')
               %gen('MDRFRAME(GENERATOR)':handle);

*Inlr = *On;
```

```
   PROCESS varchar
           apost
           nomonoprc
           nosync
           nostdtrunc

   IDENTIFICATION DIVISION.
   PROGRAM-ID.   CNS001.
   AUTHOR.       Midrange Dynamics.
  * Consumer program example with POST HTTP method.
  *>  CRTCBLMOD MODULE(&O/&ON) SRCFILE(&L/&F) DBGVIEW(&DV) -
  *>            OPTION(*SOURCE *EVENTF *IMBEDERR)
  *>  CRTPGM PGM(&O/&ON) MODULE(&O/&ON) BNDDIR(MDRFRAME) -
  *>         ACTGRP(*NEW)
  ****************************************************************
   ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-I.
   OBJECT-COMPUTER. IBM-I.

   SPECIAL-NAMES.
  * MDRest4i Special Names for iCore Procedures
  * Please insert any program specific SEPCIAL NAMES BEFORE this
  * COPYBOOK, and the COPYBOOK delimits the divsion with a period
   COPY MDRCBLSPC OF MCLBL.

  ****************************************************************
   DATA DIVISION.
  ****************************************************************
   WORKING-STORAGE SECTION.
  * MDRest4i REQ/RSP Payload Schemas
   COPY CNS001C OF   QLBLSRC.
  * MDRest4i Framework Variables
   COPY MDRCBLWSC OF MCLBL.
  * MDRest4i Large Framework Variables
  * Set sizes accoprding to maximum size required in this program
  * This will effect the size of the compiled object

  * MDR-DOC - by default this paramater is OMMITTED in the SECTIONs
  * MDR-DATAGEN and MDR-DATAINTO. For more information see comments
  * for MDR-DATAGEN
  * MAXSIZE: 16000000
  ****************************************************************
  * 01  MDR-DOC          PIC X(16773100).

  * MDR-PAYLOAD MAXSIZE: 16000000
  ****************************************************************
   01  MDR-PAYLOAD PIC X(10000).

  * MDR-CLIENT-CONFIG MAXSIZE: 32767
  ****************************************************************
   01  MDR-CLIENT-CONFIG PIC X(4096).

  * MDR-MESSAGE MAXSIZE: 32767
  ****************************************************************
   01  MDR-MESSAGE       PIC X(4096).

  * MDR-STRING MAXSIZE: 65536
  ****************************************************************
   01  MDR-STRING       PIC X(4096).

  * MDR-INPUT-BUFFER MAXSIZE: 5000000
  ****************************************************************
```

1.4 SDK Web-UI Help

1.4.1 MDRest4i SDK

MDRest4i Software Development Kit (SDK) provides automation for building and documenting REST Provider and Consumer programs.

It is a web application that is hosted on the IBM i with all of the extraction, parsing, generation and data being executed/stored on the IBM i via REST API's, built using MDRFRAME.

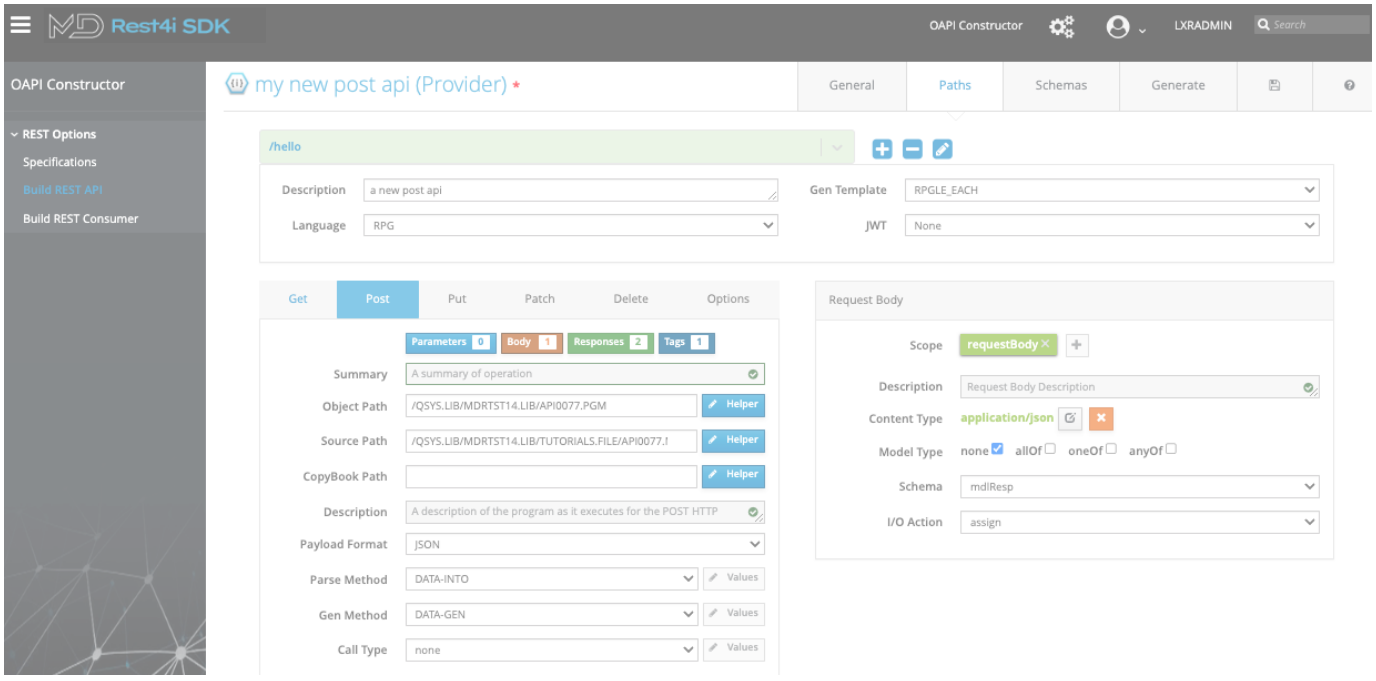
> **Info**
>
> This section provides an overview on the MDRest4i SDK. Specific details about each screen are available as online help from within the UI itself by selecting the "**?**" icon on any given page.

The main parts of the SDK are as follows:

SDK WEB UI

A graphical, form-based web UI hosted from the IBM i HTTP server. Simplifies the editing of SWAGGER definitions, used to generate Consumer and Provider programs in RPG/COBOL. SWAGGER editor and SWAGGER UI are embedded in the SDK UI, which provides a rich set of additional validation and testing capabilities to the SDK.

Form-based editing in the SDK UI



SWAGGER/OAPI

Consumer and Provider definitions are edited via the web UI and submitted to the REST API code generator.

SWAGGER definitions can be edited from basic templates provided in the SDK, imported from externally created SWAGGER/OAPI docs, or imported from POSTMAN collections.

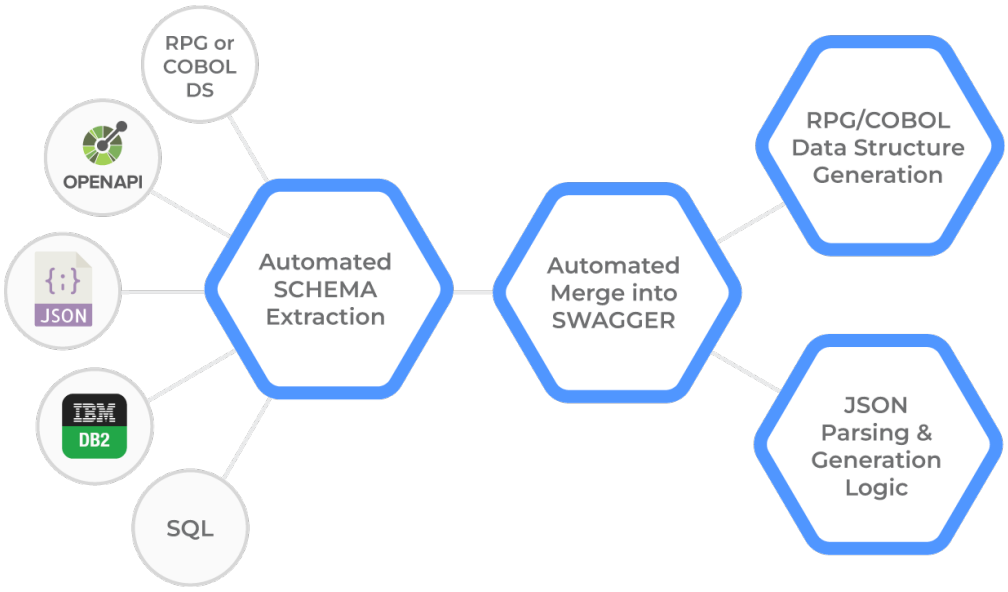
Extensions to SWAGGER are used by the SDK to specify information needed by the code generators, such as objects names, IBM i field attributes, parsing methods, etc... below is an example of the main MDRGenoptions extension added to the SWAGGER definition by the SDK:

```
x-mdrGen:
  RESTtype: provider
  genTemplate: RPGLE_EACH
  language: RPGLE
  jwt: 'N'
  mdrGenOptions:
    post:
      object: /QSYS.LIB/MDRTST14.LIB/api0077.PGM
      source: /QSYS.LIB/MDRTST14.LIB/TUTORIALS.FILE/api0077.MBR
      copybookPath: ''
      lastmdrGen: '2024-03-07T06:42:48.541Z'
      payloads:
        format: JSON
        parseMethod: DATA-INTO
        genMethod: DATA-GEN
        parseIntoFormat: UTF-8
```

SWAGGER Schema Extractors

A set of ILE REST API's called by the web UI of via IBM i commands, that generate the SWAGGER schemas and merge them into the Consumer/Provider SWAGGER definitions.

Automated SWAGGER Schema Extraction



Consumer & Provider Generators

A Rest API and ILE generator plugins, that accept SWAGGER from the SDK web UI, and generate RPGLE or COBOL code as Consumer or Provider programs.

The SDK web UI provides the user with Consumer or Provider SWAGGER(OAPI) templates

SDK USER MANAGEMENT

> **Info**
>
> SDK Portal Users and SDK Console users are shared.

SDK Users have a "User Class" assigned to them. SDK users must exist as IBM i users, before registering. No special IBM i authority is required for an SDK user

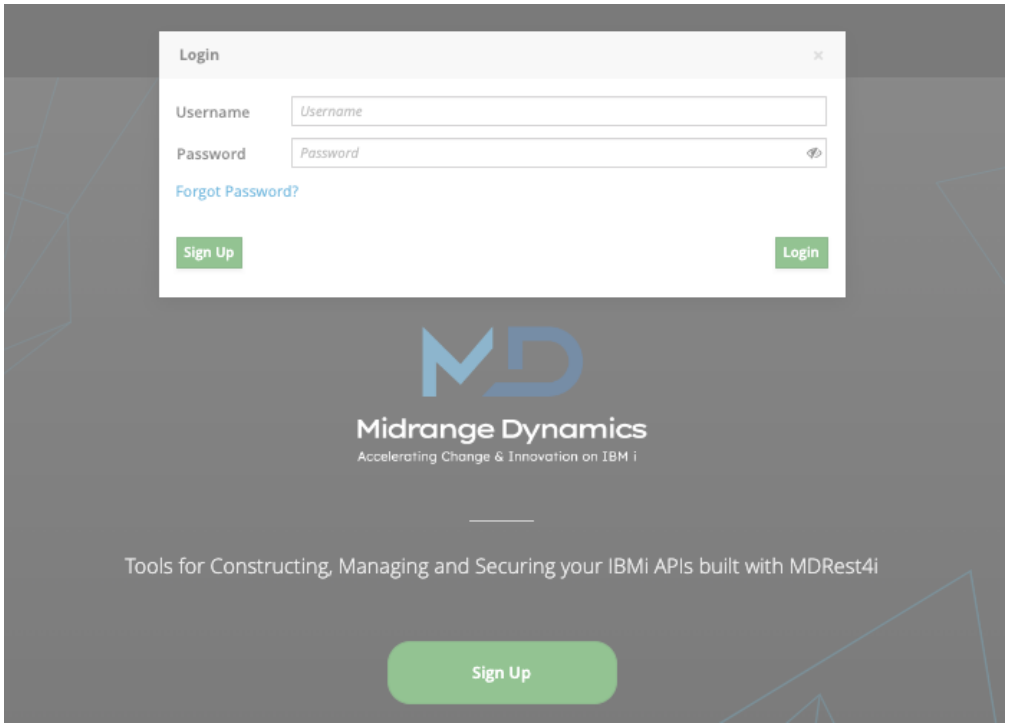
The MDRST/MDRSDKINS installation command requires a user name (ADMPRF) to use as the default SDK Administrator.

This user can be used to login to the SDK Web UI for the first time.

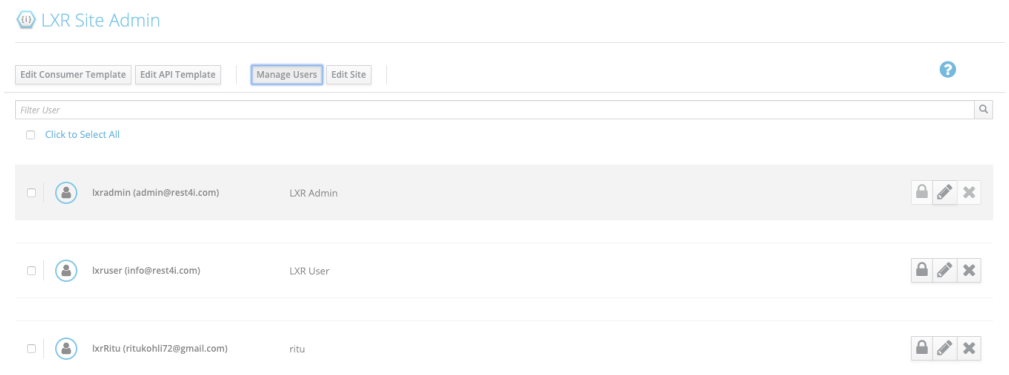
New users can then use the "Sign Up " option at the SDK Login screen.

New users must then be authorized by an Admin level user, after registering. This is accomplished via the SDK Site Admin - Manage Users section, clicking on the locked padlock icon, to "unlock" the user.

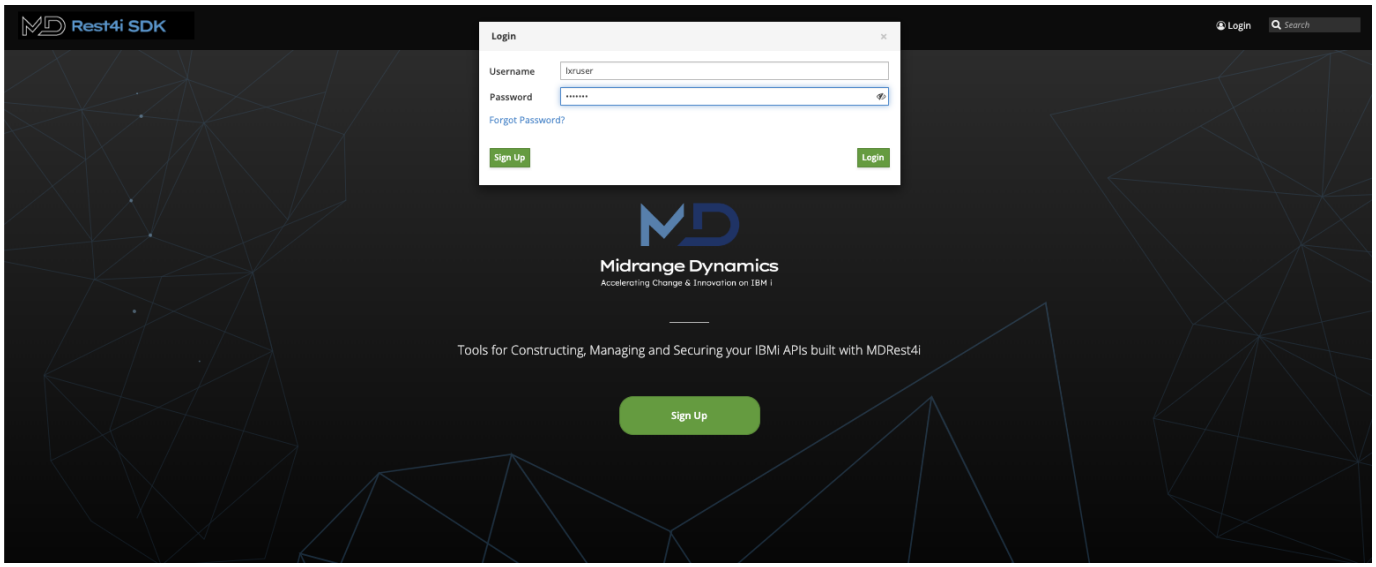
SDK Web UI Login



Site Admin, Manage Users UI



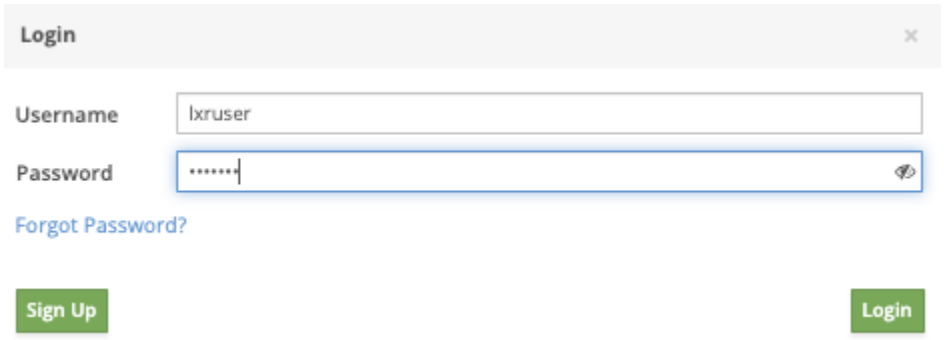
1.4.2 Login



Login

Use the 

button on the top right to sign in to the application.



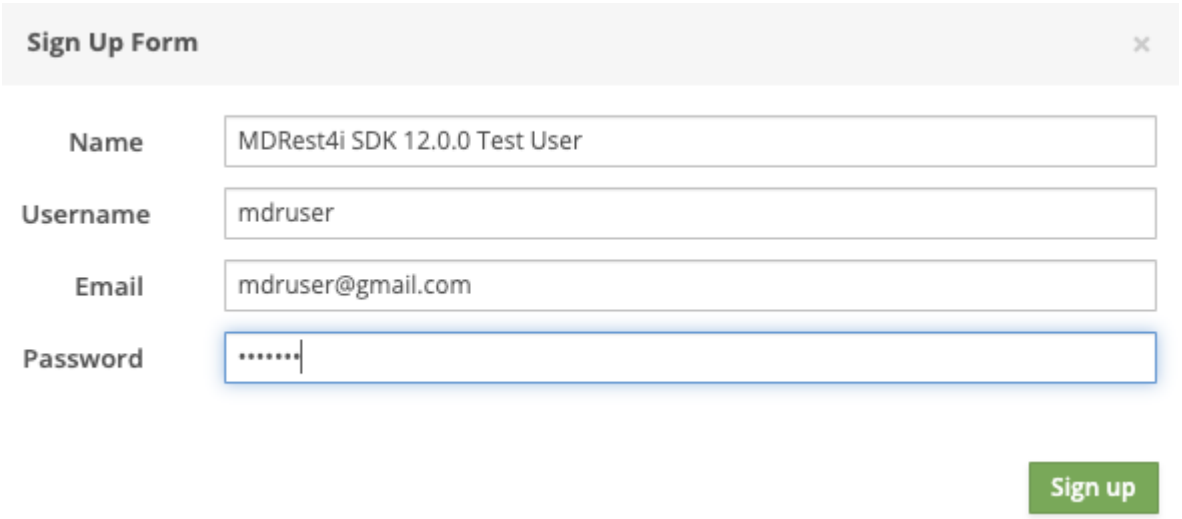
On successful login, the next screen shows up.

Sign Up

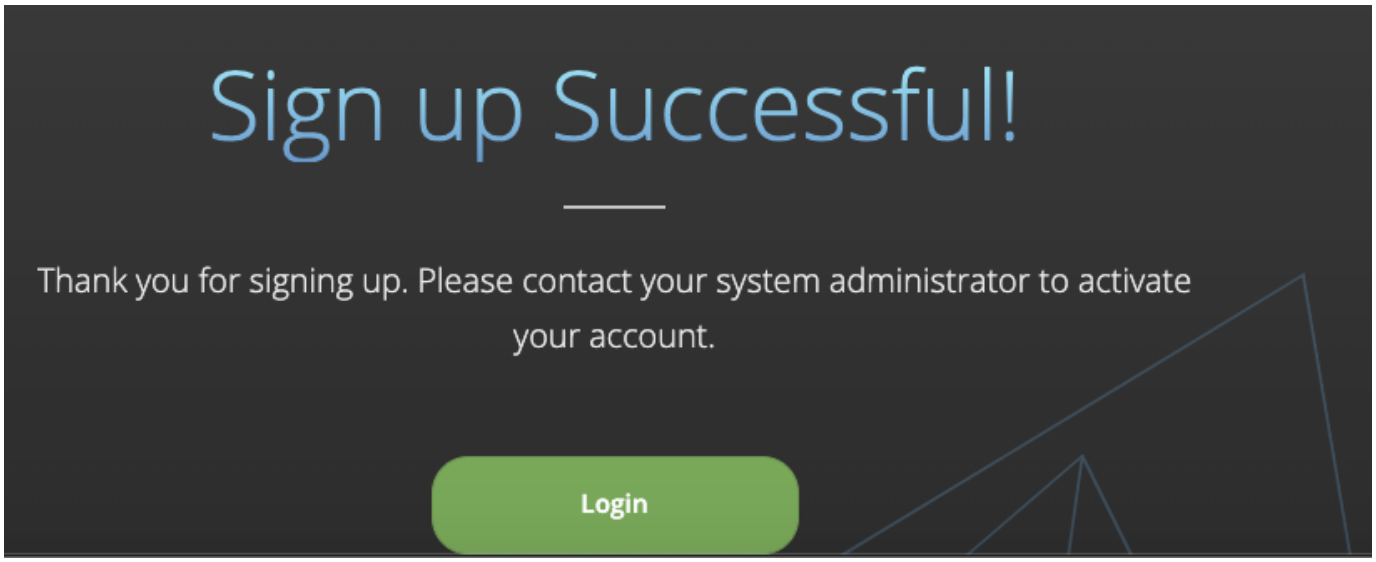
Use the 

button to create a new account. This opens the signup form.

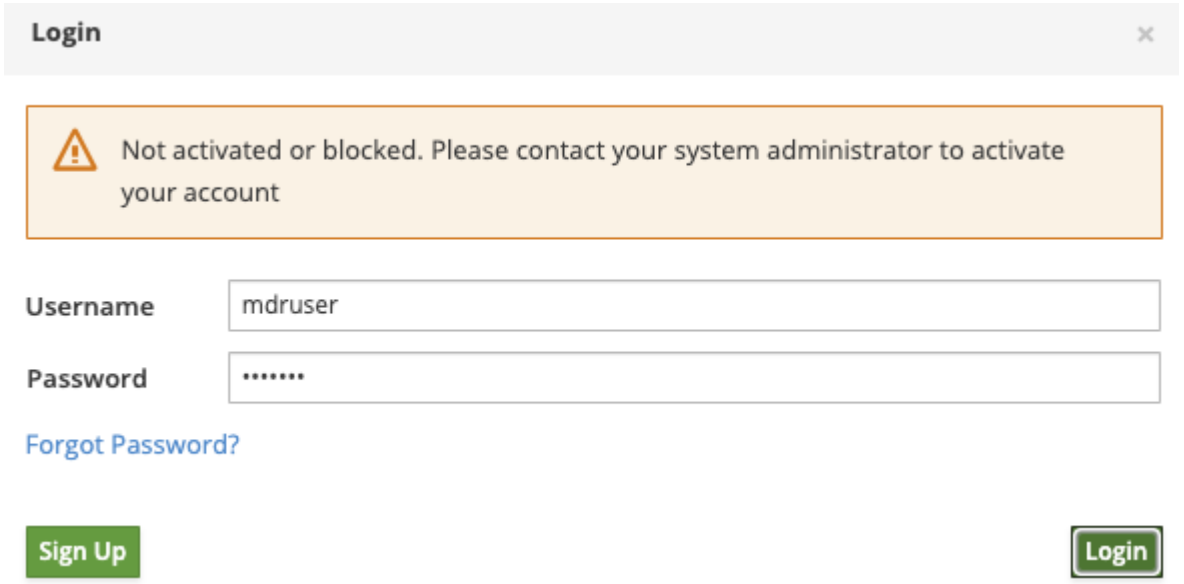
Fill the form and click the "Sign up" button.



If the username and the email id entered are unique, the account gets created successfully!



Now if you try to login using your credentials, you get an error message



You need to contact your system administrator to activate your account in the Site Admin.

1.4.3 SDK Site Administration

ONLY users with **Admin** class, are enabled to access the site administration section, including managing users.

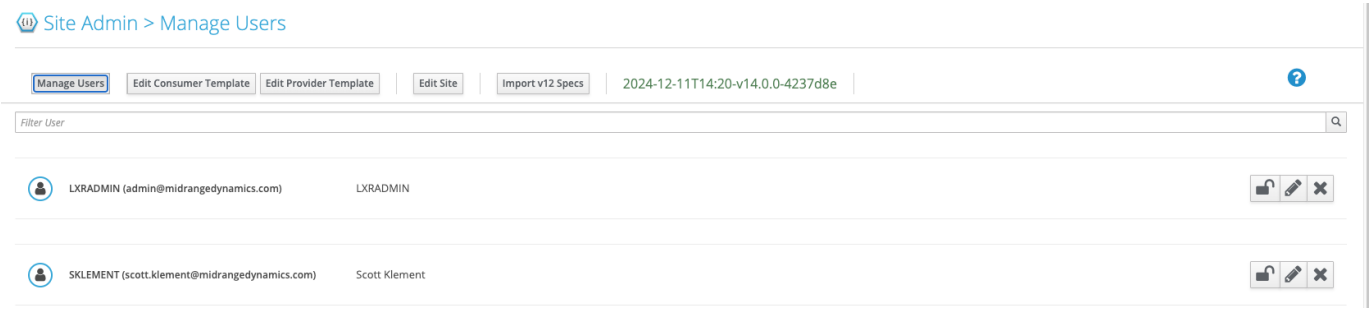
To access this, select the  button.

This displays the site administration options tool bar and opens the Manage Users window by default:

Manage Users	Manage the users - block/delete the selected users or edit a user profile
Edit Consumer Template	Edit the standard template for Consumer
Edit Provider Template	Edit the standard template for Provider
Edit Site	Edit the configuration settings for the application
Import v12 Specs	Bulk import of V12 SDK specs into v14
2024-04-30T10:28-V14.0.0-9b1548a	The build Date of the SDK and the commit hash

Manage Users:

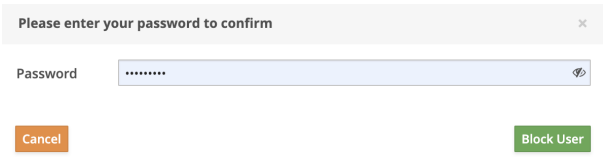
Options to activate/de-activate, edit profile or delete a user



USER ACTIONS

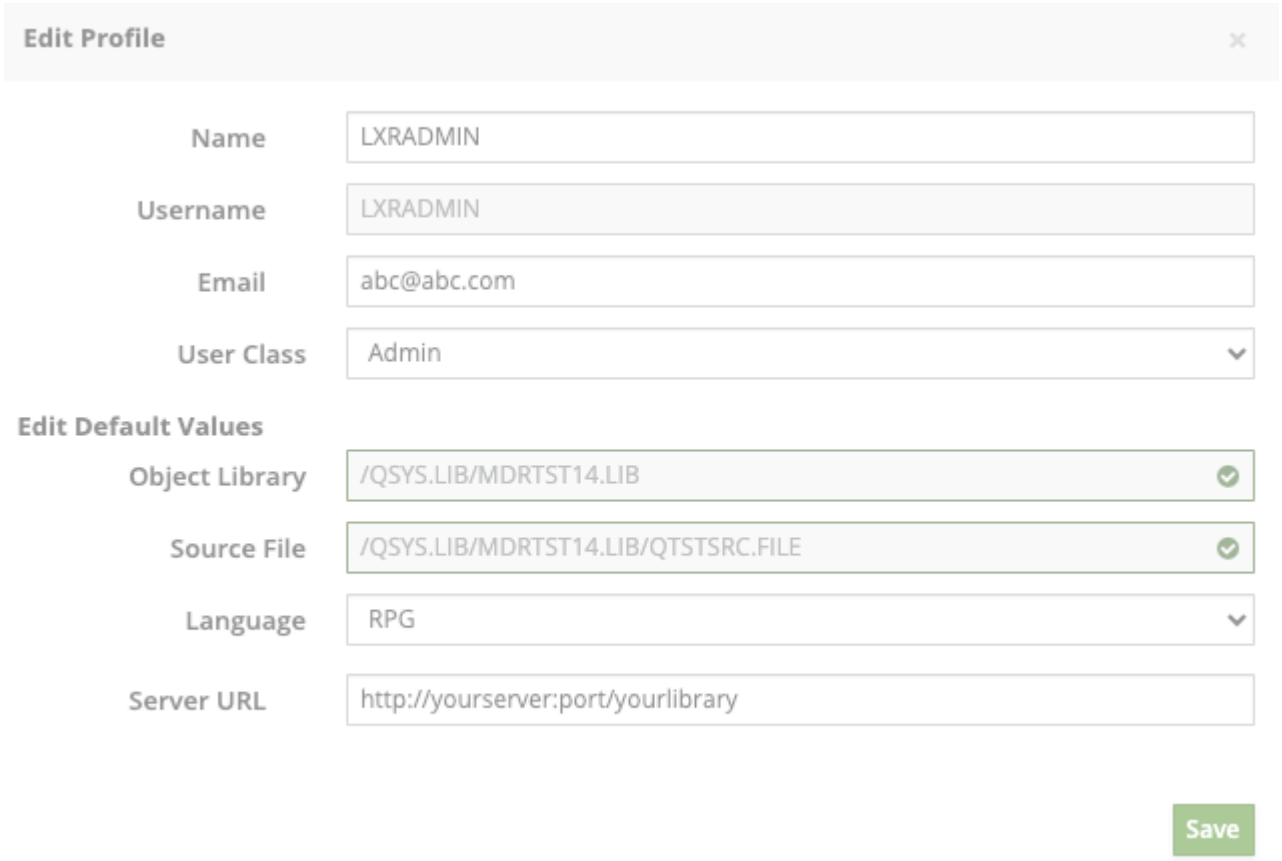
Note

Any of the User Actions below require you to provide your login password to authorize the change.



	Used to unlock/lock a user after signing up
	To edit the profile of a particular user, select the user and click on the pencil icon
	To delete a user

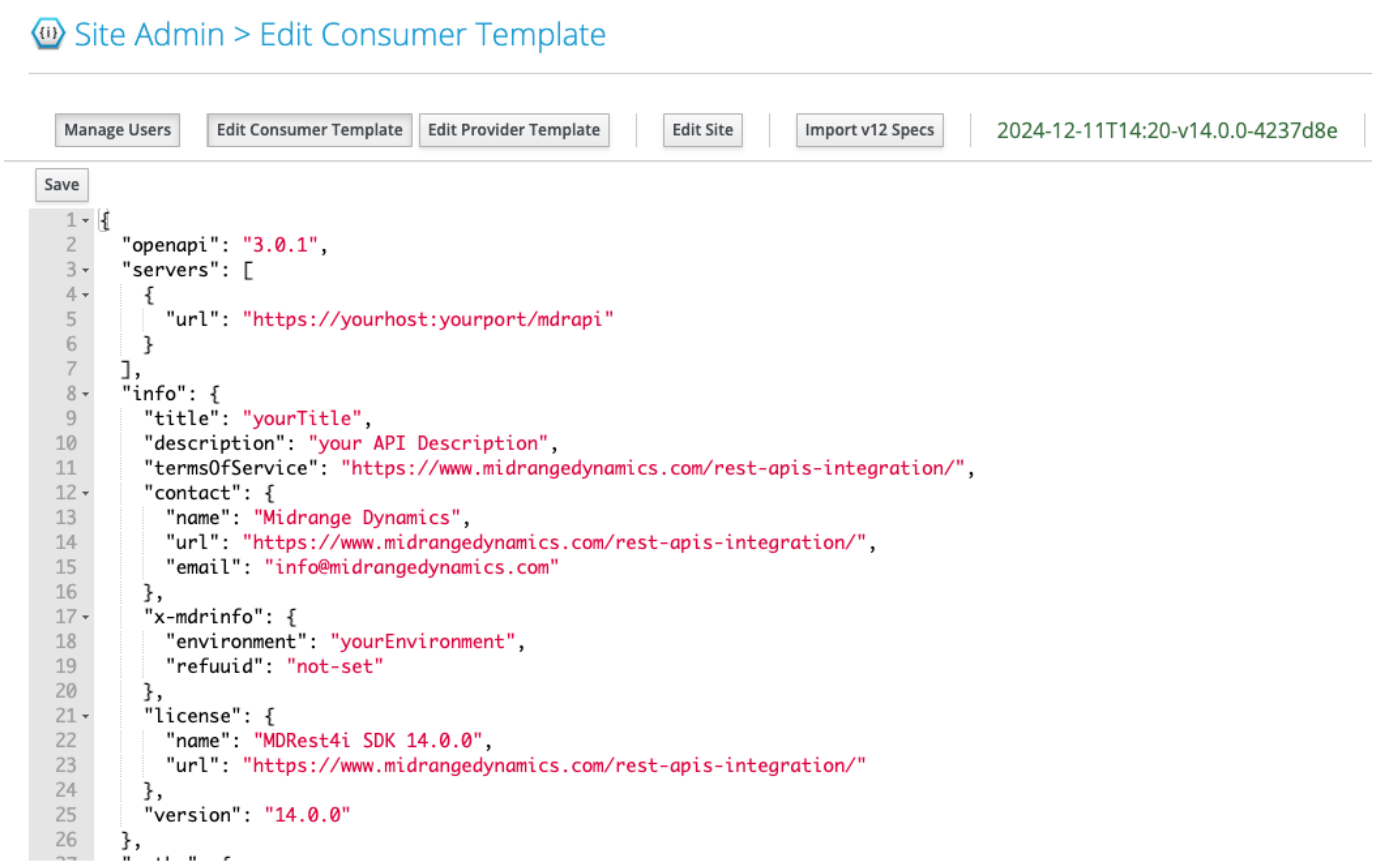
EDIT PROFILE



Email	Not in use - for reference only
User class	Admin, User or Developer to be selected from the dropdown list
Source File	The default source file path used when the user creates a new consumer/Provider
Object Library	The default object library path used when the user creates a new Consumer/Provider
Language	The default language in which the code is generated
Server URL	The default Server/API URL when the user creates a new Provider

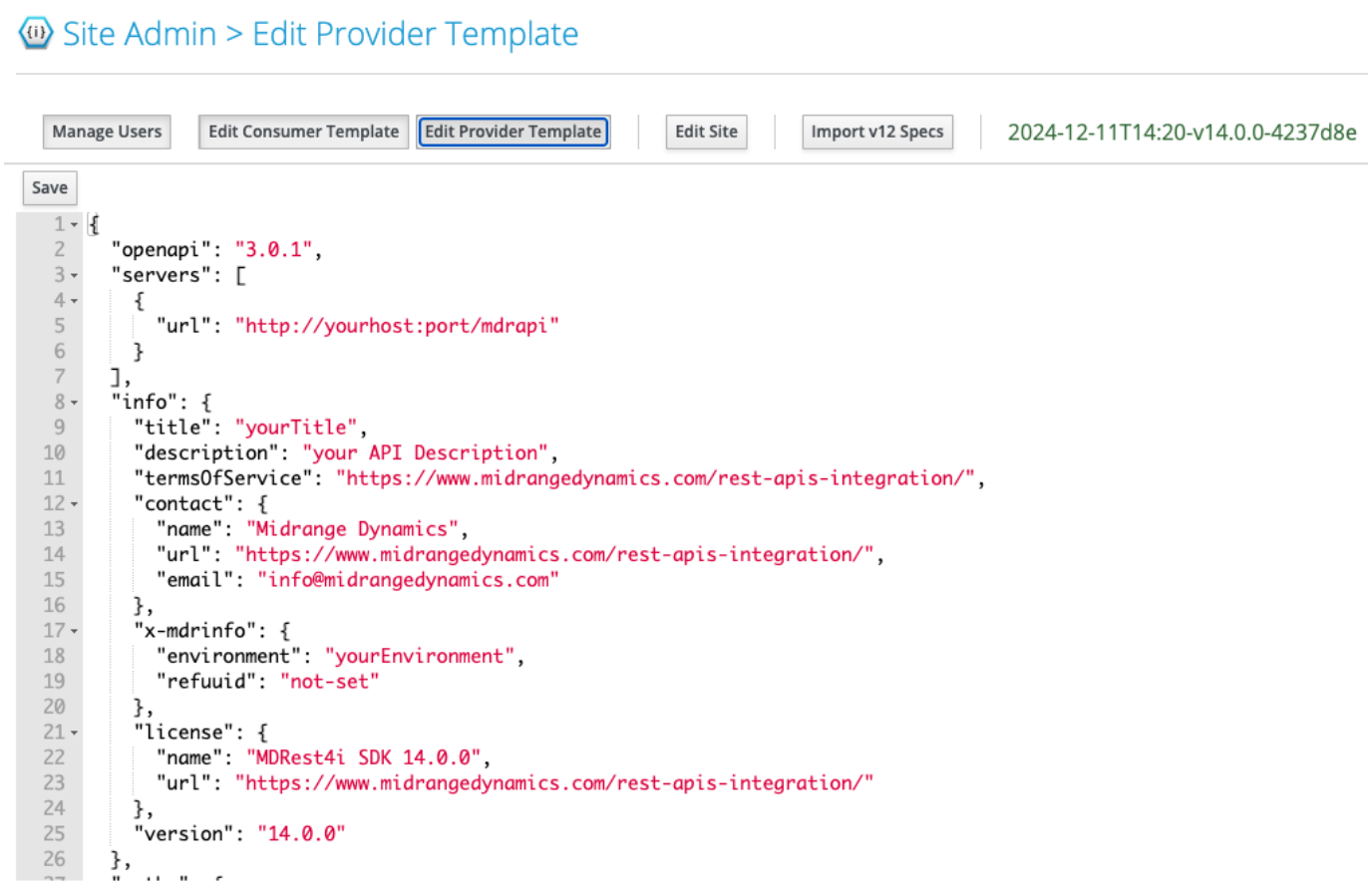
Edit Consumer Template

Use this option to edit the standard template for "Consumer"



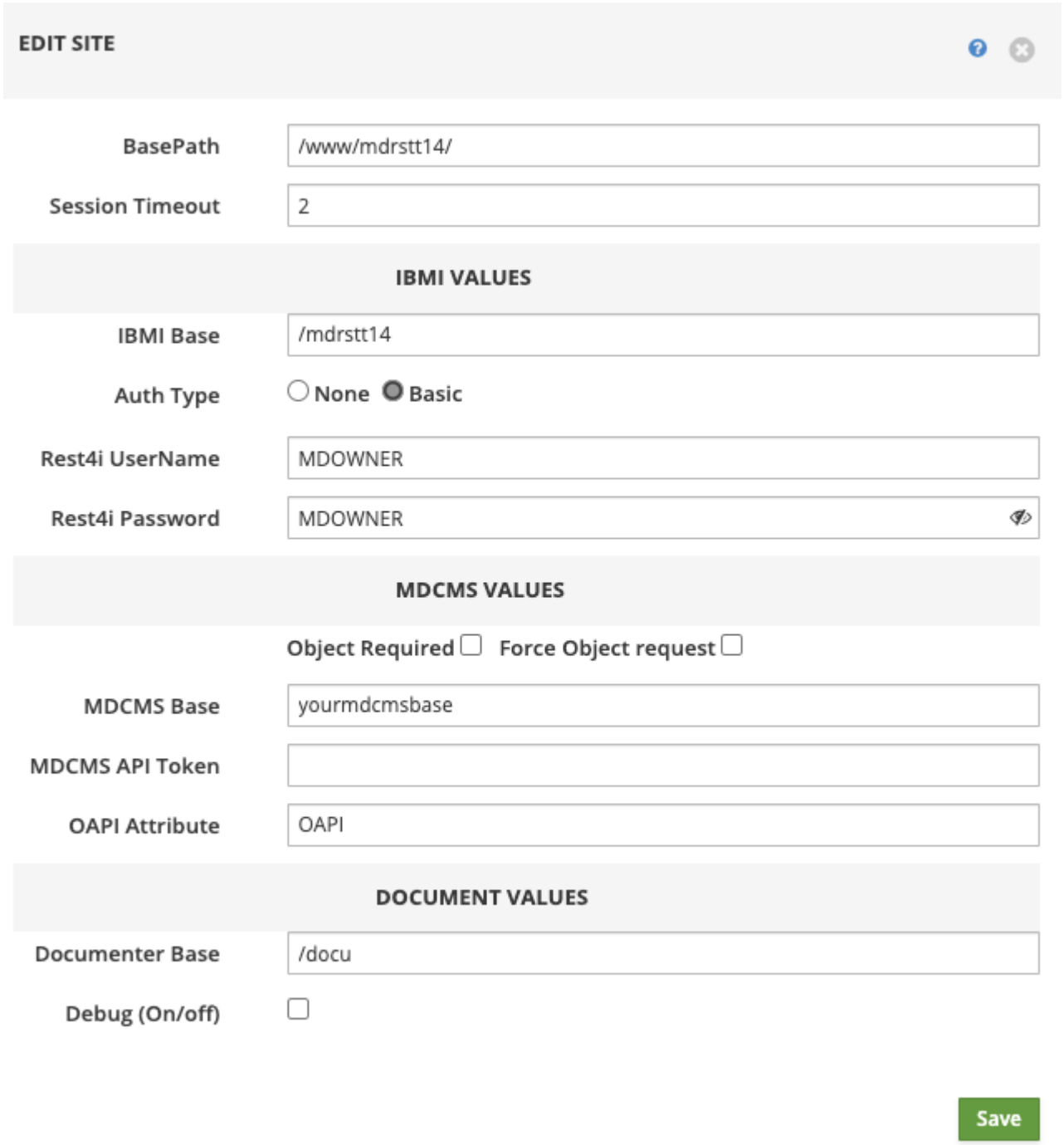
Edit Provider Template

Use this option to edit the standard template for "Provider"



Edit Site

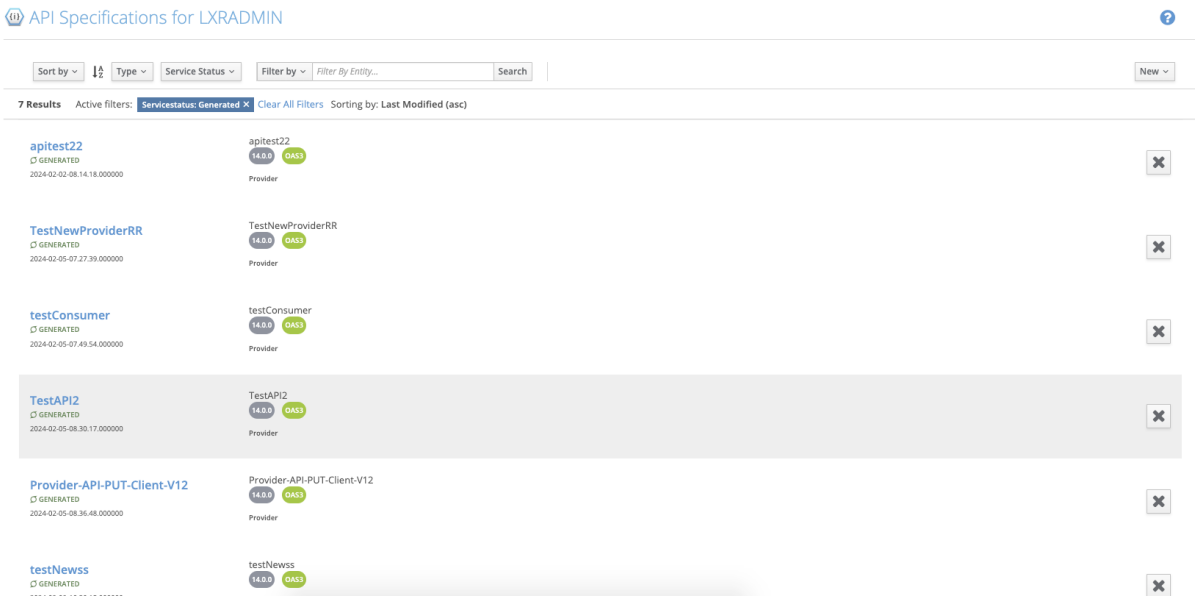
Enter/edit the values to set the configuration for the application.



Base Path	The absolute path where the application is installed on the IBMI server eg. /www/mdrstt12/
Session Timeout	Length of the session, the session timeout duration in hours
IBMI Base	The base address that connects the UI to the MDREST4i API generators
Rest4i Username & Password	Credentials to login to the your IBMI machine running MDREST4i API generators.
Object required	If checked, the application will let the user to create an MDCMS object request
Force Object request	If checked, the application will not let the user create an API unless the MDCMS object request is created.
MDCMS Base	Must be configured for MDCMS Object requests
MDCMS API Token	The Authorization Bearer token value for making any request to any MDCMS API
OAPI attribute	The MDCMS attribute used to check out the OAPI/SWAGGER specification
Documenter Base	Base address of the MDREST4i SDK portal application.

1.4.4 User Specifications List

This screen allows the logged in user, to manage their OAPI/SWAGGER specifications.

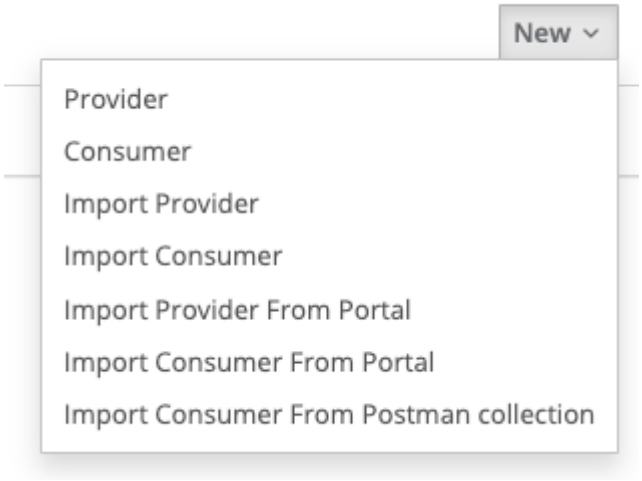


Header Filters

Allow the user to control what specs are displayed in the main section

Sort by	Allows user to sort the list by Entity name or last modified date using a dropdown list in both ascending and descending orders
Type	Type of API: Provider or Consumer
State	The status: Generated or Ungenerated
Filter by	Entity or title
Search	Button to filter the results based on the search criteria entered

New Actions



New Provider	Creates a new Provider specification
New Consumer	Creates a new consumer specification
Import Provider	Imports a new Provider specification from a local OAPI/SWAGGER file
Import Consumer	Imports a new Consumer specification from a local OAPI/SWAGGER file
Import Provider from Portal	Imports a new Provider specification from an environment in MDRest4i-SDK-Documenter
Import Consumer from Portal	Imports a new consumer specification from an environment in MDRest4i-SDK-Documenter
Import Consumer from Postman collection	Imports a new consumer specification from a postman collection export

NEW PROVIDER

Use this option to create a new Provider spec.

Add New Provider Specification

Title	Provider API Get Client V14
Refuuid	Provider-API-Get-Client-V14
Description	Produces and API Provider the returns an array of clients from the LXCLIENT file
Path	/clients
Method	get
Gen Template	RPGLE_EACH
Program Name	GETCLNTDTL
Object path	/QSYS.LIB/MDRTST14.LIB Helper
Source File	/QSYS.LIB/MDRTST14.LIB/EXAMPLE.FILE Helper

Done

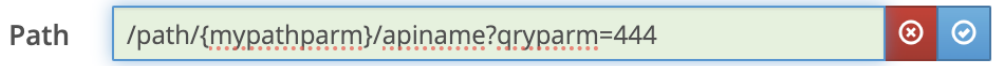
Title	The Provider title
Refuuid	Auto-generated from the field "Title" but editable. This refuuid is used as a key for the spec in the MDRDOAPI file, as well as the refuuid for the documenter (MDRDPAGE file)
Description	Description
Path	The Path is mandatory. Unlike the consumer, only the path part is required (not the host and port information). Path parameters can be specified as part of the path. For example: `/apis/myapi/{nameid}` **nameid** will automatically be added as an inpath parameter in the specification
Method	The http method, can be selected from the dropdown list (get, post, put, patch, delete, options)
Gen Template	This determines what generation pattern will be used. Possible values are: **RPGLE_EACH** - One program per path/method combination in RPGLE. **COBOL_DFT** - One program per path/method combination in COBOL ILE
Program Name	The compiled object name of the provider program
Object Path	Library where the generated provider program will be compiled. Uses IFS format with a helper button. For example: `/QSYS.LIB/MDRSTT14.LIB` Picks up the default object path stored in the user profile, and can be edited.
Source File	Location where the provider source will be generated. Uses IFS format with a helper button. MDRest4i supports either an IFS path `/mdrtst14/src/skdemo.rpgle` or source file `/QSYS.LIB/APISRCLIB.LIB/QRPGLESRC.FILE/SKDEMO.MBR`. Picks up the default source file name stored in the user profile, can be edited

The General tab screen now appears.

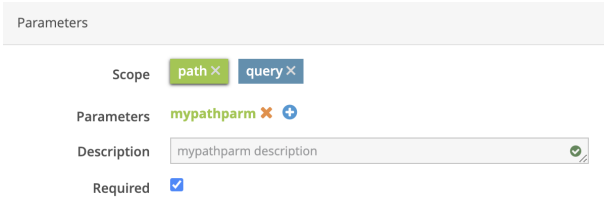
Hot Tip

If you include query or path parameters in the path variable, while adding a new Provider spec, the SDK will automatically add these parameters into the specification.

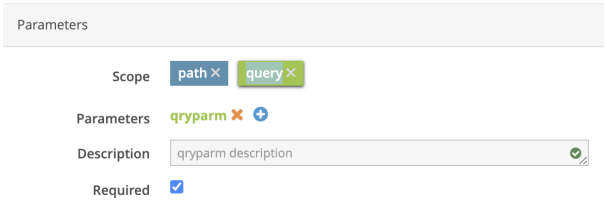
For example this path:



Creates a path parameter



and a query parameter

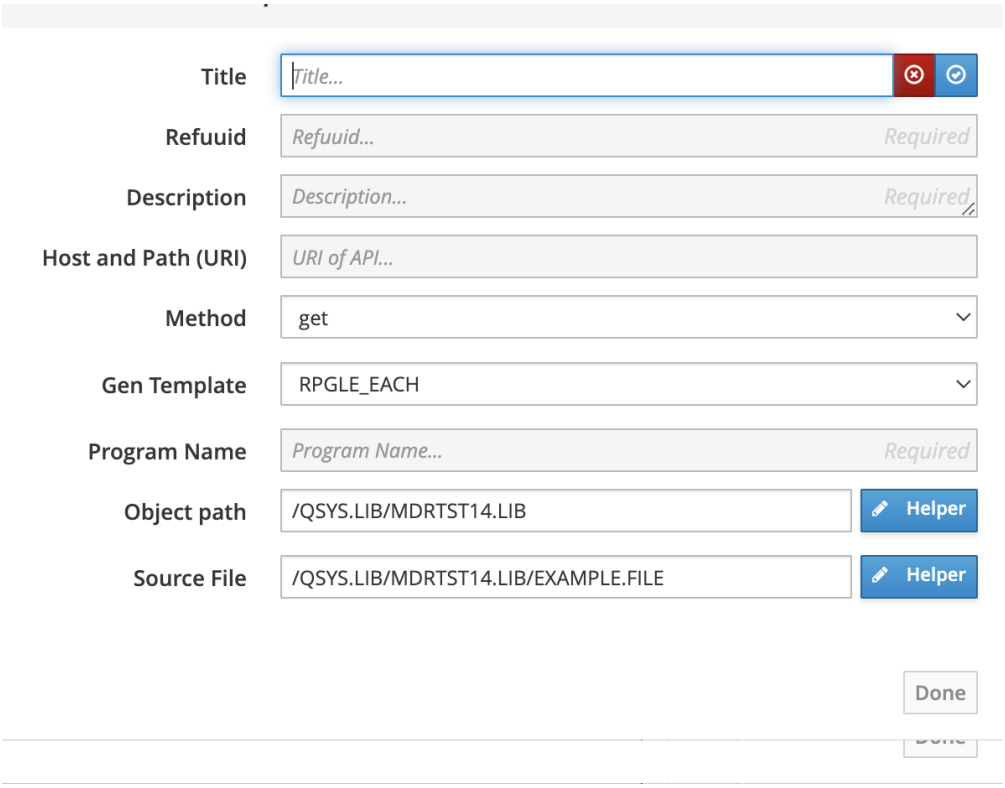


To the OAPI spec:

```yaml
parameters:
  - in: path
    name: mypathparm
    description: mypathparm description
    required: true
    schema:
      type: string
  - in: query
    name: qryparm
    description: qryparm description
    required: true
    schema:
      type: string
```

NEW CONSUMER

Use this option to create a new Consumer spec



Title	The Consumer Title
Refuuid	uto-generated from the field "Title" but editable. This refuuid is used as a key for the spec in the MDRDOAPI file, as well as the refuuid for the documenter
Description	Description
Host and Path (URI)	The consumer API URL. In-path parameters can also be used in the URI. For example: `https://myhost.com/apis/myapi/{nameid}` **nameid** will automatically be added as an inpath parameter in the specification
Method	The http method, can be selected from the dropdown list (get, post, put, patch, delete, head)
Gen Template	This determines what generation pattern will be used. Possible values are: **RPGLE_EACH** - One program per path/method combination in RPGLE. **COBOL_DFT** - One program per path/method combination in COBOL ILE
Program Name	The compiled object name of the consumer program
Object Path	Library where the generated provider program will be compiled. Uses IFS format with a helper button. For example: `/QSYS.LIB/MDRSTT14.LIB` Picks up the default object path stored in the user profile, and can be edited.
Source File	Location where the consumer source will be generated. Uses IFS format with a helper button. MDRest4i supports either an IFS path `/mdrtst14/src/skdemo.rpgle` or source file `/QSYS.LIB/APISRCLIB.LIB/QRPGLESRC.FILE/SKDEMO.MBR`. Picks up the default source file name stored in the user profile, can be edited

Click the "Done" button (which is enabled only after validating all the input fields).

The General tab screen now appears.

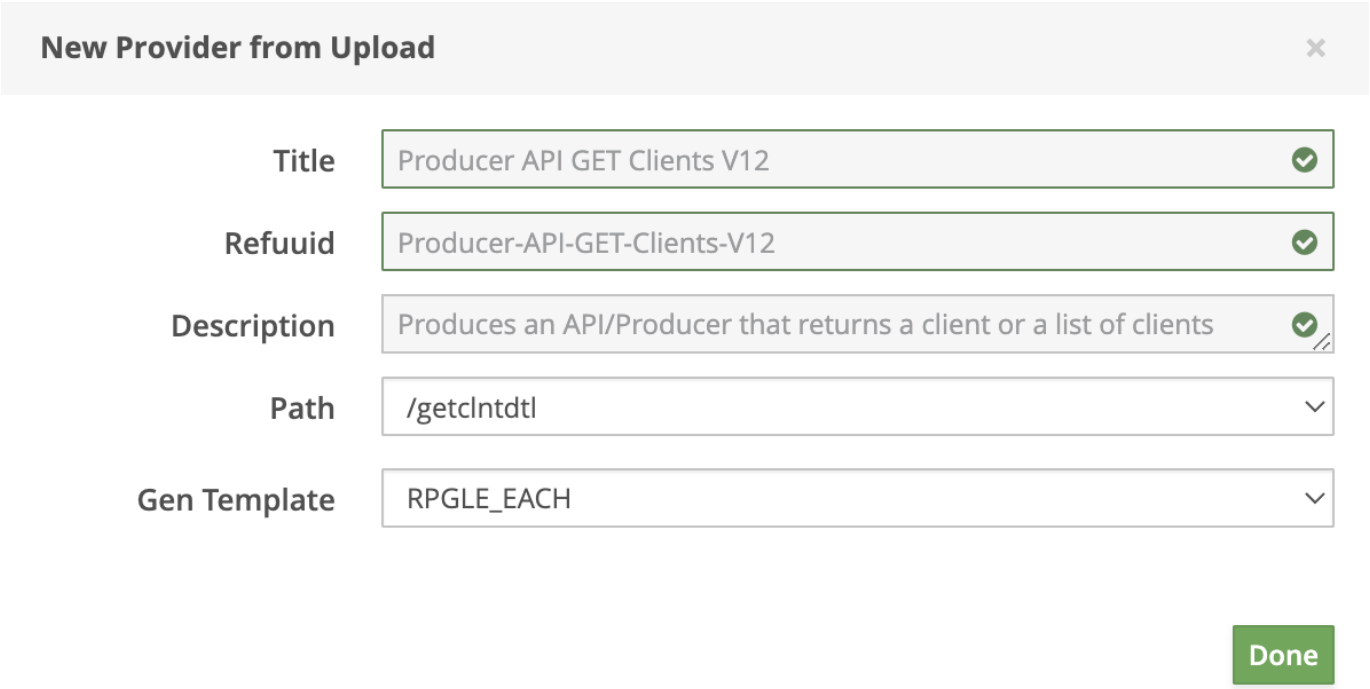
IMPORT PROVIDER

> **Warning**
>
> The SDK supports OPAI 3.0.0 upwards

Use this option to import a new Provider from the provided Swagger specification file.

The SDK adds its own extensions to the OAPI/SWAGGER content on import.

Selecting this option opens the file browser from where you select the file to upload. If the import happens successfully, the following screen appears:



The Helper buttons are also available to help you create the source and object paths.

The input fields in the form are picked up from the default values entered in the user profile.

Click the "Done" button (which is enabled only after validating all the input fields). The General tab screen now appears.

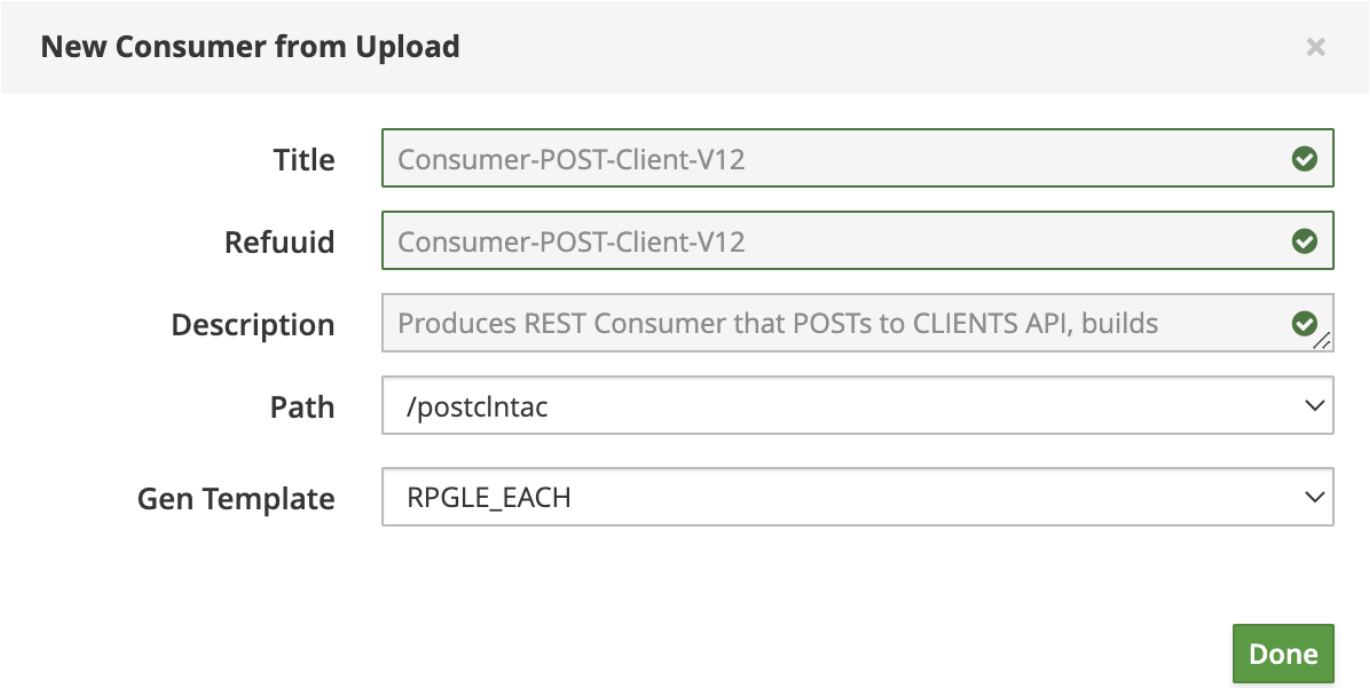
IMPORT CONSUMER

> **Warning**
>
> The SDK supports OAPI 3.0.0 upwards

Use this option to import a new consumer from the provided Swagger specification file.

The SDK adds its own extensions to the OAPI/SWAGGER content on import.

Selecting this option opens the file browser from where you select the file to upload. If the import happens successfully, the following screen appears:



The input fields in the form- Object Library, Source File and language are picked up from the default values entered in the user profile.

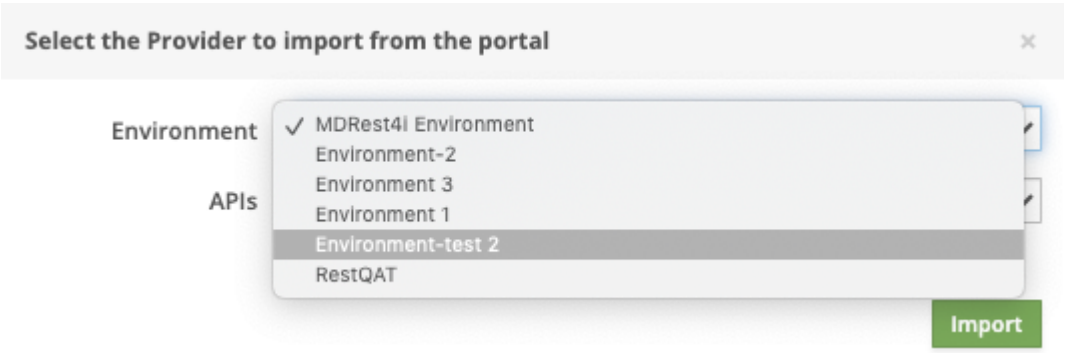
The other values would be picked up from the file imported.

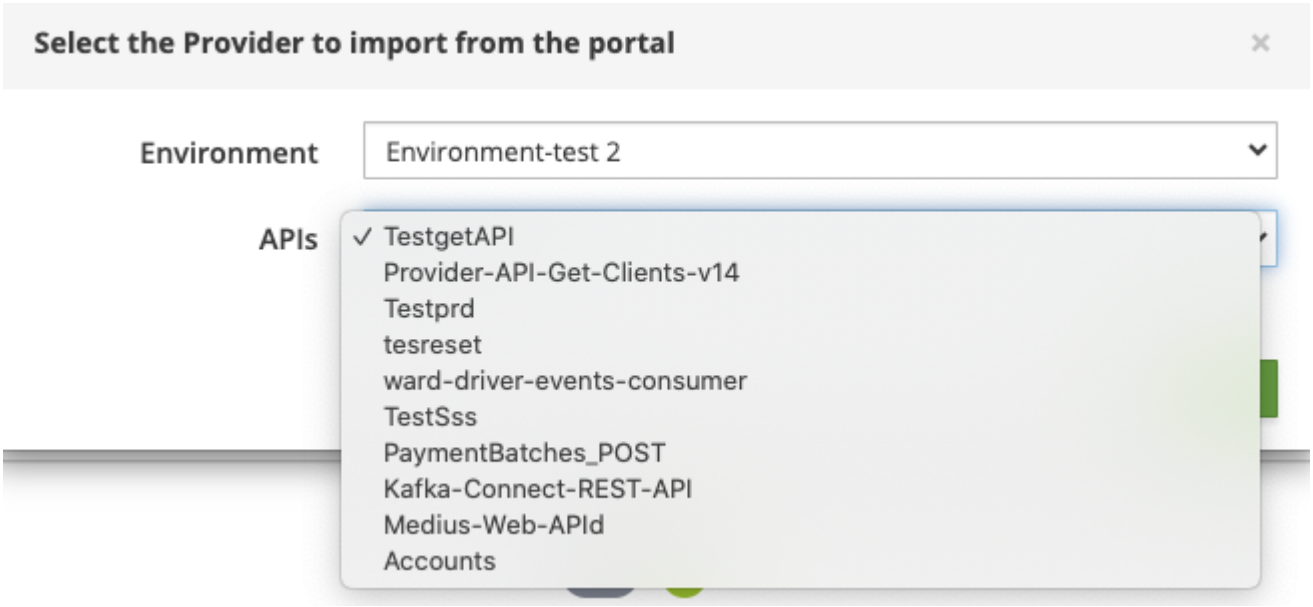
Click the "Done" button (which is enabled only after validating all the input fields). The General tab screen now appears.

IMPORT PROVIDER FROM PORTAL

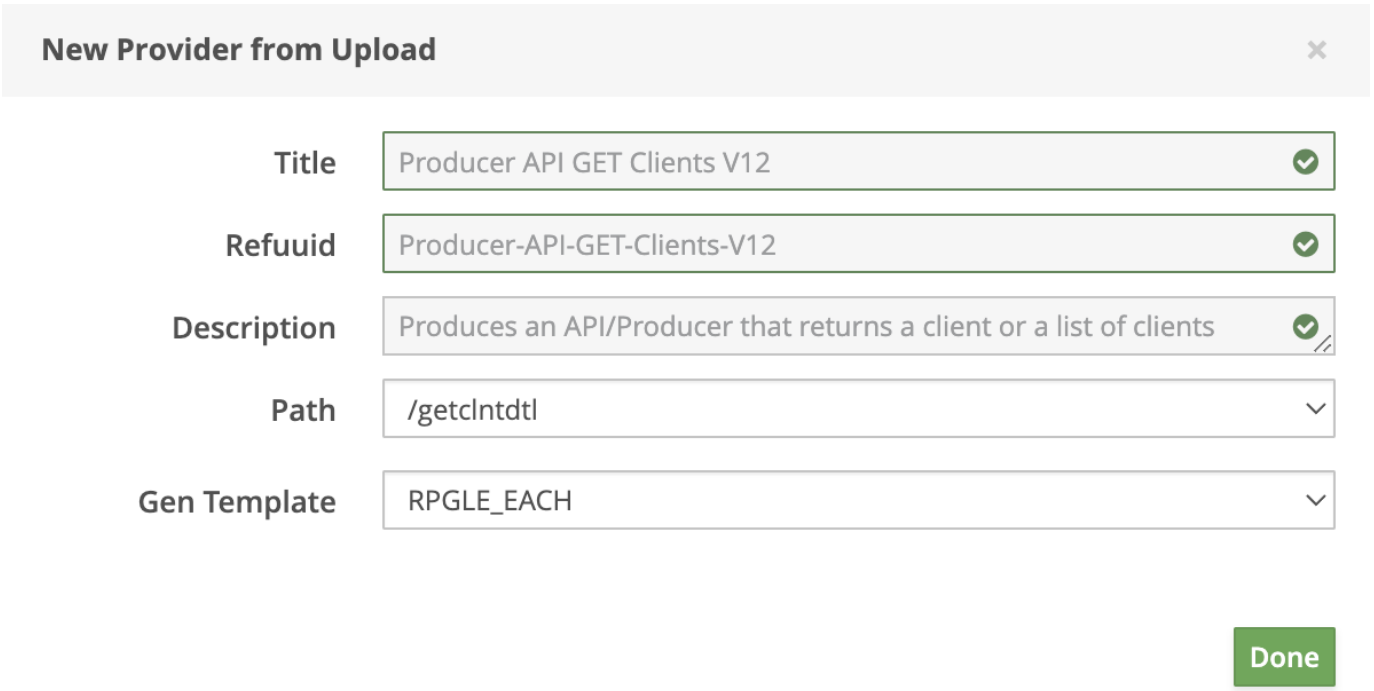
Use this option to import a new Provider from the documenter portal.

Selecting this option opens the following popup to select the environment and the API file to upload.





If the import happens successfully, the following screen pops up:



The input fields in the form- Object Library, Source File and library are picked up from the default values entered in the user profile.

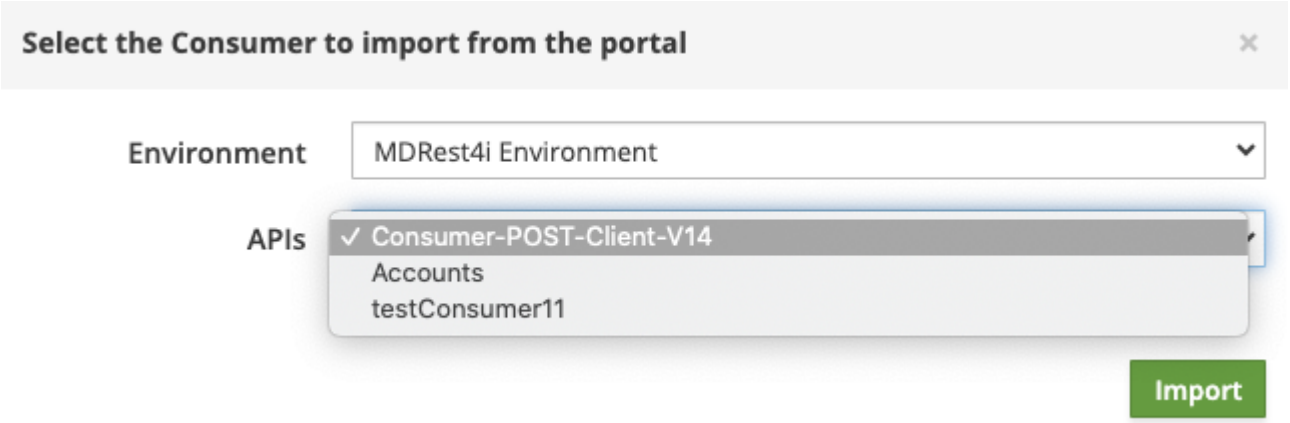
The other values would be picked up from the file imported.

Click the "Done" button (which is enabled only after validating all the input fields). The General tab screen now appears.

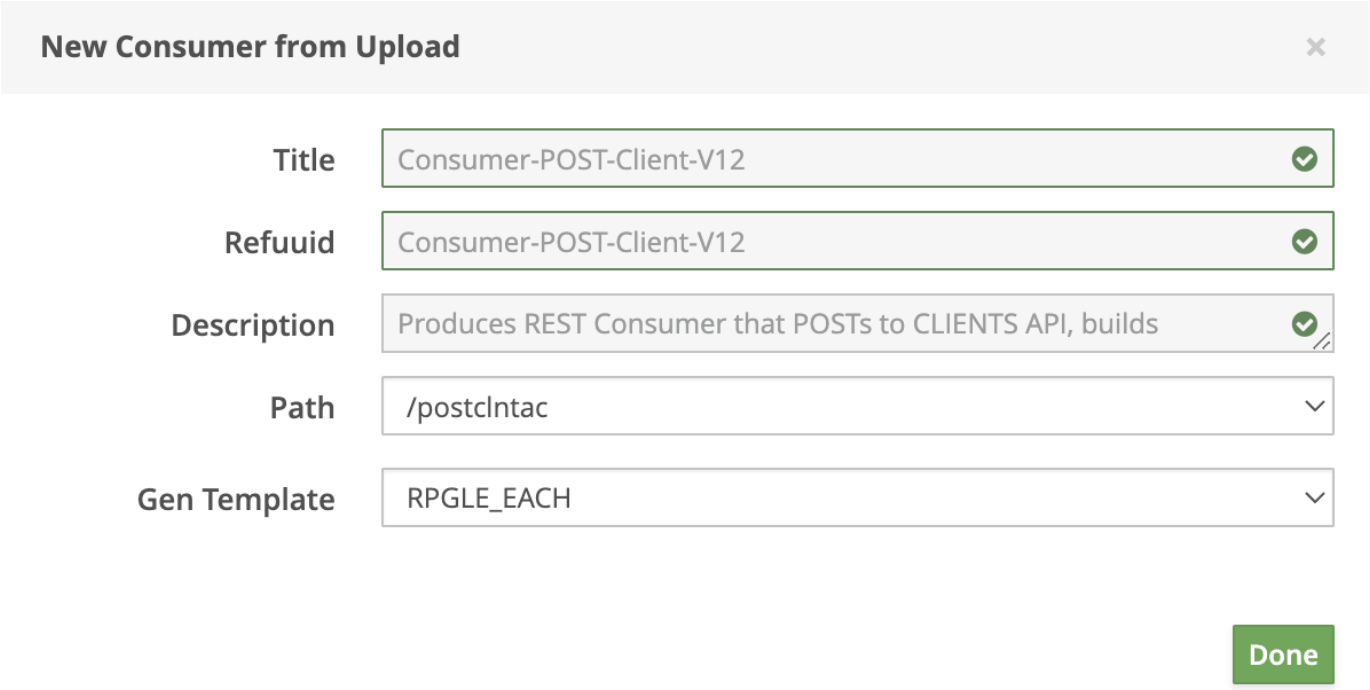
IMPORT CONSUMER FROM PORTAL

Use this option to import a new Consumer from the portal.

Selecting this option opens the following popup to select the environment and the API file to upload.



If the import happens successfully, the following screen pops up:



The input fields in the form- Object Library, Source File and library are picked up from the default values entered in the user profile.

The other values would be picked up from the file imported.

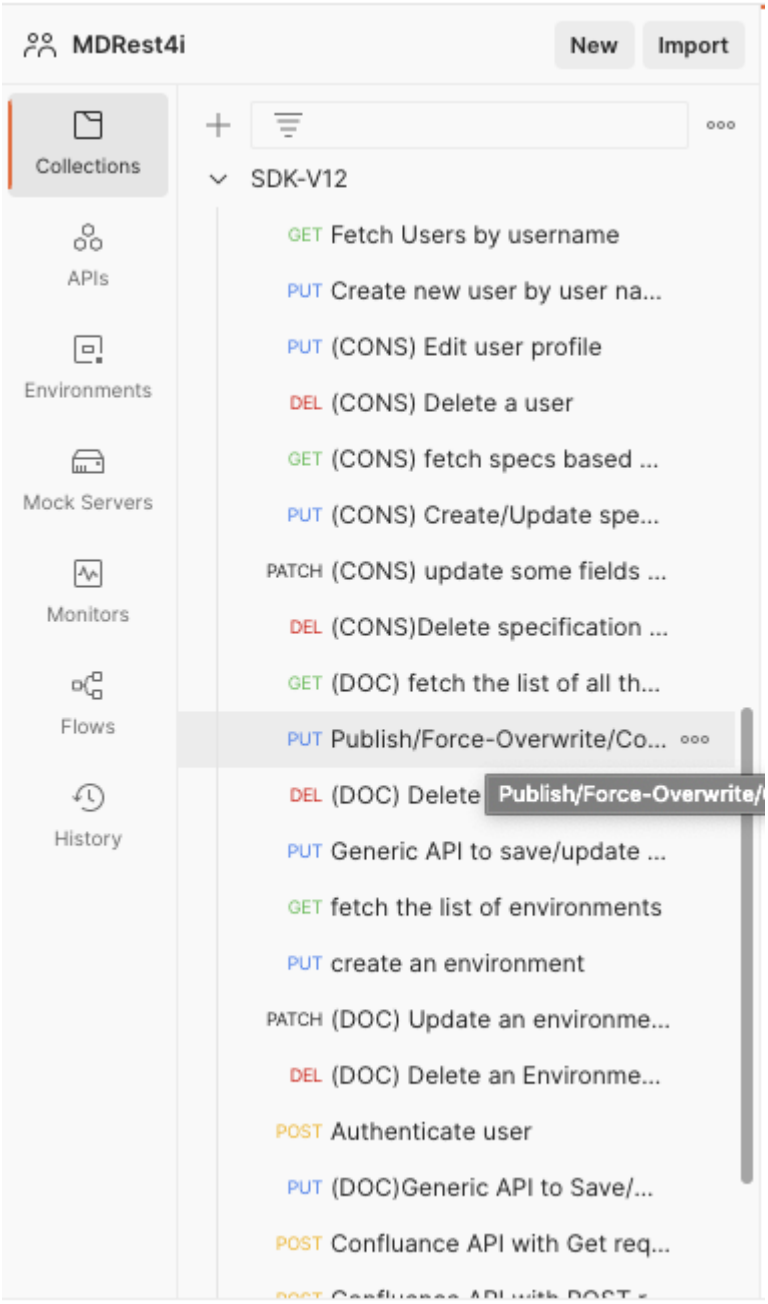
Click the "Done" button (which is enabled only after validating all the input fields). The General tab screen now appears.

IMPORT CONSUMER FROM POSTMAN COLLECTION

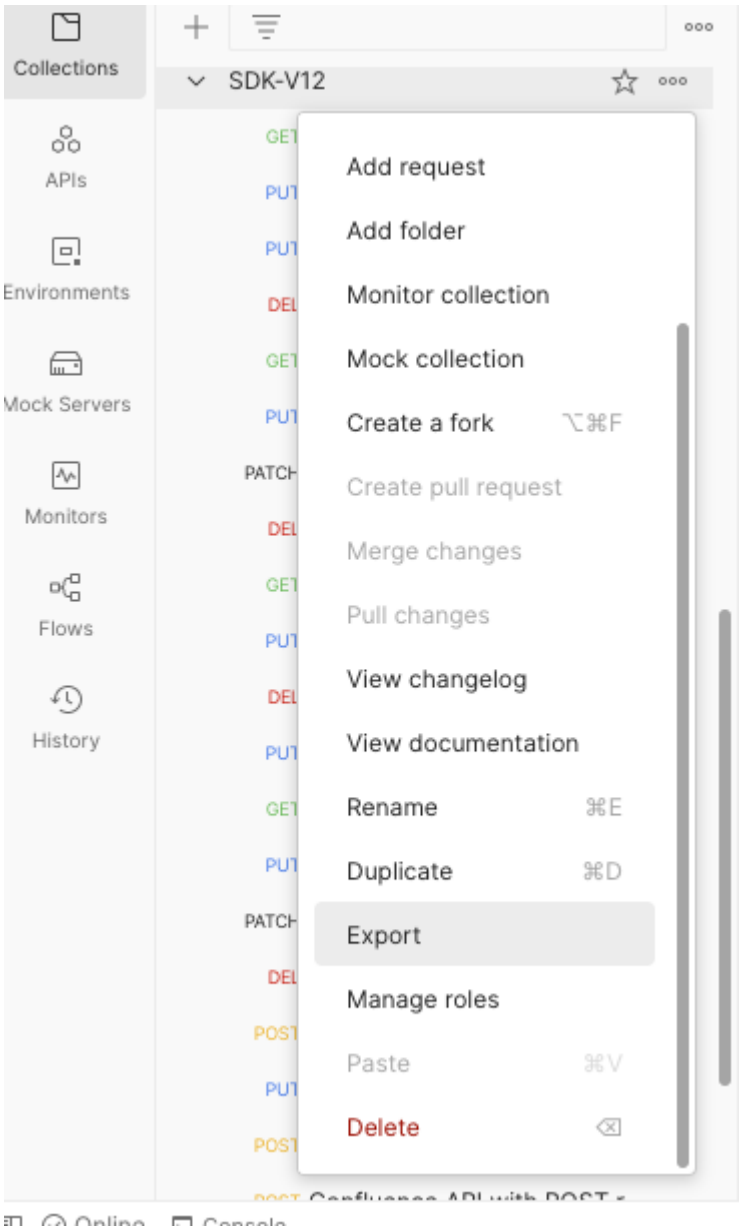
How to export from a postman collection

Use this option to import a new consumer from a postman collection export file.

Consider this example in postman where we have created a collection named SDK-V12



Click on the ellipses ••• to bring up the menu and select the option - "**Export**".



A postman collection may or may not contain multiple folders and sub-folders containing the different requests.

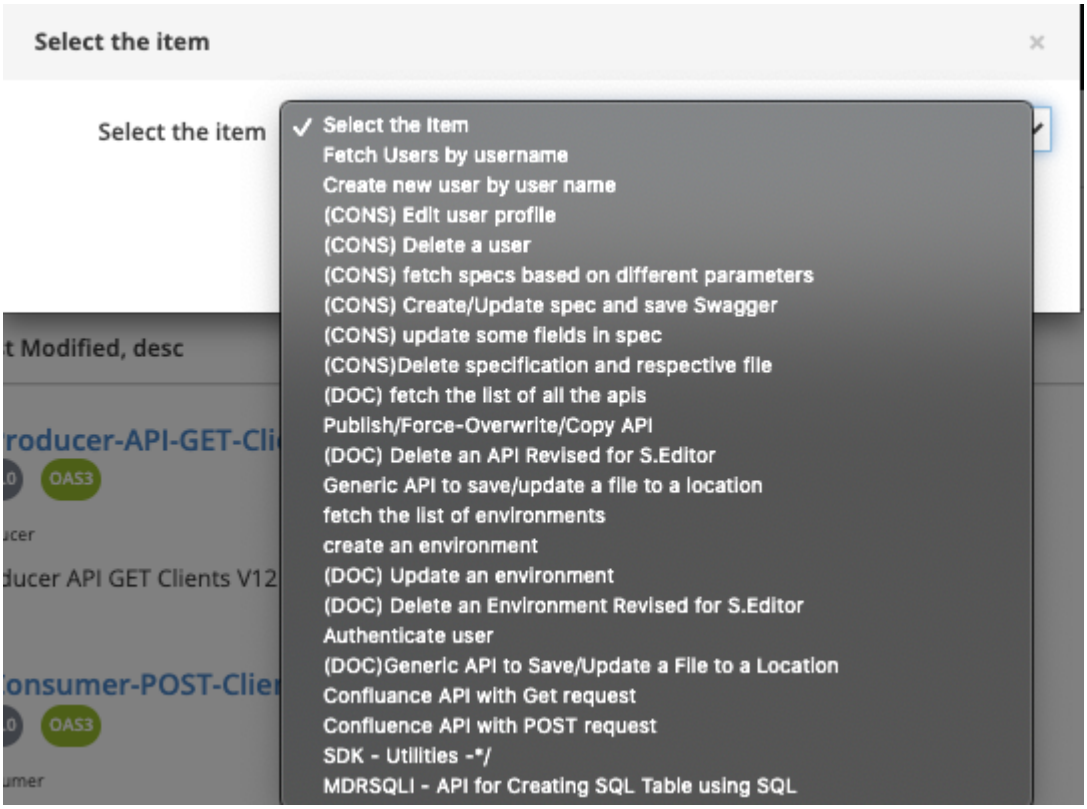
On selecting this option, the postman export file gets saved on your local file system.

Import from the exported postman collection file

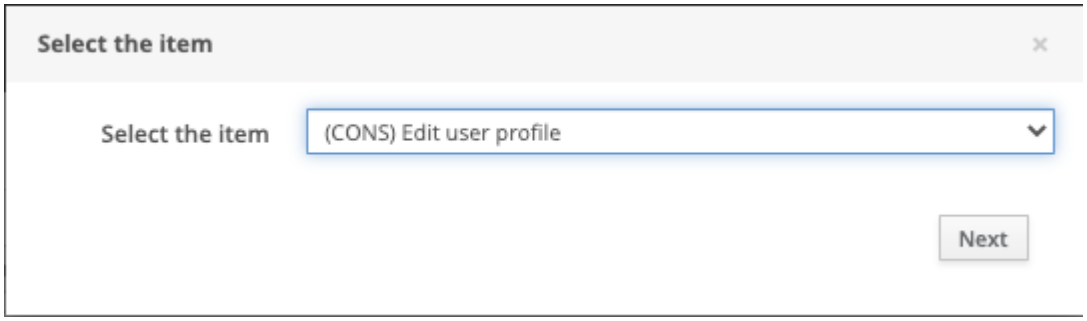
Select the option "Import Consumer From Postman Collection" brings up the file explorer where you can select the file you wish to import. After the file is selected, a popup shows up.

A complete dropdown list of all the APIs is shown and the user can select any one to import.

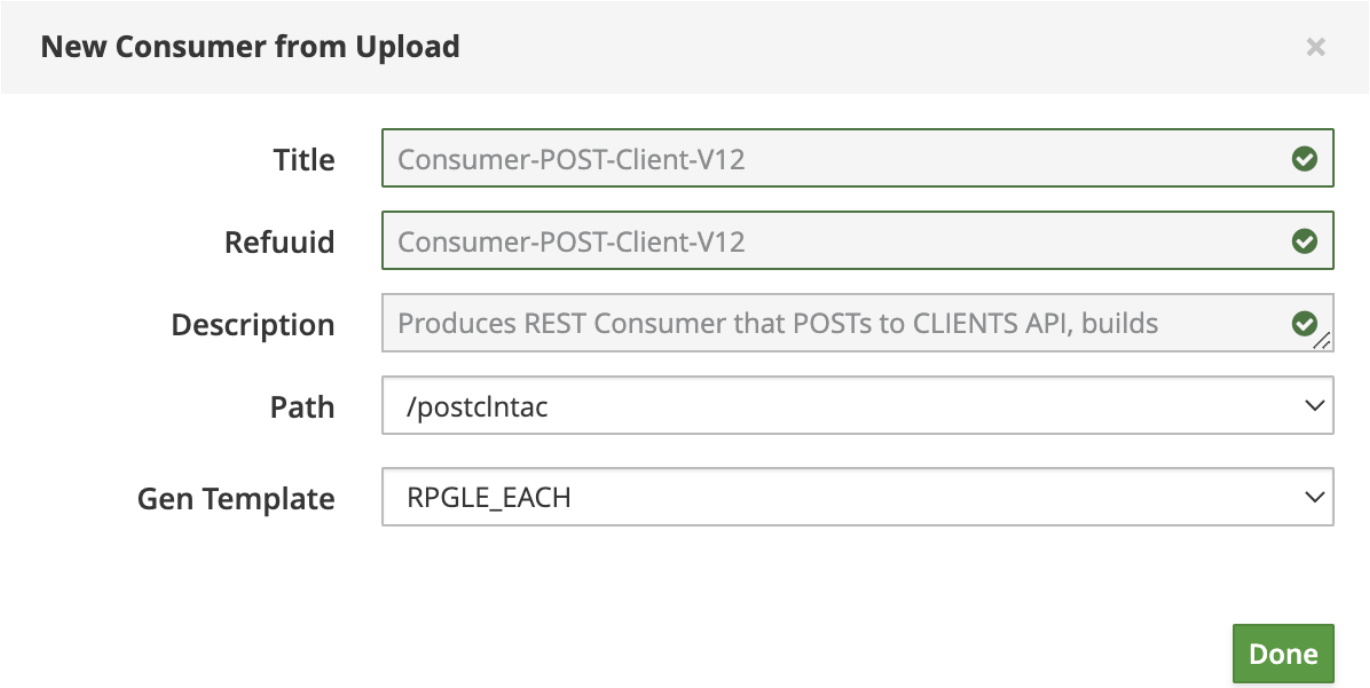
All the folders and the sub-folders (if any) and the APIs would be listed in hierarchical order.



Let us, for example, select the option 3 from the list.



Click the **Next** button.



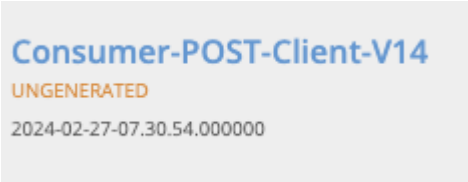
Edit the title and Refuuid and remove the invalid characters, if any.

Click "Done". The import is done and the General tab screen now appears!

Main Content List

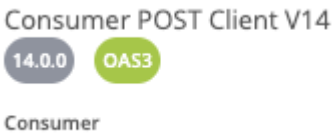
LEFT MAIN CONTENT

Shows the Entity name, Generated status, last generated date/time



CENTER MAIN CONTENT

Shows the title,version, the OAS type, consumer or Provider type



RIGHT MAIN CONTENT

DELETE A SPEC

Use the  button to delete a spec.

Helper buttons

MDRest4i supports iAsp, IFS and Source file options for compilation.

To allow for this a member **APIPGM** in source file **QRPGLESRC** in library **APISRCLIB**, must be entered using this IFS syntax:

```
/QSYS.LIB/APISRCLIB.LIB/QRPGLESRC.FILE/APIPGM.MBR
```

The same source file in the folder APISRC would use this syntax:

```
/APISRC/APIPGM.RPGLE
```

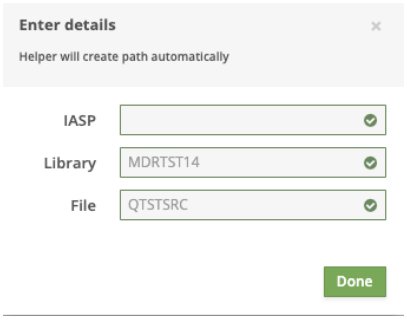
Using this format, an object path would look like this:

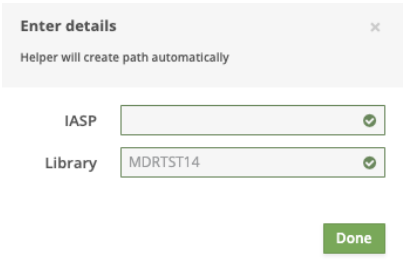
```
/QSYS.LIB/YOURLIB.LIB/APIPGM.PGM
```

For those users not familiar with this syntax for lib/file/member or objects, a 

popup has been created.

This popup allows the traditional manner of editing these details.





Unless you are using a specific iASP, this value can be left blank for *SYSBAS iASP.

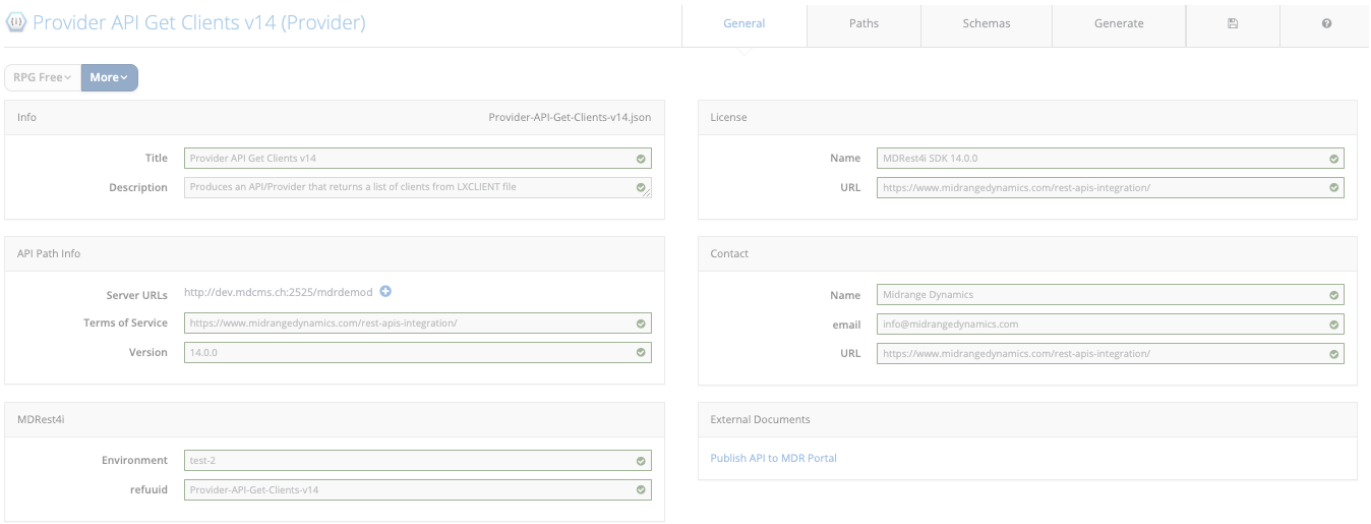
The previous form and underlying OAPI/SWAGGER are updated saved after the **Done** button is clicked.

1.4.5 Spec Editor

General Tab - Spec Editor

The General Screen displays the details of Consumer / Provider spec selected from the List of Specifications on the Specs page. This screen also comes up when a new Provider / Consumer is created.

GENERAL SCREEN - PROVIDER



Buttons

We have 2 buttons on the left side of the screen (Figure 2)

RPG Free	Enabled if the API is generated and can be used to view the RPG code of the option selected from the dropdown list
More	
Upload	Import a Provider Spec
Save	Save the changes
Save As	Save with a different refuuid
Move to user	Move the spec to another user
Rename	Rename the spec (changes the refuuid)
Download	Download the json spec

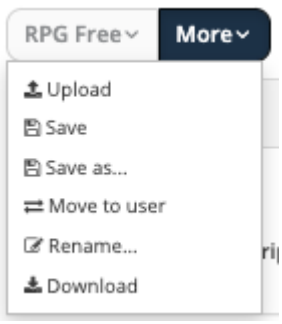


Figure 2: Buttons

GENERAL SCREEN - CONSUMER

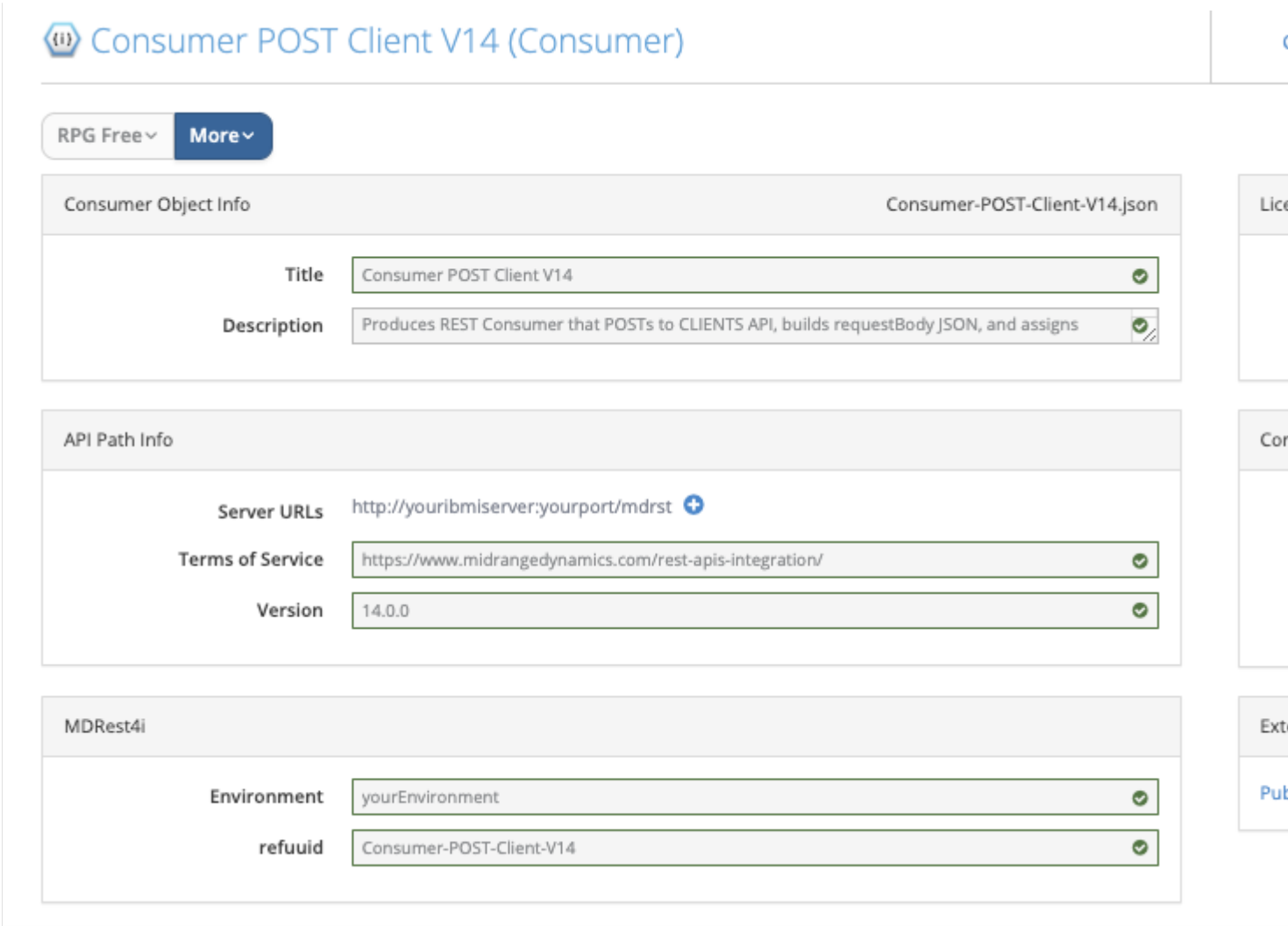


Figure 3: General Screen-

The only thing that needs updating is the server URLs. Add the host name, Port number and library name used when setting up the iCore HTTP server during installation. The default value is picked up from the value set in the **User Profile**. The rest is general information and can be left as is.

Buttons

We have 2 buttons on the left side of the screen (Figure 2)

RPG Free	Enabled if the API is generated and can be used to view the RPG code of the option selected from the dropdown list
More	
Upload	Import a Consumer Spec
Save	Save the changes
Save As	Save with a different refuuid
Move to user	Move the spec to another user
Rename	Rename the spec (changes the refuuid)
Download	Download the json spec

THE EXTERNAL DOCUMENTS

MDR Portal

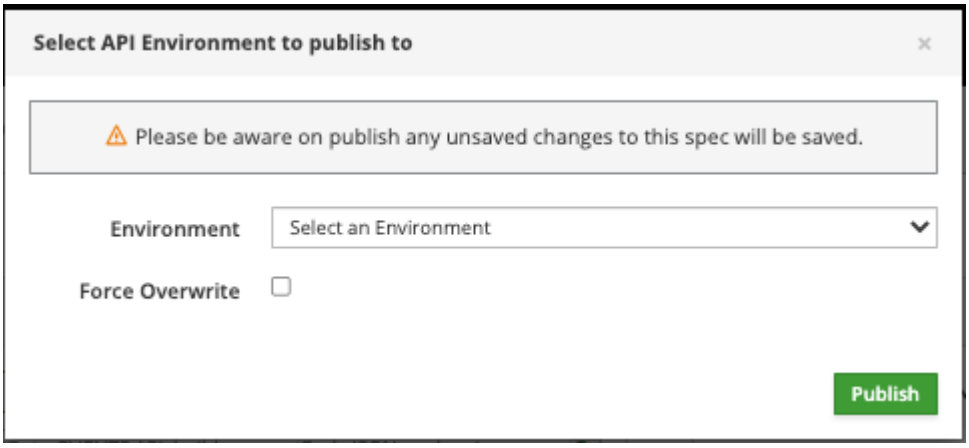


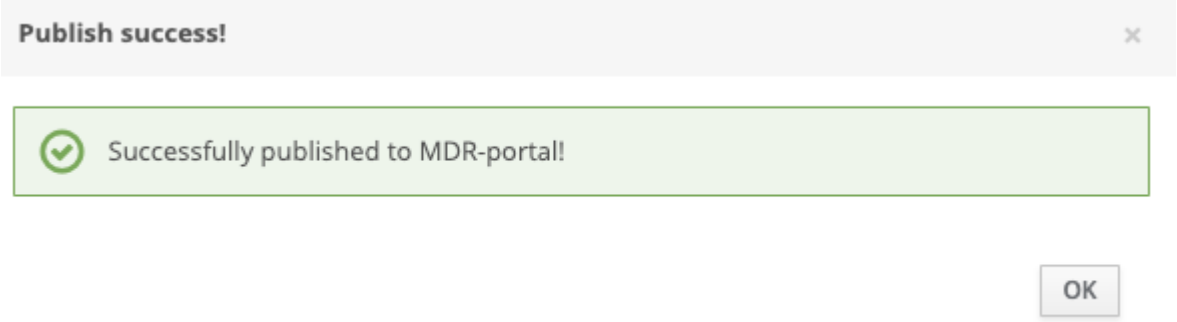
Figure 9(a): Publish to Portal

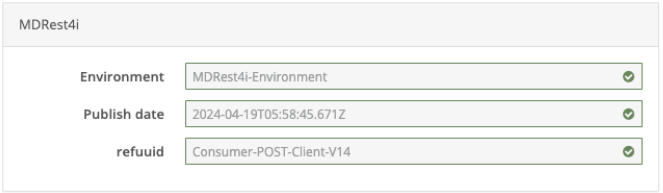
Select the Environment from the dropdown list and click 

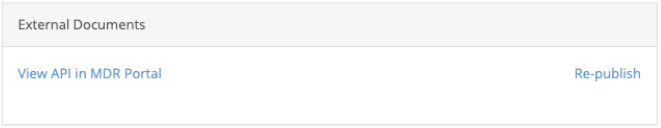
If an older version of the document is already published on the documenter, you get an error message

Figure 9(b): Publish to Portal

If we check the "Force Overwrite" checkbox, then the copy on the MDR Portal (MDRESt4i-SDK-Documenter) is over-written.







Once the API is published, the link

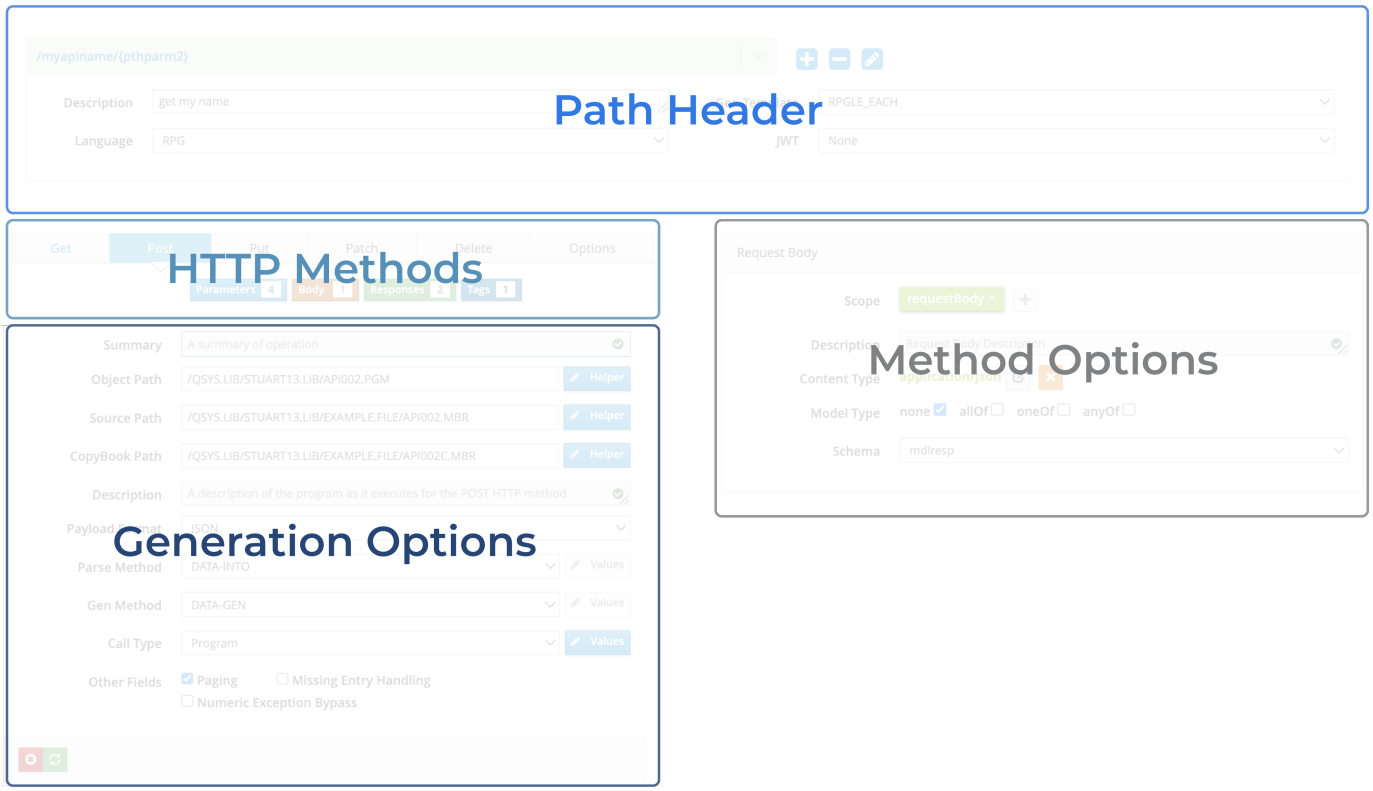
Publish API to MDR Portal changes to View API in MDR Portal

Provider Path Tab

SCREEN LAYOUT

The screen below appears when editing a Provider spec and the "Path" tab is selected.

Each section controls a different aspect of OAPI/SWAGGER editing, and providing details for the RPG/COBOL generators.



Path Header - adding, deleting, selecting paths from the OAPI/SWAGGER specification.

HTTP Methods - Select or enable HTTP methods, and control what appears in the Method Options section such as parameters, response payload details etc.

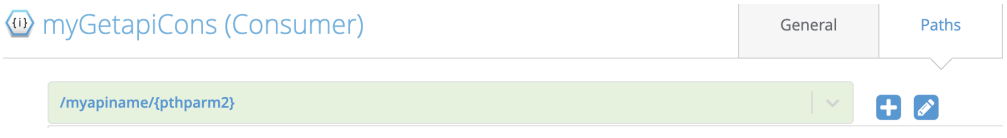
Generation Options - Edit details that control the specifics of the generation of provider programs.

Method Options - Maintain details related to parameters, response body, request body, and tags for each path/method combination. Activated by selecting a button from the HTTP methods section.

PATH HEADER

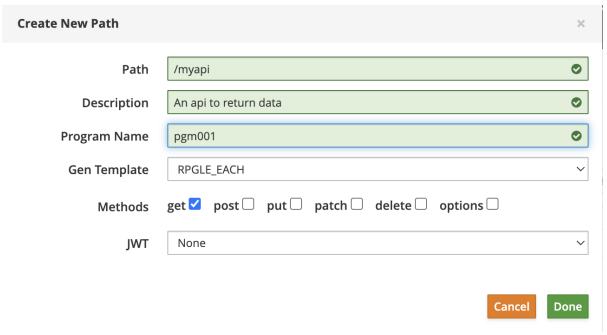
Select a path

The Paths are shown in the dropdown list on the left hand side.



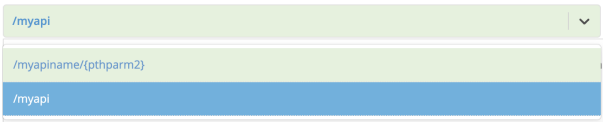
Add a New Path

Click the  button to add a New path.



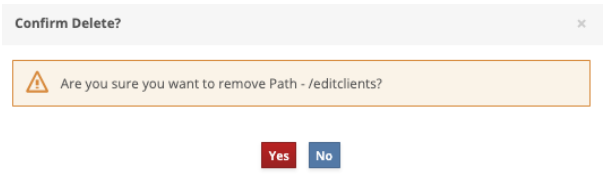
Field	Description
URI Path	Enter path that will be used at runtime to call the program via the HTTP server API request. For details on hwo to map URI's to programs see the MDRSTCFG command documentation.
Description	Description of the path. .This is used as the text for the generated source member and the compiled program
Program Name	Name of the program.
Gen Template	This determines what generation pattern will be used. Possible values are: **RPGLE_EACH** - One program per path method combination in RPGLE. **COBOL_DFT** - One program per path method combination in COBOL ILE
Methods	Select the methods by clicking the checkboxes
JWT	Possible options: None, Yes and Remote

Click the Done button, and the path is added to the underlying OAPI/SWAGGER, and the dropdown list updated.



Delete a path

Click the button to delete the selected path.



Select "Yes" to delete the path.

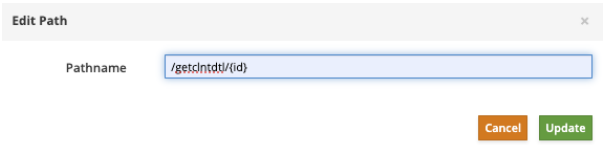
Edit a path name

To change the pathname, click the pencil icon in the button bar. A popup window appears on click.

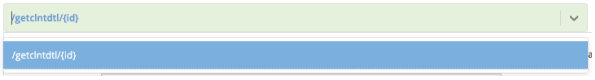
Make the necessary changes and press the Update button. This will update corresponding in the underlying OAPI/SWAGGER.

Adding Path Parameters

Path parameters can be added to the path name by enclosing the path parameter in curly braces as shown in the example below:



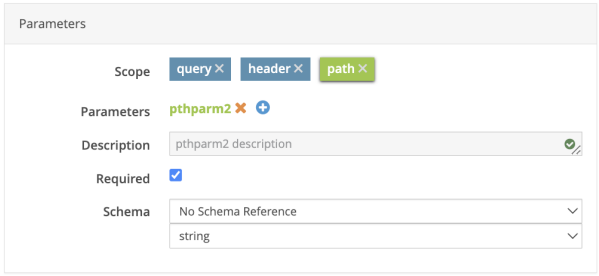
Click Update and the paths list gets updated as:



This will add an in: Path parameter to the requests section of the underlying OAPI/SWAGGER:

```
- in: path
  name: id
  description: id description
  required: true
  schema:
    type: string
```

This parameter will also appear in the HTTP Options when the parameters button is selected:



This tells the generator to add code in the provider program, to extract the value of the path parameter:

```
// Get Path Parameters
if MDR_getPathVar(handle: 'myapiname':1) <> '*NOTFOUND';
  pthparm2 = MDR_getPathVar(handle: 'myapiname':1);
else;
  errorMsg = 'Mandatory path parameter: pthparm2 not supplied';
  MDR_setError( handle: 'API0002': errorMsg: 10000: 20 : 400);
  *Inlr = *On;
  Return;
Endif;
```

Path Header Options

This section below the path editing line, includes more details for the selected path. These details correspond to the "x-mdrGen" object in the OAPI section of the selected path.

Option	Description
Description	A text description of the path. This description is used to as the text for the generated source member and object.
Gen Template	Specifies which generation template is used to generate the code in what language. **RPGLE_EACH** - One program per path per HTTP method is generated in RPG Free. **COBOL_DFT** - One program per path per HTTP method is generated in ILE COBOL.
Program	This is the IFS formatted path to the Library/object name of the prgram to be generated. The library and object names can be edited using the Helper popup. **EXAMPLE**: `/QSYS.LIB/YOURLIB.LIB/APIPGM.PGM`
JWT	Specifies if generator should add logic to retrieve and validate the Authorization header using MDRest4i JWT functions.

Important note about MDRDCFG records

The "Program" field in the header options and the "Object Path" in the generation options determine how the generator adds records to MDRDCFG file.

If the "Object Path" in the generation options has a value in it, an entry for this specific METHOD is added to the MDRDCFG file.

If the "Program" field in the header options has a value, and the "Object Path" in the generation options does not, then a record is added to the MDRDCFG file with a blank method.

At runtime when MDRAPI reads MDRDCFG to find which program to call, it first checks for the specific path and method provided in the request, and then if it doesn't find it, it looks for a the path and a blank method.

For more information about maintaining records path to program mapping and API logging, see URI to API/Provider Program Mapping

HTTP METHODS

Shows the lists of HTTP methods as different tabs

Enable Method

To enable additional methods, select the method required and, if this method is not found in the OAPI/SAWGGER for this path, select the Enable xxxx Method button for that method: Enable patch method

This will add this method, with default values from the Provider template for that method, to the OAPI/SWAGGER and update the screen, to show Generation Options for that method.

GENERATION OPTIONS FOR PROVIDER

All these values are added/edited in the "**mdrGenOptions**" object of the selected method inside for the "**x-mdrGen**" object for the selected path.

Note

MDRest4i supports iAsp, IFS and Source file options for compilation.

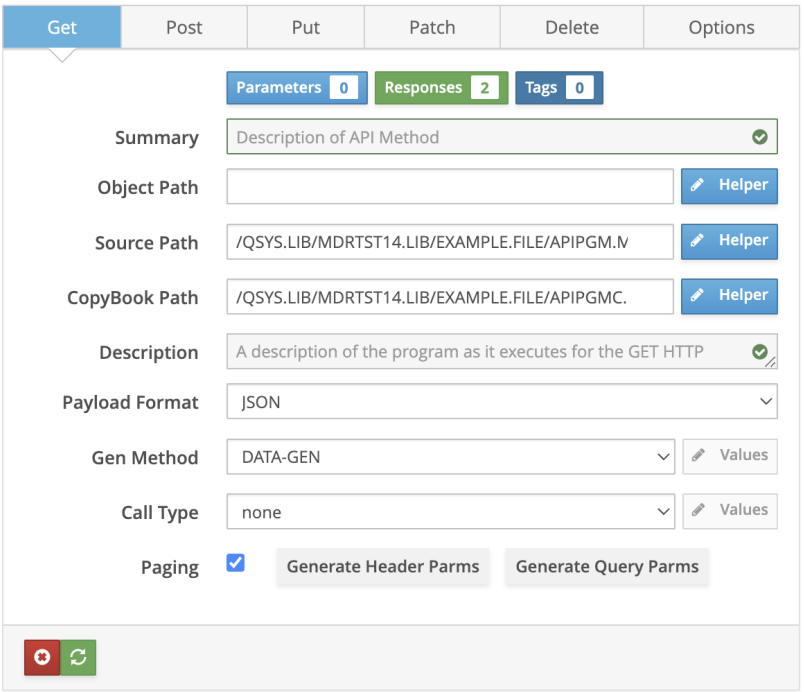
To allow for this a member **APIPGM** in source file **QRPGLESRC** in library **APISRCLIB**, must be entered using this IFS syntax:

`/QSYS.LIB/APISRCLIB.LIB/QRPGLESRC.FILE/APIPGM.MBR`

For those users not familiar with this syntax for lib/file/member or objects, a **Helper** popup has been created.

This popup allows the traditional manner of editing these details.

Generation Options for a Provider



Summary

Short description of the API path. Used for documentation purposes only.

Object Path

This is the IFS formatted path to the Library/object name of the API. The library and object names can be edited using the Helper popup.

EXAMPLE: `/QSYS.LIB/YOURLIB.LIB/APIPGM.PGM`

Source Path

Location of source file/mbr for the Provider program.

EXAMPLE `/QSYS.LIB/STUART13.LIB/EXAMPLE.FILE/API002.MBR`

Copybook Path

Location of copy book source file/mbr for payloads. The generator will create all payload data structures in this copybook, and add the `/copy` statement to the copybook in the generated provider.

EXAMPLE `/QSYS.LIB/STUART13.LIB/EXAMPLE.FILE/API002.MBR`

Description

Description of the selected method for that Provider

Payload Format

This can be JSON, XML, or TXT. The generator will create logic to write or parse the correct payload syntax

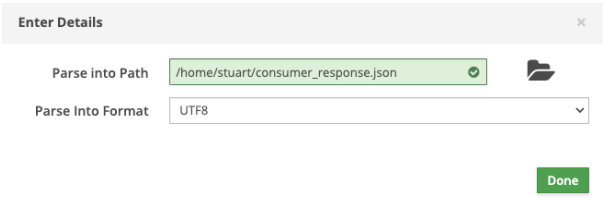
Parse Method

Determines what parsing functions are used to process the inbound payload in the generated code. Select DATA-INTO, JPATH(used for mdr_JSONPATHV etc), or IFS.

If IFS is selected, the **path** can be entered using the 

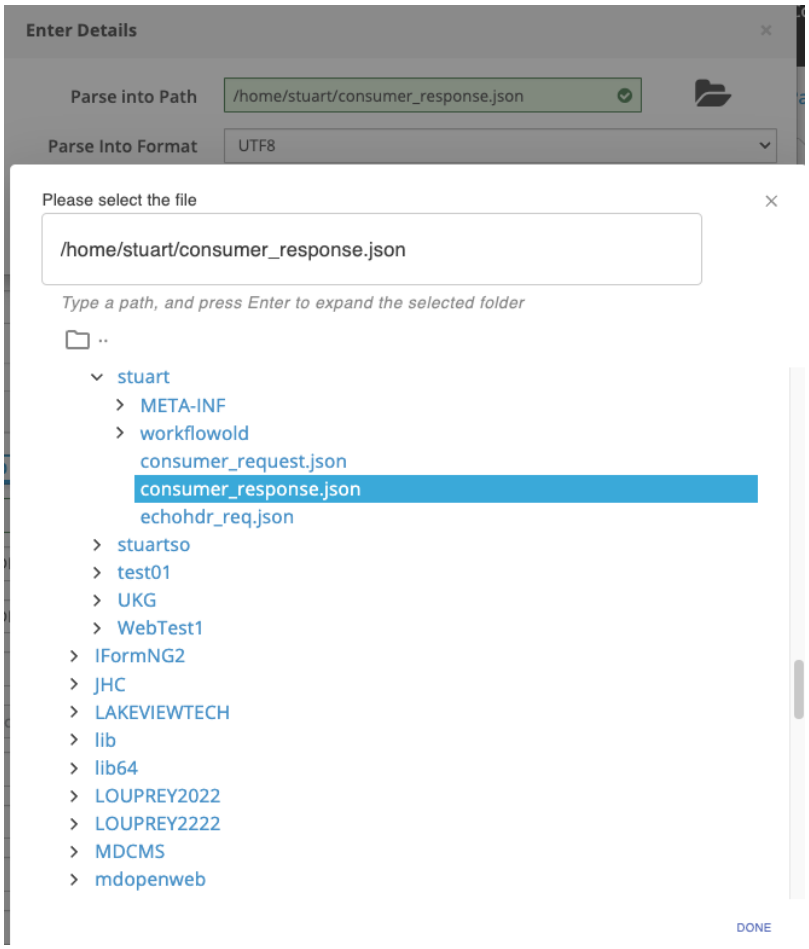
button.

This will display a popup that allows the adding of path name to the IFS object to be used in the program.



 button is provided next to the input box to use the File explorer to easily select the desired path.

The File explorer opens when the button is clicked.



Gen Method

Determines how the outbound payload is handled in the generated code.

Select DATA-GEN, MDR-YAJL(used for mdr_addchar() etc), or IFS.

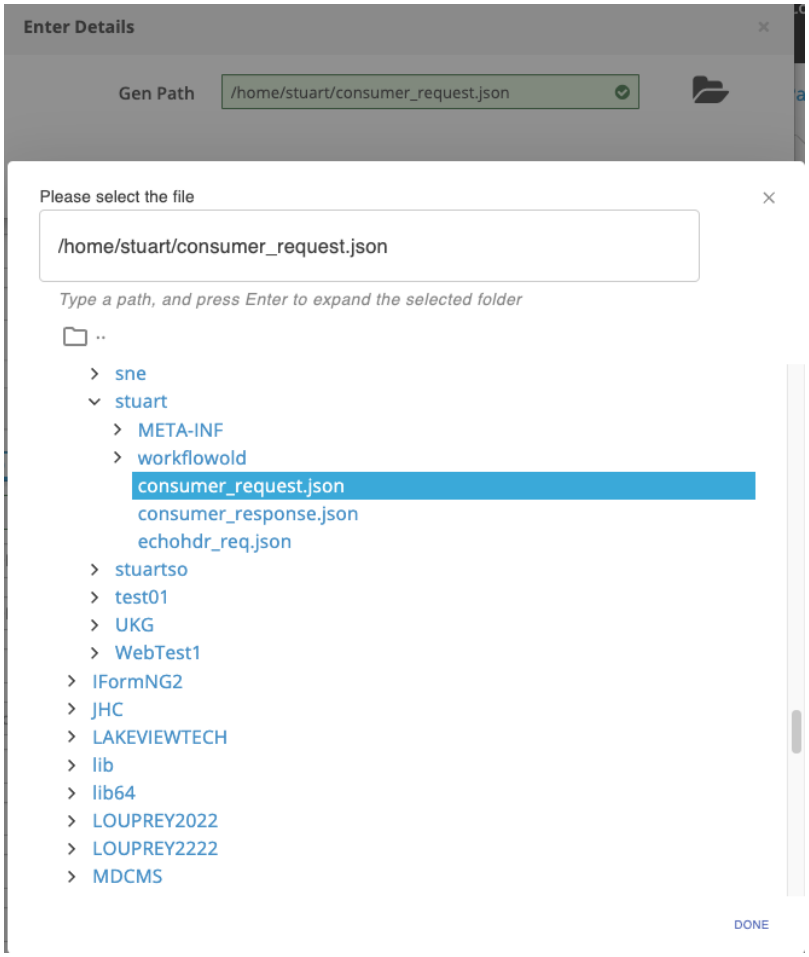
If IFS is selected, the values can be entered using the 

button.

This will display a popup that allows the adding of path name to the IFS object to be used in the program.

 button is provided next to the input box to use the File explorer to easily select orthe desired path.

The File explorer opens when the button is clicked.



CALL TYPE

Generator creates logic for calling a module/program/service-program from a provider. Options available: none, Program, Module, Service Program

Selecting any option other than "none", enables the 

button which can be used to enter the details used by the generator to create the call logic.

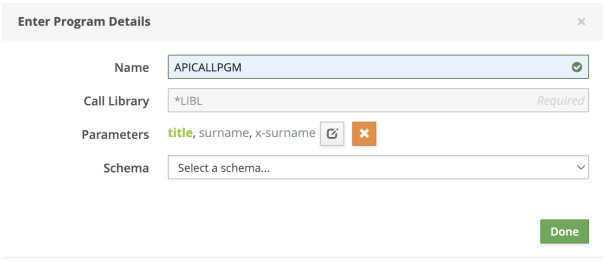
Each of the call types specified below allows the entry of what to call, and what parameters should be passed with the call.

The provider generator will add the necessary prototypes, bindings and calls to these objects.

Each type shows a different popup, and generates different code in the provider program:

- Program
- Module
- Service Program

Program

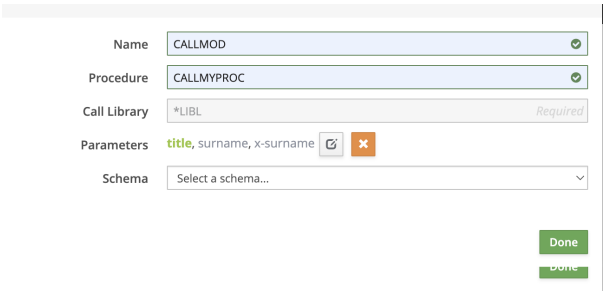


Name	Enter name of **program** to be called.
Call Library	Library name of *LIBL
Parameters	See Adding Call Parameters below to add parameters to the generated call.
Schema	Only applies to HTTP methods POST, PUT, PATCH. If a schema has been assigned to the request body, the request body schema name will appear. See Passing Schemas below to add schema parameters to the generated call.

The code generated for a program call is as follows:

```
dcl-pr APICALLPGM Extpgm('APICALLPGM');
  *n like(title);
  *n like(surname);
  *n like(x_surname);
end-pr;
...
callp(e) APICALLPGM(title:surname:x_surname);
```

Module

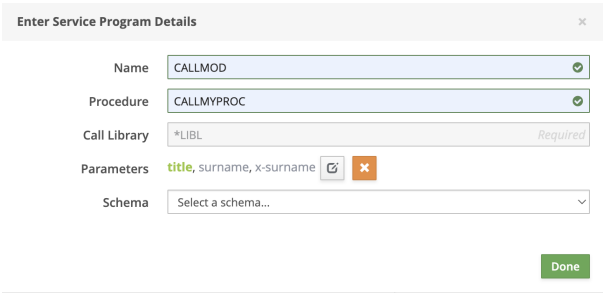


Name	Enter name of the **module** to be bound in to the provider program.
Procedure	Enter name of procedure to be called by the provider program.
Call Library	Library name of *LIBL
Parameters	See Adding Call Parameters below to add parameters to the generated call.
Schema	Only applies to HTTP methods POST, PUT, PATCH. If a schema has been assigned to the request body, the request body schema name will appear. See Passing Schemas below to add schema parameters to the generated call.

The code generated for the call is as follows:

```
dcl-pr CALLMYPROC Extproc('CALLMYPROC');
  *n like(title);
  *n like(surname);
  *n like(x_surname);
end-pr;
...
callp(e) CALLMYPROC(title:surname:x_surname);
```

Service Program



Name	Enter name of **service program** to be bound in to the provider program.
Procedure	Enter name of procedure to be called by the provider program.
Call Library	Library name of *LIBL
Parameters	See Adding Call Parameters below to add parameters to the generated call.
Schema	Only applies to HTTP methods POST, PUT, PATCH. If a schema has been assigned to the request body, the request body schema name will appear. See Passing Schemas below to add schema parameters to the generated call.

The code generated for the call is as follows:

```
dcl-pr CALLMYPROC Extproc('CALLMYPROC');
  *n like(title);
  *n like(surname);
  *n like(x_surname);
end-pr;
...
callp(e) CALLMYPROC(title:surname:x_surname);
```

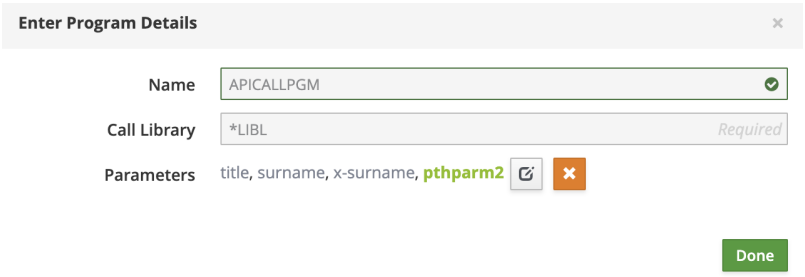
Adding Call Parameters

Provider parameters including query, path and headers can be passed to the called pgm/mod/srvpgm. The SDK generator will add logic to extract these parameters, and assigns them to the call wheer specified

Add these using the select parameters  button.

When this is clicked, any inbound parameter defined for this path and method (including query parameters, headers and path parameters) will be displayed as a list from which any can be selected. For example:

Once selected and confirmed with the  button, the screen is updated, and the underlying OAPI/SWAGGER is updated in mdrGenOptions for that method:



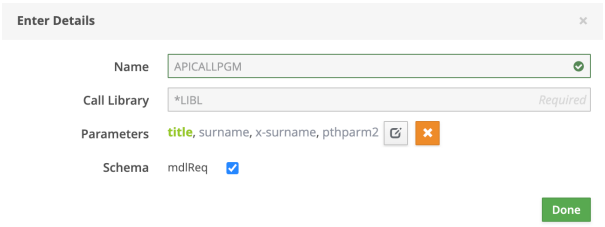

```
called:
  callname: APICALLPGM
  callparms:
    inpth:
      - pthparm2
    inqry:
      - title
      - surname
    inhdr:
      - x-surname
```

These parameters can be removed individually using the  , while the parameter is selected in green.

Passing Request Schema

The request body schema (POST, PUT, PATCH methods) can be passed as a parameter to the called program/procedure.

This adds the request body data structure (created from the request body schema) to the end of the call to the specified program/module/service pgm.

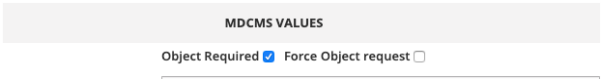



```
called:
  callname: APICALLPGM
  callparms:
    inpth:
      - pthparm2
    inqry:
      - title
      - surname
    inhdr:
      - x-surname
    schema: '#/components/schemas/mdlReq'
    schemaFlds:
      - '*SCHEMA'
```

METHOD OPTIONS

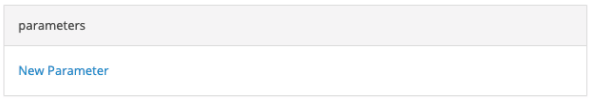
For the **Get** & **Delete** Methods, the following buttons appear:

Other methods (**POST**, **PUT**, **PATCH**) display the following buttons:

Option	Description
Parameters	Select to display and edit request parameters
Body	Select to display and edit request body details. (only for **POST**, **PUT**, **PATCH** method requests)
Responses	Select to display and edit response body details.
Security	Select to display and edit response body details.
Tags	To create tags
Object Request	For MDCMS users. This appears only if “Object Required" box is checked in the Edit Site 

Parameters

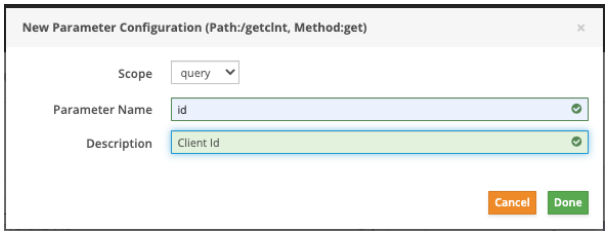
Click on the  button



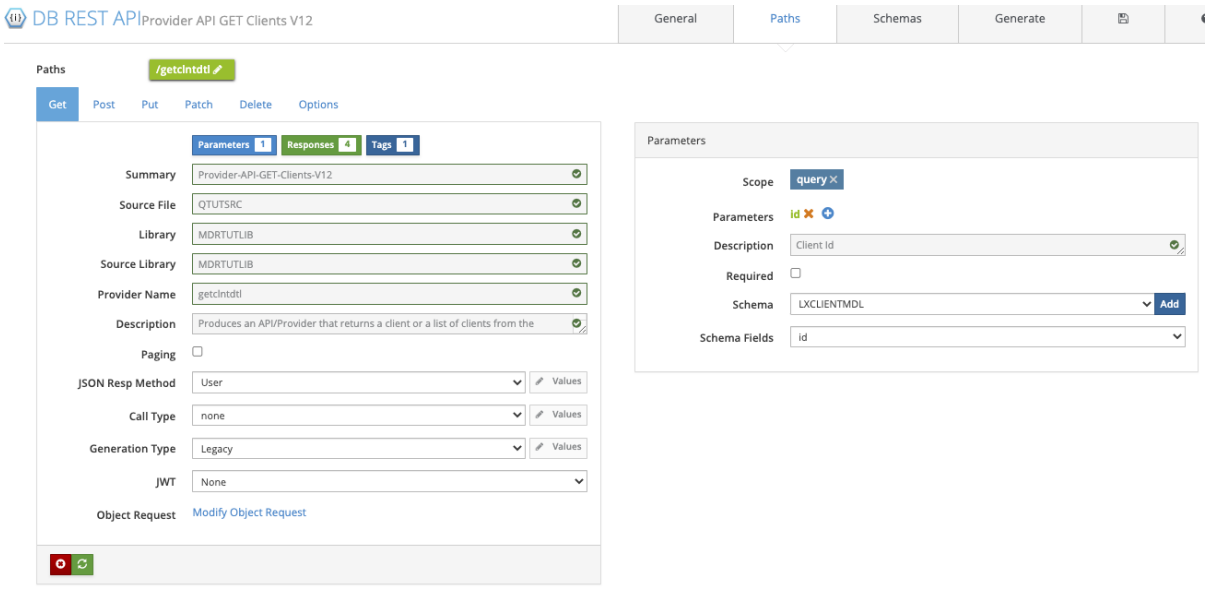
To adda new parameter, click on the  link

A new popup window appears

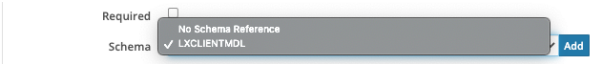
Select the scope (query/header). Enter the parameter name and description and click 



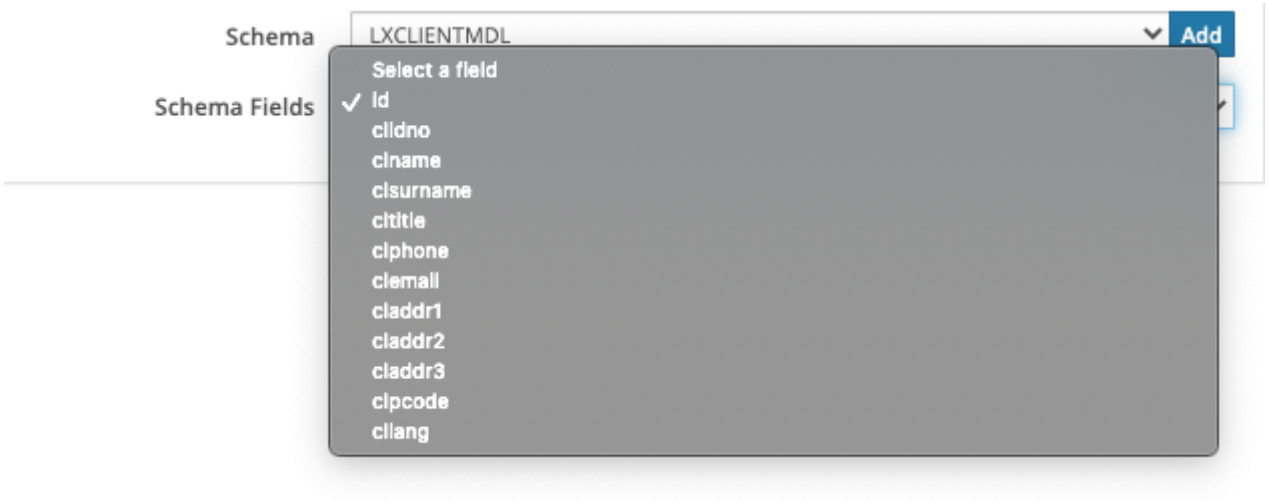
The "Paths" tab screen now looks like:



The schema can be selected from the dropdown list of schemas created using the schema tab.



On selecting the schema, the list of fields automatically is displayed in the next input field, Schema Field.

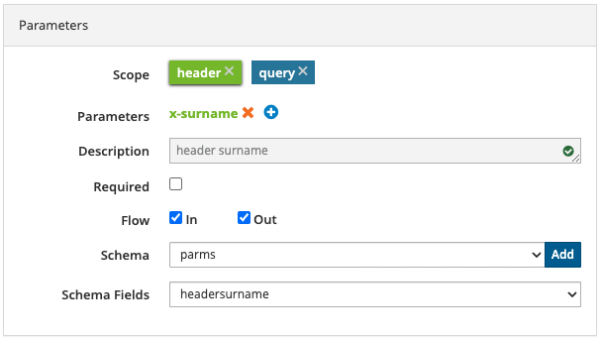


The Swagger definition now adds the "parameters" field .

```yaml
parameters:
  - in: query
    name: id
    description: Client Id
    required: false
    schema:
      x-schemaFld: id
      x-schema: '#/components/schemas/LXCLIENTMDL'
```

Here, the Swagger Extension fields x-schema, x-schemaFld are used.

In case of "header" type parameters, an additional input "flow" is required.




```yaml
parameters:
  - in: header
    name: x-surname
    description: header surname
    required: false
    schema:
      x-schemaFld: headersurname
      x-schema: '#/components/schemas/parms'
    x-flow: both
```

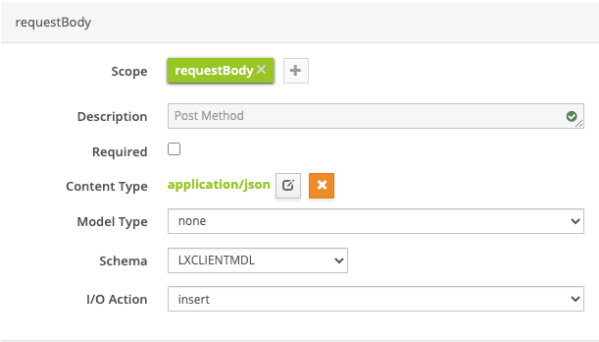
The Swagger Extension field x-flow is used which can have the value "in", "out" or "both" depending on the boxes checked.

Body

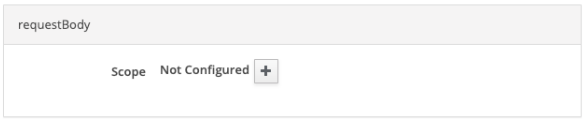
Click on the 

button to edit the request body. The "Body" option is available only for "**POST**", "**PUT**" and "**PATCH**" methods.

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.

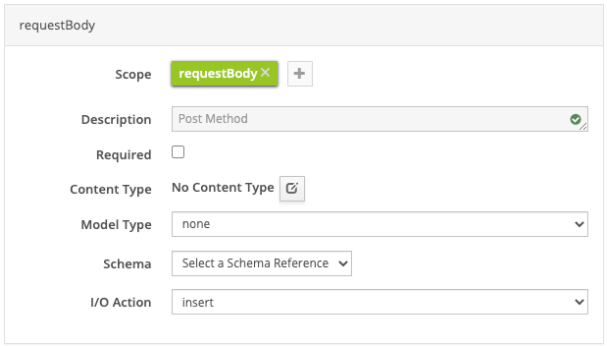


When no request body has been created, the section looks like:



Click on the  button to create one.

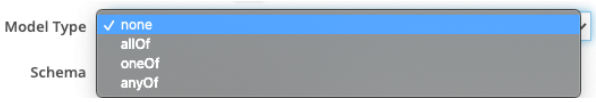
Click 



Select the Content Type as application/json from the dropdown list.

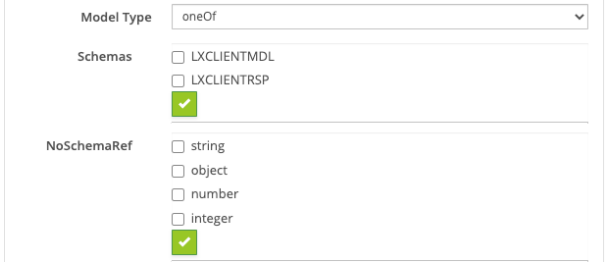
Select the Model type from the dropdown provided which lets you select if the type: allOf, anyOf, oneOf, if you wish to combine schemas.

- `oneOf` – validates the value against exactly one of the subschemas
- `allOf` – validates the value against all the subschemas
- `anyOf` – validates the value against any (one or more) of the subschemas

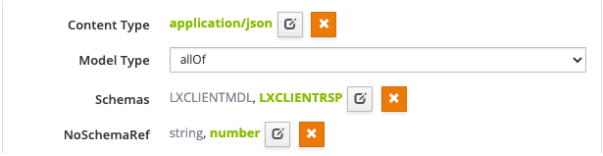


Selecting none lets you select a single schema.

Selecting oneOf, allOf, anyOf brings up the entry fields to select/deselect the multiple schemas and the multiple no-schema reference objects too. Click the  button to close the dropdown list.



Select the schemas and no schema references, if any.



The following object: requestBody is added to the swagger.

```
requestBody:
  x-ioAction: insert
  description: Post Method
  content:
    application/json:
      schema:
        allOf:
          - $ref: '#/components/schemas/LXCLIENTMDL'
          - $ref: '#/components/schemas/LXCLIENTRSP'
          - type: string
          - type: number
```

Select the schema from the dropdown list which shows the list of schemas created in the schema tab.

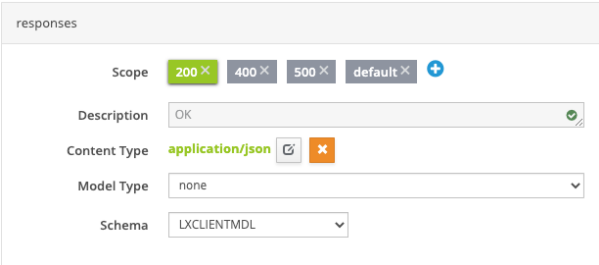
Following object: requestBody is added to the swagger.

```
requestBody:
  x-ioAction: insert
  description: Post Method
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/LXCLIENTMDL'
```

Responses

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.

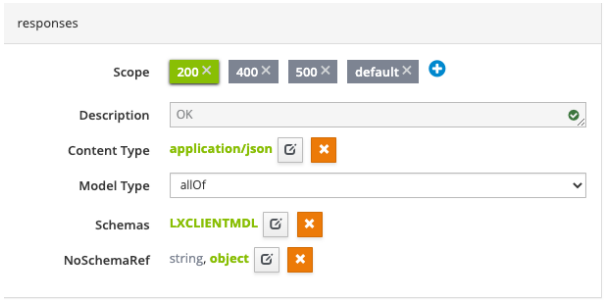
From the “Responses” section on the right hand side, select the content type, Model Type and then select a schema(s) created earlier from the Schema drop down depending on the model type selected.



The "responses" object is updated like so:

```
responses:
  '200':
    description: OK
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/LXCLIENTMDL'
  '400':
    description: Bad Request from GET Request
  '500':
    description: Internal Server Error
  default:
    description: Unexpected error occurred
tags:
```

For model type other than "none":




```
responses:
  '200':
    description: OK
    content:
      application/json:
        schema:
          allOf:
            - $ref: '#/components/schemas/LXCLIENTMDL'
            - type: string
            - type: object
  '400':
    description: Bad Request from GET Request
  '500':
    description: Internal Server Error
  default:
    description: Unexpected error occurred
```

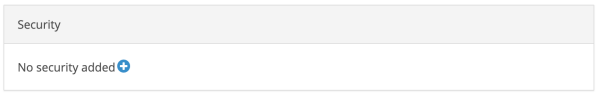
Security

Click on the 

button to edit the request body.

When this button is selected the “Security” section appears.

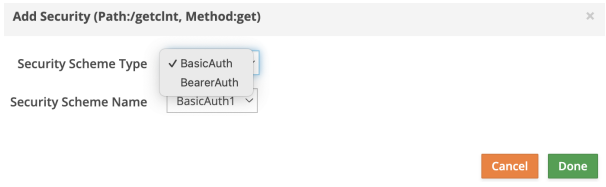
If no security schemes have been added to the method, the section looks like:



Click on the  button to create one.

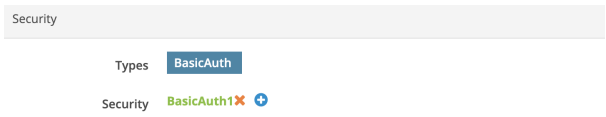
Note

You can add a security scheme only when the global security schemes have been created using the Security tab.



Click 

button



The umderlying Swagger gets updated as:

```
get:
      ...
```

```
    security:
      - BasicAuth1: []
```

More security schemes can be added using the "plus" button.

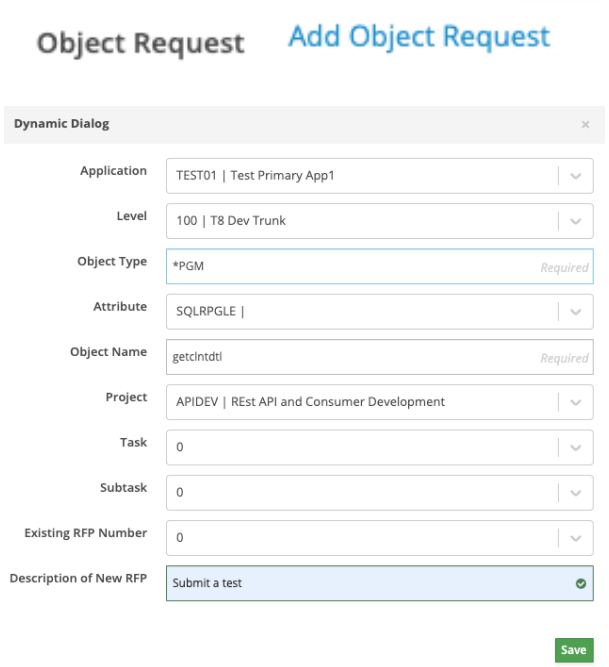
Tags

The "tags" array is now added to the Swagger definition.

Use the  button to create more tags. The first one is created by default and takes the program name.

Object Request

When selected the dynamic dialogue below appears. The fields are initialised with the default values fetched from the IBMi server using the Rest API.



Click 

This updates the swagger spec as per the section below:

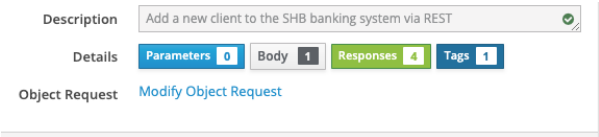
```
x-mdcms:
  vref: '*MDR'
  user: RITUK
  appl: TEST01
  lvl: '100'
  objt: '*PGM'
  attr: SQLRPGLE
  objn: getclntdtl
  folb: MDRSTT14
  fsfl: EXAMPLE
  fslb: MDRSTT14
  proj: APIDEV
  task: '0'
  stsk: '0'
  arfp: '*AUTO'
  rfp: ''
  rfpd: Submit a test
```

Read more about **x-mdcms** here

The values can be edited and saved.

The Add Object Request changes to Modify Object Request

now which lets you modify the object definition anytime.



HELPER BUTTONS

MDRest4i supports iAsp, IFS and Source file options for compilation.

To allow for this a member **APIPGM** in source file **QRPGLESRC** in library **APISRCLIB**, must be entered using this IFS syntax:

`/QSYS.LIB/APISRCLIB.LIB/QRPGLESRC.FILE/APIPGM.MBR`

The same source file in the folder APISRC would use this syntax:

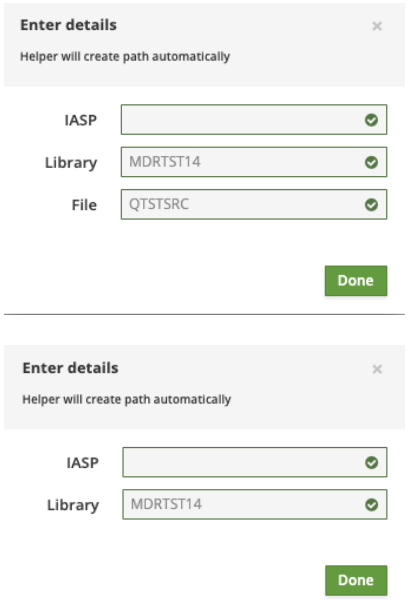
`/APISRC/APIPGM.RPGLE`

Using this format, an object path would look like this:

`/QSYS.LIB/YOURLIB.LIB/APIPGM.PGM`

For those users not familiar with this syntax for lib/file/member or objects, a Helper popup has been created.

This popup allows the traditional manner of editing these details.



Unless you aer using a specific iASP, this value can be left blank for *SYSBAS iASP.

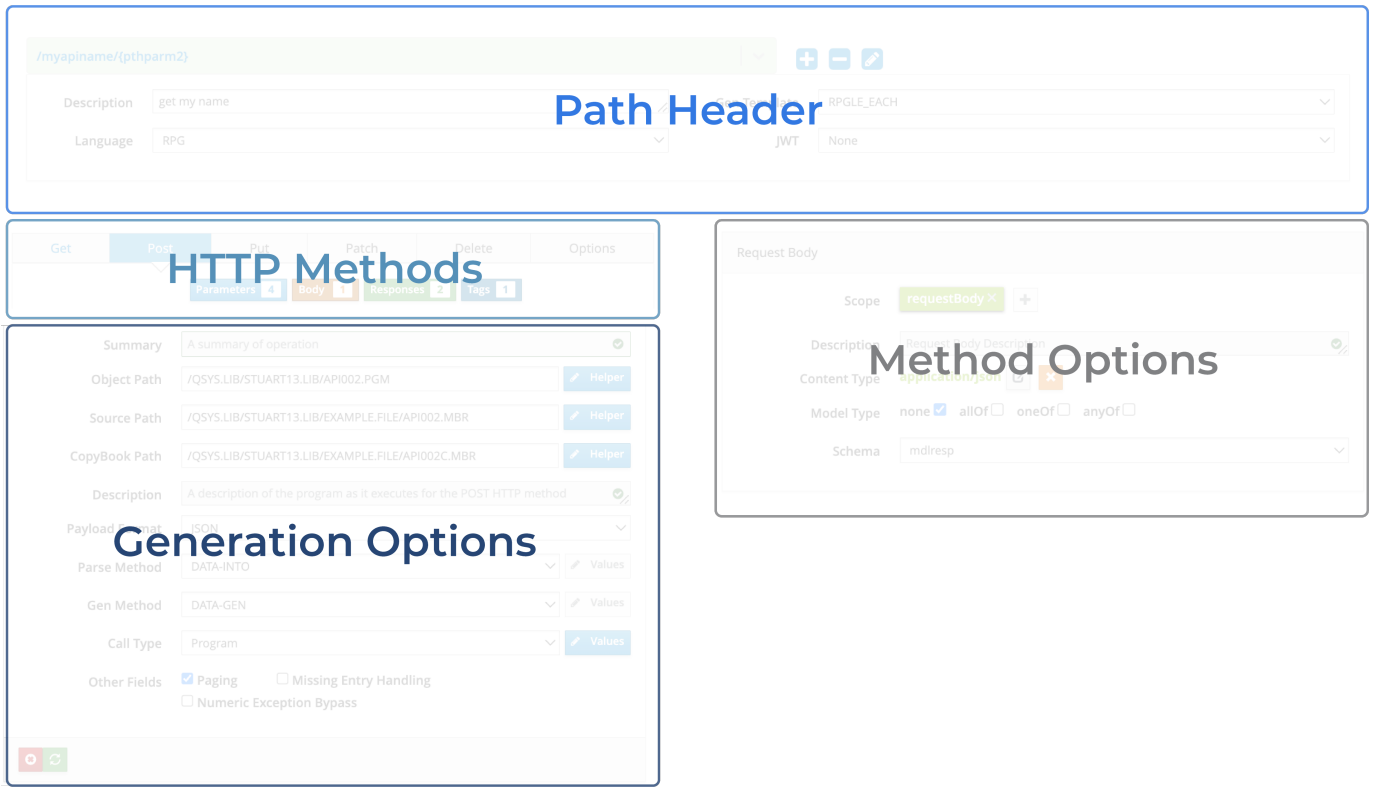
The previous form and underlying OAPI/SWAGGER are updated saved after the **Done** button is clicked.

Consumer Path Tab

SCREEN LAYOUT

The screen below appears when editing a Consumer spec and the "Path" tab is selected.

Each section controls a different aspect of OAPI/SWAGGER editing, and providing details for the RPG/COBOL generators.



Path Header - adding, deleting, selecting paths from the OAPI/SWAGGER specification.

HTTP Methods - Select or enable HTTP methods, and control what appears in the Method Options section such as parameters, response payload details etc.

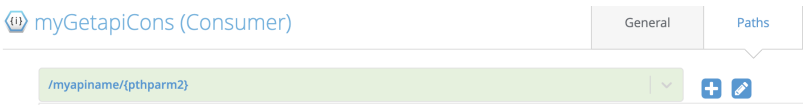
Generation Options - Edit details that control the specifics of the generation of Consumer programs.

Method Options - Maintain details related to parameters, response body, request body, and tags for each path/method combination. Activated by selecting a button from the HTTP methods section.

PATH HEADER

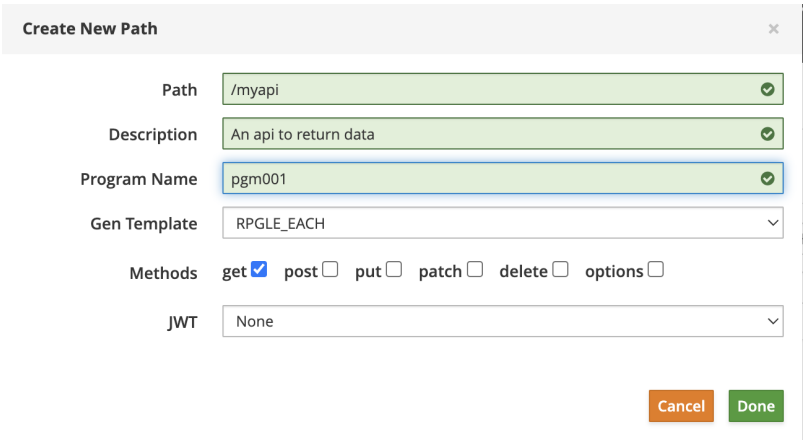
Select a path

The Paths configured in the OAPI/SWAGGER spec are displayed in the dropdown list on the left hand side.



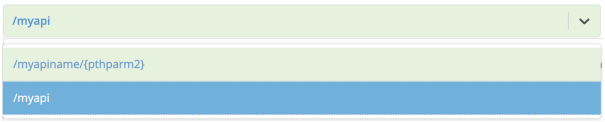
Add a New Path

Click the  button to add a New path.



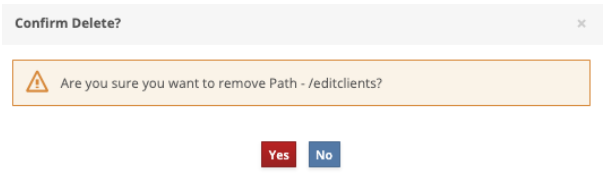
Field	Description
URI Path	Mandatory path that will be used to call the remote API
Description	Description of the path
Program Name	Name of the program. / becomes the path if URI path is left blank
Gen Template	This determines what generation pattern will be used. Possible values are: **RPGLE_EACH** - One program per path method combination in RPGLE. **COBOL_DFT** - One program per path method combination in COBOL ILE
Methods	Select the methods by clicking the checkboxes
JWT	Possible options: None, Yes and Remote

Click the Done button, and the path is added to the underlying OAPI/SWAGGER, and the dropdown list updated.



Delete a path

Click the button to delete the selected path.



Select "Yes" to delete the path.

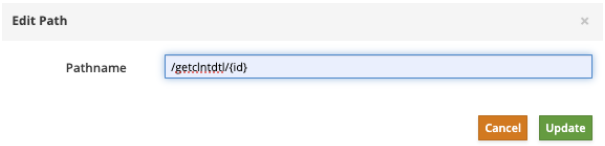
Edit a path name

To change the pathname, click the pencil icon in the button bar. A popup window appears on click.

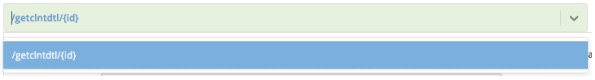
Make the necessary changes and press the Update button. This will update corresponding in the underlying OAPI/SWAGGER.

Adding Path Parameters

Path parameters can be added to the path name by enclosing the path parameter in curly braces as shown in the example below:



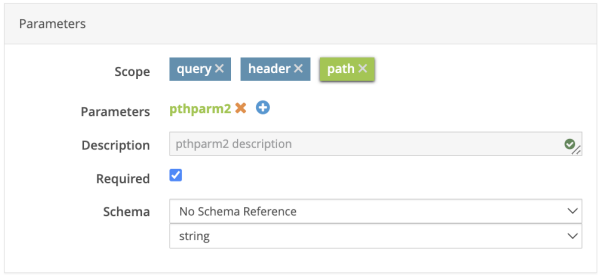
Click Update and the paths list gets updated as:



This will add an in: Path parameter to the requests section of the underlying OAPI/SWAGGER:

```
- in: path
  name: id
  description: id description
  required: true
  schema:
    type: string
```

This parameter will also appear in the HTTP Options when the parameters button is selected:

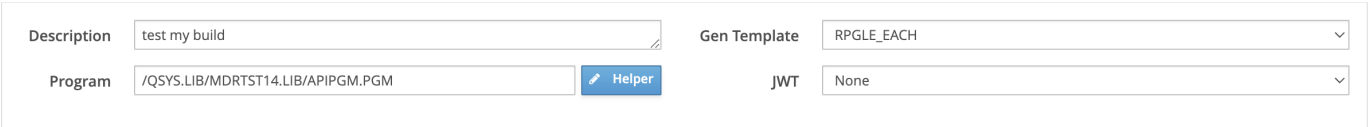


This tells the generator to add function MDR_encode() in the consumer program, to add this parameter to the URI in the request:

```
// Path parameter variables.
dcl-s  pthparm2  varchar(10);
...
// Set the PATH parameters
MDR_encode(handle : encodeVar : %trimr(pthparm2));
uri += '/' + encodeVar;
```

Path Header Options

This section below the path editing line, includes more details for the selected path. These details correspond to the x-mdrGen object in the OAPI/SWAGGER section of the selected path.



Description	A text description of the path. This description is used to as the text for the generated source member and object.
Gen Template	Specifies which generation template is used to generate the code. **RPGLE_EACH** - One program per path per HTTP method is generated.
Language	Language the Consumer is generated in. **RPG** - RPGLE **COBOL** - COBOL - (available June 2024)
JWT	Specifies if generator should add logic to add the Authorization header using MDRest4i JWT functions. (available June 2024)

HTTP METHODS

Shows the lists of HTTP methods as different tabs

Enable a Method

To enable additional methods, select the method required and, if this method is not found in the OAPI/SAWGGER for this path, select the Enable xxxx Method button for that method: 

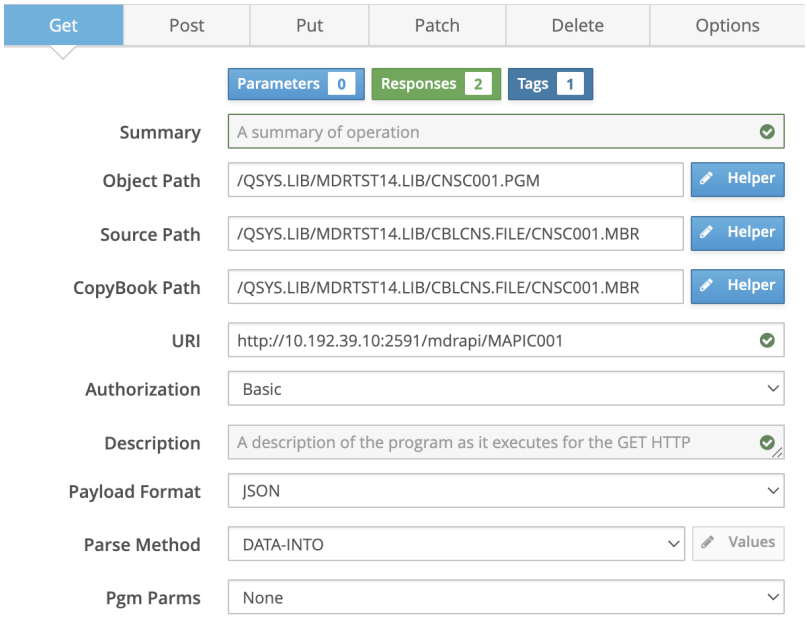
This will add this method, with default values from the Consumer template for that method, to the OAPI/SWAGGER and update the screen, to show Generation Options for that method.

GENERATION OPTIONS FOR CONSUMER

These values are updated in the underlying OAPI/SWAGGER **mdrGenOptions** object of the selected method, inside for the x-mdrGen object for the selected path. They are used to determine what code is generated and where it is generated.

> **Note**
>
> MDRest4i supports iAsp, IFS and Source file options for compilation.
>
> To allow for this a member **APIPGM** in source file **QRPGLESRC** in library **APISRCLIB**, must be entered using this IFS syntax:
>
> `/QSYS.LIB/APISRCLIB.LIB/QRPGLESRC.FILE/APIPGM.MBR`
>
> For those users not familiar with this syntax for lib/file/member or objects, a Helper popup has been created.
>
> This popup allows the traditional manner of editing these details.

Generation Options for a Consumer



Summary

Short description of the API path.

Object Path

This is the IFS formatted path to the Library/object name of the consumer program. The library and object names can be edited using the Helper popup.

Example: `/QSYS.LIB/YOURLIB.LIB/APIPGM.PGM`

Source Path

Location of source file/mbr for the Consumer program.

Example: `/QSYS.LIB/STUART13.LIB/EXAMPLE.FILE/API002.MBR`

Copybook Path

Location of copy book source file/mbr for payloads. The generator will create all payload data structures in this copybook, and add the `/copy` statement to the copybook in the generated Consumer.

Example: `/QSYS.LIB/STUART13.LIB/EXAMPLE.FILE/API002.MBR`

Authorization

Defines what type of authorization will be used in the request.

The generator adds variables and logic to add the Authorization header, to the request. The default value of string is created by the generator for these values. The developer should amend appropriately.

Selecting a value in the Authorization drop down, adds/sets the `security` object in the OAPI/SWAGGER for the path and method being edited in the paths tab.

> **Warning**
>
> In the generated consumer code, the value string is used by default. The appropriate values must be set by the developer.

Available options are:

Basic

When **Basic** is selected, the OAPI/SWAGGER is updated to:

OAPI/SWAGGER

```
/myapiname:
...
  get:
...
    security:
      - BasicAuth: []
```

The following logic is generated in the consumer program:

```
dcl-s authUser     varchar(256:4);
dcl-s authPwd      varchar(2048:4);
...
// TODO: Assign values for Basic authorization
authUser = 'string';
authPwd  = 'string';

// Set request Authorization header
opts = 'authtype=basic user=' + authUser + ' password=' + authPwd;
MDR_setClientCfg(handle:opts);
```

See MDR_setClientCfg for more information on setting these values.

Bearer

When **Basic** is selected, the OAPI/SWAGGER is updated to:

OAPI/SWAGGER

```
/myapiname:
...
  get:
...
    security:
      - BearerAuth: []
```

The following logic is generated in the consumer program:

```
// Variables used in Authorization process
dcl-s opts         varchar(32767:4);
dcl-s authToken      varchar(2048:4);
...
// Set the Bearer authorization by assigning the bearer token
// to the variable "authToken". e.g. authToken = 'Pnx9PH59vj0'
authToken = 'string'
opts = 'authtype=bearer authtoken=' +  authToken;
MDR_setClientCfg(handle:opts);
```

See MDR_setClientCfg for more information on setting these values.

OAPI securitySchemes

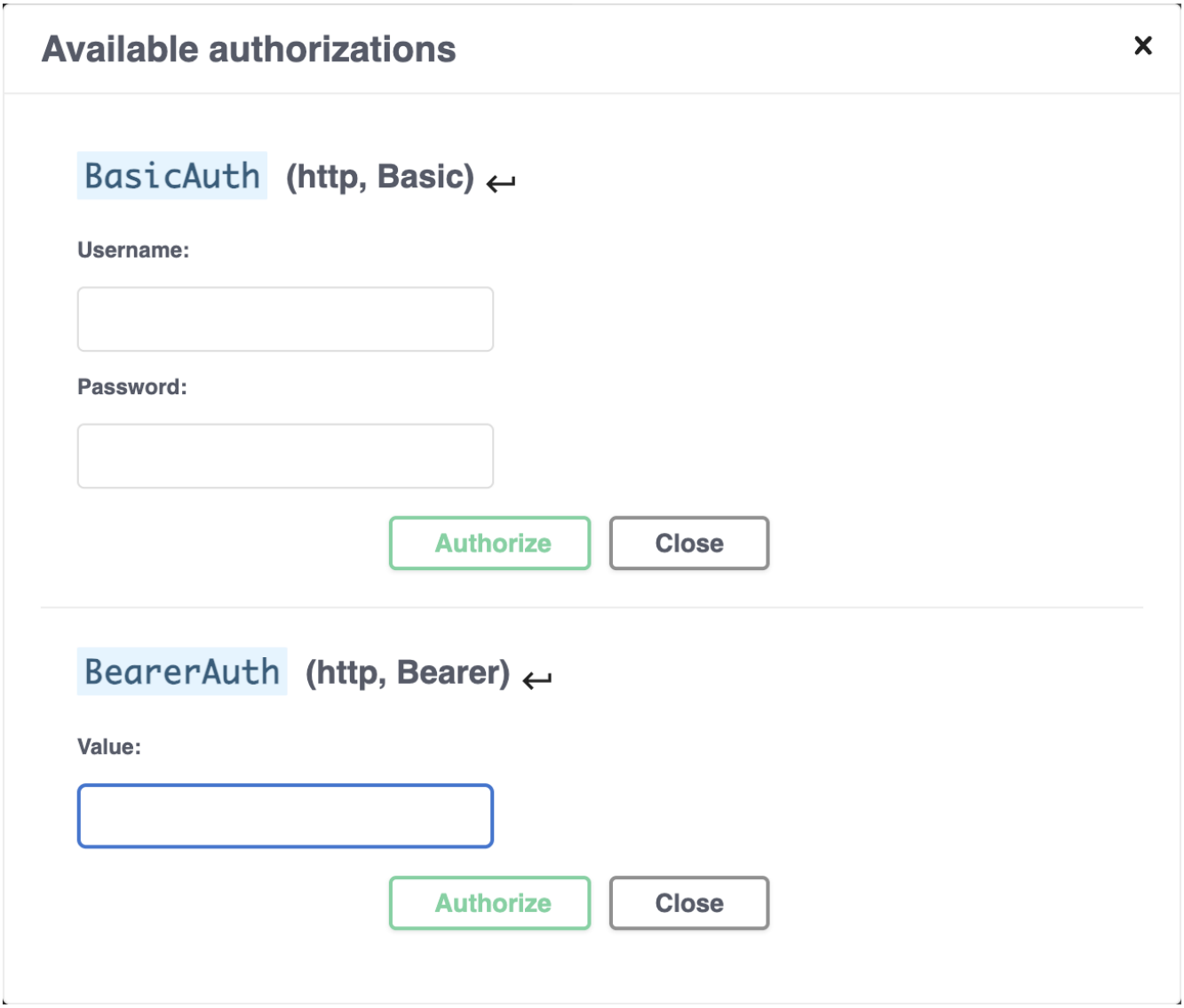
When a new consumer spec is created, these standard `components.securitySchemes` options are created from the consumer OAPI/ SWAGGER template.

Only Basic and Bearer are currently supported.

```
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
    BearerAuth:
      type: http
      scheme: bearer
```

Setting this will also enable Authentication credentials to be entered in General tab, SWAGGGER UI section when testing. This is done by selecting the Authorize button.

Enter the appropriate values in the popup:



These are then passed as the Authorization header with the request.

Description

Description of the selected method for that Consumer

Payload Format

This can be JSON, XML, or TXT. The generator will create logic to write or parse the correct payload syntax

Parse Method

Determines what parsing functions are used to process the inbound payload (response from api) in the generated code.

Available options are:

DATA-INTO - Uses the generated, qualified data structure (eg. mdlResp), assigned in the Responses section, IBM's DATA-GEN, and the MDRFRAME parser. It generates the following code:

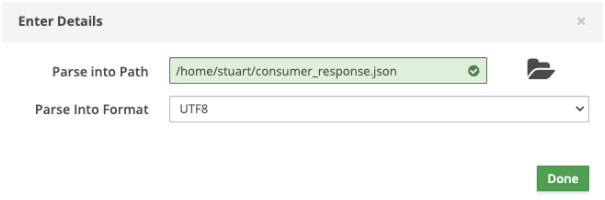
```
// Process Response Data for HTTP Status 200
// Parse JSON response into ds "mdlresp" using DATA-INTO
MDR_genParseOptions(handle: 'document_name=mdlresp');
data-into mdlresp %data('': 'case=convert +
                      countprefix=num_ +
                      allowmissing=yes allowextra=yes')
                %parser('MDRFRAME(PARSER)':handle);
```

JPATH Uses the generated, qualified data structure (eg. mdlResp), assigned in the Responses section, and generates MDRest4i JSON parsing functions such as MDR_jsonPathV etc in the consumer code. For example:

```
  dcl-s errorMsg   varchar(500:4);      // MaxSize: 4096
  dcl-s docnode    pointer;

  reset mdlresp;

  // Process Response Data for HTTP Status 200
  docnode = mdr_tree_parse(handle:*omit:*omit:errorMsg);
  if docnode <> *null;
  // Parse JSON response into ds "mdlresp" using JPathXX
    mdlresp.qrytitle = MDR_jsonPathV(handle:'qrytitle');
    mdlresp.qrysurname = MDR_jsonPathV(handle:'qrysurname');
    mdlresp.headersurname = MDR_jsonPathV(handle:'headersurname');
    mdlresp.cookie = MDR_jsonPathV(handle:'cookie');

  // Free memory resources used by YAJL parsing
  MDR_tree_free(handle:docNode);
```

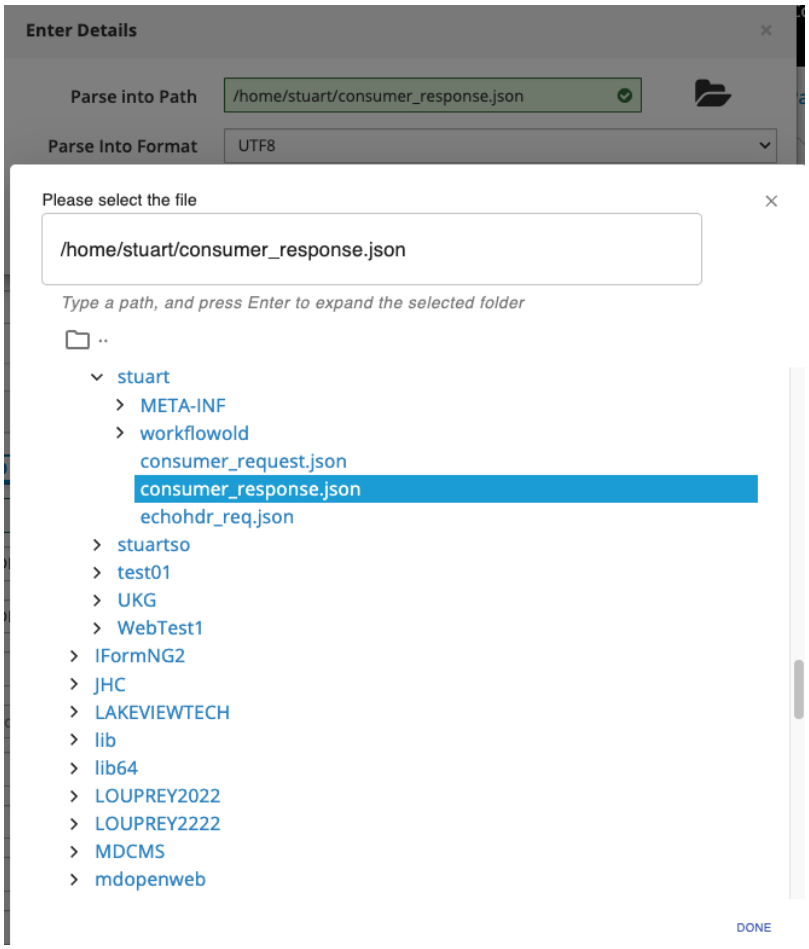
IFS - No parsing occurs. Generates logic in the consumer program to write the inbound payload directly to the IFS. Add the IFS path to be used using the 

button. This will display a popup that allows the adding of path name to the IFS object to be used in the program.



 button is provided next to the input box to use the File explorer to easily select or navigate to the desired path.

The File explorer opens when the button is clicked.



The following code is generated in the consumer program:

```
// Receive JSON response into an IFS
rcvstr = 'ifs:/home/stuart/consumer_response.json';
// Make an HTTP request. Return the HTTP status code (200=success)
// or -1 if there was a client-side error occurred.
code = MDR_request( handle
                  : method
                  : uri
                  : *omit
                  : rcvstr
                  );
```

Warning

rcvstr value is set with an "ifs:" prefix. Function MDR_request automatically writes the JSON response to this path as a result.

If the 'ifs:' prefix is not used, this function will not handle this automatic assignment.

See MDR_request for more details.

Gen Method

ONLY applicable to **POST**, **PUT**, **PATCH** HTTP methods. Determines how the outbound (consumer request) payload is handled in the generated code. Available options are:

DATA-GEN: Uses the generated, qualified data structure (eg. mdlReq), assigned in the Body section, and IBM's **DATA-GEN** and the MDRFRAME parser. It generates the following code:

```
// Create JSON request payload using DATA-GEN.
MDR_genParseOptions(handle: 'document_name=mdlReq');
data-gen mdlReq %data(result: 'doc=string +
                               countprefix=num_ +
```

```
                                renameprefix=name_')
                    %gen('MDRFRAME(GENERATOR)':handle);
```

MDR-YAJL: Generates MDRest4i YAJL functions such as mdr_addChar which use the generated, qualified data structure (eg. mdlReq - assigned in the Request Body) section to create JSON for the request body:

```
// Start writing the JSON using YAJL functions.
mdr_startJson(*OFF:*OFF);
// Build JSON request body using YAJL functions.
mdr_beginObject();
mdr_addChar('field1':mdlReq.field1);
mdr_addChar('field2':mdlReq.field2);
mdr_beginObject('object');
mdr_addChar('field3':mdlReq.object.field3);
mdr_addChar('field4':mdlReq.object.field4);
mdr_endObject();
mdr_endObject();

// Retrieve JSON loaded in buffer memory
mdr_getJson(*zero:JsonPtr:%size(JsonStr):rtnlen);

// End the JSON write
mdr_endJson();
// Assign the JSON into SendStr to send JSON request to the API
// using MDR_request.
If rtnlen > *zero;
  sendStr = %subst(JsonStr:1:rtnlen);
endif;
```

MDR-YAJL-NULL: Exactly the same mechanism as MDR-YAJL, but adds a parameter(isNull) to each function call ***ON**.

@param isNull (input) = if *ON output is null, otherwise is value supplied in second parameter

Example:

```
mdr_beginObject();
mdr_addChar('name':mdlReq.name:mdlReq.name=' ');
mdr_addNum('age':mdlReq.age:mdlReq.age=0);
mdr_endObject();
```

If `mdlReq.name` is blank, then the JSON will look like this: `{"name": null}`

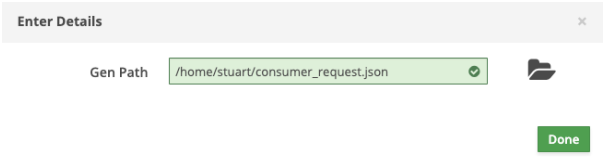
if `mdlReq.name` is 'stuart', then the JSON will look like this: `{"name": "stuart"}`

For more details on writing JSON with this set of ILE functions in RPG or COBOL, see the MDRFRAME - JSON Functions sections of this guide.

IFS: Generates logic in the consumer program, to write the outbound payload, directly from the contents of an IFS file.

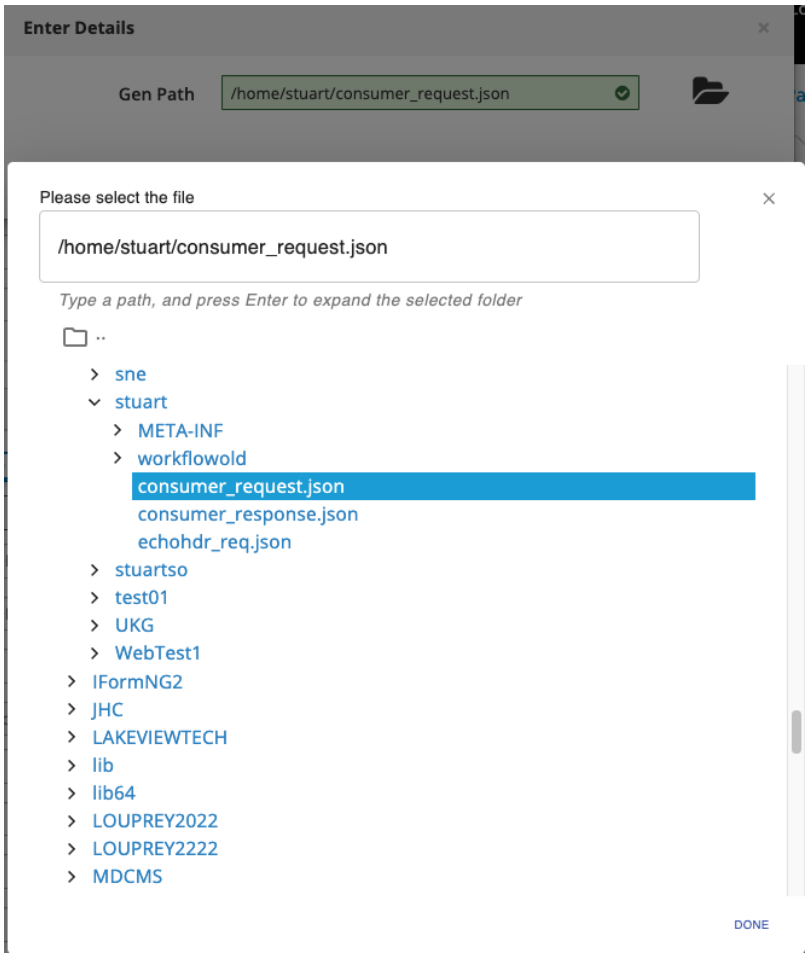
Add the IFS path to be used using the 

button. This will display a popup that allows the adding of path name to the IFS object to be used in the program.



 button is provided next to the input box to use the File explorer to easily select or navigate to the desired path.

The File explorer opens when the button is clicked.



The following code is generated in the consumer program when IFS is used:

```
// Send JSON request from ifs
SendStr = 'ifs:/home/stuart/consumer_request.json';
// Make an HTTP request. Return the HTTP status code (200=success)
// or -1 if there was a client-side error occurred.
code = MDR_request( handle
                  : method
                  : uri
                  : sendstr
                  );
```

Warning

The **sndstr** value is set with an '**ifs:**' prefix. Function MDR_request automatically writes the request payload from this path into the request.

If the 'ifs:' prefix is not used, this function will not handle this automatic assignment.

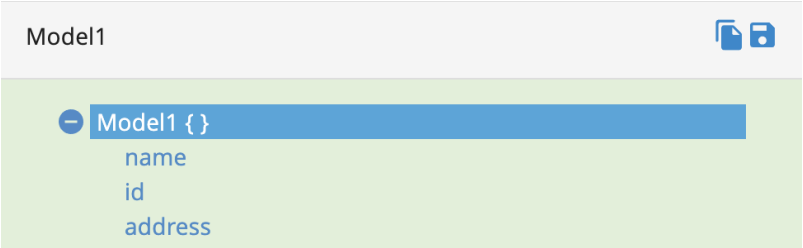
See MDR_request for more details.

Pgm Parms

This drop down allows selection of a schema. This schema will then be generated as a data structure which is then used as the entry parameters for the program.

Here are examples of the code generated in RPGLE and COBOL when these options are selected.

Example Schema:



Assigned to PGM Parms

Pgm Parms: Model1

Generated Code

RPGLE COBOL

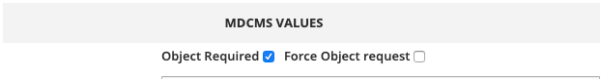
```
**FREE
// Consumer program with entry param

ctl-opt dftactgrp(*no) actgrp(*new) option(*srcstmt)
        bnddir('MDRFRAME');

// Standard MDRest4i Copybooks
/copy MDRUSRCPY
/copy MDRFRAME

// Main program PI definition.
dcl-pi CNSEPARMS;
  PgmParam     likeds(Model1);
end-pi;

// DS definitions for program call parameter
dcl-ds Model1 qualified;
  name    varchar(15)     inz;
  id      int(10)         inz;
  address varchar(18)     inz;
end-ds;
```

```
       LINKAGE SECTION.
   COPY MDRCBLLSC OF QLBLSRC.

   * Data Structure definition for call parameter
     01  MDR-MODEL1.
       03  MDR-NAME    PIC X(15).
       03  MDR-ID      PIC S9(9) USAGE BINARY.
       03  MDR-ADDRESS PIC X(18).
```

METHOD OPTIONS

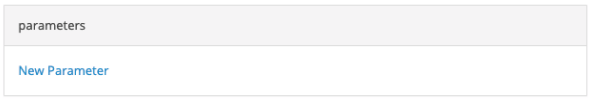
For the **GET**, **DELETE** and **OPTIONS** HTTP Methods, the following buttons appear:

HTTP methods **POST**, **PUT**, **PATCH** display the following buttons:

Option	Description
Parameters	Select to display and edit request parameters
Body	Select to display and edit request body details. (only for **POST**, **PUT**, **PATCH** method requests)
Responses	Select to display and edit response body details.
Security	Select to display and edit response body details.
Tags	To create tags
Object Request	For MDCMS users. This appears only if “Object Required" box is checked in the Edit Site 

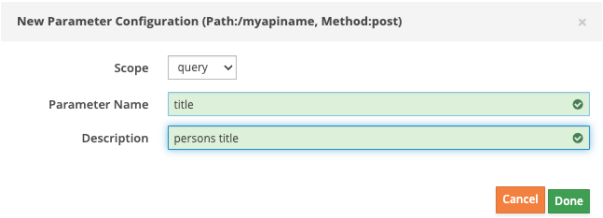
Parameters

Click on the  button



To add a new parameter, click on the  link

In the popup window, select the scope (query/header). Enter the parameter name and description and click  button



Header parameters can be added using the same method.

The **Parameters** section is now updated:

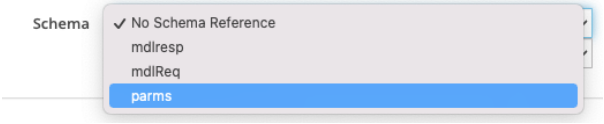
Using function MDR_encode, the generator creates code that URL encodes the query parameter, then appends this encoded value to the uri variable. The uri variable is used to make the final request. In the above example, it also generates the function MDR_setHeader() to set the outbound HTTP header for the request:

```
// Set the HTTP header request parameters
MDR_setHeader(handle:'x-surname':x_surname);
// Set request URI
uri = 'https://host.com/myapiname';

// Set the URI parameters
if title <> *blanks;
  MDR_encode(handle : encodeVar : %trimr(title));
  uri += '?title=' + encodeVar;
endif;
```

Detailed Parameter Attributes

OAPI Schemas allow specifying of detailed attributes of variables. Using this mechanism, detailed attributes of the query, header and path variables can specified for the generated code.

Select a schema from the dropdown list of existing schemas.



On selecting the schema, the list of fields automatically is displayed in the next input field, Schema Field.

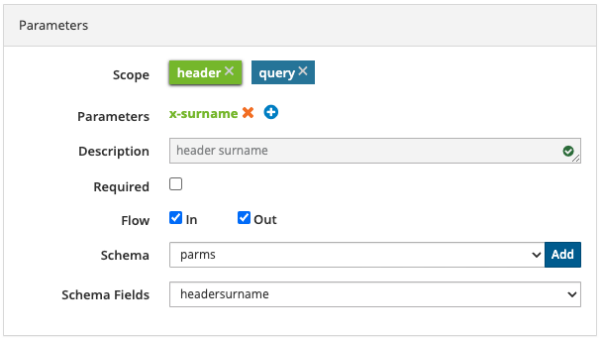
The OAPI/Swagger definition now adds the **parameters** field to the path and method selected, with references to the schema.

```
parameters:
  - in: header
    name: x-surname
    description: header surname
    required: false
    schema:
      x-schemaFld: headersurname
      x-schema: '#/components/schemas/parms'
    x-flow: both
  - in: query
    name: title
    description: persons title
    required: false
    schema:
      x-schemaFld: qrytitle
      x-schema: '#/components/schemas/parms'
```

Here, the Swagger Extension fields x-schema, x-schemaFld are used.

Header Parameter Flow

In the case of **header** type parameters, an additional value **flow** can be specified.




```yaml
parameters:
  - in: header
    name: x-surname
    description: header surname
    required: false
    schema:
      x-schemaFld: headersurname
      x-schema: '#/components/schemas/parms'
    x-flow: both
```

The Swagger Extension x-flow field is used which can have the value "in", "out" or "both" depending on the boxes checked.

in requires the header value to be read by the consumer program, from the HTTP response.

out requires the header to be added by the consumer to the request.

both requires the header value to be sent with the request, AND read in the response. The following code uses function MDR_getHeader() to read the header from the response:

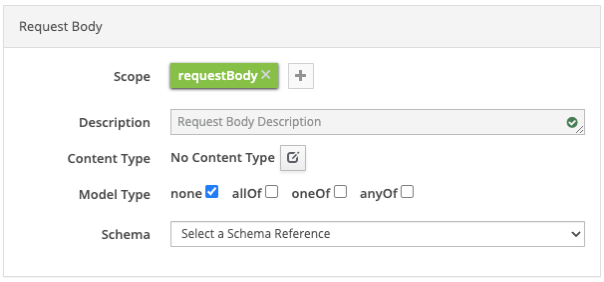
```
// Get the value of HTTP header
x_surname = MDR_getHeader(handle:'x-surname');
```

Body

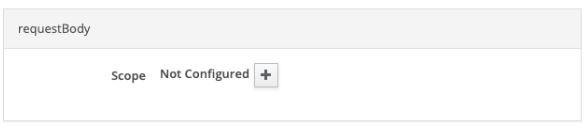
Click on the 

button to edit the request body. The **Body** option/button is available only for **POST**, **PUT** and **PATCH** methods.

When this button is selected the **Request Body** section appears on the right hand side. Similar to the **response** section, it allows the request body schema and content-type to be selected.

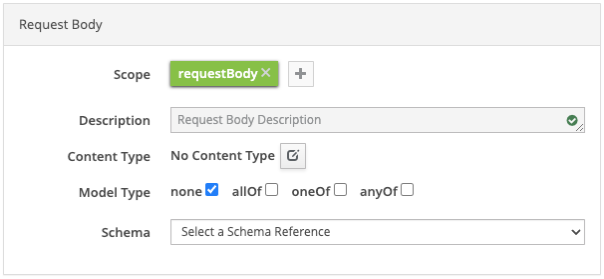


When no request body has been added to the specification, the **Request Body** section looks like this:



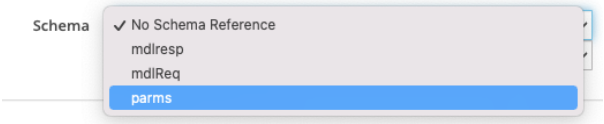
Click on the  button to create one.

Click the Done button.

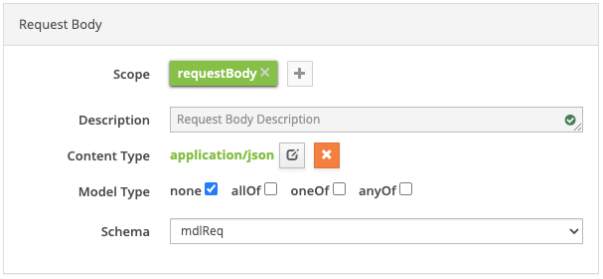


Select the Content Type as application/json from the dropdown list.

Using the Schema drop down, select the schema that will be used for the outbound request body.



The Request Body section in the UI is updated, and the following object: **requestBody** is updated in the OAPI/Swagger.




```
requestBody:
  description: Request Body Description
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/mdlReq'
```

Request Body Logic

In a consumer program, the schema in the body details above (object requestBody from the OAPI/SWAGGER), determines the payload definition. This schema is generated as a qualified data structure in RPG/COBOL.

For example here is a JSON sample from the schema used in body above:

```
{
  "field1": "something of value",
  "field2": "something of value2",
  "object": {
    "field3": "something of value3",
    "field4": "something of value4"
  }
}
```

The schema extracted from this sample by the SDK while creating a schema, is called **mdlReq** (or any other name you chose when creating the schema). Here is the schema tree diagram from the schema tab:

Below is the qualified data structure created from this by the SDK generator:

```
// Request payload definitions
dcl-ds mdlReq qualified;
  field1 varchar(18)     inz;
  field2 varchar(19)     inz;
  dcl-ds object;
    field3 varchar(19)     inz;
    field4 varchar(19)     inz;
  end-ds;
end-ds;
```

The logic used in the generated consumer program, to create the JSON request body payload, is determined by the Gen Method as described above.

Responses

By default, two HTTP responses are added to each consumer. Response 200 is the default **status** HTTP header value, consumer, that everything is OK.

HTTP Status 4xx is used to indicate a problem. This can be amended to whatever standard or value is desired. Here is a guideline that can be used to determine what codes are used in general:

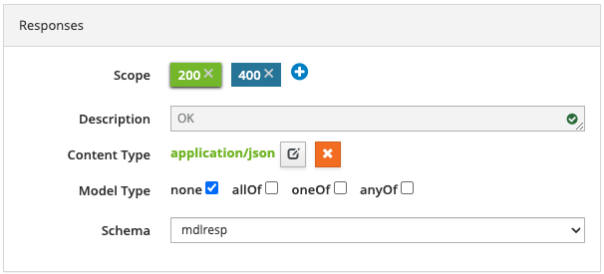
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

> **Tip**
>
> The concept of REST is about collaborative development between often disparate development teams. Following industry norms wherever is a very good practice. This avoids unnecessary errors, when developers and users assume industry standards are being followed, especially where developers don't have direct communication, or API documentation is not of a high standard, or up to date.

Select the 

button.

From the **Responses** section on the right hand side, select the content type, and then select a schema created earlier from the Schema drop down (leaving model type as none).



The **responses** object in the underlying OAPI/SWAGGER is updated :

```
responses:
  '200':
    description: OK
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/mdlresp'
```

Response Body Logic

In a consumer program, the schema in the Responses details above (object responses from the OAPI/SWAGGER), determines the payload definition. This schema is generated as a qualified data structure in RPG/COBOL.

For example, the following JSON sample from the schema used in body above is:

```
{
  "qrytitle": "mr",
  "qrysurname": "smith",
  "headersurname": "header smith",
  "cookie": "bourbon biscuit"
}
```

The schema extracted from this sample by the SDK while creating a schema, is called **mdResp** (or any other name you chose when creating the schema). Here is the schema tree diagram from the schema tab:

Below is the qualified data structure created from this by the SDK generator:

```
// Response payload definitions
dcl-ds mdlresp qualified;
  qrytitle      varchar(15)     inz;
  qrysurname    varchar(30)     inz;
  headersurname varchar(30)     inz;
  cookie        varchar(30)     inz;
end-ds;
```

The logic used in the generated consumer program, to parse the JSON response body payload, is determined by the Parse Method as described above in the Generation Options section.

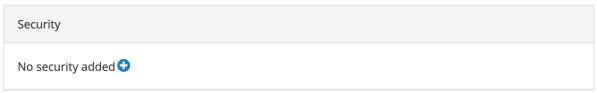
Security

Click on the 

button to edit the request body.

When this button is selected the “Security” section appears.

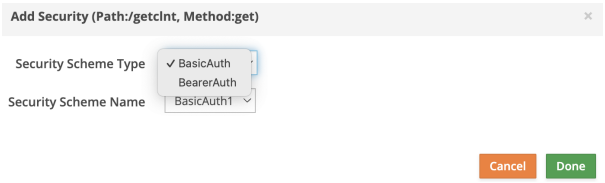
If no security schemes have been added to the method, the section looks like:



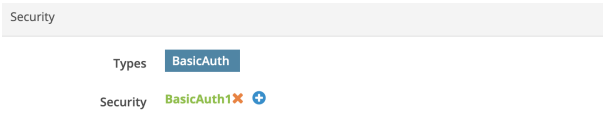
Click on the  button to create one.

Note

You can add a security scheme only when the global security schemes have been created using the Security tab.



Click  button



The umderlying Swagger gets updated as:

```
get:
      ...
      security:
        - BasicAuth1: []
```

More security schemes can be added using the "plus" button.

HELPER BUTTONS

MDRest4i supports iAsp, IFS and Source file options for compilation.

To allow for this a member **APIPGM** in source file **QRPGLESRC** in library **APISRCLIB**, must be entered using this IFS syntax:

`/QSYS.LIB/APISRCLIB.LIB/QRPGLESRC.FILE/APIPGM.MBR`

The same source file in the folder APISRC would use this syntax:

`/APISRC/APIPGM.RPGLE`

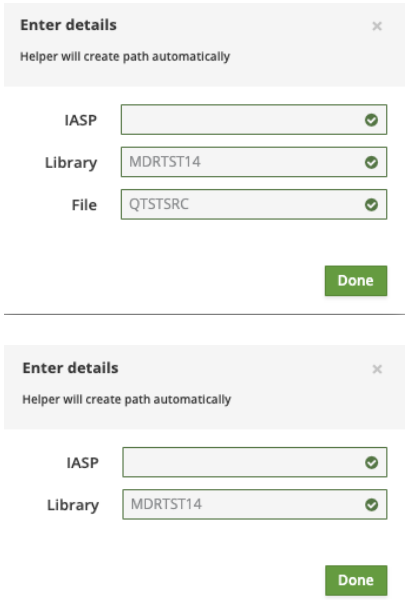
Using this format, an object path would look like this:

`/QSYS.LIB/YOURLIB.LIB/APIPGM.PGM`

For those users not familiar with this syntax for lib/file/member or objects, a 

popup has been created.

This popup allows the traditional manner of editing these details.



Unless you are using a specific iASP, this value can be left blank for *SYSBAS iASP.

The previous form and underlying OAPI/SWAGGER are updated saved after the **Done** button is clicked.

Generate Tab

The "Generate" tab is a custom version of a full version of SWAGGER Editor and SWAGGER UI combined.

The page is split into two sections- the editor and the UI section

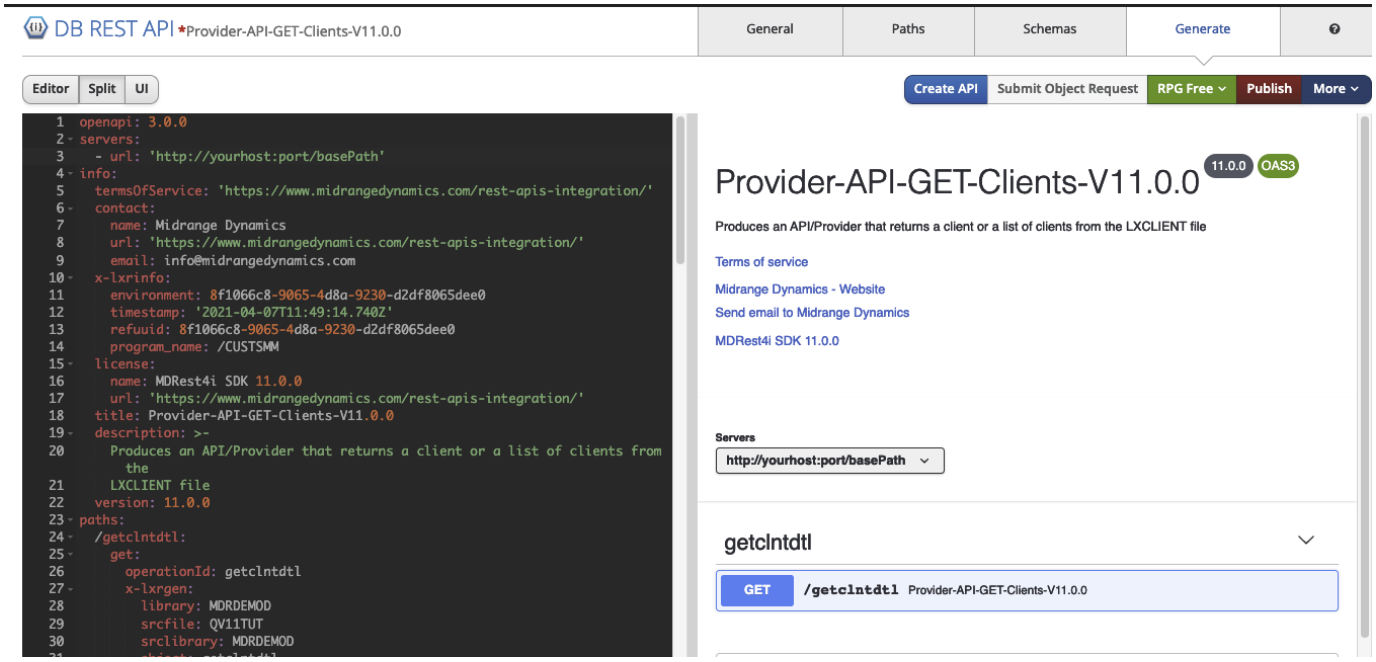
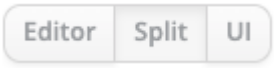


Figure 1: Generate Screen

The left hand window - Swagger editor enables editing of the SWAGGER specification directly, and the right-hand side - the Swagger UI shows the documentation and allows testing of the Provider / consumer once it is generated and completed.

BUTTON BARS

Left



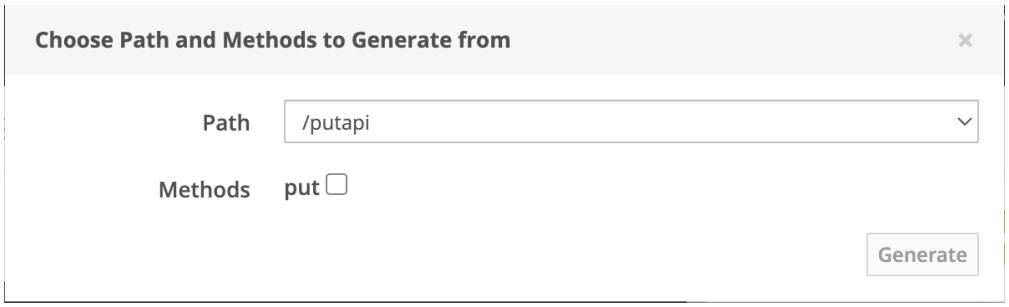
Editor	The Swagger Editor is displayed on the full screen
Split	The Swagger Editor and Swagger UI sections both are displayed
UI	Only the Swagger UI is displayed on full screen

Right

Create Program	Used to create the API
Submit Object Request	To submit the Object request to the MDCMS, visible only if "Object required" under "MDCMS values" is set to true in the Configuration settings
View Source	To display the generated code (Enabled only if the API is successfully generated). Brings up a dropdown list of all generated programs
Publish / Documentation	To publish the API specification to MDRest4i Portal/Documenter or view the published document
More	The dropdown menu same as in the General tab.

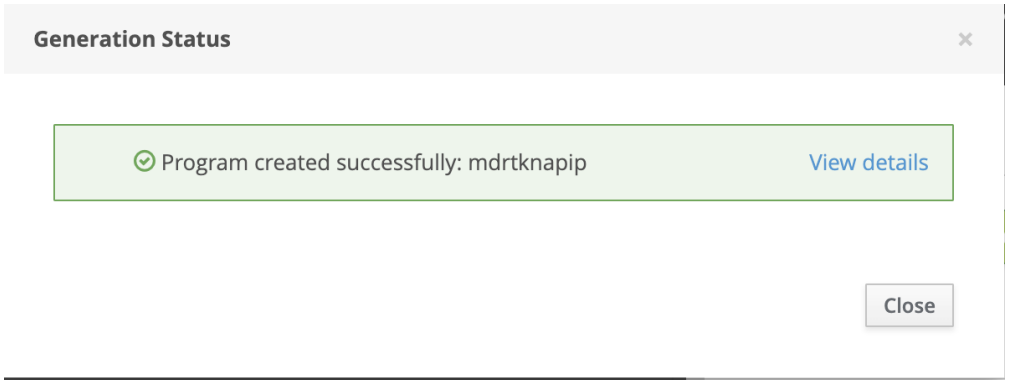
Create Program

Click the Create Program button



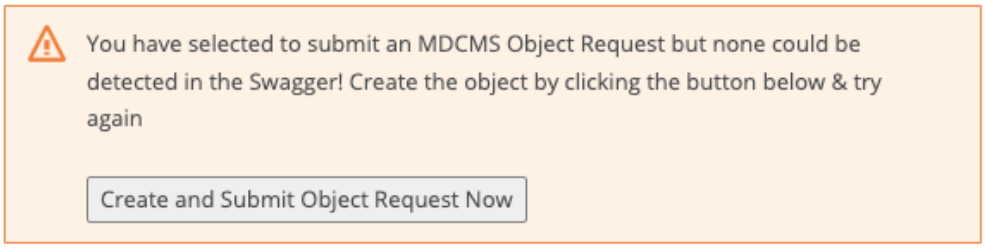
Select the required path and method and click the “Generate” button.

Upon completion of the code generation process on the IBM i, the following success message popup appears



Object Request Warning!

If MDCMS Force Object Request is checked in Configuration settings, and an MDCMS Object Request has not been created prior to trying to Create Program, then the following warning will appear:



The **Create and Submit Object Request Now** button lets you both add/modify and submit the object request.

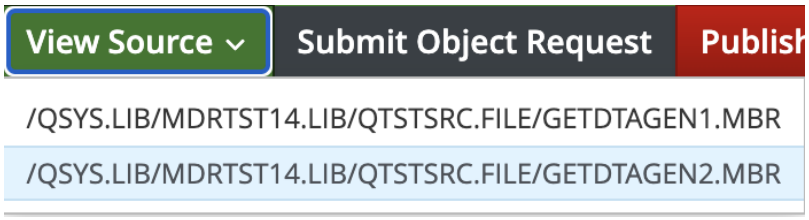
After successfully submitting an MDCMS Object Request, click the 

button again.

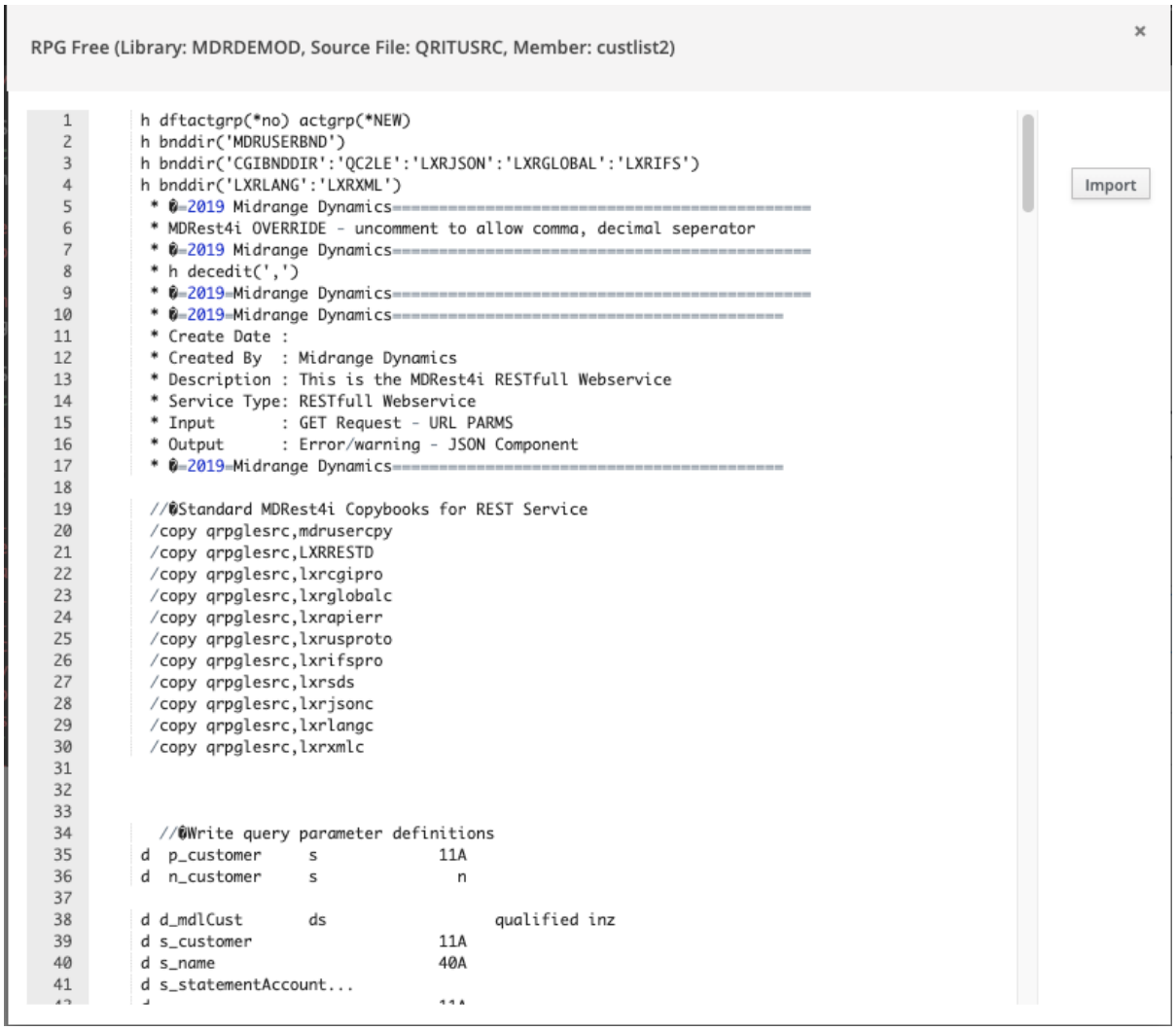
In case, the object request has been created already but not submitted, the object request submission will happen automatically before creating the program.

View Source

The dropdown button which lists all the generated members.



This option lets you **view** the generated RPG source. The user can select the program from the dropdown list.



Publish/ Documentation

Select this button to publish the API specification to the MDRest4i Portal/Documenter.

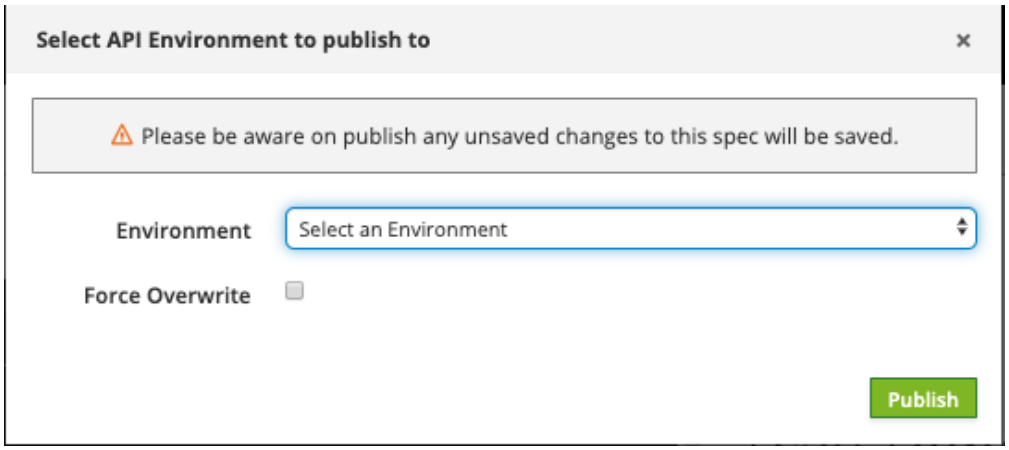
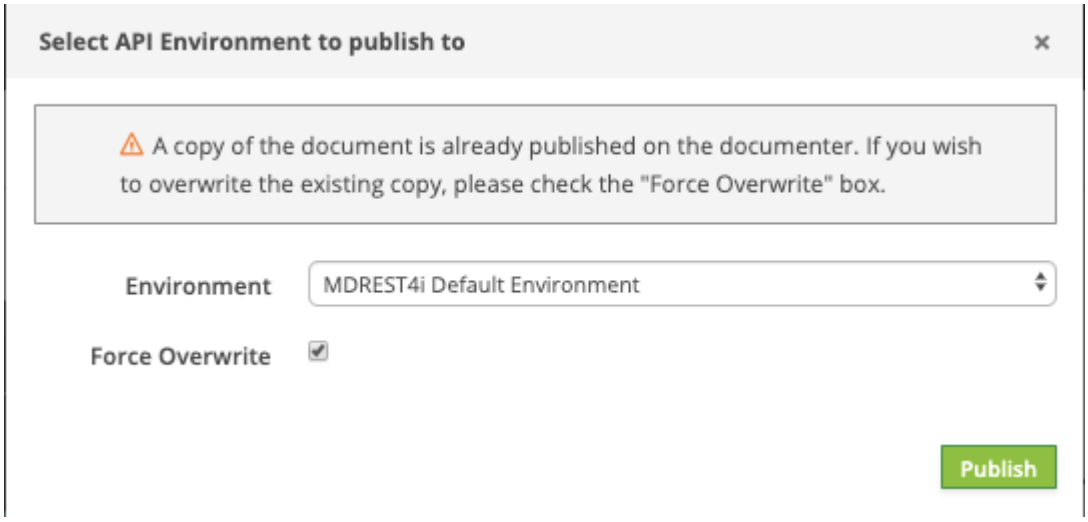


Figure 9: Publish to documenter

Select the Environment from the dropdown list and click Publish

If an older version of the document is already published on the documenter, you get an error message

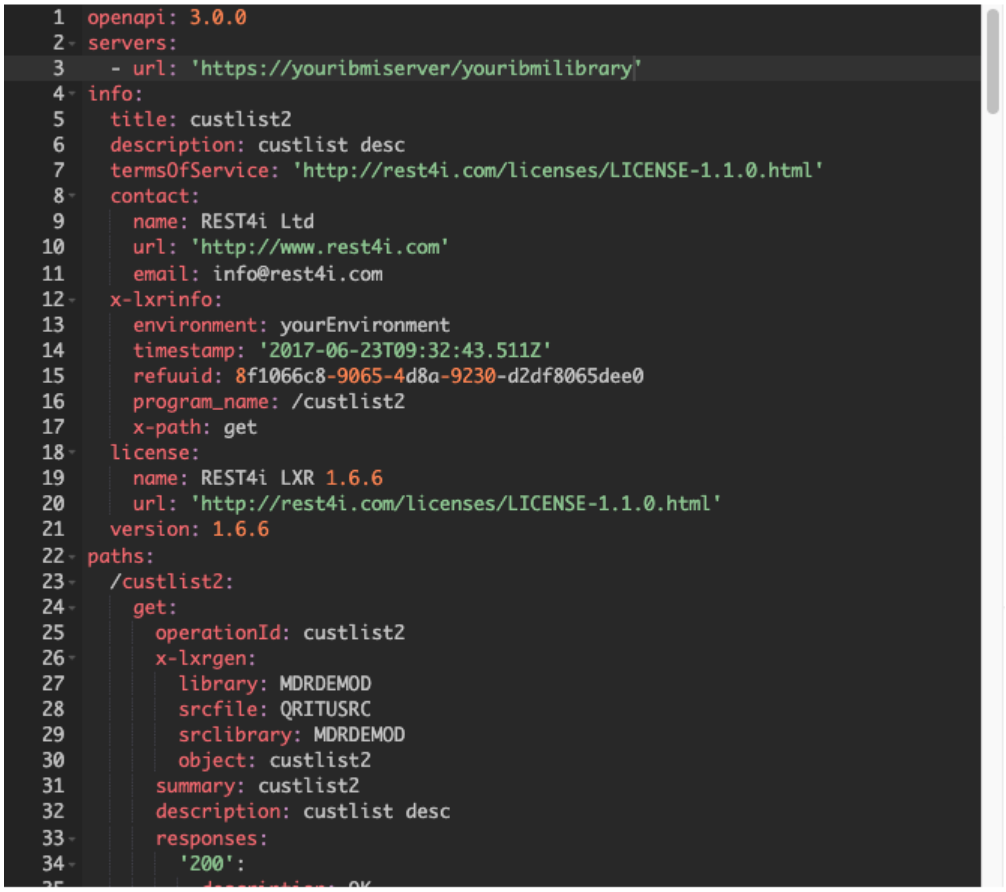


If we check the "Force Overwrite" checkbox, and select then the copy in MDRest4i Portal/Documenter is over-written.

SWAGGER EDITOR

The Swagger Editor allows you to define and document RESTful APIs using the Swagger Specification.

We created the Swagger definition using the MDREST4i-SDK-Console Web GUI. Any changes made using the editor would be reflected in the MDREST4i-SDK-Console Web GUI and vice-versa.



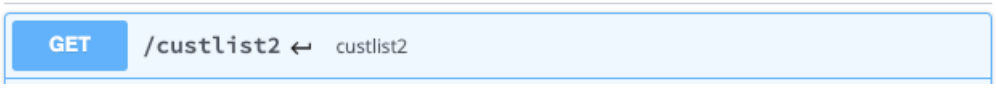
```yaml
openapi: 3.0.0
servers:
  - url: 'https://youribmiserver/youribmilibrary'
info:
  title: custlist2
  description: custlist desc
  termsOfService: 'http://rest4i.com/licenses/LICENSE-1.1.0.html'
  contact:
    name: REST4i Ltd
    url: 'http://www.rest4i.com'
    email: info@rest4i.com
  x-lxrinfo:
    environment: yourEnvironment
    timestamp: '2017-06-23T09:32:43.511Z'
    refuuid: 8f1066c8-9065-4d8a-9230-d2df8065dee0
    program_name: /custlist2
    x-path: get
  license:
    name: REST4i LXR 1.6.6
    url: 'http://rest4i.com/licenses/LICENSE-1.1.0.html'
  version: 1.6.6
paths:
  /custlist2:
    get:
      operationId: custlist2
      x-lxrgen:
        library: MDRDEMOD
        srcfile: QRITUSRC
        srclibrary: MDRDEMOD
        object: custlist2
      summary: custlist2
      description: custlist desc
      responses:
        '200':
```

Figure 10: Swagger Editor

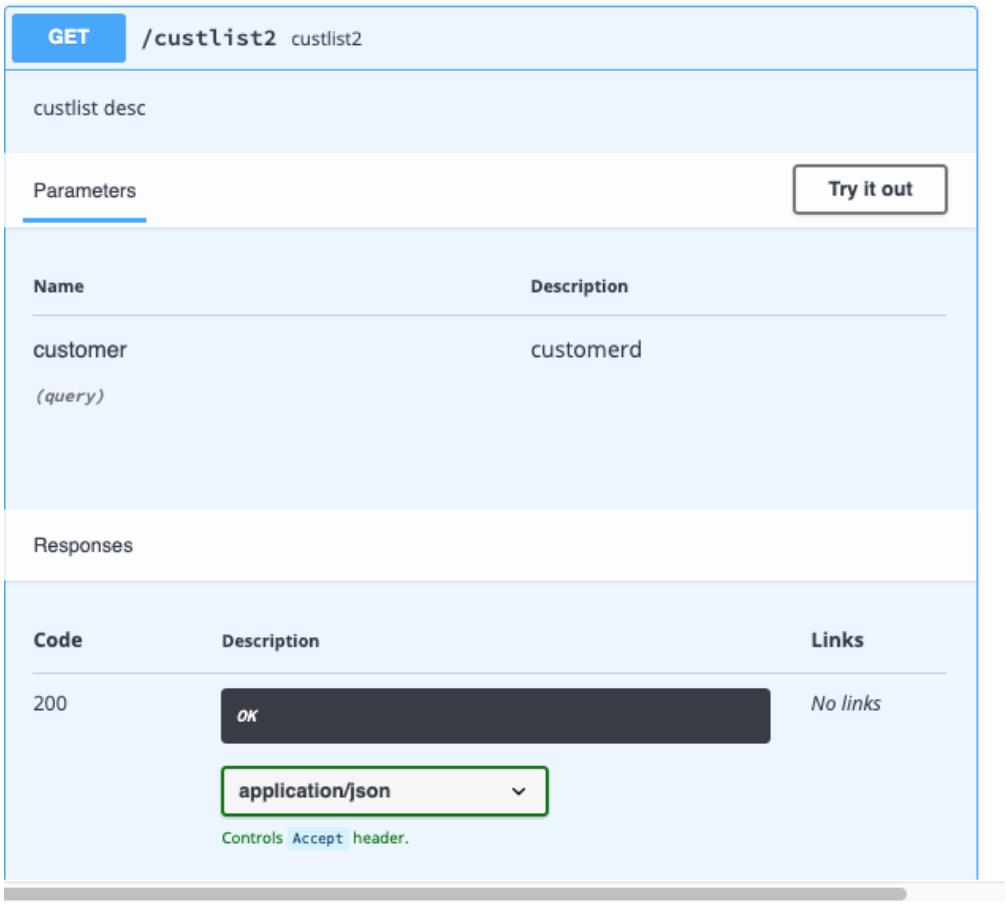
SWAGGER UI

TEST THE GENERATED SERVICE

In the UI section on the right, scroll down until the server and “GET” button (or whichever method you have cerated) appears. Click on the “GET” Button to expand it.



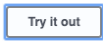
Get Button for testing



|*** Get Button expanded ***

Now scroll down to the “Parameters” section

Scroll down to the “Server response” section.

and click the  button.

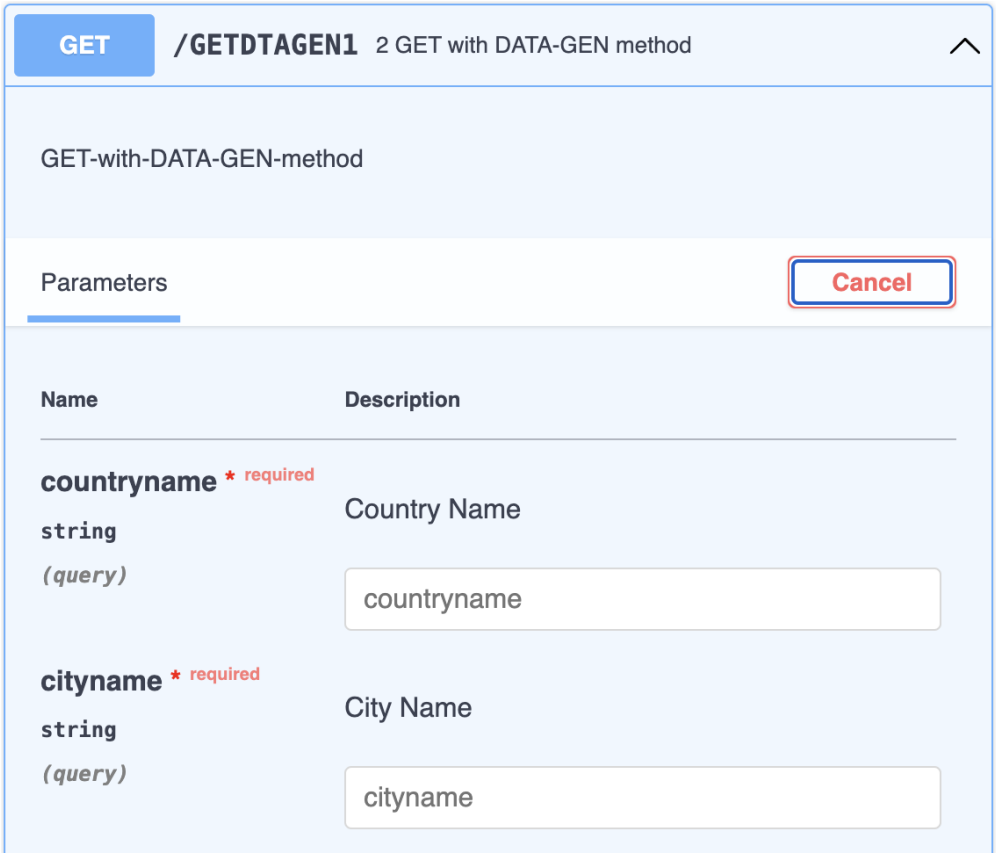
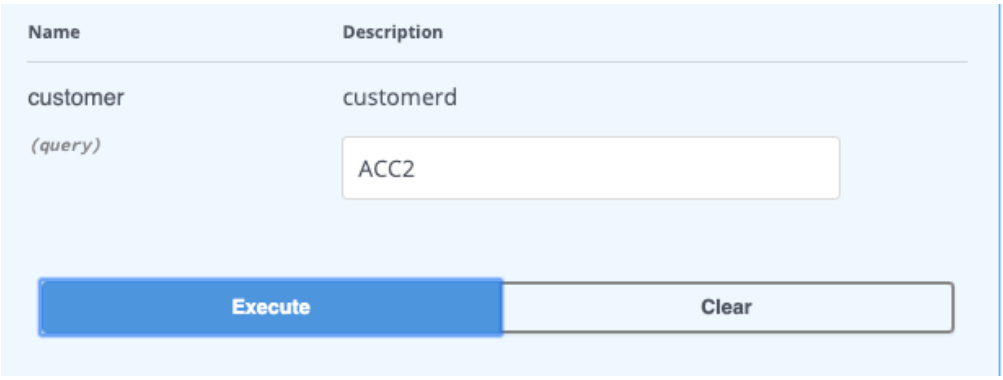


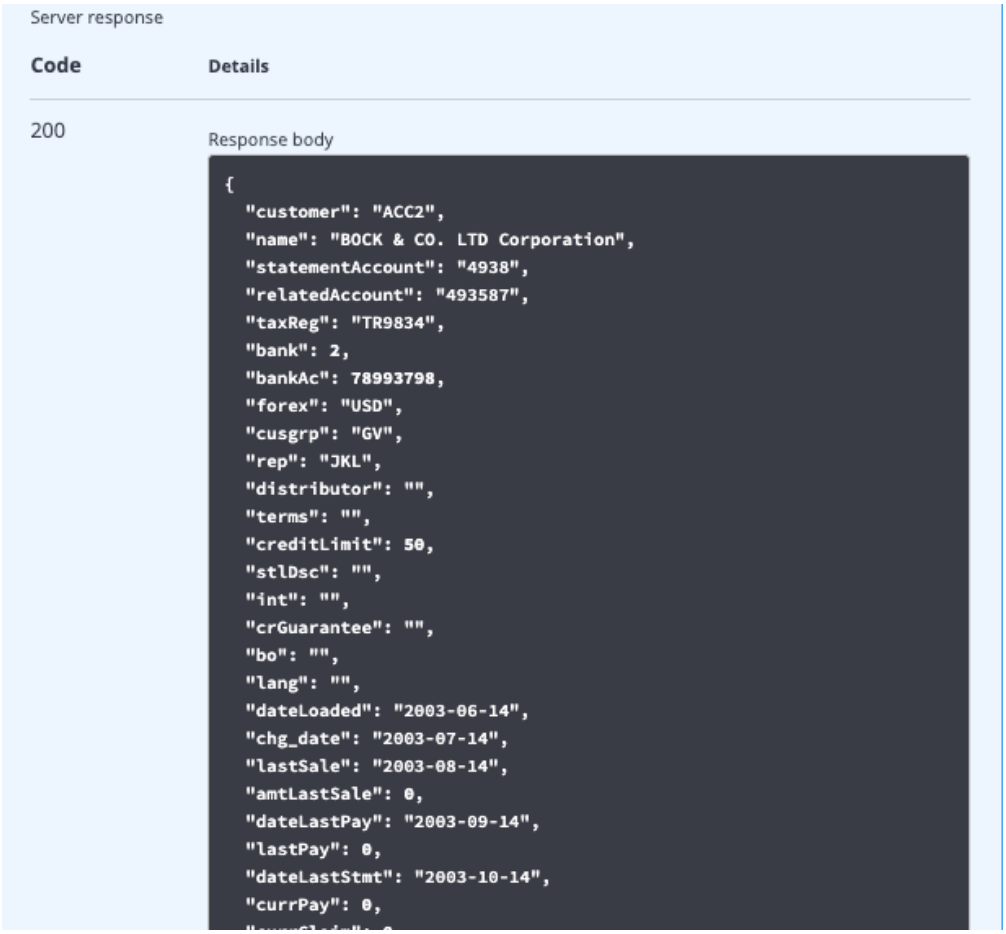
Figure 13: Parameters Section expanded

Enter the value of the parameters as necessary, and select the "execute" button. This sends a request to the generated Provider



Parameters Section expanded

Scroll down to the “Server response” section. The response body can be seen as JSON in this window.



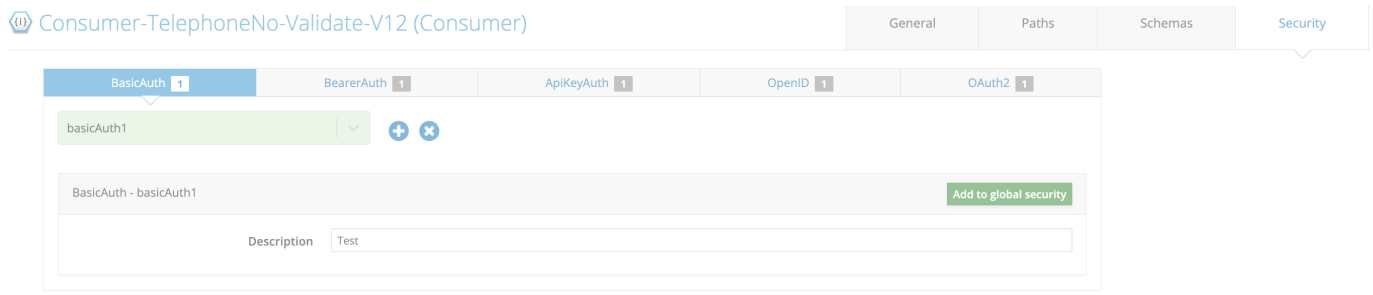
Security Tab - Spec Editor

The Security tab lets you create, view and edit the security schemes for the spec selected from the List of Specifications on the Specs page.

All security schemes used by the API must be defined in the global components/securitySchemes section in the OpenAPI spec. This section contains a list of named security schemes, where each scheme can be of type:

http – for Basic, Bearer and other HTTP authentications schemes apiKey – for API keys and cookie authentication oauth2 – for OAuth 2 openIdConnect – for OpenID Connect Discovery

SECURITY SCHEMES SCREEN



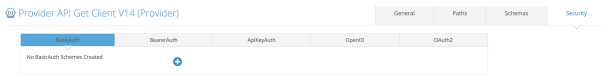
The screen shows the tabs for the available 5 types of security schemes:

1. **BasicAuth**
2. **BearerAuth**
3. **APIKeyAuth**
4. **OpenID**
5. **OAuth2**

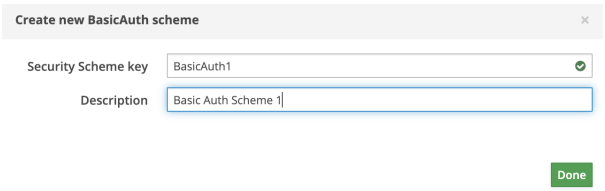
BasicAuth

Create a BasicAuth Scheme

When no schemes are created yet, the screen looks like:



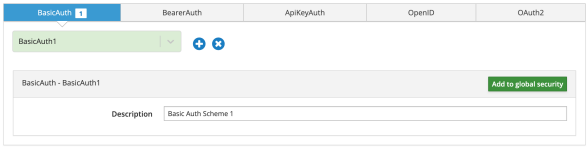
Select the tab BasicAuth on the screen and click the  button.



Enter the security scheme name which is an arbitrary name that will be used to refer to this definition from other places in the spec, and the description. Click the 

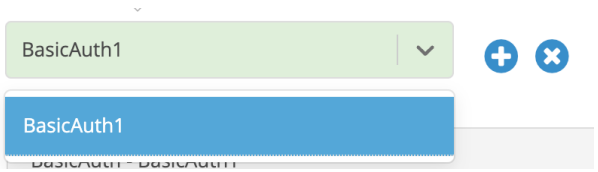
button.

The basic security scheme gets created and the Security screen is updated.



The tab now shows the number of Basic auth schemes created.

Also a dropdown list of the security schemes created appear now to which the newly created Security Scheme is added.



Use the  button to delete the selected scheme.

The created Security Scheme gets added to global components/securitySchemes section of the underlying OAPI/SWAGGER:

```
  securitySchemes:
    BasicAuth1:
      type: http
      scheme: basic
      description: Basic Auth Scheme 1
```

After you have defined the security schemes in the securitySchemes section, you can apply them to the whole API or individual operations by adding the security section on the root level or operation level, respectively. When used on the root level, security applies the specified security schemes globally to all API operations, unless overridden on the operation level.

The button Add to global security lets you add the security scheme to the root or the global level.

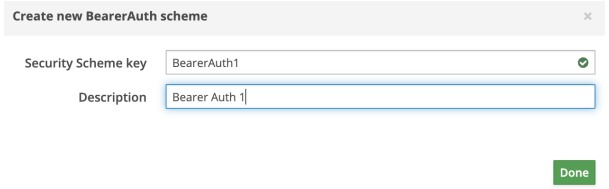
When the button is clicked, the underlying OAPI/SWAGGER gets updated. The selected security scheme is added to the global "security" array:

```
security:
  - BasicAuth1: []
components:
  securitySchemes:
    BasicAuth1:
      type: http
      scheme: basic
      description: Basic Auth Scheme 1
```

BearerAuth

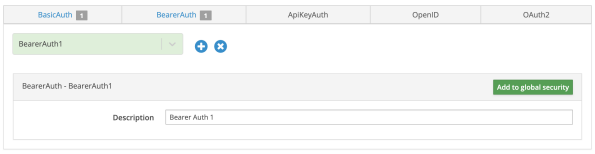
Create a BearerAuth Scheme

Select the tab BearerAuth on the screen and click the  button.

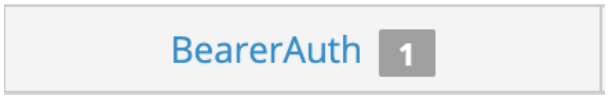


Enter the security scheme name which is an arbitrary name that will be used to refer to this definition from other places in the spec, and the description. Click the Done button.

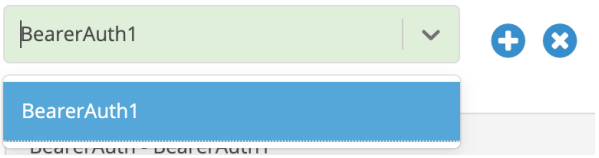
The bearer security scheme gets created and the Security screen is updated.



The tab now shows the number of Bearer auth schemes created.



Also a dropdown list of the security schemes created appear now to which the newly created Security Scheme is added.



Use the button to delete the selected scheme.

The created Security Scheme gets added to global components/securitySchemes section of the underlying OAPI/SWAGGER:

```
components:
 securitySchemes:
   BasicAuth1:
     type: http
     scheme: basic
     description: Basic Auth Scheme 1
   BearerAuth1:
     type: http
     scheme: bearer
     description: Bearer Auth 1
```

After you have defined the security schemes in the securitySchemes section, you can apply them to the whole API or individual operations by adding the security section on the root level or operation level, respectively. When used on the root level, security applies the specified security schemes globally to all API operations, unless overridden on the operation level.

The button Add to global security lets you add the security scheme to the root or the global level.

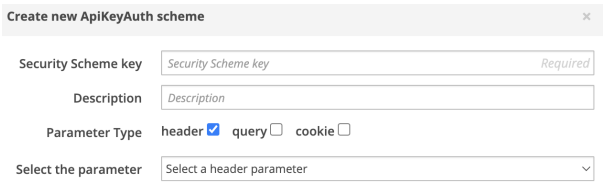
When the button is clicked, the underlying OAPI/SWAGGER gets updated. The selected security scheme is added to the global "security" array:

```
security:
  - BasicAuth1: []
  - BearerAuth1: []
components:
  securitySchemes:
    BasicAuth1:
      type: http
      scheme: basic
      description: Basic Auth Scheme 1
    BearerAuth1:
      type: http
      scheme: bearer
      description: Bearer Auth 1
```

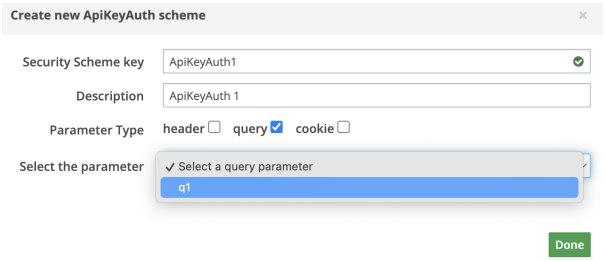
APIKeyAuth

Create an ApiKeyAuth scheme

Select the tab ApiKeyAuth on the screen and click the button.



Fill the form. Enter the security scheme name which is an arbitrary name that will be used to refer to this definition from other places in the spec, and the description. Select the parameter type and the actual parameter or cookie to be used from the dropdown. Click the Done button.



The APIKeyAuth security scheme gets created and the Security screen is updated.

The tab now shows the number of ApiKeyAuth schemes created.

Also a dropdown list of the security schemes created appear now to which the newly created Security Scheme is added.

Use the button to delete the selected scheme.

The created Security Scheme gets added to global components/securitySchemes section of the underlying OAPI/SWAGGER:

```
components:
 securitySchemes:
   BasicAuth1:
     type: http
     scheme: basic
     description: Basic Auth Scheme 1
   BearerAuth1:
     type: http
     scheme: bearer
     description: Bearer Auth 1
   ApiKeyAuth1:
     type: apiKey
     in: query
     name: q1
     description: ApiKeyAuth 1
```

After you have defined the security schemes in the securitySchemes section, you can apply them to the whole API or individual operations by adding the security section on the root level or operation level, respectively. When used on the root level, security applies the specified security schemes globally to all API operations, unless overridden on the operation level.

The button Add to global security lets you add the security scheme to the root or the global level.

When the button is clicked, the underlying OAPI/SWAGGER gets updated. The selected security scheme is added to the global "security" array:

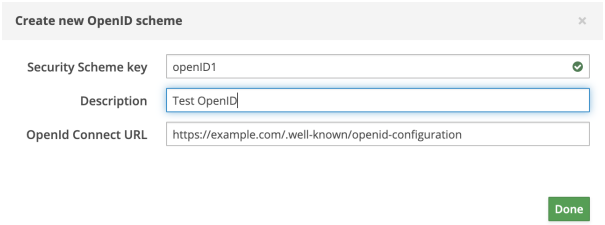
```
security:
  - BasicAuth1: []
  - BearerAuth1: []
  - ApiKeyAuth1: []
```

```
components:
  securitySchemes:
    BasicAuth1:
      type: http
      scheme: basic
      description: Basic Auth Scheme 1
    BearerAuth1:
      type: http
      scheme: bearer
      description: Bearer Auth 1
    ApiKeyAuth1:
      type: apiKey
      in: query
      name: q1
      description: ApiKeyAuth 1
```

OpenID

Create an OpenID scheme

Select the tab OpenID on the screen and click the  button.



Fill the form. Enter the security scheme name which is an arbitrary name that will be used to refer to this definition from other places in the spec, and the description.

The default value of the openID connect/server URL is provided in the form as https://server.com/.well-known/openid-configuration.

Click the 

button.

The OpenID security scheme gets created and the Security screen is updated.

The tab now shows the number of OpenID schemes created.

Also a dropdown list of the security schemes created appear now to which the newly created Security Scheme is added.

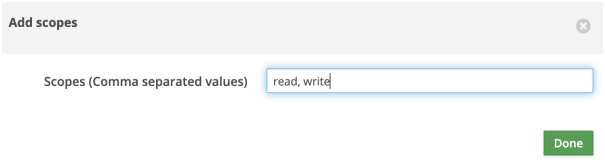
Use the  button to delete the selected scheme.

The created Security Scheme gets added to global components/securitySchemes section of the underlying OAPI/SWAGGER:

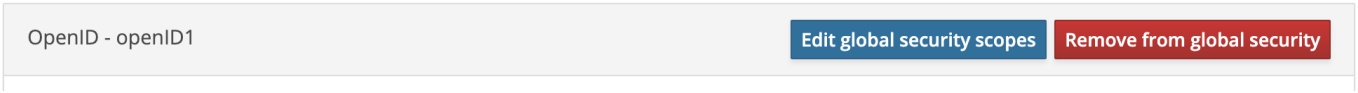
```
components:
 securitySchemes:
   BasicAuth1:
     type: http
     scheme: basic
     description: Basic Auth Scheme 1
   BearerAuth1:
     type: http
     scheme: bearer
     description: Bearer Auth 1
   ApiKeyAuth1:
     type: apiKey
     in: query
     name: q1
     description: ApiKeyAuth 1
   openID1:
     type: openIdConnect
     openIdConnectUrl: 'https://example.com/.well-known/openid-configuration'
     description: Test OpenID
```

After you have defined the security schemes in the securitySchemes section, you can apply them to the whole API or individual operations by adding the security section on the root level or operation level, respectively. When used on the root level, security applies the specified security schemes globally to all API operations, unless overridden on the operation level.

The button Add to global security lets you add the security scheme to the root or the global level. When the button is clicked, a popup appears where you are required to enter the scopes (the comma separated values).



Click the Done button. The ![](../img/SpecEditorTabSecurity/addToGlobalSecurityButton.png{width=50} button now changes to:



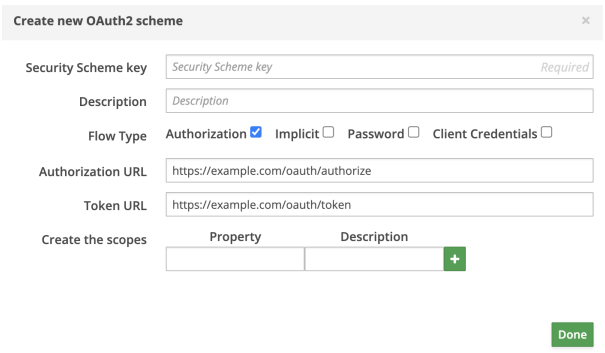
the underlying OAPI/SWAGGER gets updated. The selected security scheme is added to the global "security" array:

```
security:
  - BasicAuth1: []
  - BearerAuth1: []
  - ApiKeyAuth1: []
  - openID1:
      - read
      - write
components:
  securitySchemes:
    BasicAuth1:
      type: http
      scheme: basic
      description: Basic Auth Scheme 1
    BearerAuth1:
      type: http
      scheme: bearer
      description: Bearer Auth 1
    ApiKeyAuth1:
      type: apiKey
      in: query
      name: q1
      description: ApiKeyAuth 1
    openID1:
      type: openIdConnect
      openIdConnectUrl: 'https://example.com/.well-known/openid-configuration'
      description: Test OpenID
```

OAuth2

Create an OAuth2 scheme

Select the tab OAuth2 on the screen and click the  button.



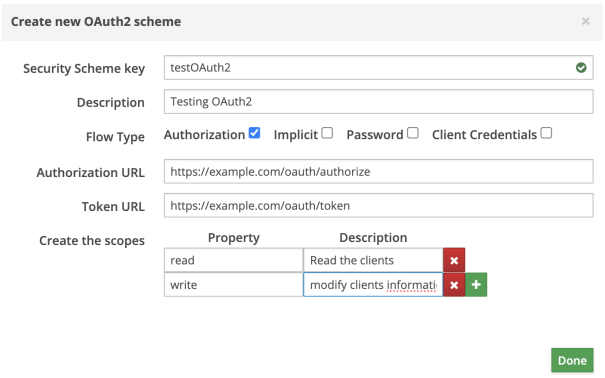
Fill the form. Enter the security scheme name which is an arbitrary name that will be used to refer to this definition from other places in the spec, and the description.

You need to select the Flow or the grant type. Read about the Oauth2 and flows here

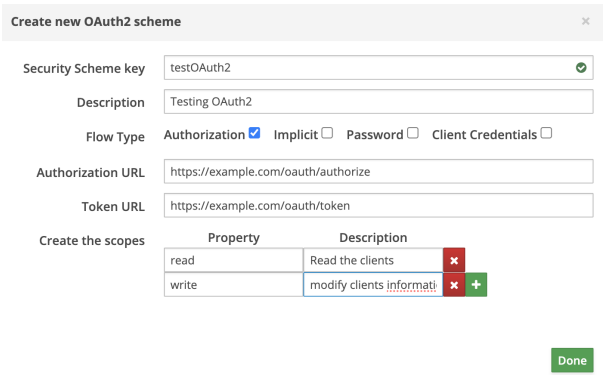
Following types of flows can be used.

1. Authorization code
2. Implicit
3. Client Credentials
4. Resource owner password credentials (or just password)

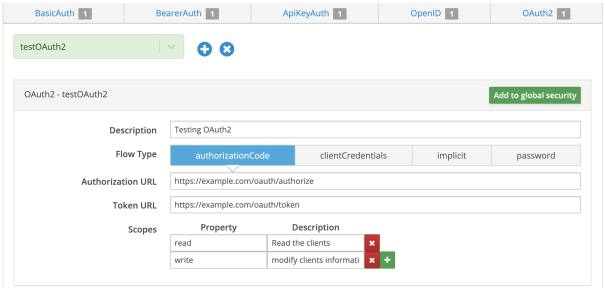
Authorization code type input form



Click the Done button.



The OAuth2 security scheme gets created and the Security screen is updated which can be edited to add more flows and scopes within the selected flow type.



The tab now shows the number of OAuth2 schemes created.

Also a dropdown list of the security schemes created appear now to which the newly created Security Scheme is added.

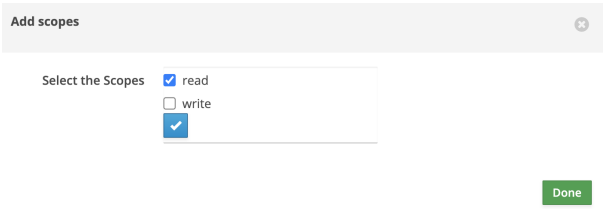
Use the ![](../img/SpecEditorTabSecurity/crossButton.png{width=50} button to delete the selected scheme.

The created Security Scheme gets added to global components/securitySchemes section of the underlying OAPI/SWAGGER:

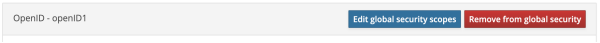
```
components:
 securitySchemes:
   BasicAuth1:
     type: http
     scheme: basic
     description: Basic Auth Scheme 1
   BearerAuth1:
     type: http
     scheme: bearer
     description: Bearer Auth 1
   ApiKeyAuth1:
     type: apiKey
     in: query
     name: q1
     description: ApiKeyAuth 1
   openID1:
     type: openIdConnect
     openIdConnectUrl: 'https://example.com/.well-known/openid-configuration'
     description: Test OpenID
   testOAuth2:
     type: oauth2
     description: 'Testing OAuth2 '
     flows:
       authorizationCode:
         authorizationUrl: 'https://example.com/oauth/authorize'
         tokenUrl: 'https://example.com/oauth/token'
         scopes:
           read: Read the clients
           write: modify clients information in your account
```

After you have defined the security schemes in the securitySchemes section, you can apply them to the whole API or individual operations by adding the security section on the root level or operation level, respectively. When used on the root level, security applies the specified security schemes globally to all API operations, unless overridden on the operation level.

The button Add to global security lets you add the security scheme to the root or the global level. When the button is clicked, a popup appears where you are required to enter the scopes (the comma separated values).



Click the Done button. The Add to global security button now changes to:



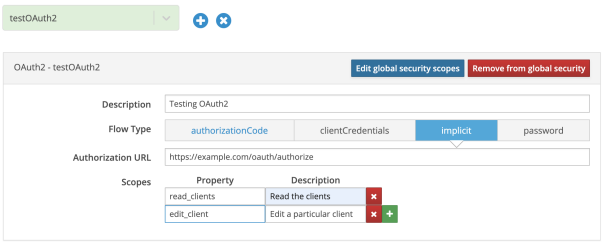
the underlying OAPI/SWAGGER gets updated. The selected security scheme is added to the global "security" array:

```
security:
  - BasicAuth1: []
  - BearerAuth1: []
  - ApiKeyAuth1: []
  - openID1:
      - read
      - write
  - testOAuth2:
      - read
components:
  securitySchemes:
    BasicAuth1:
      type: http
      scheme: basic
      description: Basic Auth Scheme 1
    BearerAuth1:
      type: http
      scheme: bearer
      description: Bearer Auth 1
    ApiKeyAuth1:
      type: apiKey
      in: query
      name: q1
      description: ApiKeyAuth 1
    openID1:
      type: openIdConnect
```

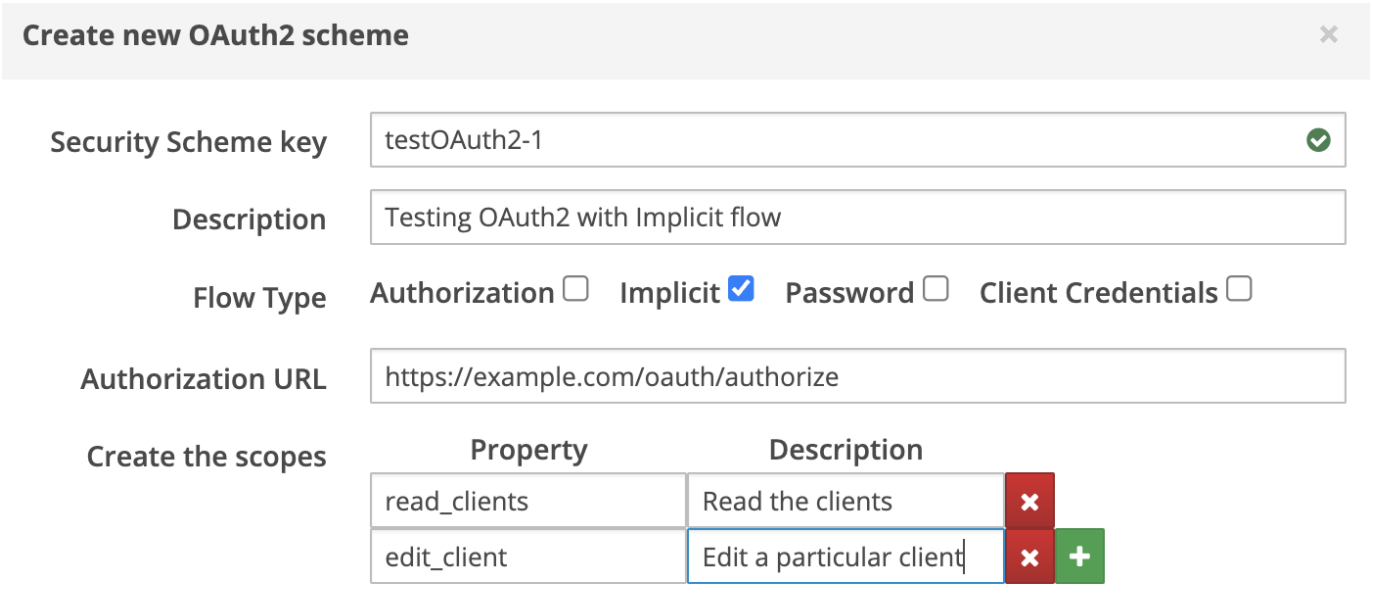
```
    openIdConnectUrl: 'https://example.com/.well-known/openid-configuration'
    description: Test OpenID
  testOAuth2:
    type: oauth2
    description: 'Testing OAuth2 '
    flows:
      authorizationCode:
        authorizationUrl: 'https://example.com/oauth/authorize'
        tokenUrl: 'https://example.com/oauth/token'
        scopes:
          read: Read the clients
          write: modify clients information in your account
```

Implicit type input form

To add an implicit type flow in the existing scheme created above (testOAuth2), use the form:



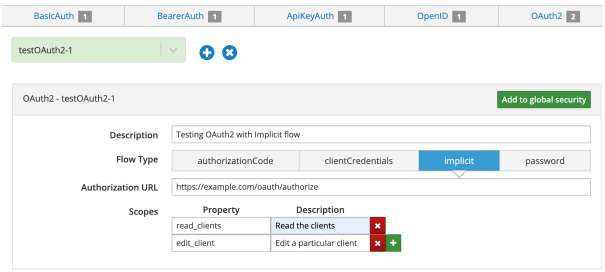
To create a new OAuth2 scheme with implicit type flow, click the  button.



and click the 

button.

The OAuth2 security scheme gets created and the Security screen is updated which can be edited to add more flows and scopes within the selected flow type.



The tab now shows the number of OAuth2 schemes created.

Also a dropdown list of the security schemes created appear now to which the newly created Security Scheme is added.

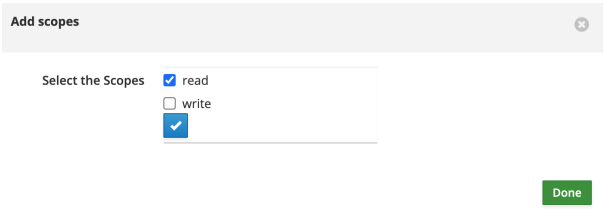
Use the  button to delete the selected scheme.

The created Security Scheme gets added to global components/securitySchemes section of the underlying OAPI/SWAGGER:

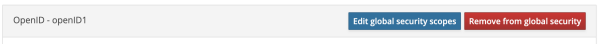
```yaml
components:
 securitySchemes:
   BasicAuth1:
     type: http
     scheme: basic
     description: Basic Auth Scheme 1
   BearerAuth1:
     type: http
     scheme: bearer
     description: Bearer Auth 1
   ApiKeyAuth1:
     type: apiKey
     in: query
     name: q1
     description: ApiKeyAuth 1
   openID1:
     type: openIdConnect
     openIdConnectUrl: 'https://example.com/.well-known/openid-configuration'
     description: Test OpenID
   testOAuth2:
     type: oauth2
     description: 'Testing OAuth2 '
     flows:
       authorizationCode:
         authorizationUrl: 'https://example.com/oauth/authorize'
         tokenUrl: 'https://example.com/oauth/token'
         scopes:
           read: Read the clients
           write: modify clients information in your account
       implicit:
         authorizationUrl: 'https://example.com/oauth/authorize'
         scopes:
           read_clients: Read the clients
           edit_client: Edit a particular client
   testOAuth2-1:
     type: oauth2
     description: Testing OAuth2 with Implicit flow
     flows:
       implicit:
         authorizationUrl: 'https://example.com/oauth/authorize'
         scopes:
           read_clients: Read the clients
           edit_client: Edit a particular client
```

After you have defined the security schemes in the securitySchemes section, you can apply them to the whole API or individual operations by adding the security section on the root level or operation level, respectively. When used on the root level, security applies the specified security schemes globally to all API operations, unless overridden on the operation level.

The button Add to global security lets you add the security scheme to the root or the global level. When the button is clicked, a popup appears where you are required to enter the scopes (the comma separated values).



Click the Done button. The Add to global security button now changes to:



the underlying OAPI/SWAGGER gets updated. The selected security scheme is added to the global "security" array:

```yaml
security:
  - BasicAuth1: []
  - BearerAuth1: []
  - ApiKeyAuth1: []
  - openID1:
      - read
      - write
  - testOAuth2:
      - read
components:
  securitySchemes:
    BasicAuth1:
      type: http
      scheme: basic
      description: Basic Auth Scheme 1
    BearerAuth1:
      type: http
      scheme: bearer
      description: Bearer Auth 1
    ApiKeyAuth1:
      type: apiKey
      in: query
      name: q1
      description: ApiKeyAuth 1
    openID1:
      type: openIdConnect
      openIdConnectUrl: 'https://example.com/.well-known/openid-configuration'
      description: Test OpenID
    testOAuth2:
      type: oauth2
      description: 'Testing OAuth2 '
      flows:
        authorizationCode:
          authorizationUrl: 'https://example.com/oauth/authorize'
          tokenUrl: 'https://example.com/oauth/token'
          scopes:
            read: Read the clients
            write: modify clients information in your account
```

Client Credentials type input form

Similar to Implicit Type.

Password type input form

Similar to Implicit Type.

Schema Tab

SCHEMA TAB

A schema is a SWAGGER/OAPI definition of a JSON payload. It is used to generate the data structures in RPG and COBOL, Plus logic which used to write or parse JSON in requests or responses by the Providers or Consumers.

This tab is used to create/edit/delete the OAPI schemas from the specification. Following sections are covered under this guide.

Create a Schema	How to create a new Schema
Edit a Schema	How to edit/delete the created Schemas

CREATING A NEW SCHEMA

A new schema can be created manually using schema editing tools, or importing information from various sources.

Imputing is handled by MDRest4i SDK reverse engineering APIs, that derive this structure from various input types. The derived schema is then inserted as a schema component in the underlying OAPI/SWAGGER specification in the compo.

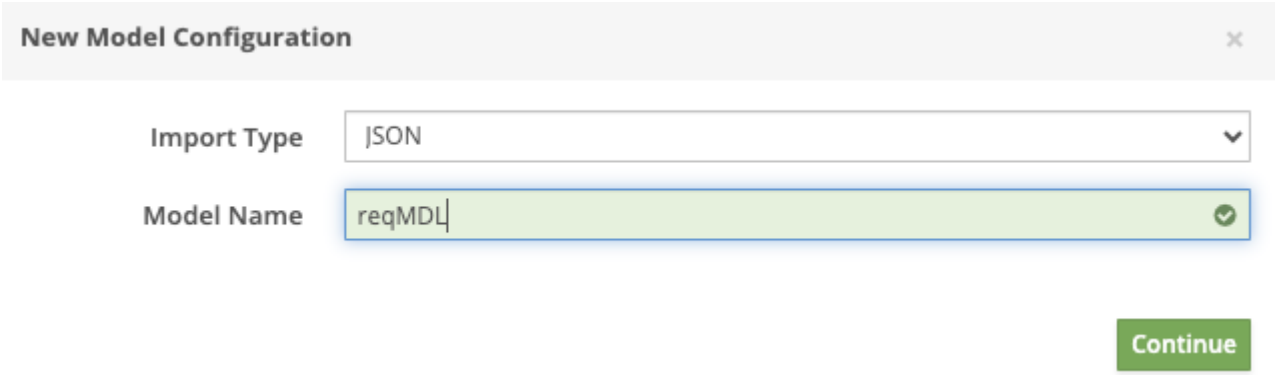
The schema can be derived from the following import types:

Import JSON	Import JSON sample from a local file, or edit sample JSON manually
Importing DB2 DDS	Import from DB2 DDS/Tables on IBM i
Importing DS RPGLE	Extract OAPI schema(s) from RPGLE Data Structure in source file
Importing DS COBOL	Extract OAPI schema(s) from Cobol Data Structure in source file
Importing Saved Schemas	Upload the schema from a JSON schema file saved on the server

Importing JSON

JSON can either be imported from a local file, from an IFS file on the server or add schema manually in the JSON editor. The JSON is then parsed and the OAPI schema inserted as a schema to the components section of the OAPI specification.

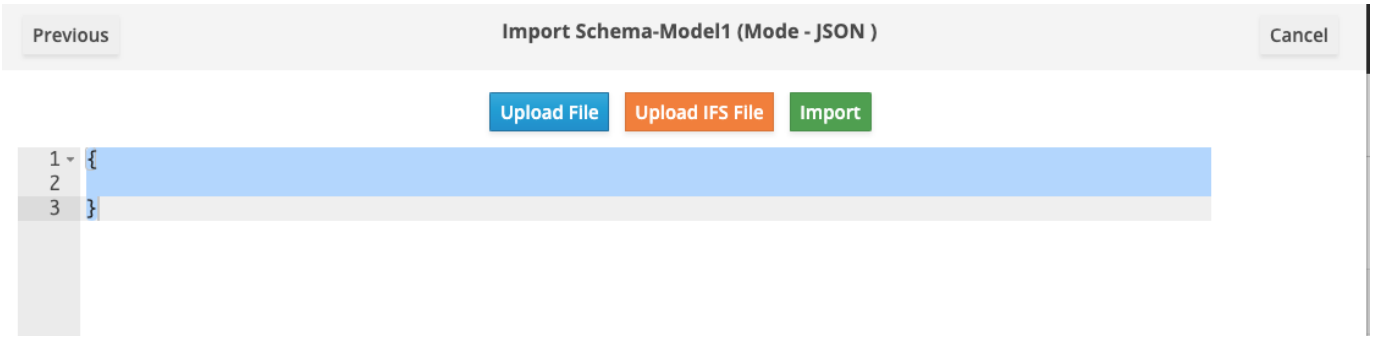
1. In the "Schema" tab, click the button to add a new schema
2. Enter the model name



JSON Import Schema Creation

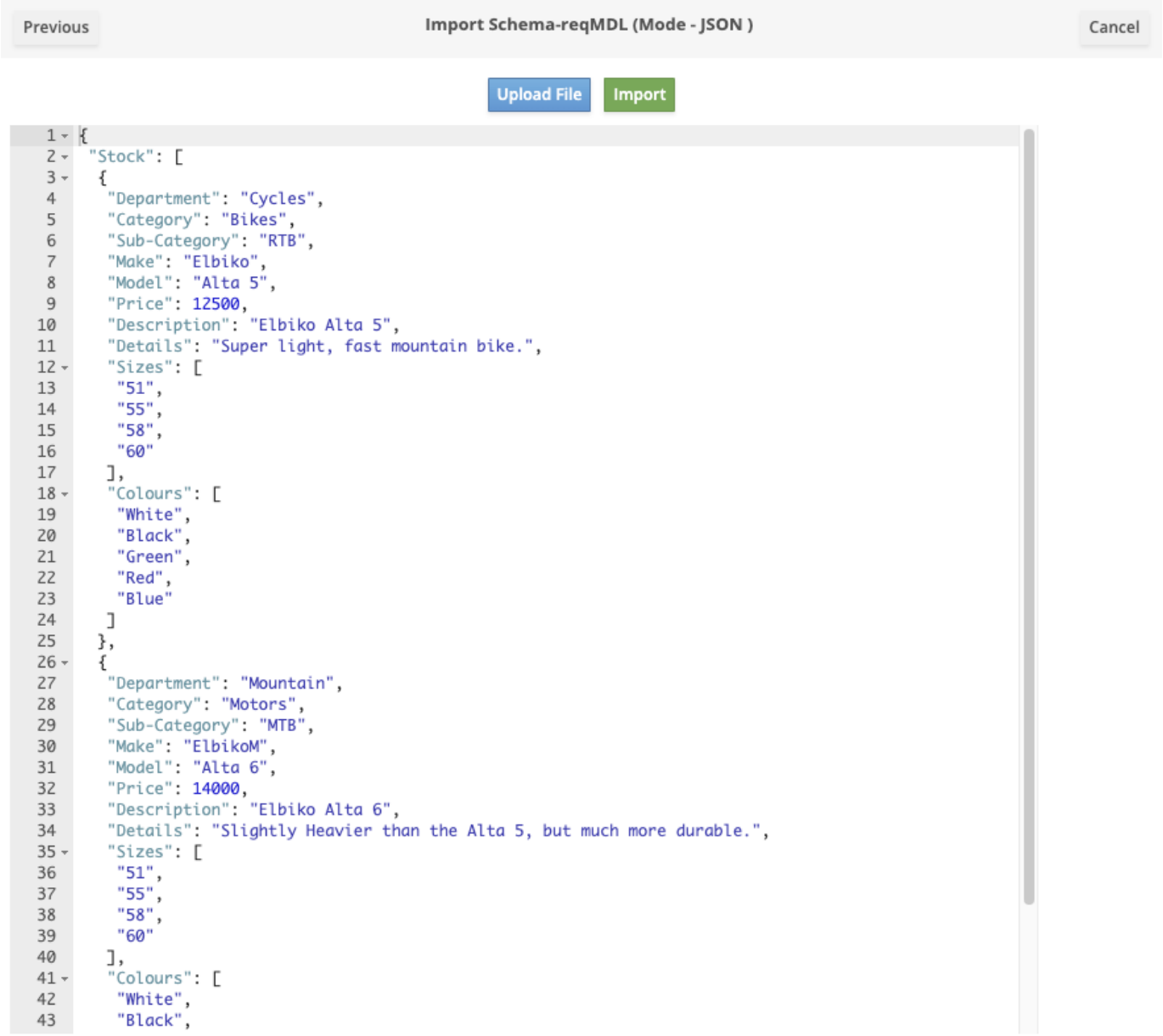
Click Continue

The schema import editor comes up.

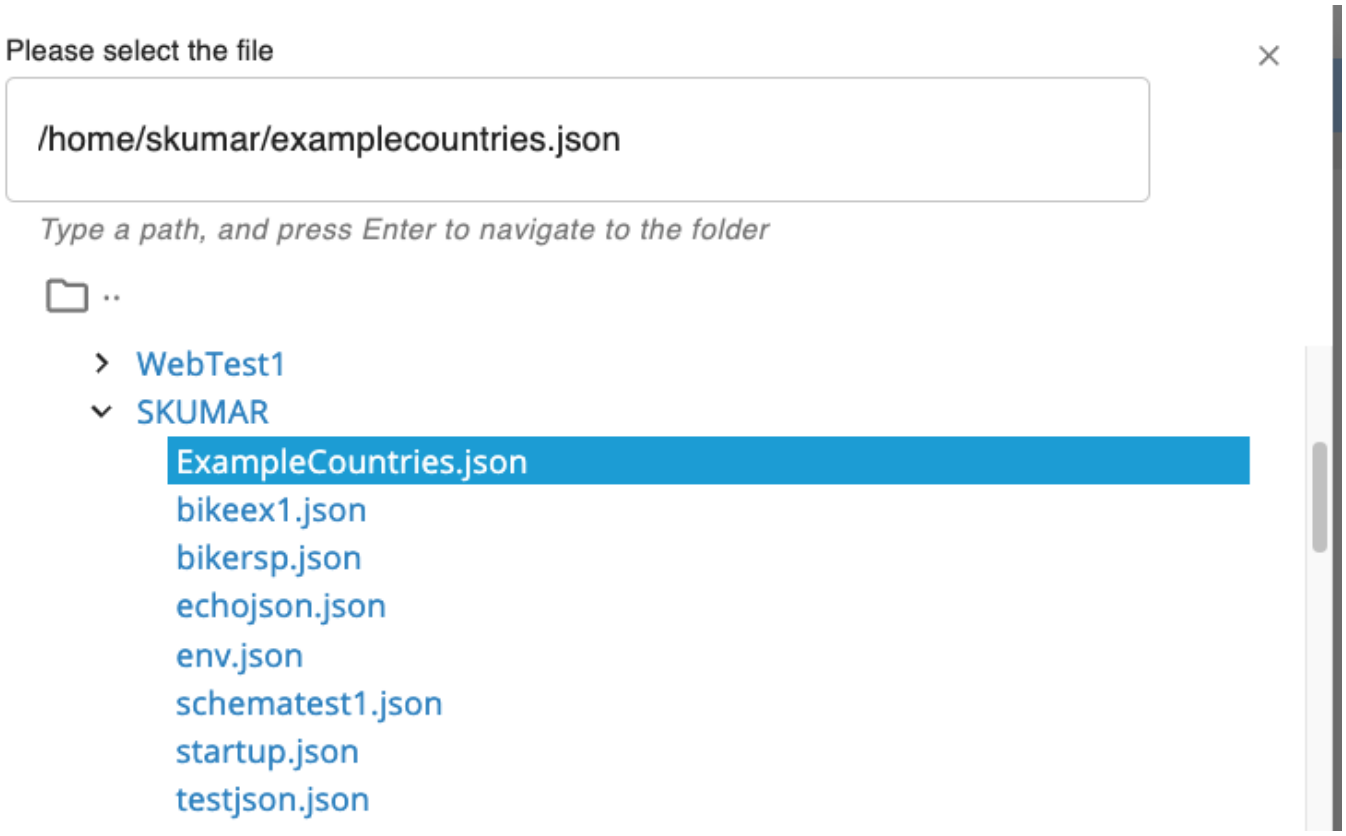


There are 2 possible ways to provide the json for import.

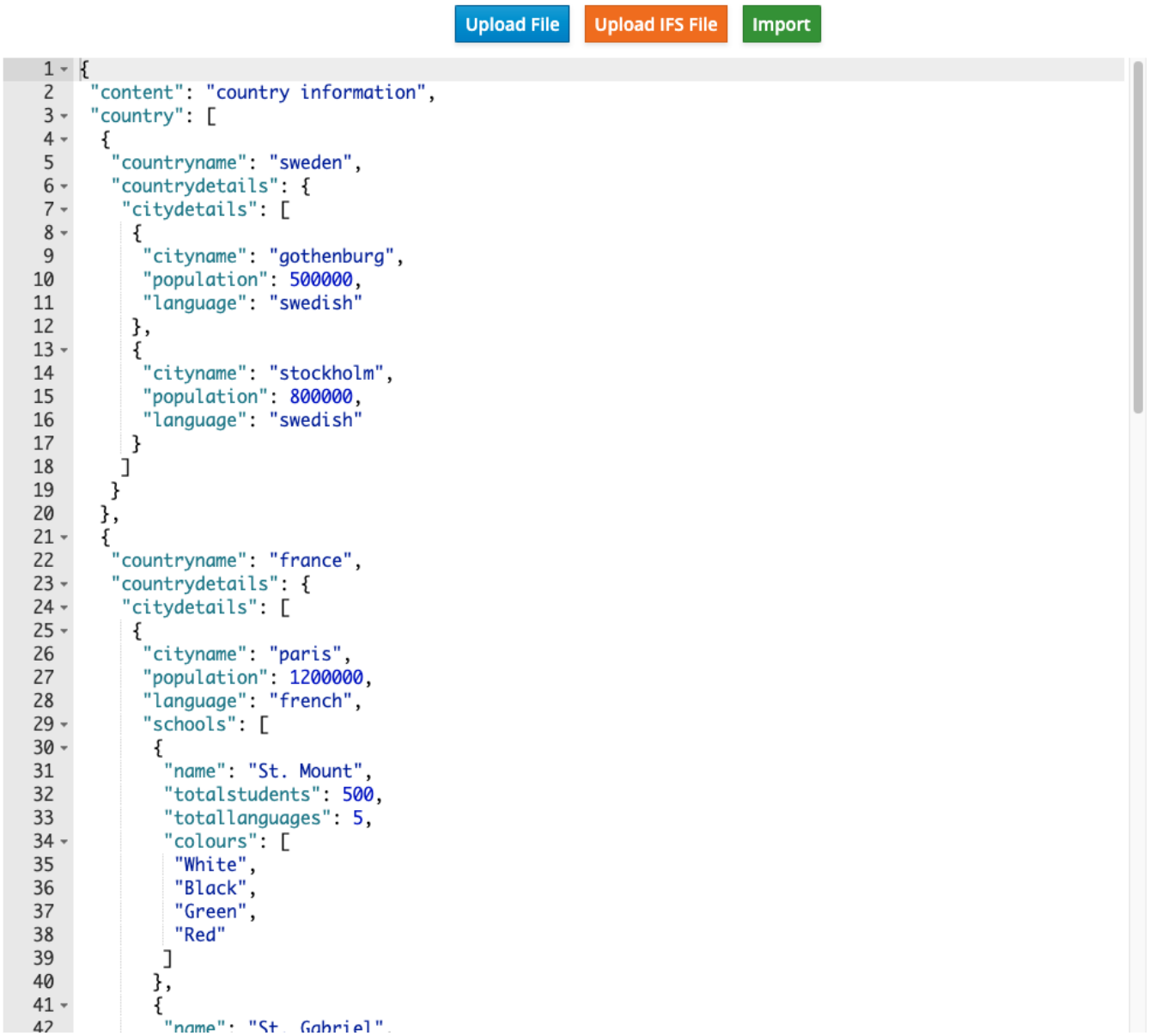
1. Local file on your system - Click the **Upload File** button. You can then select the file on your system by navigating to the required file using File explorer.



1. The IFS file on the server - Click the **Upload IFS File** button. You can then select the file on your system by navigating to the required file using File explorer.



Double click or click the "Done" button provided below to select the json file.



Click the Import button.

In the example used in case 1, the Components object in the Swagger specification would be updated as:

```yaml
components:
  schemas:
    reqMDL:
      type: object
      properties:
        Stock:
          type: array
          items:
            type: object
            properties:
              Department:
                description: Department_desc
                type: string
                example:
                  - Cycles
              Category:
                description: Category_desc
                type: string
                example:
                  - Bikes
              Sub-Category:
                description: Sub-Category_desc
                type: string
                example:
                  - RTB
              Make:
                description: Make_desc
                type: string
                example:
                  - Elbiko
              Model:
                description: Model_desc
                type: string
                example:
                  - Alta 5
              Price:
                description: Price_desc
                type: integer
                example:
                  - 12500
              Description:
                description: Description_desc
                type: string
                example:
                  - Elbiko Alta 5
              Details:
                description: Details_desc
                type: string
                example:
                  - 'Super light, fast mountain bike.'
```

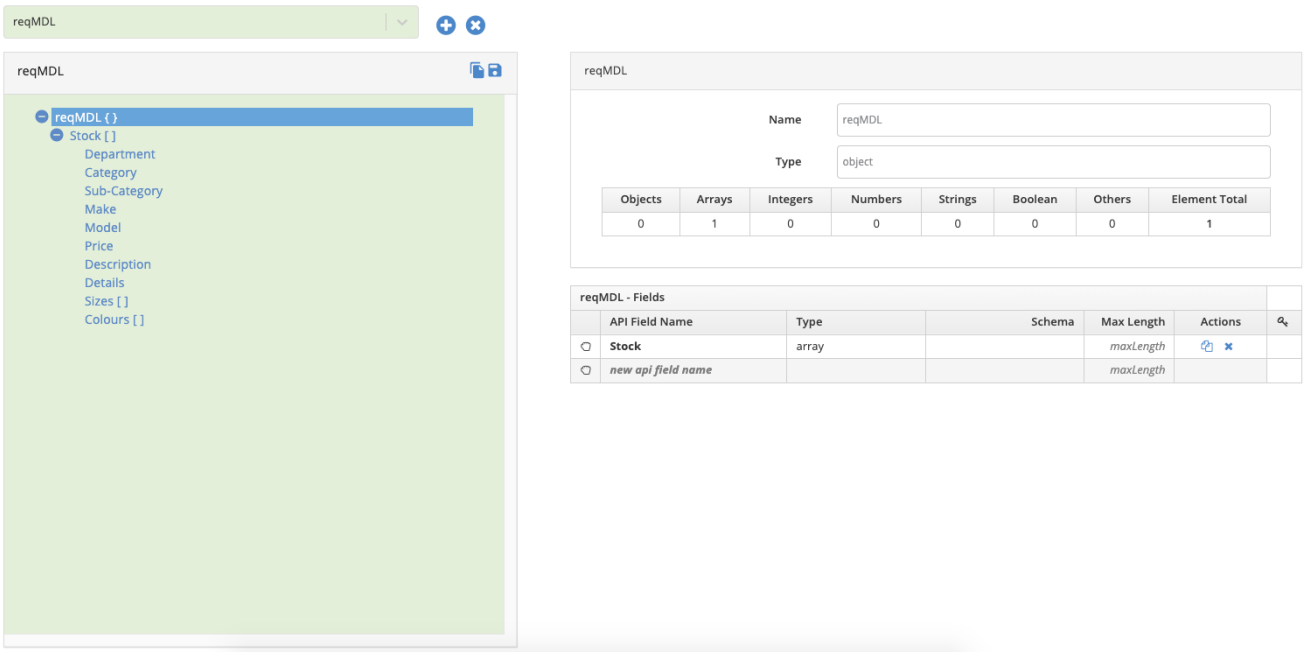
Swagger definition update

In the example used in case 2, the Components object in the Swagger specification would be updated as:

```
reqMDL:
  type: object
  properties:
    content:
      description: content_desc
      type: string
      maxLength: 19
      example: country information
    country:
      type: array
      items:
        type: object
        properties:
          countryname:
            description: countryname_desc
            type: string
            maxLength: 6
            example: sweden
          countrydetails:
            type: object
            properties:
              citydetails:
                type: array
                items:
                  type: object
                  properties:
                    cityname:
                      description: cityname_desc
                      type: string
                      maxLength: 10
                      example: gothenburg
                    population:
```

Swagger definition update

The schema gets created!



The created new Schema

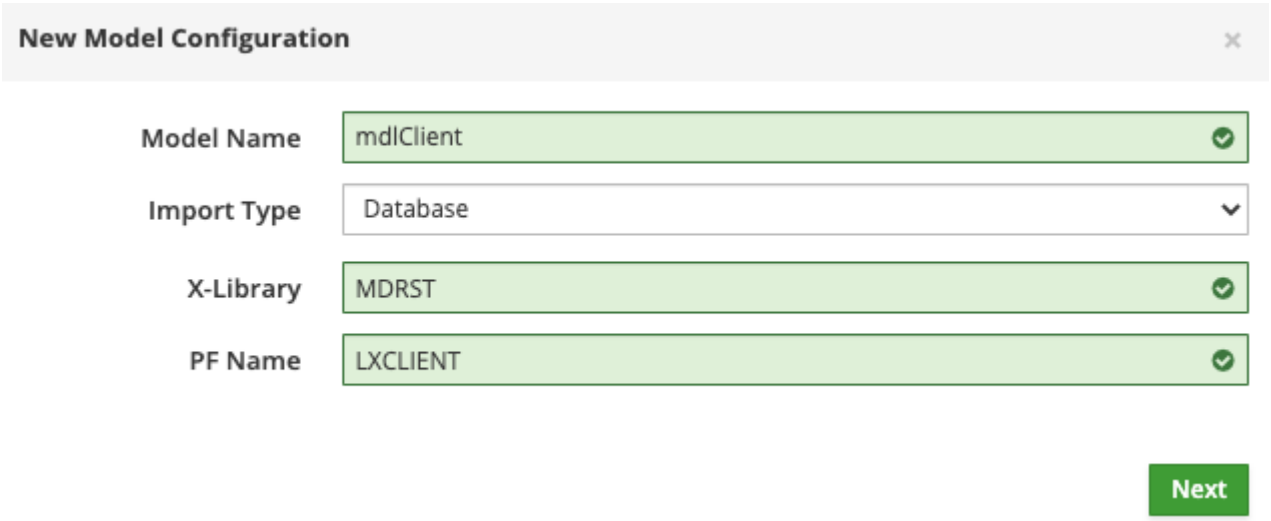
> **Adding Schema Manually**
>
> To manually create a schema using the schema editing tools, select **JSON** as the type of import, use either an empty object or array sample and select import

Importing DB2 DDS

MDRest4i uses an API on the IBM i to derive a JSON schema structure from the physical file definition.

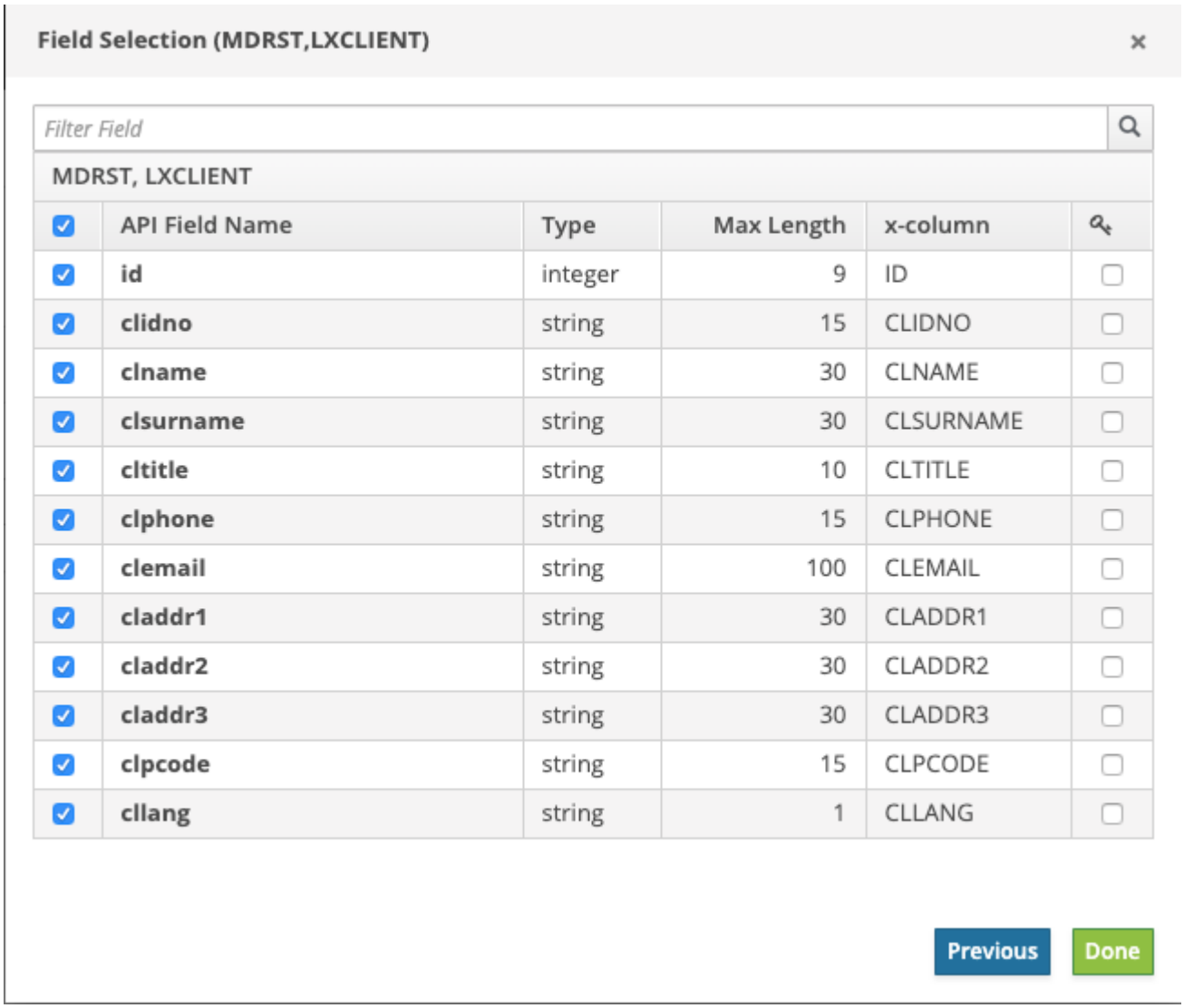
1. Click the button to add a new schema
2. Select the **"import Type"** as **"Database"** from the dropdown list.
3. Enter the library name (**X-Library**)and the physical file name(**PF Name**) which would be validated from the server.
4. Click the Next button

Note: The library name and PF name are validated on the IBM i using an Accelerator REST API written using iCore. So the **"Next"** button would be enabled only after they are validated (the green checkbox on the right of each field) and if model with the entered name does not exist.



Database Schema Extraction

After the Next button is clicked, the list of fields in the PF file LXCLIENT in Library MDRST comes up



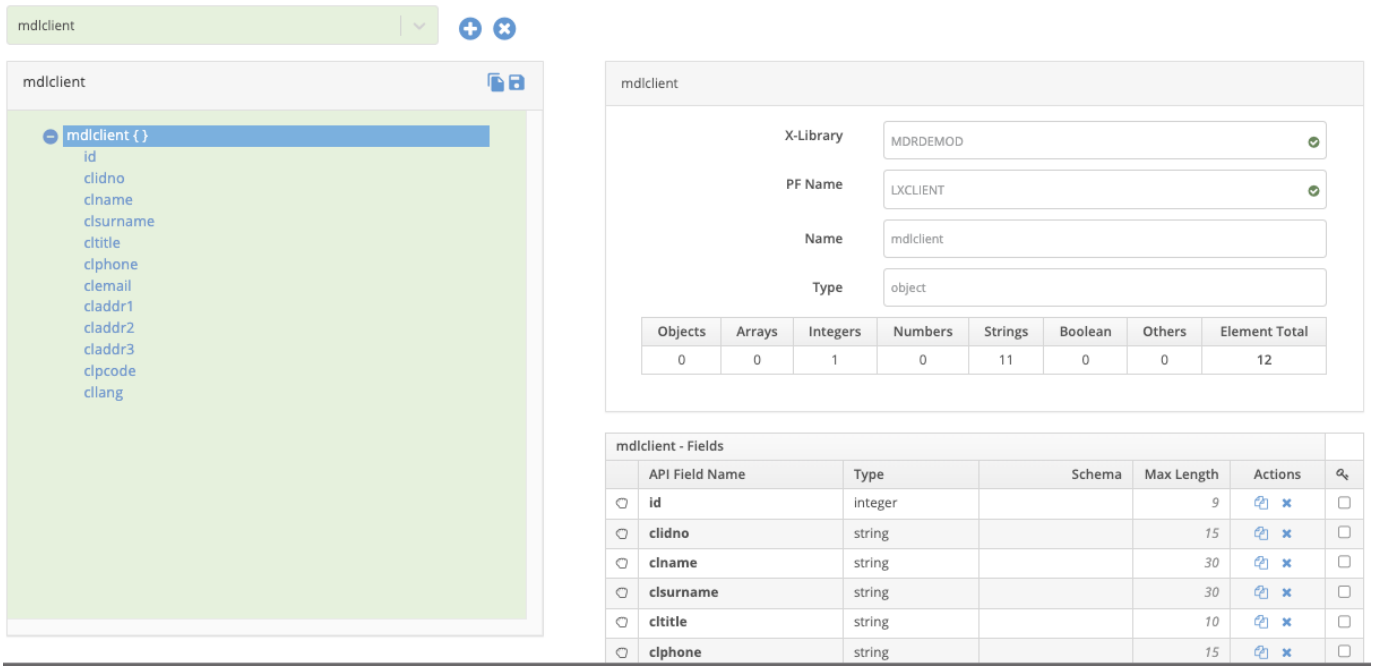
Field Selection

Select all the 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.

Click Done to continue.

The newly created schema is gets created and is added to the list of models.

!



Created Schema

The Components object would contain a new object in the Swagger specification.

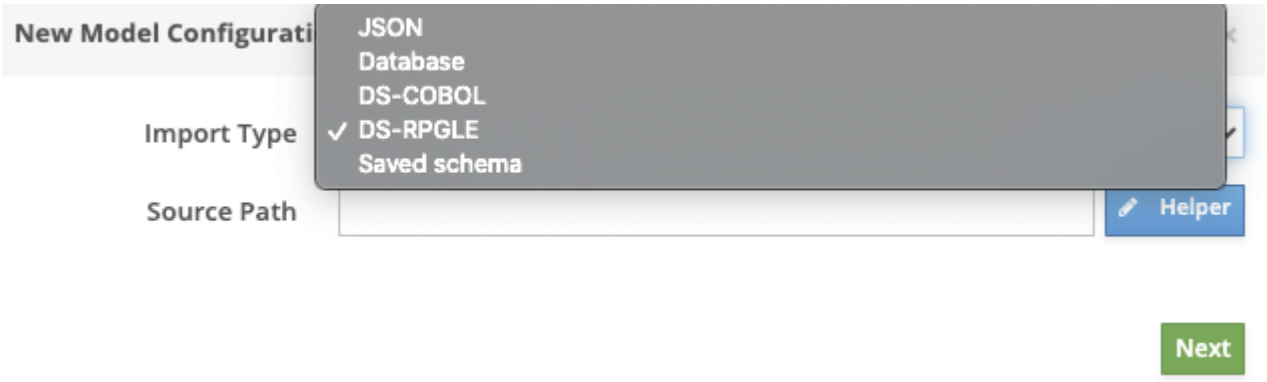
```yaml
components:
  schemas:
    mdlclient:
      x-library: MDRDEMOD
      x-pfname: LXCLIENT
      properties:
        id:
          type: integer
          description: Description of id
          maxLength: 9
          x-column: ID
        clidno:
          type: string
          description: Description of clidno
          maxLength: 15
          x-column: CLIDNO
        clname:
          type: string
          description: Description of clname
          maxLength: 30
          x-column: CLNAME
        clsurname:
          type: string
          description: Description of clsurname
          maxLength: 30
          x-column: CLSURNAME
        cltitle:
          type: string
          description: Description of cltitle
          maxLength: 10
          x-column: CLTITLE
        clphone:
          type: string
          description: Description of clphone
```

Updated Swagger

Importing from RPGLE Data Structure

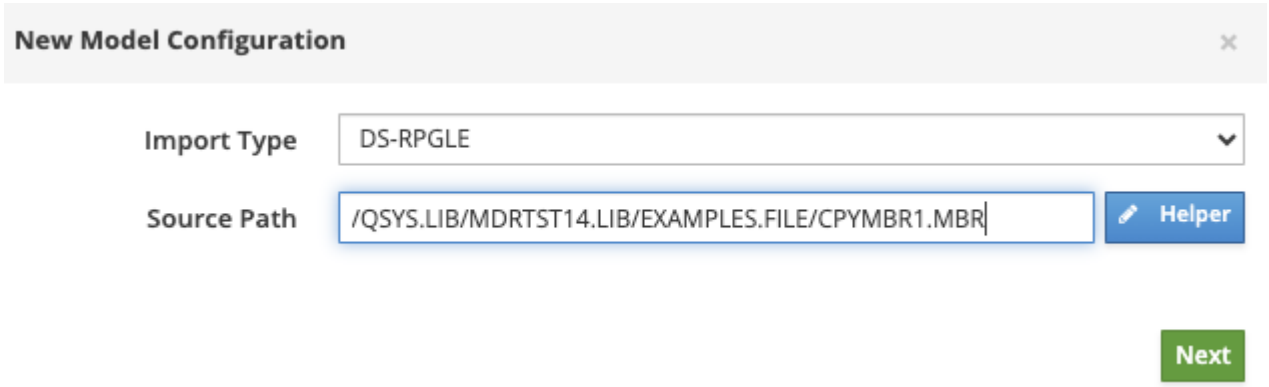
MDRest4i SDK can derive a payload definition from data structures defined in RPGLE source.

1. In the "Schema" tab, click the  button to add a new schema
2. Enter the model name



DS-RPGLE Import Schema extraction

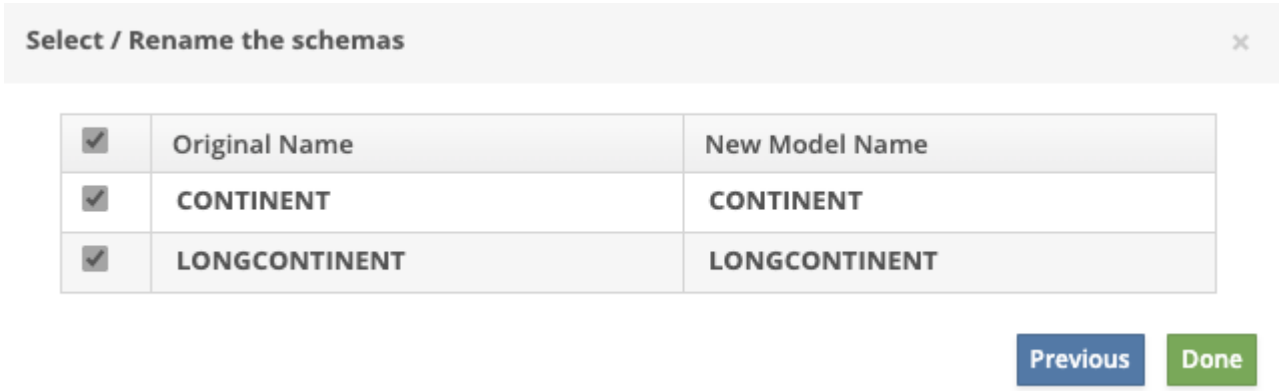
1. Provide the complete path for the Cobol file in the "Source Path"



DS-RPGLE Import Schema extraction - Source Path

Click Next

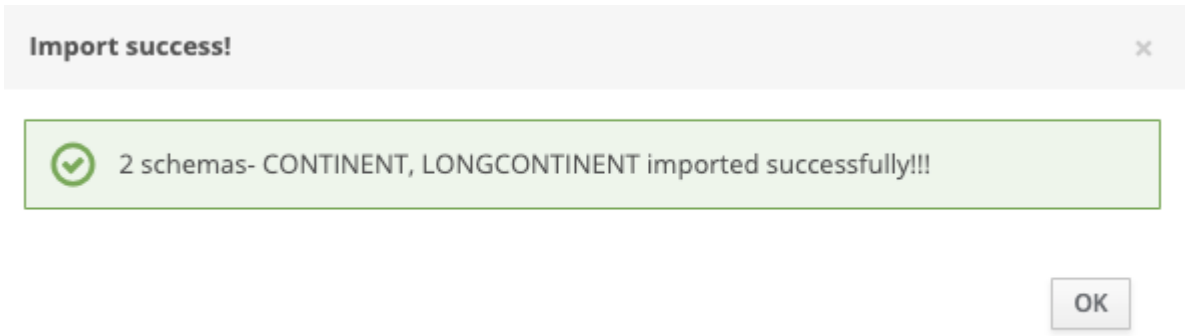
The schemas extracted from the Cobol data-structure are listed.



Schemas extracted

Please select the schemas you want to import. The model names can also be renamed by editing the names in the second column of the table.

Click the Done button.

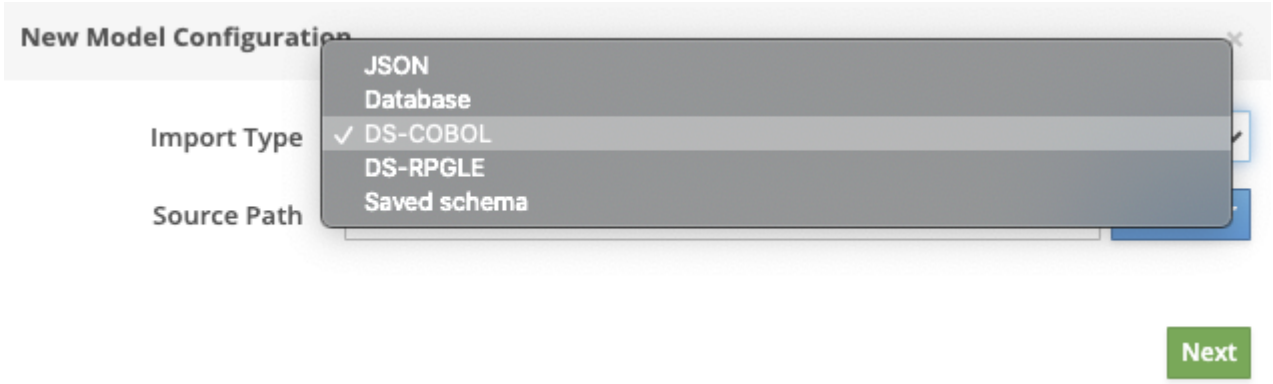


Successful Schemas extraction

Importing from COBOL Data Structure

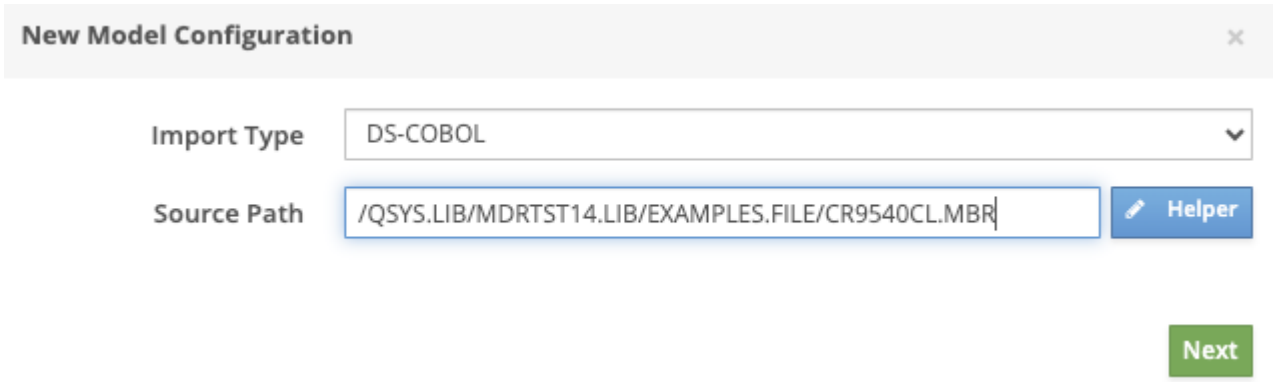
MDRest4i SDK can derive a payload definition from data structures defined in COBOL source.

1. In the "Schema" tab, click the button to add a new schema
2. Enter the model name



DS-COBOL Import Schema extraction

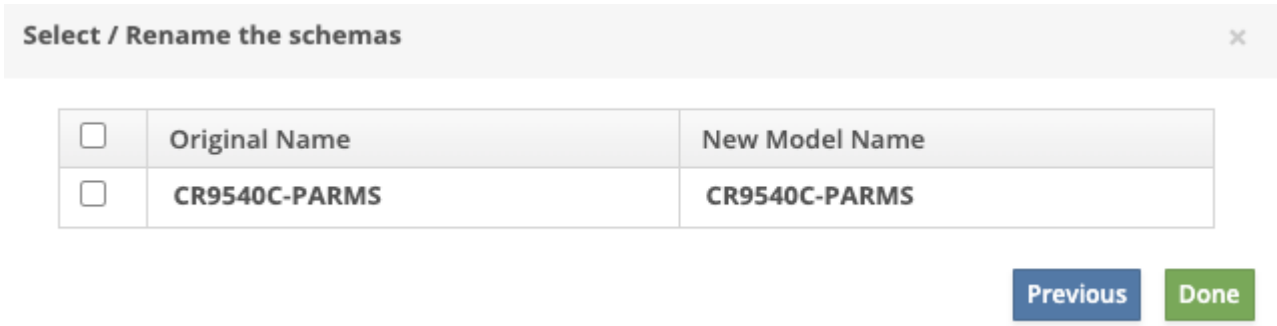
1. Provide the complete path for the Cobol file in the "Source Path"



DS-COBOL Import Schema extraction - Source Path

Click Next

The schemas extracted from the Cobol data-structure are listed.

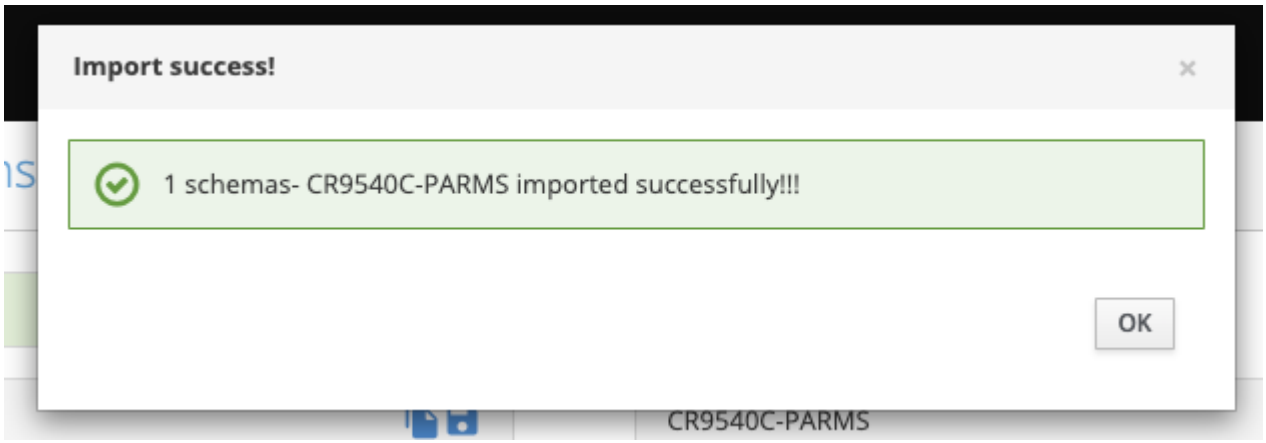


Schemas extracted

Please select the schemas you want to import. The model names can also be renamed by editing the names in the second column of the table.

Note: In our source file, there was only one schema. There could be multiple schemas too, in other cases.*

Click the Done button.

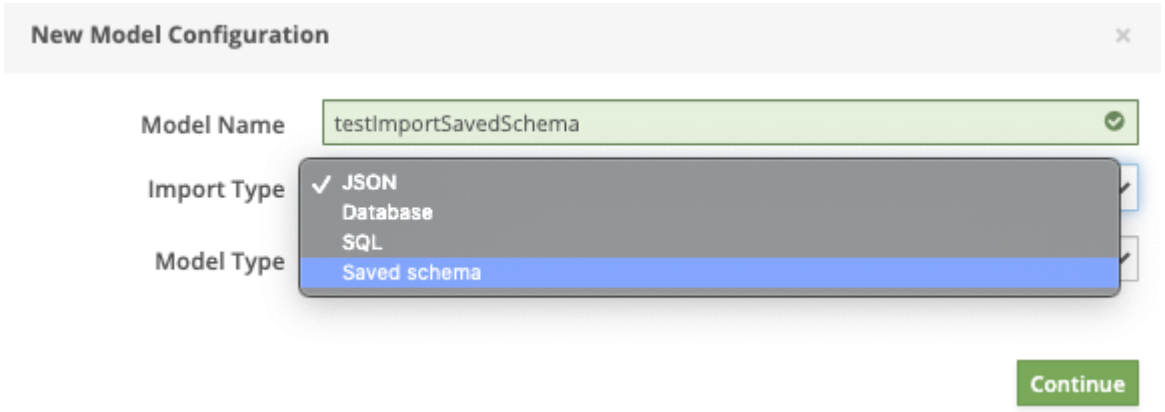


Successful Schemas extraction

Importing Saved Schemas

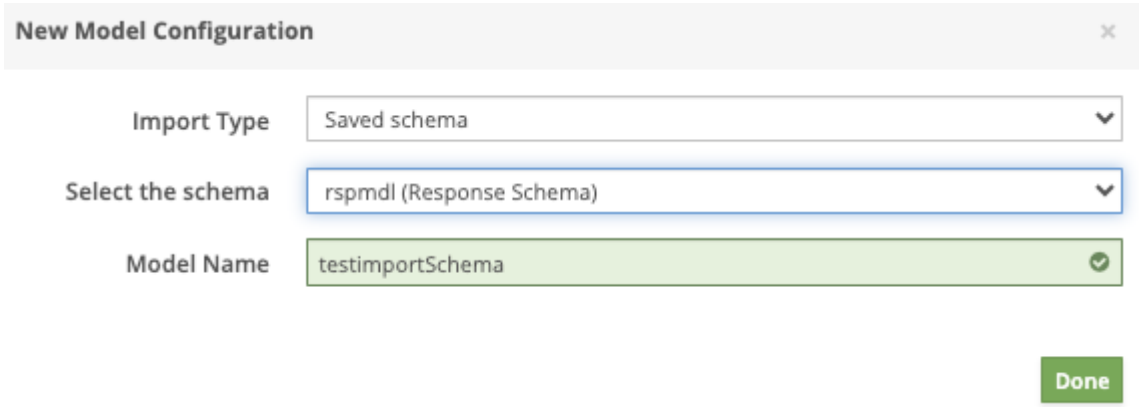
Schemas already created and saved on the server, can be imported to a specification.

Use this option to import a schema already saved on the server by any user using the Save Schema button.



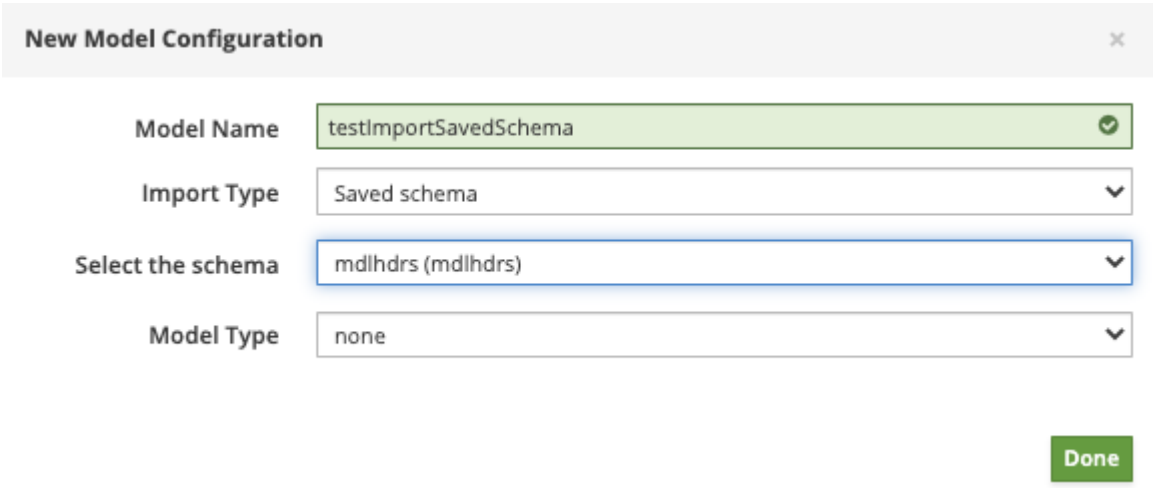
Import Saved Schema

On selecting the "Import Type", the dropdown list of the saved schemas appears from which you can select the one you wish to import.

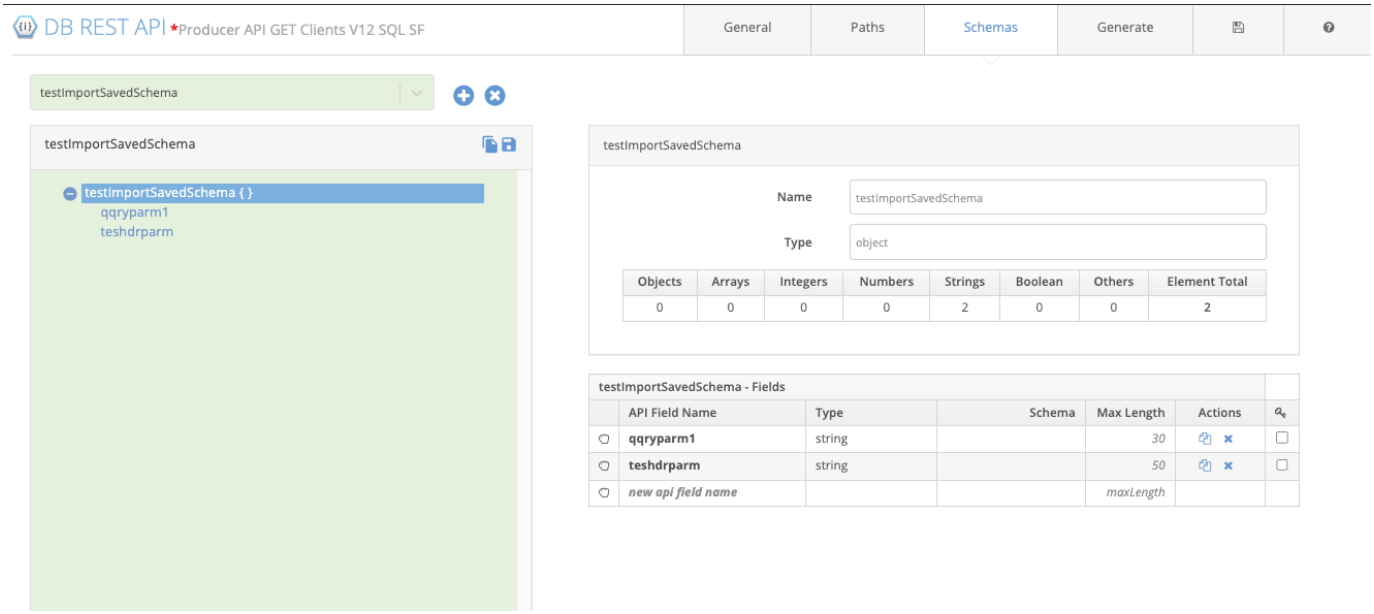


Schema Selection

Select the "Done" button.



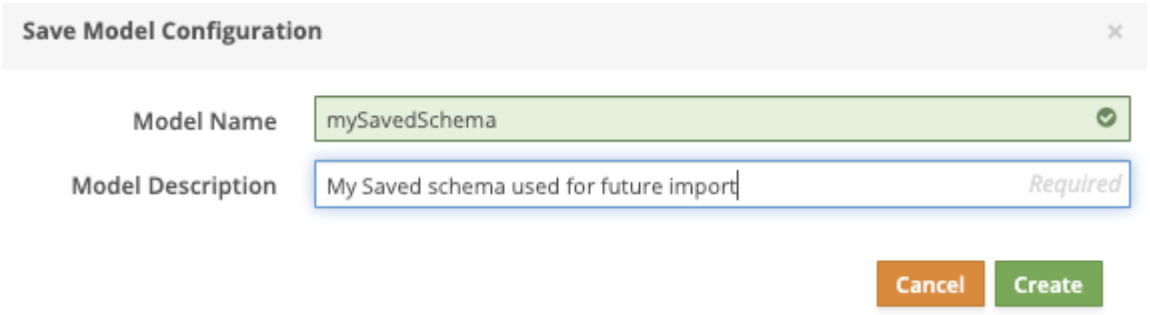
The new schema gets imported and created.



SAVING SCHEMA

Schemas can be saved on the server, and shared with other developers by importing saved schemas.

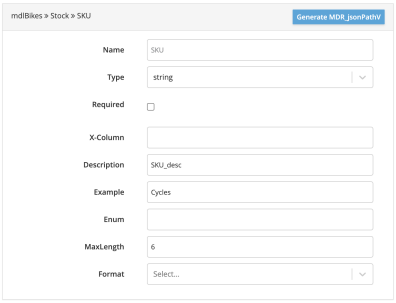
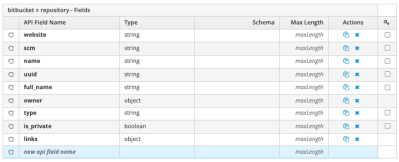
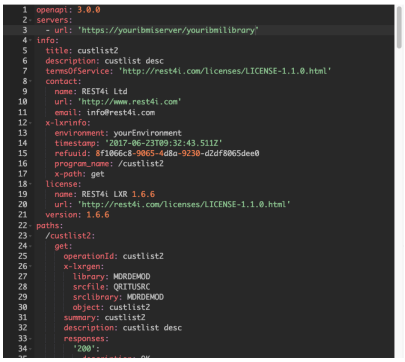
Use the  button on the schema tree (left side) to create a copy of the schema on the server. When you click this button, a popup form appears.



Enter the desired name and description and click  button. The schema is saved!

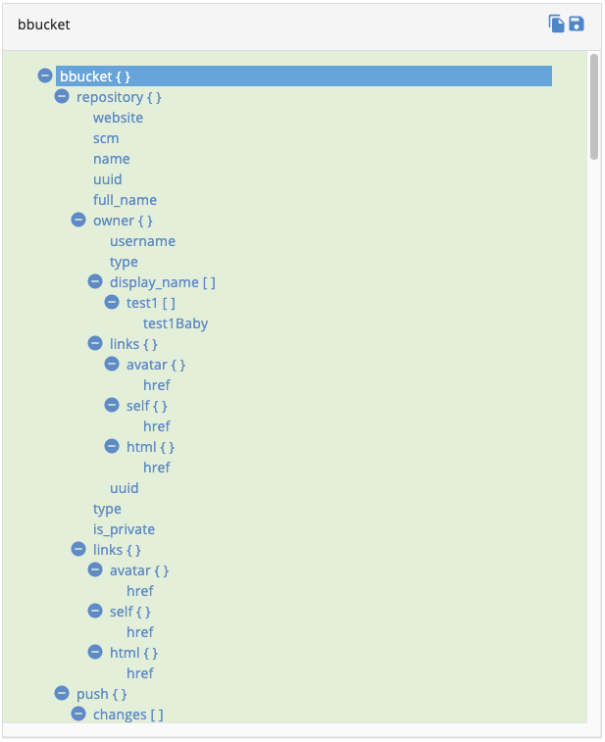
EDITING SCHEMAS

A schema can be edited in three ways:

Detailed Element Editing	 Edit the details of individual element selected from the Schema Tree
Elements Editor Table	 A table of sub-elements of a parent object selected from the Schema Tree
SWAGGER Editor	 Edit OAPI directly in the **Generate** tab, SWAGGER editor section.

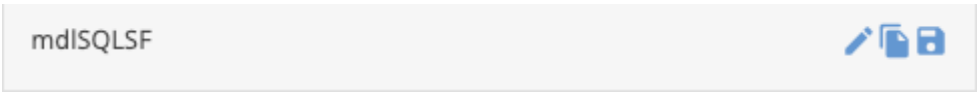
The Schema Tree

The **Schema tree** in the **Schema** tab looks like:



It contains the two sections- **Header** and the **Main Section**.

Header



Sections

Schema name / Selected schema	Displays the schema name / complete navigation path
Edit Schema Button	Used to edit schema (only if the schema was created using the SQL import)
Duplicate Schema Button	Used to copy the schema with a different name
Save Schema Button	Used to save the schema on the server

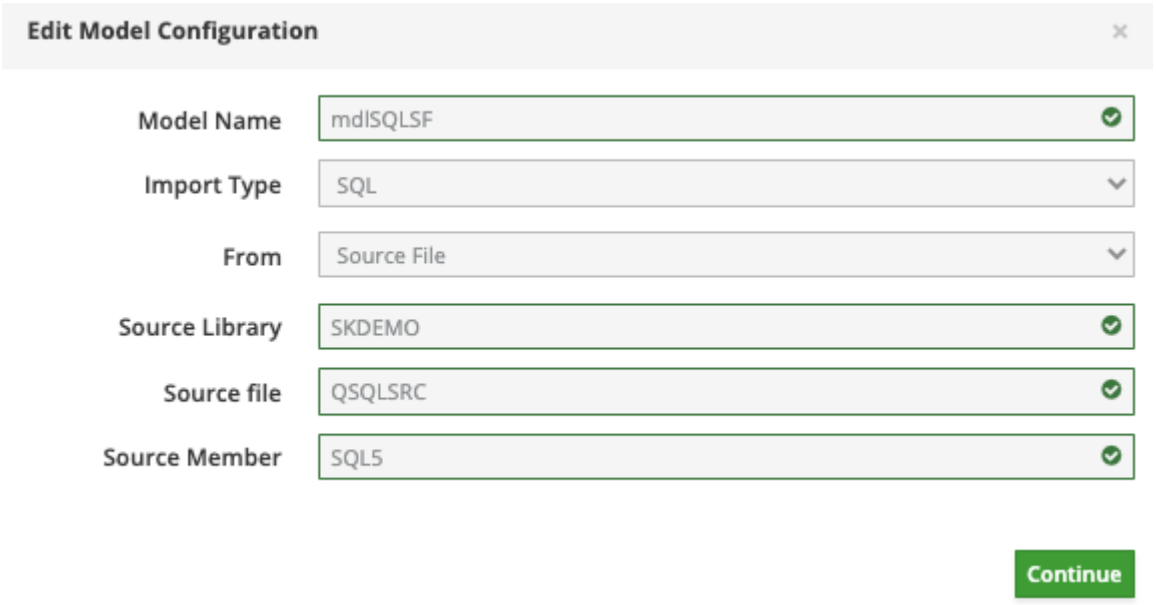
Schema name / Selected schema

The name or the schema or the navigation path in case you navigate to the other schema for viewing the schema refs.

Edit Schema Button

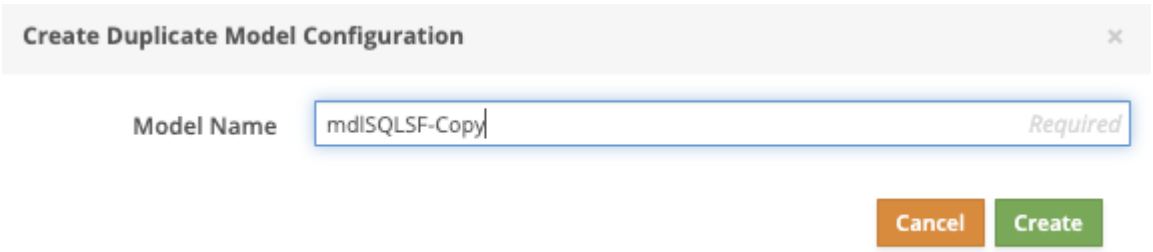
Edit the schema using the button. This button is visible only if the schema was created using the SQL import.

After the schema is created, it can be edited anytime later by clicking the button.

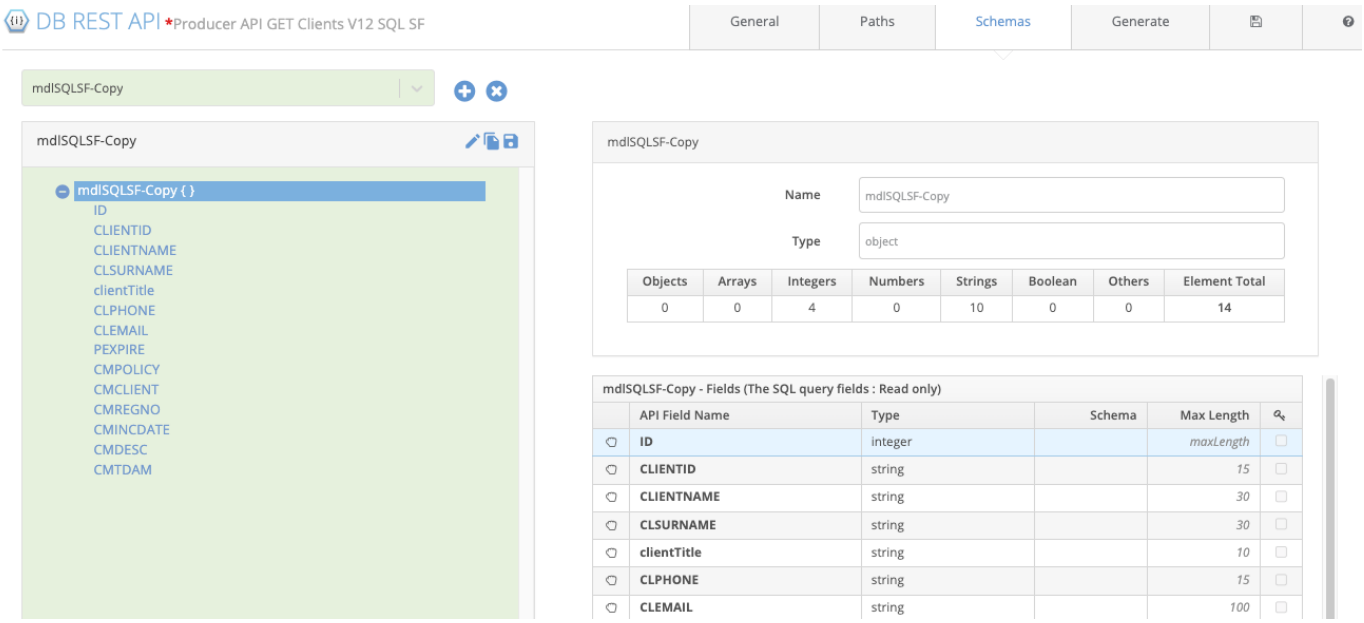


Duplicate Schema Button

Use the button to create a copy of the existing schema. On clicking the button, a popup form appears

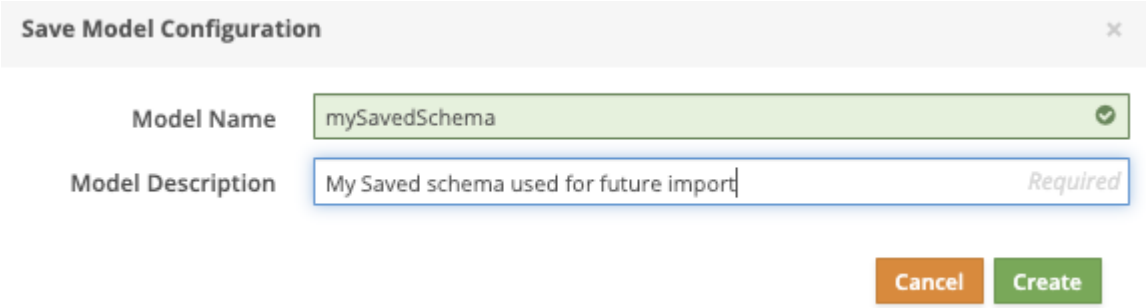


Enter the name of the new schema you want to create and click "Create". The duplicate schema is created!



Save Schema Button

Use the button to create a copy of the schema on the server. When you click this button, a popup form appears.



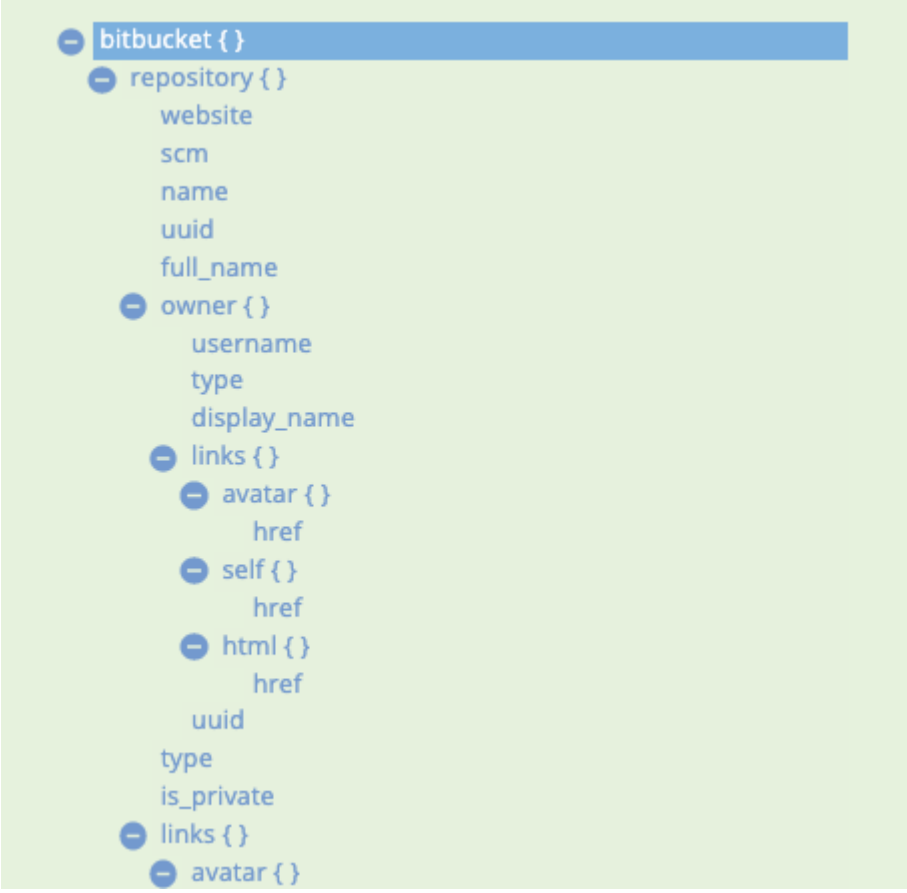
Enter the desired name and description and click Create The schema is saved!

Schema Tree Section

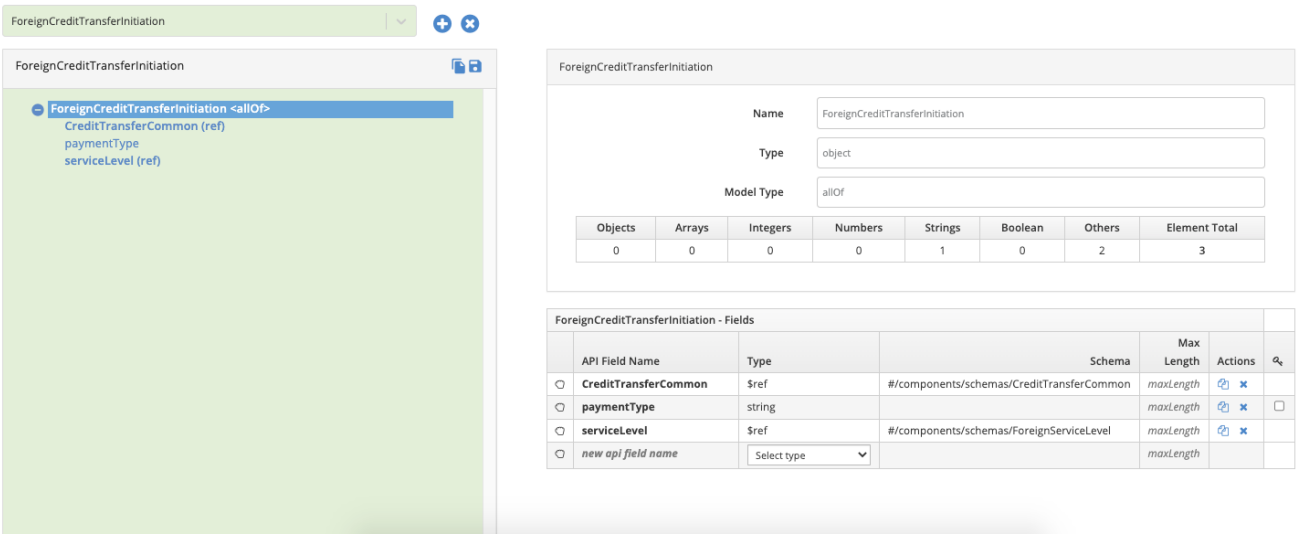
This section shows the expandable/collapsible hierarchical tree view of the schema selected.

The curly braces {} after the field name indicate that the element is an object type.

The square brackets [] after the field name indicate that the element is array type whose elements could be object type or the elementary data types.

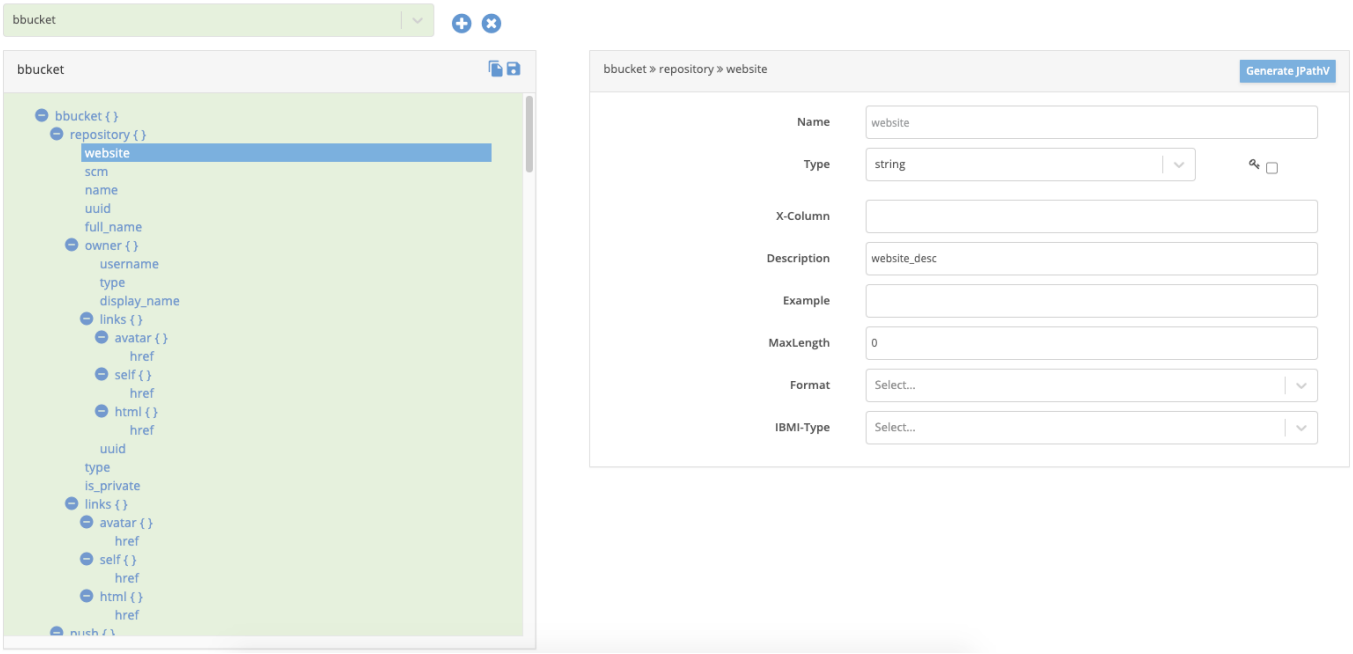


The model Types: **allOf**, **oneOf** and **anyOf** and the **ref** type fields are also depicted in the schema tree, like in the following example.



Detailed Element Editing

The basic or elementary data type properties can be edited by selecting an the element in the schema tree and editing the detail in the form which appears on the right side.

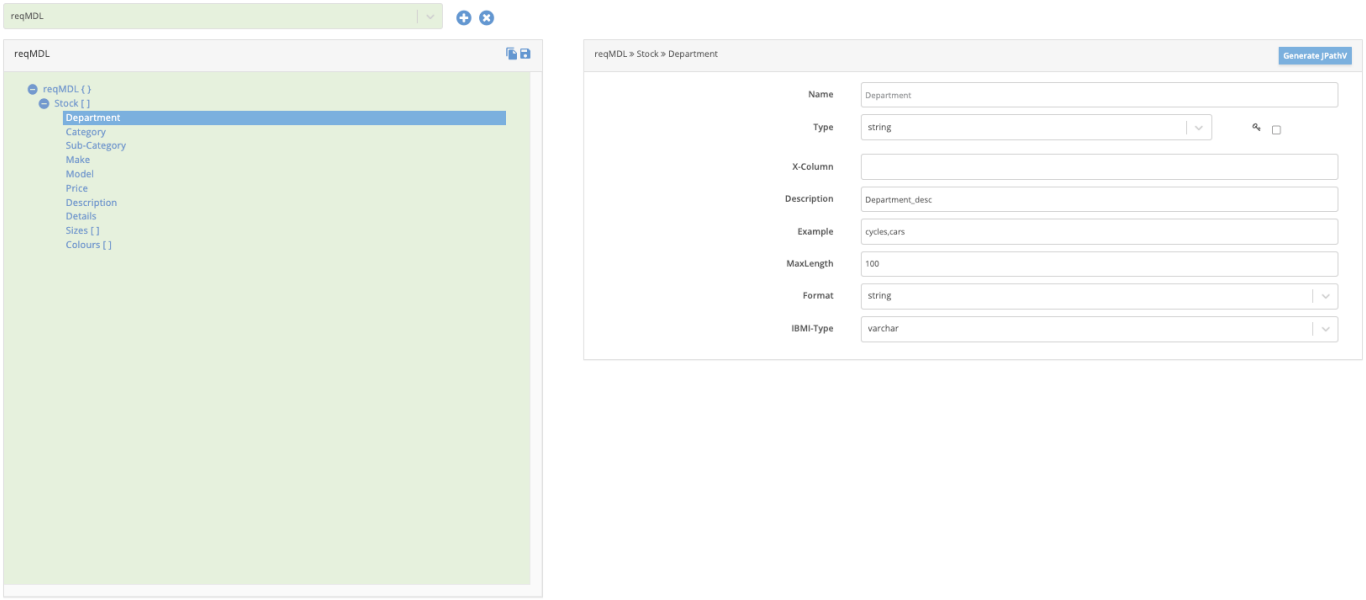


The detail form contains a combination of the following input fields shown in the table below. The Details Form format varies depending on the basic data type.

Field	Description	Corresponding Swagger field
Type	The type of element to be selected from the dropdown list	type
Example	Editable, can be comma separated multiple values	example (an array)
X-Column	Column name in the DB2 Table or Physical File	x-column
Key	If the selected field is the key-field	x-key (boolean)
Description	Description of the field	description
MaxLength	The maximum length in case of **string** type elements	maxLength
Maximum	This parameter is also considered for identifying the maximum decimal positions when the data type is **“number”**.	maximum
For example, if the maximum is set to 9.9999999, that means the variable will be declared with 10s,6 (if **“x-ibmitype”** is set as “zoned”).		
Otherwise, it will be defined as 10p, 6.	maximum	
IBMI-Type	Possible values are “zoned”, “packed” or "binary" when the item type is **“number”** or **“integer”**	x-ibmitype
Possible values are “char” or “varchar” when the item type is **“string”**		
Format	The possible values for this entry could be “int32” or "int64” for **“integer”** type	format
and that for **“number”** type could be “float” or “double". The format **“int32”** causes the variable definition with 10i,0 or 10p,0 or 10s,0 depending on whether **IBMI-Type** is specified or not		
When “int64” is there, the length 10 becomes 20. The format “float” causes the variable to be defined as 9b,5		
whereas "double" causes 16p,11 or 16s,11 or 9b,5 definition depending on **IBMI-Type**.		
For **"string"** type, the possible values are 'date' or 'string'*		
MultipleOf	This parameter tells the domain of the values but here it is used to identify the number of decimal positions.	multipleOf
As an example, if the value is **multipleOf: 0.0001** and the data type is **“number”**, that means the decimal positions are 4.		

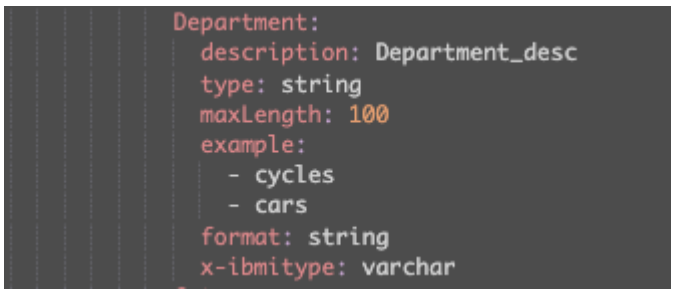
Basic Data Types String Type

Consider the following schema example. The detail form for editing the string type variables looks like:



Basic type - String:Schema Detail

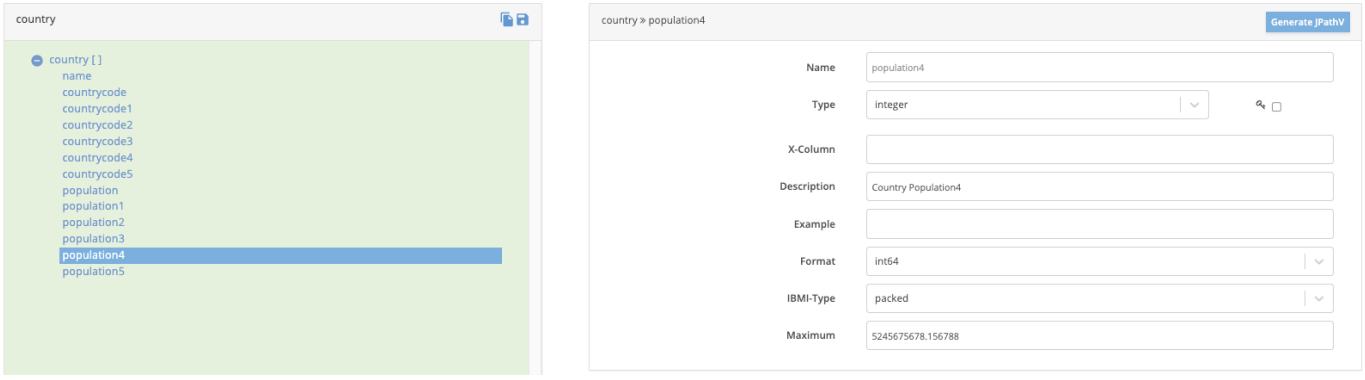
The corresponding Swagger object for the selected property **"Department"** looks like:



Basic type - String:Schema Detail

Integer Type

Consider the following schema example. The detail form for editing the string type variables looks like:



Basic type - Integer:Schema Detail

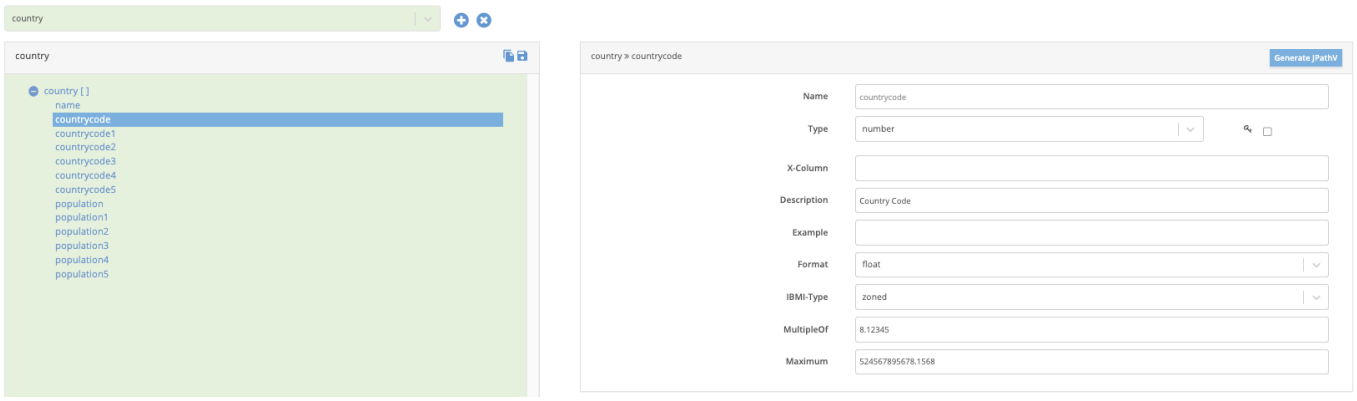
The corresponding Swagger object for the selected property **"population4"** looks like:

```
population4:
  type: integer
  x-ibmitype: packed
  format: int64
  description: Country Population4
  maximum: 5245675678.156788
```

Basic type - Integer:Schema Detail

Number Type

Consider the same schema example used for the **integer** type property. The detail form for editing the number type variables looks like:



Basic type - Number:Schema Detail

We are viewing the details of the number type field: **"countrycode"**

The corresponding Swagger object for the selected property **"countrycode"** looks like:

The corresponding Swagger object for the selected property **"Department"** looks like:

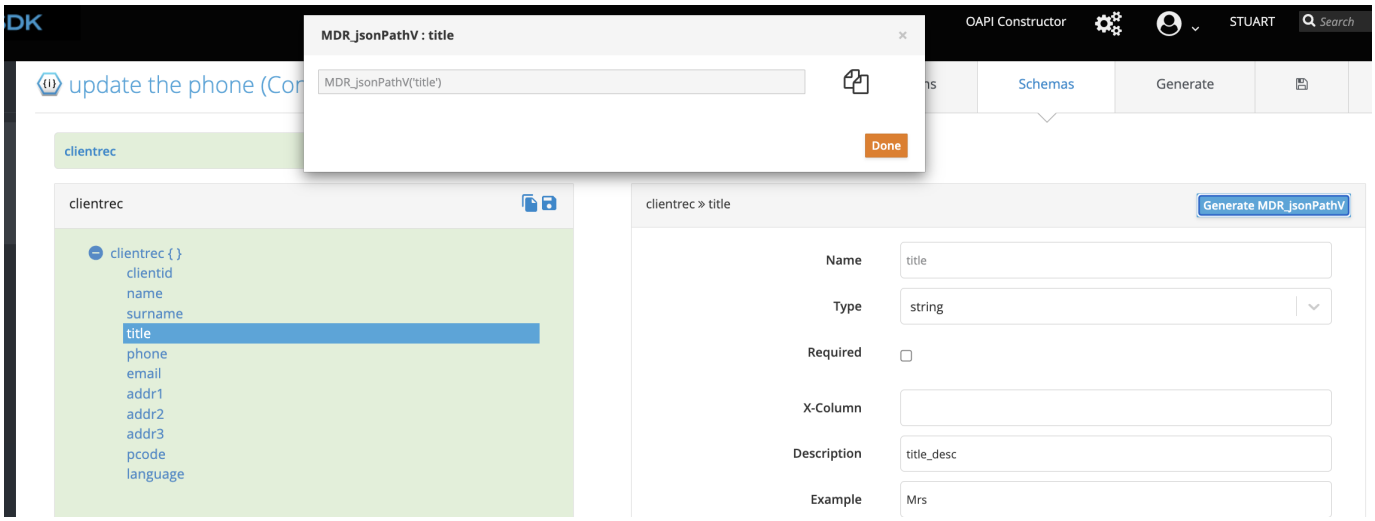
```
countrycode:
  type: number
  x-ibmitype: zoned
  description: Country Code
  multipleOf: 8.12345
  maximum: 524567895678.1568
  format: float
```

Basic type - String:Schema Detail

Generate MDR_jsonPathV

Use the Generate MDR_jsonPathV button on the header of the Detail Section, to generate the RPGLE snippet. This logic can be used to parse the value from this schema element of the selected property in the schema.

In the example below we have generated the **MDR_jsonPathV** syntax of the field named "display_name". The popup shows the generated MDR_jsonPathV statement



The value can be copied using the "Copy" button provided alongside and pasted into your RPGLE program.

For example, take the request body JSON sample below:

```
{
  "clientid": "8403211024083",
  "name": "Daisy",
  "surname": "Doolittle",
  "title": "Mrs",
  "phone": "0115557777",
  "email": "daisy@soc.gov",
  "addr1": "1 Impala Avenue",
  "addr2": "Buckburg",
  "addr3": "",
  "pcode": "9203",
  "language": "E"
}
```

In a provider program, the rpg logic below, will return the value Mrs into the title subfield for qualified data structure clientrec:

```
dcl-ds clientrec qualified;
  clientid  varchar(15)     inz;
  name      varchar(30)     inz;
  surname   varchar(30)     inz;
  title     varchar(30)     inz;
  phone     varchar(15)     inz;
  email     varchar(100)    inz;
  addr1     varchar(30)     inz;
  addr2     varchar(30)     inz;
  addr3     varchar(30)     inz;
  pcode     varchar(15)     inz;
  language  varchar(1)      inz;
end-ds

clientrec.title = MDR_jsonPathV(handle:'title');
```

Elements Editor Table

This section lets you view / edit / add / delete fields in the object / array element selected on the Schema Tree

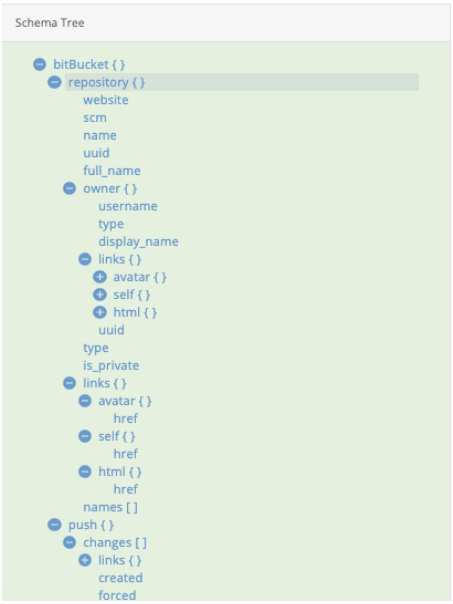
bitbucket » repository - Fields

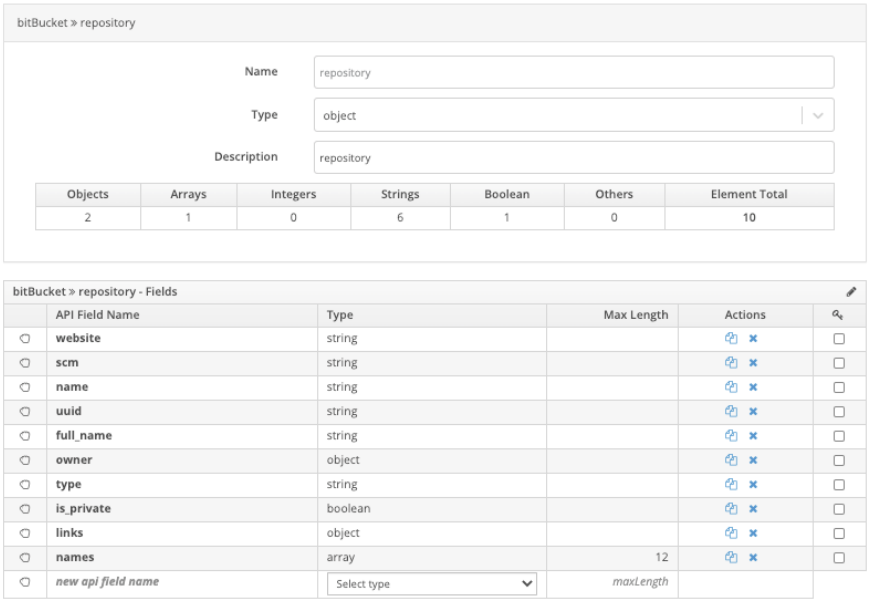
	API Field Name	Type	Schema	Max Length	Actions	
	website	string		maxLength		
	scm	string		maxLength		
	name	string		maxLength		
	uuid	string		maxLength		
	full_name	string		maxLength		
	owner	object		maxLength		
	type	string		maxLength		
	is_private	boolean		maxLength		
	links	object		maxLength		
	new api field name			maxLength		

Re-Order the fields

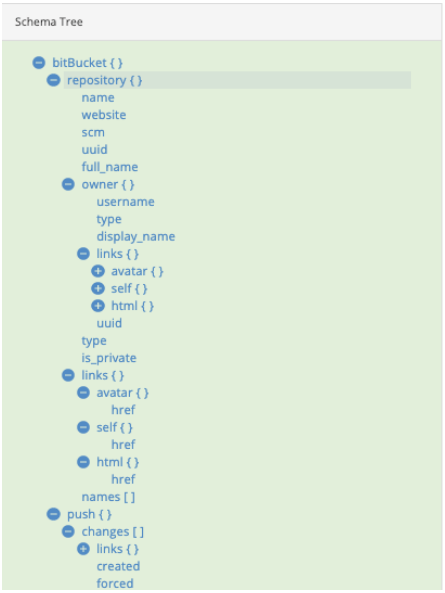
Click on the icon to drag and drop the fields.

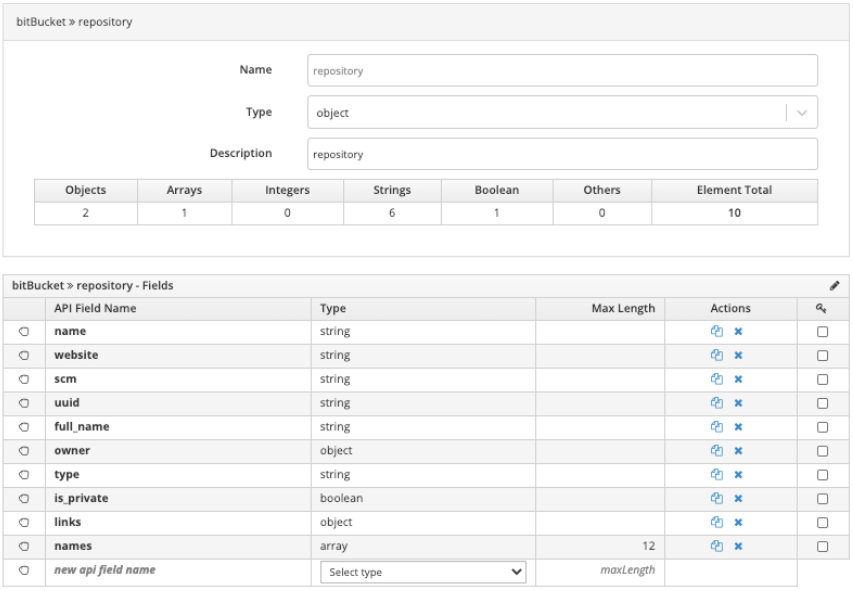
For Example:





If we bring the field "name" to the top, the schema screen looks like





Delete a field

Click on the ✖ button under the Actions column to delete a field.

Copy a field

Click on the button under the Actions column to copy a field.

Add a field

Add the API Field name, Select the data type from the dropdown list and add the Max Length. The new field is created!

JSON Structure Editing:

The non-basic element types are included in this section.

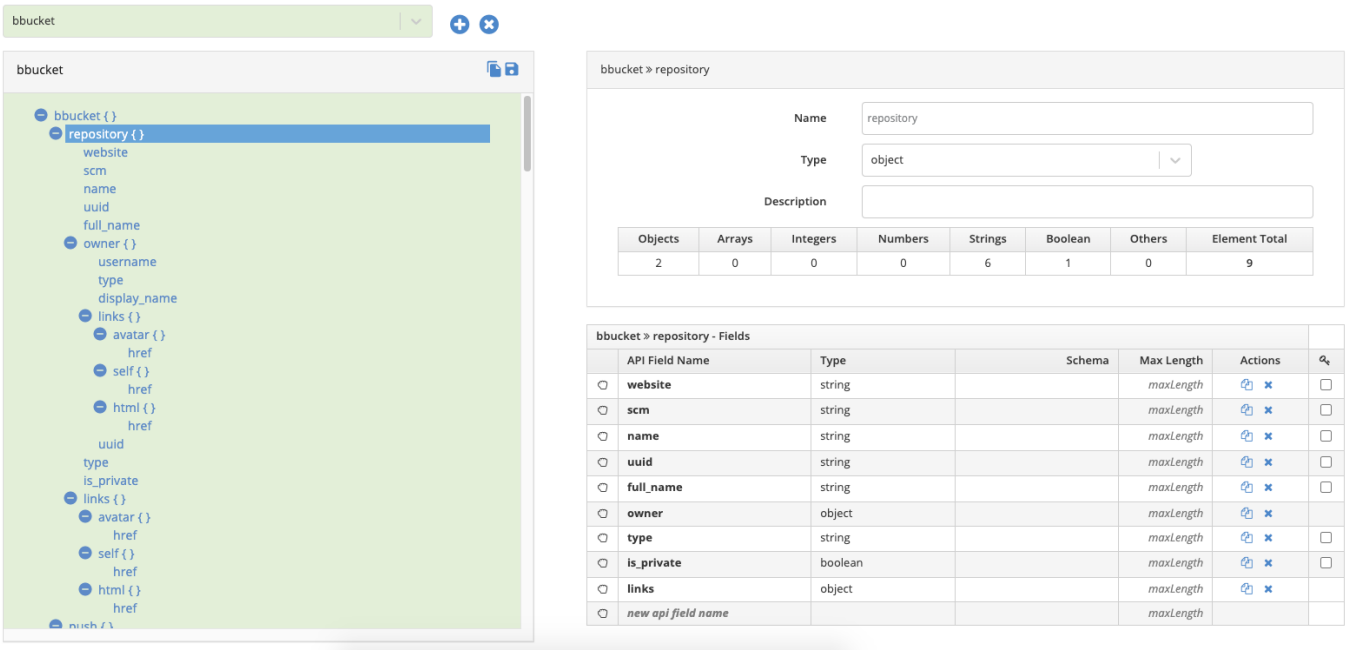
- **Objects**
- **Arrays**
- **Refs**

The JSON Structure can be edited using the Details Form.

Objects {}

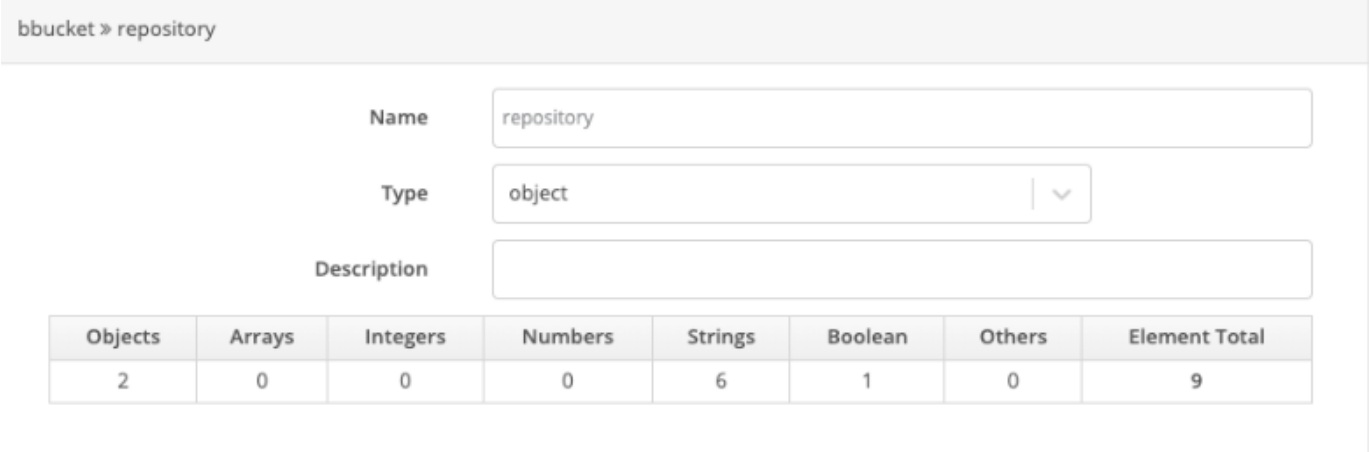
Clicking on the object element in the schema brings up the details of the selected object. The data type can be changed by selecting the value from the dropdown list but it comes with a word of caution. Changing the data type from object to any other data type would delete all the fields (the immediate children) that the object contains.

Consider the following schema example where we click the field **"repository"** which is object type. Following details come up:



Object type Field Details

The table shown inside the details form gives summarised detail of the child fields of the selected object. In our example the two fields owner and links are object type, so the total number of object fields is 2. Similarly it gives the summarised information for all the data types.



Object Type Detail Form

Below the details form, the Sub-Element Editor Table containing the fields that are the immediate children of the selected object. You can add, edit, re-order, delete, copy the fields in the table.

bbucket » repository - Fields

	API Field Name	Type	Schema	Max Length	Actions	
	website	string		*maxLength*		
	scm	string		*maxLength*		
	name	string		*maxLength*		
	uuid	string		*maxLength*		
	full_name	string		*maxLength*		
	owner	object		*maxLength*		
	type	string		*maxLength*		
	is_private	boolean		*maxLength*		
	links	object		*maxLength*		
	new api field name			*maxLength*		

Object Type Sub-Element Editor Table

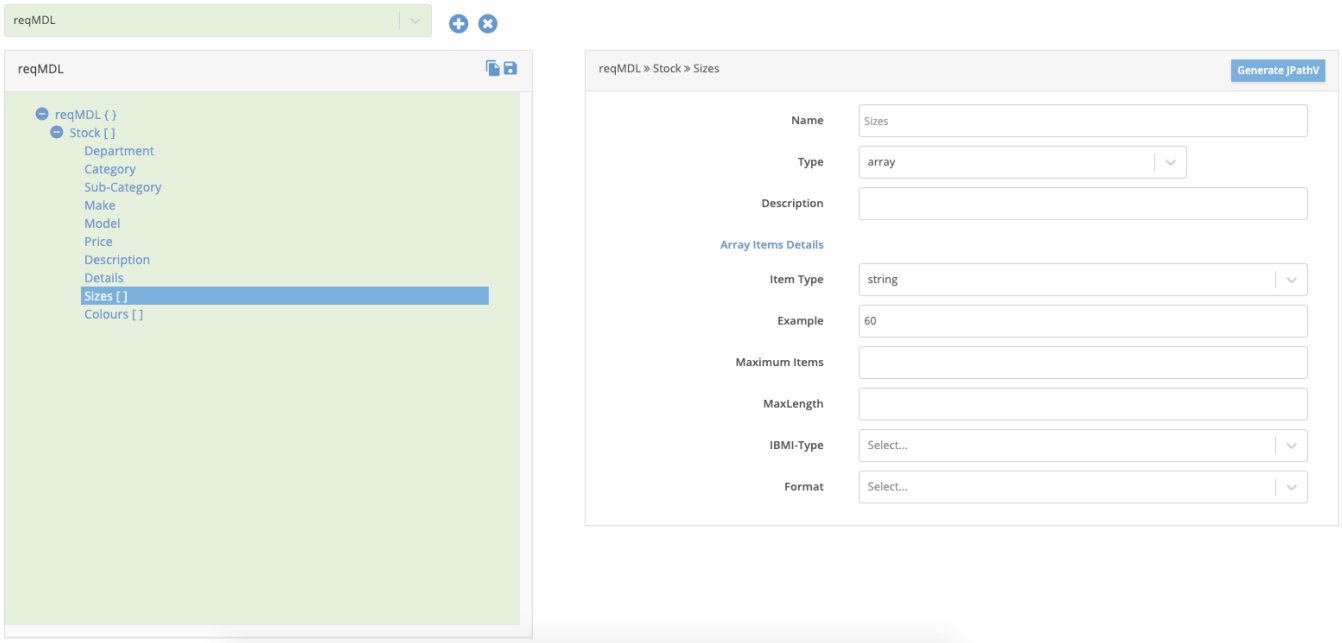
Arrays []

Arrays are defined and have Items. Item types can be any of the following:

Basic Item Type

(string, boolean, integer, number)

Consider the following example.

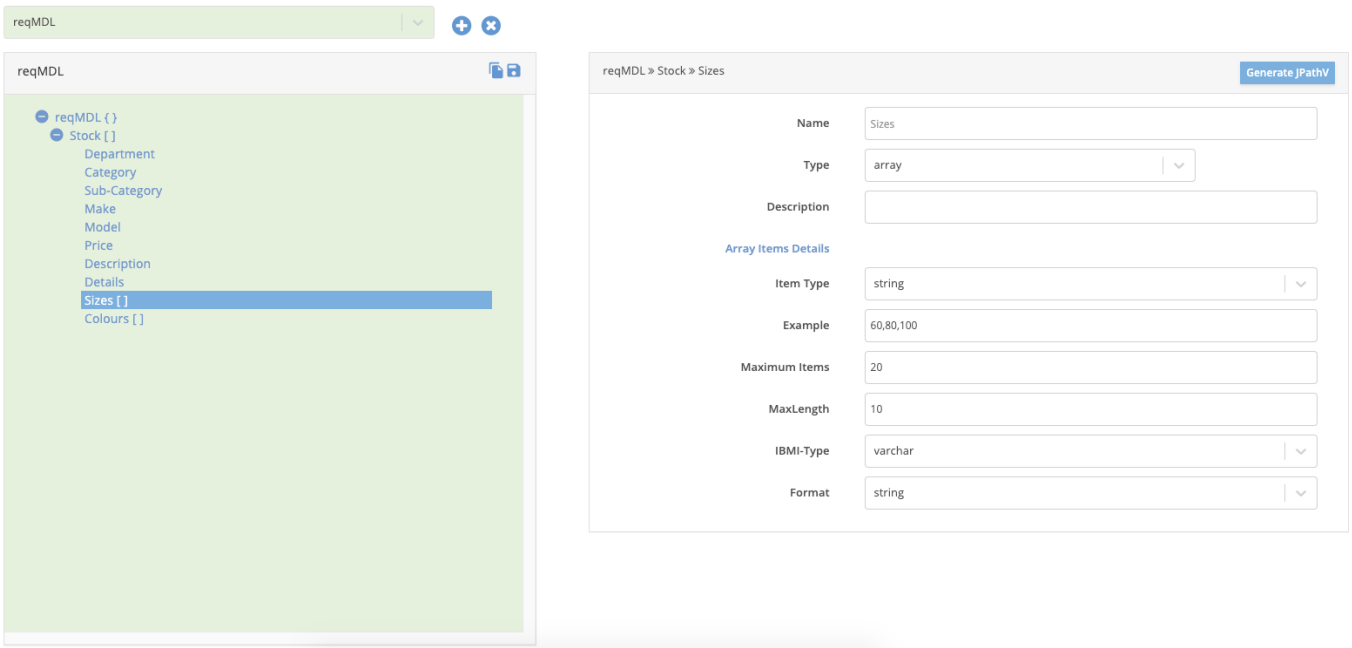


Detail section- Array with basic type element

The selected field - **"Sizes"** is an array of string type elements.

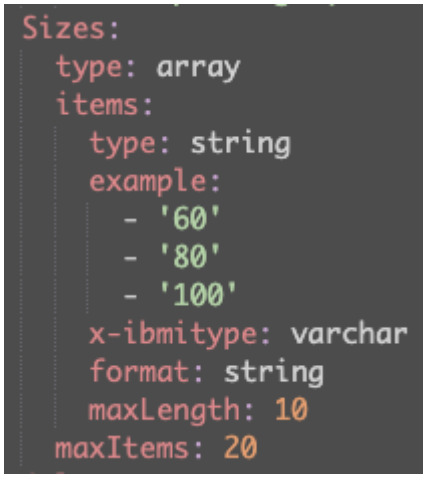
In case of array type elements and basic/elementary data type elements, following additional fields are provided.

Field	Description	Corresponding Swagger key in items object
Item Type	The type of array elements to be selected from the dropdown list	type
Example	Editable, can be comma separated multiple values in case of string elements	example (an array)
Maximum Items	The maximum size of the array	maxItems
MaxLength	The maximum length in case of string type array elements	maxLength
Maximum	This parameter is also considered for identifying the maximum decimal positions when the data type is “number”. For example, if the maximum is set to 9.9999999, that means the variable will be declared with 10s,6 (if “x-ibmitype” is set as “zoned”). Otherwise, it will be defined as 10p, 6.	maximum
IBMI-Type	Possible values are “zoned”, “packed” or "binary" when the item type is “number” or “integer”. Possible values are “char” or “varchar” when the item type is “string”	x-ibmitype
Format	The possible values for this entry could be “int32” or "int64” for “integer” type. For “number” type could be “float” or “double". The format “int32” causes the variable definition with 10i,0 or 10p,0 or 10s,0 depending on whether **IBMI-Type** is specified or not. When “int64” is there, the length 10 becomes 20. The format “float” causes the variable to be defined as 9b,5, whereas double causes 16p,11 or 16s,11 or 9b,5 definition, depending on **IBMI-Type**. For "string" type, the possible values are 'date' or 'string'	format



Edited Detail section- Array with basic type element

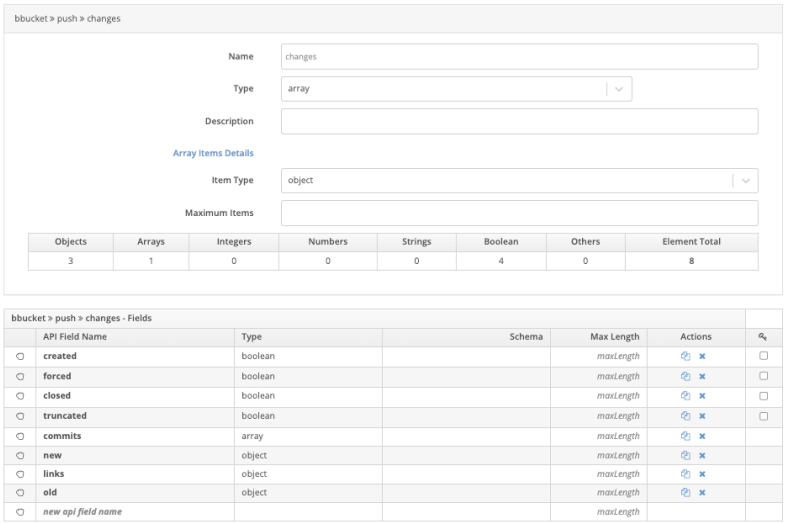
The swagger gets updated for the selected element **"Sizes"**



Updated Swagger Definition for the Edited Field

Object Item Type

Please refer to the example below. The selected field "changes" is an array with object type elements.



Array of object type items

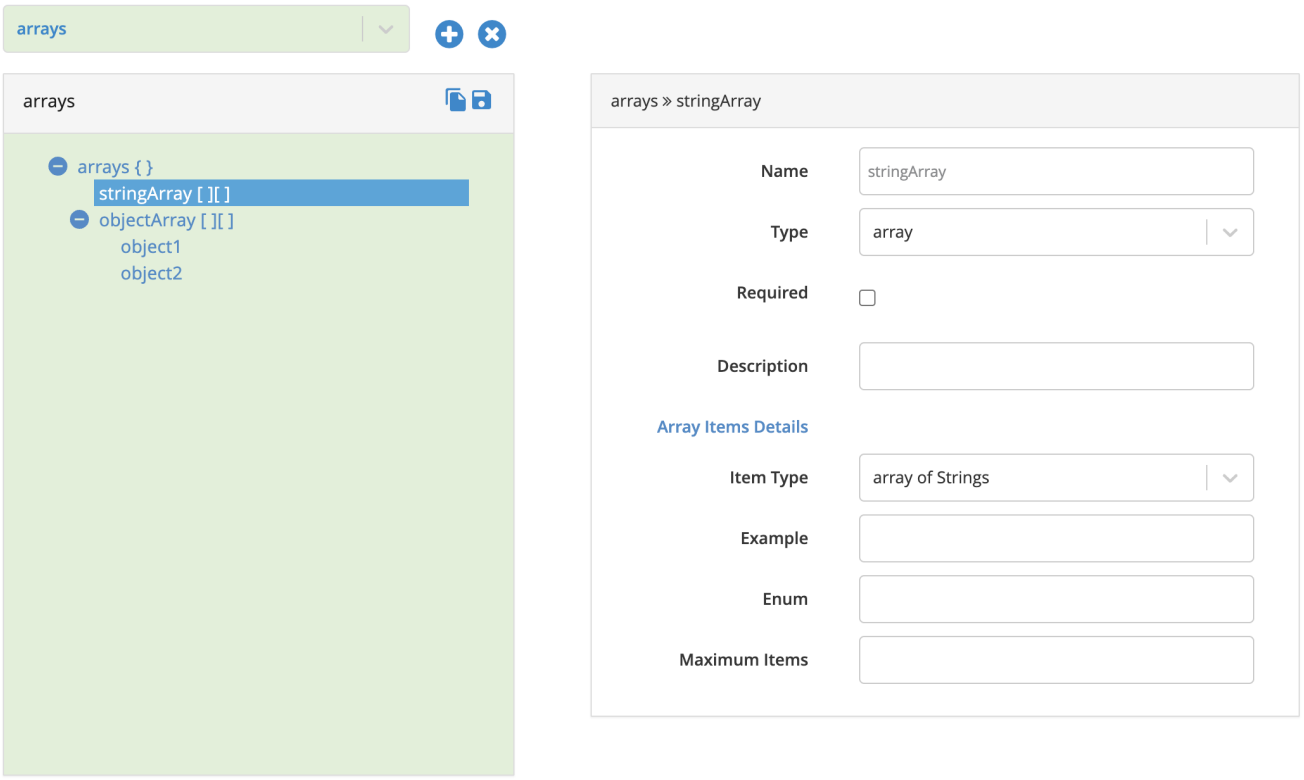
The fields of the object type array elements are also shown in the **Sub-Element Editor Table**.

Nested Arrays

Here is an example where the array element has items of type arrays of strings.

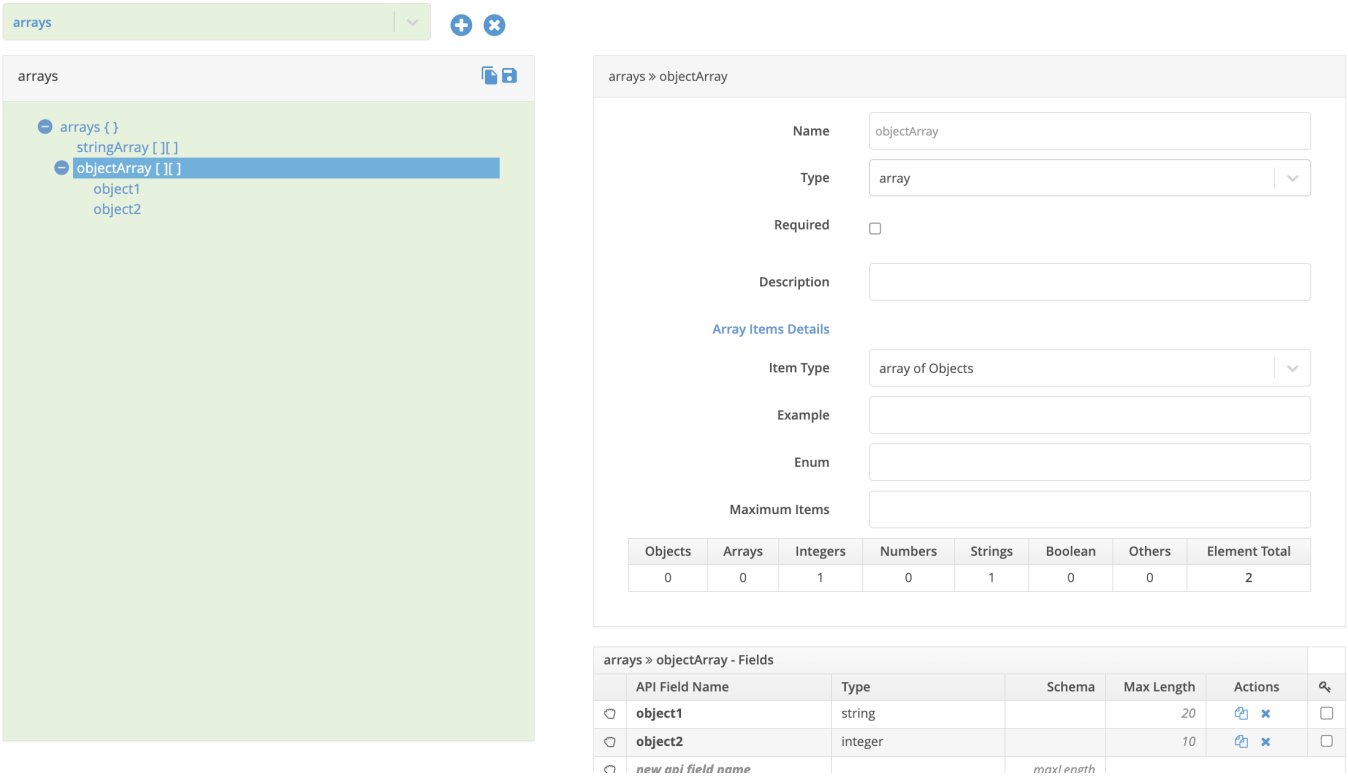
```
{
  "stringArray": [
    ["string1", "string2"],
    ["string3", "string4"]
  ],
  "objectArray": [
    {"object1": "value"},
    {"object2": "value"}
  ]
}
```

This how it will be defined in the UI



Array of arrays of strings

Here is another example where the array element has items of type arrays of objects.

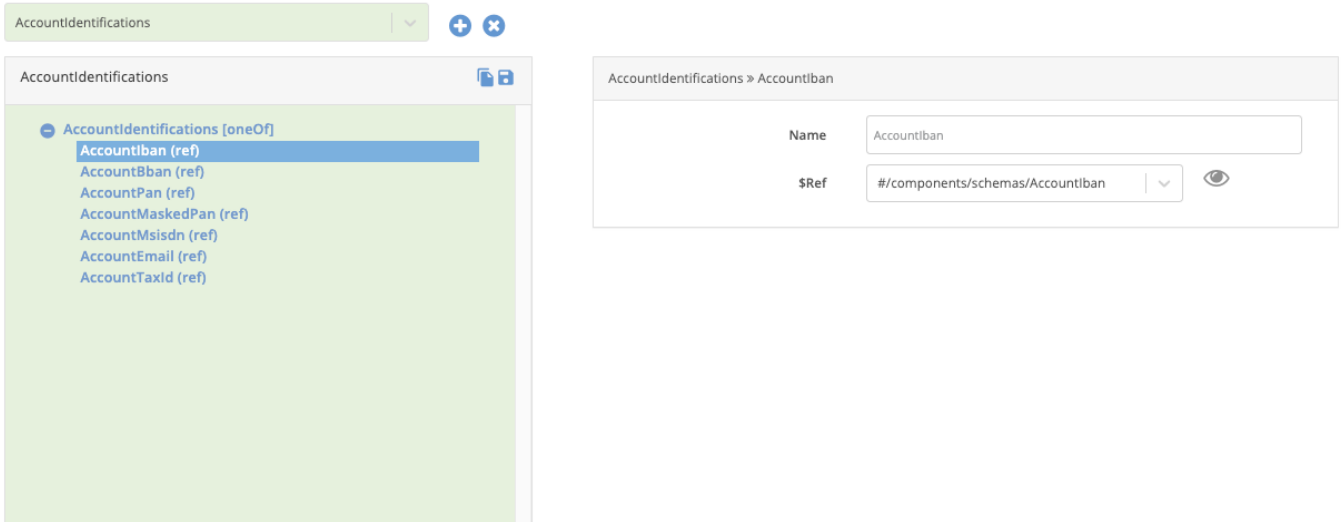


Array of arrays of objects

Again changing the data type or item type (data type of the array elements) is permitted but comes with a word of caution. You could end up getting the child elements deleted!

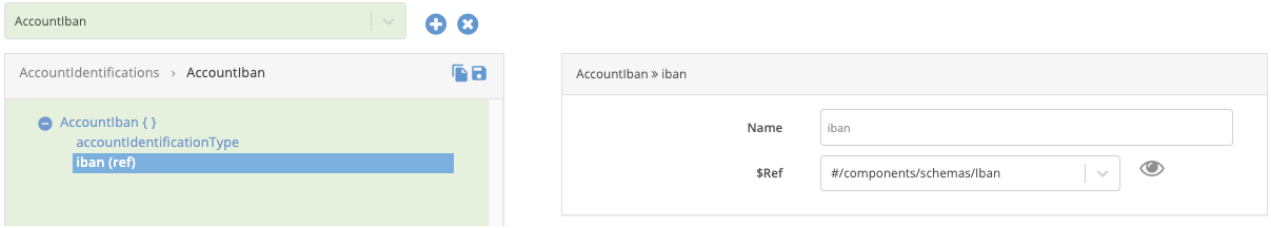
$ref Type

If you click on the ref type fields

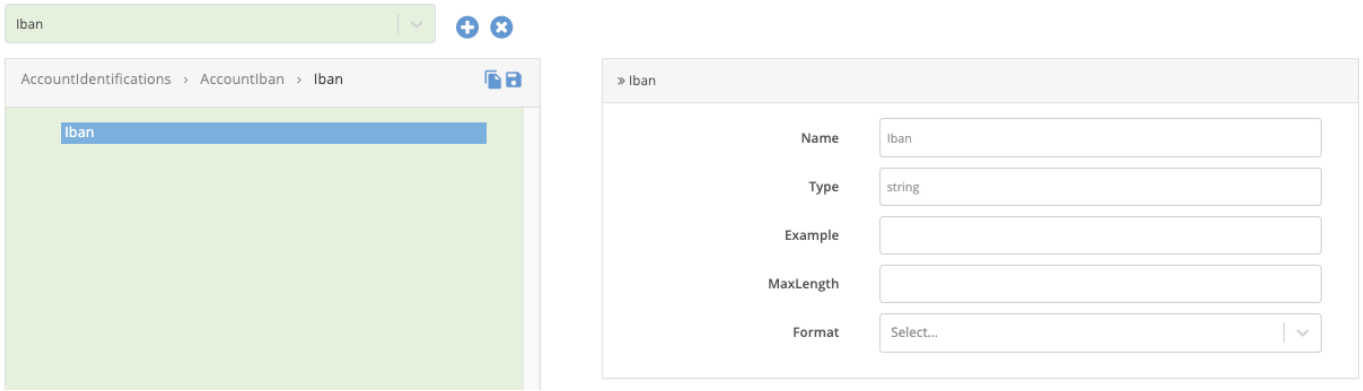


If you click on the link in the $Ref field in the details section,

the schema tree would show the selected schema



Clicking on the field iban would bring up:



Using Sub-Element Editor Table

This section lets you view / edit / add / delete fields in the object / array element selected on the Schema Tree

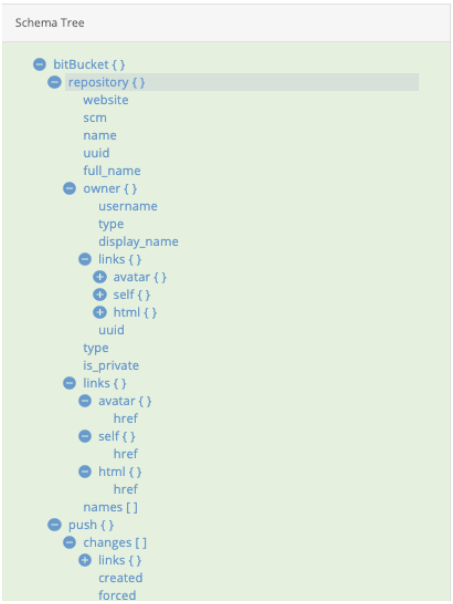
bitbucket » repository - Fields

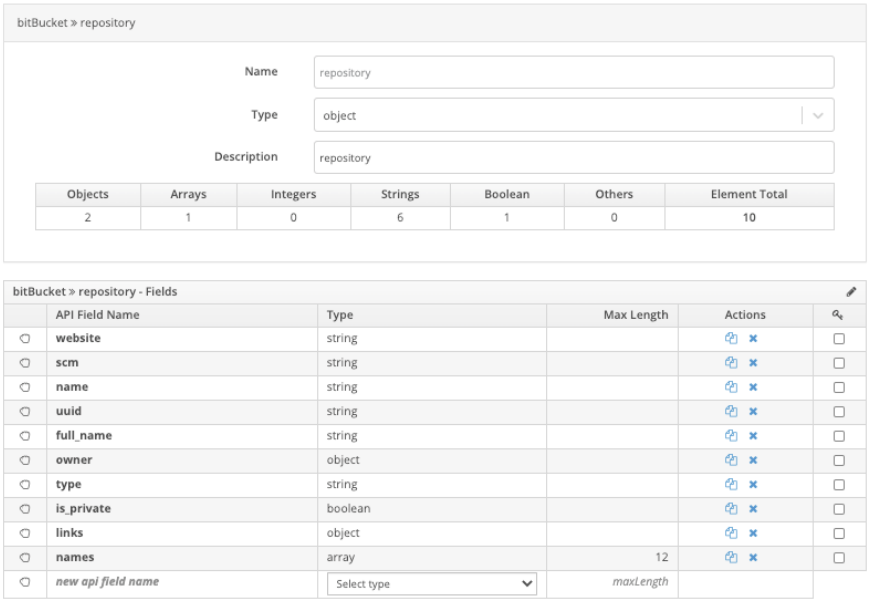
	API Field Name	Type	Schema	Max Length	Actions	
	website	string		*maxLength*		
	scm	string		*maxLength*		
	name	string		*maxLength*		
	uuid	string		*maxLength*		
	full_name	string		*maxLength*		
	owner	object		*maxLength*		
	type	string		*maxLength*		
	is_private	boolean		*maxLength*		
	links	object		*maxLength*		
	new api field name			*maxLength*		

Re-Order the fields

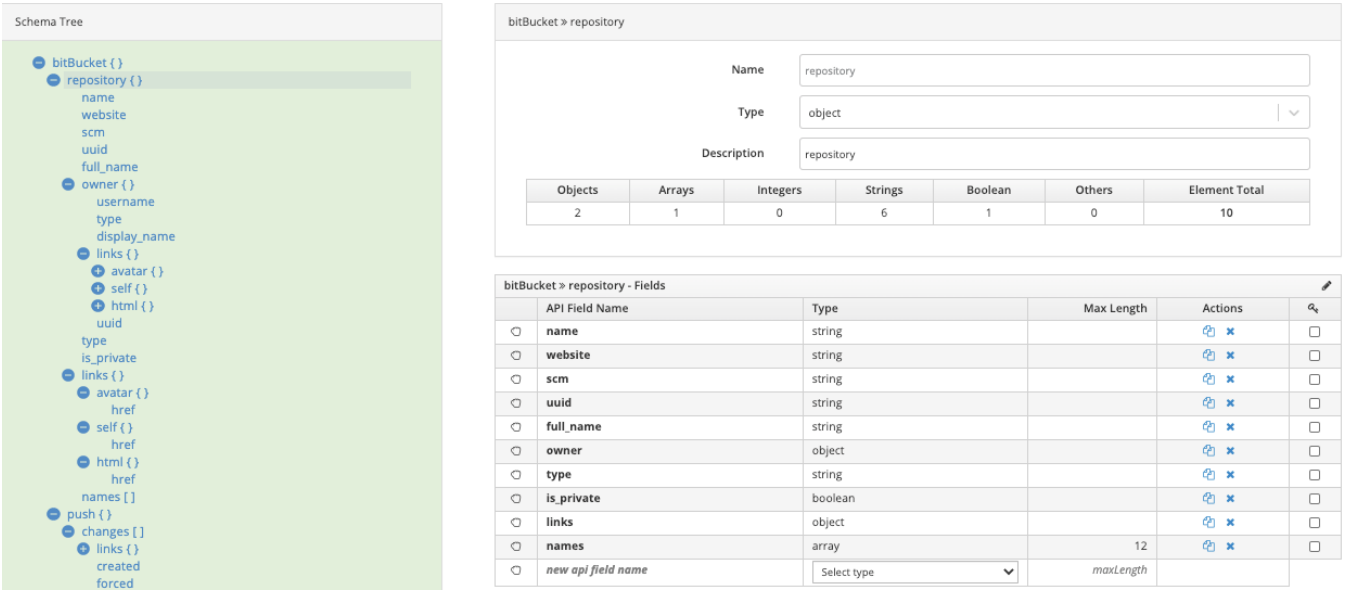
Click on the icon to drag and drop the fields.

eg.





If we bring the field "name" to the top, the schema screen looks like



Delete a field

Click on the ✖ button under the Actions column to delete a field.

Copy a field

Click on the button under the Actions column to copy a field.

Add a field

Add the API Field name, Select the data type from the dropdown list and add the Max Length. The new field is created!

1.5 Coding Guide

1.5.1 MDRest4i Coding Guide

Overview

MDRest4i MDRFRAME runtime framework is made up of these main parts:

MDRFRAME - ILE REST Framework delivered via a set of ILE Service Programs

MDRAPI - Generic REST CGi program

MDRSTCFG - API configuration

MDRFRAME CODING FRAMEWORK

MDRFRAME is a set of ILE Service Programs, with callable procedures that handle all the aspects of REST API and Rest Client program execution.

These service programs can be bound into any ILE program (RPGLE, COBOL, CLLE) and use These copybooks for procedure prototypes and variable definitions.

> **Tip**
>
> Click on the link for each framework below to navigate to the copybook documentation for this service program.

Service Program	Description
MDRFRAME	Main service program for REST API and Consumer functions
MDRYAJLR4	MDRest4i ILE wrapper for YAJL. Renames YAJL procedures (adds mdr_ prefix), plus additional, enhanced YAJL functions for IBM i
MDRTOOLS	Additional utilities used by the SDK, and also available for customer use.
MDREXPAT	The XML Parsing functions packaged by MDRest4i
MDRYAJL	This is an IBM i port of YAJL, a very fast JSON parser/generator created by Lloyd Hilaiel of Mozilla fame.

MDRAPI

MDRAPI is an ILE program acts as the REST API controller. More details can be found in the Provider section of this guide.

MDRSTCFG

MDRDCFG is a DB 2 table that holds the configuration details of API programs called by MDRAPI. It can be edited programmatically or by using command: MDRST/MDRSTCFG.

1.5.2 MDRFRAME Copybooks

MDRFRAME comes supplied with a few other copybooks. Click on any of the links below to see the details of any copybook.|

INCLUDED COPYBOOKS

The source of these can be found in MDRST/QRPGLESRC

Copybook	Description
MDRFRAME	MDRFRAME framework prototypes and templates. Find this here: `MDRST/QRPGLESRC.MDRFRAME`
MDRYAJLR4	RPG Prototypes for YAJL with mdr_ prefix. Find this here: `MDRST/QRPGLESRC.MDRYAJLR4`
MDRTOOLS	Additional MDRest4i utilities used by the SDK, and also available for customer use. Find this here: `MDRST/QRPGLESRC.MDRTOOLS`
MDRPSDS	MDRest4i program status data structure (PSDS) definitions. Find this here: `MDRST/QRPGLESRC.MDRPSDS`
MDRTOKEN	MDRest4i Token handling functions. Find this here: `MDRST/QRPGLESRC.MDRTOKEN`

COBOL COPYBOOKS

There are alternative copybooks used for COBOL usage of the MDRFRAME functions. The source of these can be found in MDRST/QLBLSRC.

Copybook	Type	Description
MDRCBLLSAC	CBLLE	MDRest4i MDRFRAME for Advance Options
MDRCBLLSC	CBLLE	MDRest4i COBOL Consumer Linkage Section
MDRCBLPRC	CBLLE	MDRest4i MDRFRAME Prodedures Sections
MDRCBLSPC	CBLLE	MDRest4i MDRFRAME Special Names Division
MDRCBLWSC	CBLLE	MDRest4i MDRFRAME COBOL Working Storage
MDRCBLWSVS	LBL	MD Rest Variables small
MDRHTTPWSC	CBLLE	MDRest4i MDRFRAME for HTTP body with 16 MB

1.5.3 MDRFrame Copybook

These are the MDRFRAME framework functions available for usage in Provider and Consumer Programs.

Some procedures are specific to REST Consumers (e.g MDR_request) or REST Providers (e.g. MDR_getPathVar), and some procedures can be used in both (e.g. MDR_jsonPathV).

> **Note**
>
> YAJL functions have been wrappered in MDRest4i - with some enhancements. The copybook for these YAJL functions in MDRest4i is included by default in the MDRFRAME copybook. The copybook content, can be found in **MDRST/MDRYAJLR4**

Contents

Program Compilation

Midrange Dynamics recommends that you compile your program with ACTGRP(CALLER) when creating a PROVIDER. CONSUMERS should use either NEW or a NAMED activation group.

If you need the MDREST4i Framework to run in a specific activation group you can accomplish this by running the following commands:

UPDPGM PGM(lib/MDRAPI) MODULE(NONE) ACTGRP(your-actgrp) UPDSRVPGM PGM(lib/MDRFRAME) MODULE(NONE) ACTGRP(your-actgrp)

This causes the MDRest4i API controller (MDRAPI) and the MDRest4i framework (MDRFRAME) to run in an activation group named your-actgrp. After this, when your program uses ACTGRP(*CALLER), it will also run in that actgrp.

It is important that MDRAPI and MDRFRAME always have the same named activation group.

MDRST/MDRFRAME copybook header

mdrframe_copybook.md

```
     /include mdryajlr4
     /if defined(MDRFRAME_H)
     /eof
     /endif
     /define MDRFRAME_H
```

Provider-Or-Consumer-Functions

MDR_HANDLE_T

Since MDRest4i supports high-speed, multi-threaded operation, a handle is needed to tell the framework which session is running in which execution thread. When your RPG program is called by the MDRest4i controller, the first parameter passed to your program is always the handle.

Each time you call an MDRest4i subprocedure, you must provide the same handle.

```
dcl-s MDR_Handle_t char(128) template;
```

MDR_SETHEADER

Set an HTTP header to be sent as output.

NOTE: This may be used with either an API provider or an API consumer, depdending on the way the handle is created

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) header = name of the header to set
- @param (input) value = value of the header to set

Example

```
MDR_setHeader( handle
: 'content-type':
: 'application/octet-stream');
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_setHeader extproc(*cwiden: 'MDR_C_setHeader');
  handle          like(MDR_handle_t);
  header          pointer value options(*string);
  value           pointer value options(*string);
end-pr;
```

MDR_SETCOOKIE

Set an HTTP cookie to be set during output

NOTE: This may be used with either an API provider or an API consumer, depdending on the way the handle is created.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) cookie = name of the cookie to set
- @param (input) value = value of the cookie to set

Example

```
SESSION_ID = '12345';
MDR_setCookie(handle: 'SessionId': SESSION_ID);
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_setCookie extproc(*cwiden: 'MDR_C_setCookie');
  handle          like(MDR_handle_t);
  cookie          pointer value options(*string);
  value           pointer value options(*string);
end-pr;
```

MDR_GETHEADER

Retrieves the value of an HTTP header on input.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) name = name of HTTP header to retrieve
- @return the value of the header. or '*NOTFOUND' if you specify a header that does not exist

Example:

`agent = MDR_getHeader(handle: 'user-agent');` by thge way this doesnt work!!!

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getHeader varchar(2048)
                     rtnparm
                     extproc(*cwiden:'MDR_RPG_getHeader');
  handle          like(MDR_handle_t);
  name            varchar(200) const;
end-pr;
```

MDR_GETOUTGOINGHEADER

Retrieves the value of an HTTP header in the response to an HTTP request.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) name = name of HTTP header to retrieve
- @return the value of the header. or '*NOTFOUND' if you specify a header that does not exist

Example: `agent = MDR_getOutgoingHeader(handle: 'user-agent');` - @info MDRFRAME

ProtoType

```
dcl-pr MDR_getOutgoingHeader varchar(2048)
                             rtnparm
                             extproc(*cwiden
                                    :'MDR_RPG_getOutgoingHeader');
  handle          like(MDR_handle_t);
  name            varchar(200) const;
end-pr;
```

MDR_GETCOOKIE

Retrieves the value of an HTTP cookie on input.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) name = name of HTTP cookie to retrieve
- @return the value of the cookie or '*NOTFOUND' if you specify a cookie that does not exist

Example: `session = MDR_getCookie(handle: 'SessionId');` - @info MDRFRAME

ProtoType

```
dcl-pr MDR_getCookie varchar(2048)
                     rtnparm
                     extproc(*cwiden:'MDR_RPG_getCookie');
  handle          like(MDR_handle_t);
  name            varchar(200) const;
end-pr;
```

MDR_GETOUTGOINGCOOKIE

Retrieves the value of an HTTP cookie in the response to an HTTP request

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) name = name of HTTP cookie to retrieve
- @return the value of the cookie or '*NOTFOUND' if you specify a cookie that does not exist

Example: `session = MDR_getOutgoingCookie(handle: 'SessionId');` - @info MDRFRAME

ProtoType

```
dcl-pr MDR_getOutgoingCookie varchar(2048)
                             rtnparm
                             extproc(*cwiden
                                    :'MDR_RPG_getOutgoingCookie');
  handle          like(MDR_handle_t);
  name            varchar(200) const;
end-pr;
```

MDR_GETHEADERCOUNT

Returns the number of **Custom headers only** in the current request

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @return the number of headers in current http request
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getHeaderCount packed(7: 0) extproc('MDR_getHeaderCount');
  handle like(MDR_handle_t);
end-pr;
```

MDR_GETCOOKIECOUNT

Returns the number of cookies in the current request

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @return the number of cookies in current http request
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getCookieCount packed(7: 0) extproc('MDR_getCookieCount');
  handle like(MDR_handle_t);
end-pr;
```

MDR_GETOUTGOINGHEADERCOUNT

Returns the number of headers in the current response

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @return the number of headers being returned in current http response
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getOutgoingHeaderCount packed(7: 0)
  extproc('MDR_getOutgoingHeaderCount');
  handle like(MDR_handle_t);
end-pr;
```

MDR_GETOUTGOINGCOOKIECOUNT

Returns the number of cookies in the current response

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @return the number of cookies being returned in current http response
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getOutgoingCookieCount packed(7: 0)
  extproc('MDR_getOutgoingCookieCount');
  handle like(MDR_handle_t);
end-pr;
```

MDR_GETHEADERNAME

Returns the name of a header based on its ordinal position in the headers array

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) pos = array position to retrieve
- @return the name of the header, or *NOTFOUND
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getHeaderName char(200) extproc('MDR_getHeaderName');
  handle like(MDR_handle_t);
  pos    packed(7: 0) const;
end-pr;
```

MDR_GETQUERYVARNAME

Returns the name of a query string variable based on its ordinal position in the query string array

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) pos = array position to retrieve
- @return the name of the query string variable, or *NOTFOUND
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getQueryVarName char(200) extproc('MDR_getQueryVarName');
  handle like(MDR_handle_t);
  pos    packed(7: 0) const;
end-pr;
```

MDR_GETCOOKIENAME

Returns the name of a cookie based on its ordinal position in the cookies array

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) pos = array position to retrieve
- @return the name of the cookie, or *NOTFOUND
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getCookieName char(200) extproc('MDR_getCookieName');
  handle like(MDR_handle_t);
  pos    packed(7: 0) const;
end-pr;
```

MDR_GETOUTGOINGHEADERNAME

Returns the name of a header in the current response based on its ordinal position in the array

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) pos = array position to retrieve
- @return the name of the header, or *NOTFOUND
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getOutgoingHeaderName char(200)
  extproc('MDR_getOutgoingHeaderName');
  handle like(MDR_handle_t);
  pos    packed(7: 0) const;
end-pr;
```

MDR_GETOUTGOINGCOOKIENAME

Returns the name of a cookie in the current response based on its ordinal position in the array

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) pos = array position to retrieve
- @return the name of the cookie, or *NOTFOUND
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getOutgoingCookieName char(200)
  extproc('MDR_getOutgoingCookieName');
  handle like(MDR_handle_t);
  pos    packed(7: 0) const;
end-pr;
```

MDR_LOCATIONINFO

Provides root IFS folder and IFS folder to use. In SRVPGM MDRTOOLS - @param (output) library: MDREST4I Library (MDRST....) - @param (output) rootIFSPath: MDREST4I root IFS path - @param (output) instFolder: instance folder name (default, T14, ...)

ProtoType

```
dcl-pr MDR_locationInfo ExtPgm('MRLOCINF');
  library       char(10);
  rootIFSPath   char(20);
  instFolder    char(7);
end-pr;
```

MDR_CHKOBJ

Check for a disk object (similar to CHKOBJ CL cmd)

- @param (input) library = library to look in (or LIBL, CURLIB)
- @param (input) object = name of object
- @param (input) type = object type (PGM, FILE etc)
- @param (input) rtnLib = Returns the library where the object was found (pass *OMIT if you don't need this)
- @return 0 if object not found, 1 if it was found
- @info MDRFRAME5

ProtoType

```
dcl-pr MDR_chkObj int(10) extproc(*cwiden:'MDR_chkObj');
  library pointer value options(*string);
  object  pointer value options(*string);
  type    pointer value options(*string);
  rtnLib  char(10) options(*omit);
end-pr;
```

MDR_B64_ENCODE

Encode string in base64 format

- @param (output) dst = destination (encoded) string
- @param (input) src = source (not encoded) string
- @return 0 if successful, -1 otherwise
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_b64_encode int(10) extproc(*cwiden:'MDR_b64_encode') opdesc;
  dst varchar(5000000:4) options(*varsize);
  src varchar(5000000:4) const options(*varsize) ccsid(*hex);
end-pr;
```

MDR_B64_DECODE(): DECODE STRING IN BASE64 FORMAT

- @param (output) dst = destination (decoded) string
- @param (input) src = source (encoded) string
- @return 0 if successful, -1 otherwise
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_b64_decode int(10) extproc(*cwiden:'MDR_b64_decode') opdesc;
  dst varchar(5000000:4) options(*varsize) ccsid(*hex);
  src varchar(5000000:4) const options(*varsize);
end-pr;
```

MDR_B64URL_ENCODE

Encode string in Base64Url format. Base64Url differs slightly from base64 in order to remove any characters that have a special meaning in a URL, including the +, / and = characters that are normally found in base64.

JWT uses Base64Url instead of standard base64

- @param (output) dst = destination (encoded) string
- @param (input) src = source (not encoded) string
- @return 0 if successful, -1 otherwise
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_b64url_encode int(10)
                         extproc(*cwiden:'MDR_b64url_encode') opdesc;
  dst varchar(5000000:4) options(*varsize);
  src varchar(5000000:4) const options(*varsize) ccsid(*hex);
end-pr;
```

MDR_B64_DECODE()

Decode string in Base64Url format. Base64Url differs slightly from base64 in order to remove any characters that have a special meaning in a URL, including the +, / and = characters that are normally found in base64.

JWT uses Base64Url instead of standard base64

- @param (output) dst = destination (decoded) string
- @param (input) src = source (encoded) string
- @return 0 if successful, -1 otherwise
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_b64url_decode int(10)
                         extproc(*cwiden:'MDR_b64url_decode') opdesc;
  dst varchar(5000000:4) options(*varsize) ccsid(*hex);
  src varchar(5000000:4) const options(*varsize);
end-pr;
```

MDR_ENCODE

URL-encode string

- @param (input) handle = handle returned by MDR_newClient.
- @param (output) dst = destination (encoded) string
- @param (input) src = source (not encoded) string
- @info MDRFRAME8

ProtoType

```
dcl-pr MDR_encode extproc(*cwiden:'MDR_encode') opdesc;
  handle like(MDR_Handle_t);
  dst    varchar(5000000:4) options(*varsize);
  src    varchar(5000000:4) const options(*varsize);
end-pr;
```

MDR_DECODE

URL-decode string

- @param (input) handle = handle returned by MDR_newClient.
- @param (output) dst = destination (decoded) string
- @param (input) src = source (encoded) string
- @info MDRFRAME8

ProtoType

```
dcl-pr MDR_decode extproc(*cwiden:'MDR_decode') opdesc;
  handle like(MDR_Handle_t);
  dst    varchar(5000000:4) options(*varsize);
  src    varchar(5000000:4) const options(*varsize);
end-pr;
```

MDR_WRITEBODYTOFILE

Write an HTTP request/response body to a stream (IFS) file.

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @param (input) body = MDR_BODY_REQUEST or MDR_BODY_RESPONSE
- @param (input) stmf = path to stream/ifs file to save to
- @param (output/optional) errmsg = variable to receive an error message if an error occurs. or *OMIT if not needed.
- @return 0 if successful, -1 upon failure
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_writeBodyToFile int(10)
       extproc(*cwiden:'MDR_writeBodyToFile') opdesc;
  handle like(MDR_Handle_t);
  body   int(10) value;
  stmf   varchar(5000:4)  const options(*varsize);
  errmsg varchar(32767:4) options(*varsize: *omit: *nopass);
end-pr;
```

Constants used for body parm

```
dcl-c MDR_BODY_REQUEST  9901;
dcl-c MDR_BODY_RESPONSE 9902;
```

Example

```
  // Variable used to hold IFS path for reading/saving response body
  dcl-s ifspath varchar(500:4);

  // Logic to write response body to IFS file
  ifspath = '/home/stuart/echohdr_resp.txt';
  MDR_writeBodyToFile(handle:MDR_BODY_REQUEST:ifsPath:errorMsg);
```

MDR_READBODYFROMFILE

Read an HTTP request/response body from a stream (IFS) file.

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @param (input) body = MDR_BODY_REQUEST or MDR_BODY_RESPONSE
- @param (input) stmf = path to stream/ifs file to read from
- @param (output/optional) errmsg = variable to receive an error message if an error occurs. or *OMIT if not needed.
- @return 0 if successful, -1 upon failure
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_readBodyFromFile int(10)
       extproc(*cwiden:'MDR_readBodyFromFile') opdesc;
  handle like(MDR_Handle_t);
  body   int(10) value;
  stmf   varchar(5000:4)  const options(*varsize);
  errmsg varchar(32767:4) options(*varsize: *omit: *nopass);
end-pr;
```

MDR_TRANSLATE

translates the input data from one CCSID to another

- @param (input) input ccsid value from which the data is to be translated
- @param (input) output ccsid value to which the data is being translated
- @param (input) address of input data for translation
- @param (input) length of the input data
- @param (input) address of output data for storing translated data
- @param (input) length available at the target pointer
- @param (input) error message
- @return the length of the translated data or -1 upon error

Note: When the value 0 is supplied in either "fromCCSID" or "toCCSID" it is taken as host CCSID value. When errorMsg is blank, that means whole input data has been translated, otherwise, error message explains the error

ProtoType

```
dcl-pr MDR_translate int(10);
  fromCCSID int(10);
  toCCSID int(10);
  inputData pointer;
  inputLen  int(10);
  targetData Pointer;
  targetLength int(10);
  errorMsg varchar(50);
end-pr;
```

Provider-Functions

MDR_ISBINARYMODE

Detects whether the API was called in binary mode.

NOTE: MDRFRAME is intended to be used in binary mode, if called from an EBCDIC or MIXED mode, functionality may be limited.

- @return ON if in binary mode, OFF otherwise

Example

```
if MDR_isBinaryMode() = *OFF;
not in binary mode!  why?
endif;
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_isBinaryMode ind extproc(*cwiden: 'MDR_isBinaryMode');
end-pr;
```

MDR_ISHTTPS

Detects whether this API was called using secure HTTPS. (aka SSL or TLS)

- @return ON if in HTTPS mode, OFF for plain HTTP

Example

```
if MDR_isHTTPS() = *OFF;
MDR_setError( handle: 'ERR9999'
: 'This API requires SSL/TLS encryption!!'
: 1 : 20: : 0 );
return;
endif;
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_isHTTPS ind extproc(*cwiden: 'MDR_isHTTPS');
end-pr;
```

MDR_PRINT

Print a string directly to the API output without format checking. This will be appended to the end of any previously sent data.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) string = the string you wish to print in the job's EBCDIC it will be converted to UTF-8 before appending.

Example

```
MDR_setHeader(handle: 'content-type': 'text/plain');
MDR_setStatus(handle: 200);
MDR_print(handle: 'This is plain text API output!');
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_print extproc(*cwiden: 'MDR_print') opdesc;
  handle like(MDR_handle_t);
  string varchar(65536:4) options(*varsize) const;
end-pr;
```

MDR_LEN_PRINT

Same as MDR_print but with a pointer and length parameter.

- @param (input) handle context handle, identifies which MDRest4i session is currently running.
- @param (input) buf = pointer to buffer containing EBCDIC data
- @param (input) len = length of data in buffer.
- @return length of string appended to API output from the string buffer

Example

```
dcl-s BUF char(1000);
dcl-s p_BUF pointer;
dcl-s LEN uns(10);

p_buf = %addr(BUF);
BUF = 'Data data data';
LEN = 14;
MDR_len_print(handle: p_BUF: LEN);
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_len_print int(10) extproc(*cwiden: 'MDR_len_print');
  handle like(MDR_handle_t);
  buf    pointer value;
  len    uns(10) value;
end-pr;
```

MDR_WRITE

This is the same as "MDR_print", above except that the MDRest4i framework will write the data as-is (no conversion to UTF-8)

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) string = the string you wish to print in UTF-8 encoding or binary data to be sent as-is

Example

```
first load a VARCHAR(4) field named imagedata with
the bytes from a .JPG file, then:

MDR_setHeader(handle: 'content-type': 'image/jpeg');
MDR_setStatus(handle: 200);
MDR_write(handle: imagedata);
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_write extproc(*cwiden: 'MDR_write') opdesc;
  handle like(MDR_handle_t);
  string varchar(65536:4) options(*varsize) const ccsid(*utf8);
  string varchar(65536:4) options(*varsize) const;
end-pr;
```

MDR_LEN_WRITE

Same as MDR_write but with a separate length parameter.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) buf = pointer to buffer containing UTF-8 encoded data
- @param (input) len = length of data in buffer.
- @return number of bytes added to the response header

Example

```
p_image = %alloc(fileSize);
lenRead = read(fd: p_image: fileSize);
```

```
MDR_len_write(handle: image: lenRead);    or
numbytes = MDR_len_write(handle: image: lenRead);
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_len_write int(10) extproc(*cwiden: 'MDR_len_write');
  handle like(MDR_handle_t);
  buf    pointer value;
  len    uns(10) value;
end-pr;
```

MDR_SETERROR

Set an error to be returned from the API

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) error_id = the 7-character message ID to be reported to the caller.
- @param (input) error_text = The full text of the message. This string will be sent to the caller as the error message. (It will not be modified according to the error_id.)
- @param (input) error_statement = statement number to report that the error occurred.
- @param (input) error_severity = severity of the error to be reported, typically should be 10 = Minor 20 = Important error 30 = Severe Error 40 = Critical Error
- @param (input) http_status = http status code to return. (0=keep same status, usually 500 for server error) 200 = success 4xx = client-side error etc.

NOTE: You can clear any pending error status by calling with: MDR_setError(handle: NULL: NULL: 0: 0: 0);

Example

```
MDR_setError( handle:
: 'TEST123'
: 'Any text can go here'
: 123
: 40
: 500 );

This will send back the following

error_id = 'TEST123'
error_text = 'Any text can go here'
statement = 123
severity = 40
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_setError extproc(*cwiden: 'MDR_C_setError');
  handle          like(MDR_handle_t);
  error_id        pointer value options(*string);
  error_text      pointer value options(*string);
  error_statement uns(10) value;
  error_severity  int(10) value;
  http_status     int(10) value;
end-pr;
```

MDR_COBOL_SETERROR

This is a wrapper around MDR_setError that used fixed-length strings for error_id and error_text. There's no reason to use it from RPG.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) error_id = the 7-character message ID to be reported to the caller.
- @param (input) error_text = The full text of the message. This string will be sent to the caller as the error message. (It will not be modified according to the error_id.)
- @param (input) error_statement = statement number to report that the error occurred.
- @param (input) error_severity = severity of the error to be reported, typically should be 10 = Minor
- 20 = Important error
- 30 = Severe Error
- 40 = Critical Error

If called from Cobol, you would refer to it by the extproc name (MDR_setError).

CALL PROCEDURE 'MDR_setError' USING LS-HANDLE WS-ERROR-ID (PIC X(7)) WS-ERROR-TEXT (PIC X(500)) WS-ERROR-STATEMENT (USAGE AS BINARY) WS-SEVERITY (USAGE AS BINARY) WS-HTTP-STATUS (USAGE AS BINARY) END-CALL.

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_COBOL_setError extproc(*cwiden: 'MDR_setError');
  handle          like(MDR_handle_t);
  error_id        char(7)   const options(*omit);
  error_text      char(500) const options(*omit);
  error_statement uns(10)   value;
  error_severity  int(10)   value;
end-pr;
```

MDR_SETERRORMSG

Use a message file to set an error returned by the API.

This routine differs from MDR_setError in that it uses the format of a message in a *MSGF object to control the contents of your message. Each &1, &2, &3, etc in the text of your message will be replaced with data from the 'msgdta' parameter.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) msgfile = An IBM *MSGF object found on disk. This parameter is treated as CHAR(21) and expects the format LIBRARY/OBJECT in uppercase.
- @param (input) msgid = The 7-character message ID from the msgfile object to be used. Must exist in 'msgfile'
- @param (input) msgdta = The message data to be used to fill-in variables from the message file.
- @param (input) msgdtalen = Length in bytes of the msgdta parameter, or 0 if no data is needed
- @param (input) error_statement = statement number to report that the error occurred.
- @param (input) error_severity = severity of the error to be reported, typically should be 10 = Minor **20** = Important error **30** = Severe Error **40** = Critical Error

Example

```
dcl-ds DS_CPF2105 qualified;
object  char(10) inz('CUSTMAST');
library char(10) inz('*LIBL');
type    char(7)  inz('*FILE');
end-ds;

MDR_setErrorMsg( handle
: '*LIBL/QCPFMSG'
: 'CPF2105'
: DS_CPF2105
: %len(DS_CPF2105)
: 123
: 40 );
```

```
This will send back the following

error_id = 'CPF2105'
error_text = 'Object CUSTMAST in *LIBL type *FILE not found'
statement = 123
severity = 40
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_setErrorMsg extproc(*cwiden: 'MDR_C_setErrorMsg');
  handle          like(MDR_handle_t);
  msgfile         pointer value options(*string);
  msgid           pointer value options(*string);
  msgdta          pointer value options(*string);
  msgdtalen       int(10) value;
  error_statement uns(10) value;
  error_severity  int(10) value;
end-pr;
```

MDR_SETSTATUS

Set an HTTP status code to be sent as output

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) status = 3-digit status code.
- @param (input/optional) = message sent to the HTTP server along with the status code. If no message is provided, the default message is sent.

Common codes are: **200** = OK; API was successful **302** = Redirect; Browser should open a different URL **400** = Bad Request; Caller sent bad data. **403** = Forbidden; You lack authority to carry out that action. **404** = Not Found; The resource you are trying to retrieve cannot be found. **500** = Server error; An error occurred within the API and the program was not successful.

Example: `MDR_setStatus(handle: 500);  // indicates a generic` error

Example: `MDR_setStatus(handle: 500: 'Program Crashed');`

Example

```
MDR_setStatus(handle: 302);  // indicates a redirect
MDR_setHeader(handle: 'location':'https://www.midrangedynamics.com');
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_setStatus extproc(*cwiden: 'MDR_setStatusOD') opdesc;
  handle          like(MDR_handle_t);
  status          uns(5) const;
  reason          varchar(50:4) const options(*omit:*nopass);
end-pr;
```

MDR_GETQUERYVAR

Retrieve an input variable from the query string. The query string is the part of the URL that follows the ?.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) name = name of query string variable to retrieve
- @return the value of the query string variable.

Example

```
URL is http://yourserver/mdrapi/customer/123?op=call

action = MDR_getQueryVar('op');
```

```
action will be set to 'call'
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getQueryVar varchar(2048)
                       rtnparm
                       extproc(*cwiden:'MDR_RPG_getQueryVar');
  handle          like(MDR_handle_t);
  name            varchar(200) const;
end-pr;
```

MDR_GETPATHVAR

Retrieve a portion of the path. The path is the part of the URL that comes after the host/port name, and before the query string.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) key = value in path that result is relative to
- @param (input) position = position within the path relative to the key. values: zero = the key negative value = positions before the key positive value = positions after the key
- @return the value of the path at that position, or '*NOTFOUND' if you specify a portion of the path that does not exist.

Example

```
URL is http://yourserver/mdrapi/customer/123/test

custid = MDR_getPathVar(handle: 'customer': 1);  // 123
test   = MDR_getPathVar(handle: 'test': -1);     // 123
x      = MDR_getPathVar(handle: 'mdrapi': 3);    // test
name   = MDR_getPathVar(handle: 'customer': 0)   // customer
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getPathVar varchar(2048)
                      rtnparm
                      extproc(*cwiden:'MDR_RPG_getPathVar');
  handle          like(MDR_handle_t);
  key             pointer value options(*string:*trim);
  position        int(10) value;
end-pr;
```

MDR_GETPATHVARCOUNT

Returns the number of path components in the current request

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @return the number of path parameter elements
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getPathVarCount packed(7: 0) extproc('MDR_getPathVarCount');
  handle like(MDR_handle_t);
end-pr;
```

MDR_GETQUERYVARCOUNT

Returns the number of query string variables in the current request

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @return the number of query parameters in current request
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getQueryVarCount packed(7: 0)
  extproc('MDR_getQueryVarCount');
  handle like(MDR_handle_t);
end-pr;
```

MDR_NEWHANDLE

Generate a new MDRest4i session handle

- @return the pointer to the handle in memory. You must call MDR_freeHandle to free the memory for this pointer.
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_newHandle pointer extproc(*cwiden:'MDR_newHandle');
end-pr;
```

MDR_FREEHANDLE

Free the memory used by an MDRest4i session handle.

- @param (input) handlePtr = pointer to a session handle created by MDR_newHandle that should be freed up.
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_freeHandle extproc(*cwiden:'MDR_freeHandle');
  handlePtr pointer value;
end-pr;
```

MDR_SENDJOBINFO

To assist with debugging and other diagnostic usages, MDRest4i supports adding a customer header to the output containing the job information.

There are two ways to enable this. It can be enabled for all programs by placing the following in your HTTP server's httpd.conf file: SetEnv MDR_JOB_HEADER Y

Alternately, it can be enabled in a per-program basis by calling the following prototype from your API: MDR_endJobInfo(handle: *ON);

- @param (input) handle = MDRest4i session handle
- @param (input) enable = pass ON to enable the header, OFF otherwise

Output: The following header will be returned from the API:

x-mdrsrvjob: pgm=library/object; curuser=userid; job=123456/user/jobname - @info MDRFRAME5

ProtoType

```
dcl-pr MDR_sendJobInfo extproc(*cwiden:'MDR_sendJobInfo');
  handle like(MDR_Handle_t);
  enable ind const;
end-pr;
```

MDR_GETAUTH

Retrieves the authorization string provided from Authorisation request header (available in provider, only.)

- @param (input/output) handle = handle provided by MDRAPI
- @param (output/optional) type = (output/optional) type of authentication used, BASIC, BEARER or DIGEST
- @param (output/optional) user = (output/optional) userid provided (blank for bearer)
- @param (output/optional) token = (output/optional) authentication token or password. For basic this is the password, for bearer it is the authentication token, for digest it is the response hash.
- @return 0 if successful, -1 if there was no authorization header, or if the header was invalid.
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getAuth int(10)
       extproc(*cwiden:'MDR_getAuth') opdesc;
  handle like(MDR_Handle_t);
  type   varchar(256:4) options(*varsize:*omit:*nopass);
  user   varchar(256:4) options(*varsize:*omit:*nopass);
  token  varchar(2048:4) options(*varsize:*omit:*nopass);
end-pr;
```

Consumer-Functions

MDR_NEWCLIENT

Create a new handle for a client connection

- @return a pointer to a dynammically allocated client handle. (call MDR_freeClient to release the memory) or *NULL upon failure
- @info MDRFRAME8

ProtoType

```
dcl-pr MDR_newClient pointer extproc(*cwiden:'MDR_newClient');
end-pr;
```

MDR_SETCLIENTOPT

Set an option within a client instance

NOTE: In addition to these options, you can also set headers with MDR_setHeader() or cookies with MDR_setCookie()

- @param (input) handle = identifies the client instance
- @param (input) opt = option to set (see constants below)
- @param (input) val = value to set option to.
- @return 0 if option set successfully, -1 upon error
- @info MDRFRAME8

ProtoType

```
dcl-pr MDR_setClientOpt int(10) opdesc
                        extproc(*cwiden: 'MDR_setClientOpt');
  handle like(MDR_Handle_t);
  option int(10) const;
  value  pointer value options(*string);
end-pr;
```

Constants for MDR_setClientOpt

Constant	Description
MDR_BUF_SIZE	size of send/receive buffer
MDR_KDB_PATH	IFS pathname of key store database (TLS/SSL)
MDR_KDB_PASSWORD	password needed for MDR_KDB_PATH, above.
MDR_KDB_LABEL	label for MDR_KDB_PATH, above.
MDR_DCM_APPID	application ID to use with digital cert mgr
MDR_TIMEOUT	timeout in millseconds to wait before giving up on network operations
MDR_IDLE_LIMIT	by default, the MDRest4i client will leave the connection to the server open between requests to improve performance. The idle limit is the maximum time (in seconds) that it will try to use this open connection. Set this to 0 to always disconnect between requests.
MDR_LAX_AUTH	Allow lax authentication for TLS/SSL certificates Y=Yes, N=No.
MDR_USEC_DELAY	Delay between retries in microseconds. See **this tip** for handling slow APIs from your consumer code.
MDR_MAX_RETRIES	Maximum amount of retries before giving up on connection.
MDR_MAX_REDIRECTS	Maximum amount of http redirects that will be followed before giving up.
MDR_FOLLOW_AUTH_HEADER	Does the Authorization: header get resent in a redirect? ('yes', 'no', default = 'no')
MDR_FOLLOW_ORIG_METHOD	Do 301,302,303 redirects keep the original HTTP method, (vs. changing it to GET) ('yes', 'no', default: 'no')
MDR_AUTH_TYPE	Type of authentication. ('basic', 'digest', 'bearer') if you set this to basic or bearer, the credentials will always be sent to the server. If this is left unset, the key will only be sent if the server requests it. You may set this to your own custom string if you need to use an unsupported authentication type. In that case, (or when using a bearer token) you should also set the const MDR_AUTH_TOKEN.
MDR_USER	UserID for authentication (if basic or digest)
MDR_PASSWORD	Password for authentication (if basic or digest)
MDR_AUTH_TOKEN	Token for authentication (if bearer token or custom)
MDR_LOG_PATH	Path to use for creating log files. - 'ifs:/path/to/file' - 'pf:library/file' - 'dtaq:library/dtaq'
MDR_LOG_PARTS	Allows selecting details to be logged (default: everything is logged) - 'request=Y requestUri=N - requestHeader=Y - requestBody=N - response=N - responseHeader=N - responseBody=Y'
MDR_PROXY_URL	URL for proxy server in this format: `http://hostname:port`
MDR_PROXY_USER	Userid to login to proxy server
MDR_PROXY_PASSWORD	Password to login to proxy server
MDR_CLEAR_LOG	Y/N should the IFS log file be cleared on the next request (default=N)

Handling Slow API Responses

If an endpoint/api being connected to has a very slow response, the following settings can be used to make the Consumer request be more "patient".

For Example:

```
// Create a new handle for a client connection and et the connection variables.
p_handle = MDR_newClient();
MDR_setClientCfg(handle:'timeout=5000 uSecDelay=50000 maxRetries=100');
```

The read operation will be performed in a loop that will run maximum 100 attempts (as per the value set in "maxRetries") and after every timeout, it will wait/sleep for 50000 microseconds which means 0.05 seconds before the next attempt of trying to read the socket.

In a specific attempt of trying to read the socket, if it gets the response, it comes out of the loop, and if it doesn't get response at all, the consumer program will run for 100 iterations of 5000 millisecond timeout at each iteration and for 0.05 seconds delay between each attempt.

```
dcl-c MDR_BUF_SIZE           const(2001);
dcl-c MDR_KDB_PATH           const(2002);
dcl-c MDR_KDB_PASSWORD       const(2003);
dcl-c MDR_KDB_LABEL          const(2004);
dcl-c MDR_DCM_APPID          const(2005);
dcl-c MDR_TIMEOUT            const(2006);
dcl-c MDR_LAX_AUTH           const(2007);
dcl-c MDR_USEC_DELAY         const(1000);
dcl-c MDR_MAX_RETRIES        const(1001);
dcl-c MDR_AUTH_TYPE          const(1002);
dcl-c MDR_USER               const(1003);
dcl-c MDR_PASSWORD           const(1004);
dcl-c MDR_AUTH_TOKEN         const(1005);
dcl-c MDR_LOG_PATH           const(1006);
dcl-c MDR_PROXY_URL          const(1007);
dcl-c MDR_PROXY_USER         const(1008);
dcl-c MDR_PROXY_PASSWORD     const(1009);
dcl-c MDR_MAX_REDIRECTS      const(1010);
dcl-c MDR_IDLE_LIMIT         const(1011);
dcl-c MDR_LOG_PARTS          const(1012);
dcl-c MDR_FOLLOW_AUTH_HEADER const(1013);
dcl-c MDR_FOLLOW_ORIG_METHOD const(1014);
dcl-c MDR_CLEAR_LOG          const(1015);
```

MDR_SETCLIENTCFG

Set client configuration within a client instance.

This is a wrapper for MDR_setClientOpt, it lets you set all of the desired options using a single property string.

NOTE: In addition to these options, you can also set headers with MDR_setHeader() or cookies with MDR_setCookie()

- @param (input) handle = (input) identifies the client instance
- @param (input) options = (input) property string representing the optionsto set.

Options are specified without the MDR_ prefix, and without underscores. For example, MDR_LOG_PATH is set as logPath="/path/to/file.txt"

Example of setting all options:

```
opts = 'bufSize=50000 kdbPath="/ifs/path/myfile.kdb" +
        kdbPassword=myPass kdbLabel="Label 123" +
        dcmAppId=ACME_CLIENT_APP timeout=30000 laxAuth=Y +
        uSecDelay=250000 maxRetries=10 user=sklement +
        password=bigboy proxyUrl=http://myprox.com:123 +
        proxyUser=scottk proxyPassword=d00fus +
        maxRedirects=3 idleLimit=5 authType=basic +
        clearLog=N logPath="/MDRest4i/logs/daily-%Y%m%d.txt"';

MDR_setClientCfg(handle: opts);
```

See MDR_setClientOpt for a description of the above options.

- @info MDRFRAME8

ProtoType

```
dcl-pr MDR_setClientCfg opdesc
                        extproc(*cwiden: 'MDR_setClientCfg');
  handle like(MDR_Handle_t);
  options varchar(32767:4) const options(*varsize);
end-pr;
```

MDR_FREECLIENT

Free the memory used by an HTTP client

- @param (input) handle = handle returned by MDR_newClient.

Once this is called, you must not use the handle again. - @info MDRFRAME8

ProtoType

```
dcl-pr MDR_freeClient extproc(*cwiden: 'MDR_freeClient');
  handle like(MDR_Handle_t);
end-pr;
```

MDR_REQUEST

Make an HTTP request with an HTTP client

- @param (input) handle = handle returned by MDR_newClient.
- @param (input) method = HTTP method (GET, POST, PUT, DELETE, etc)
- @param (input) uri = the URL to make the request to
- @param (input/optional) sendData = data to send to the HTTP server (request body) or *OMIT if no data to be sent via this parm. may be prefixed with 'ifs:' to send from a file
- @param (output/optional) receiveData = data received from the HTTP server (response body) or *OMIT if the receiveData is not to be returned via this parm. May be prefixed with 'ifs:' to save document to an IFS stream file.
- @return the HTTP status code (200=success)or a -1 if a client-side error occurred

NOTE: You may also call MDR_getHeader, MDR_getCookie MDR_getRequest to get datafrom this transaction

- @info MDRFRAME8

ProtoType

```
dcl-pr MDR_request int(10) extproc(*cwiden: 'MDR_request') opdesc;
  handle      like(MDR_Handle_t);
  method      varchar(32:4) const;
  uri         varchar(32767:4) const options(*varsize);
/if not defined(*V7R2M0)
  sendData    varchar(16773100:4) const options(*varsize:*omit:*nopass);
  receiveData varchar(16773100:4) options(*varsize:*omit:*nopass);
/elseif defined(MDR_REQUEST_TRANSLATE)
  sendData    varchar(16773100:4) const options(*varsize:*omit:*nopass)
                                  ccsid(*utf8);
  receiveData varchar(16773100:4) options(*varsize:*omit:*nopass)
                                  ccsid(*utf8);
/else
  sendData    varchar(16773100:4) const options(*varsize:*omit:*nopass)
                                  ccsid(*hex);
  receiveData varchar(16773100:4) options(*varsize:*omit:*nopass)
                                  ccsid(*hex);
/endif
end-pr;
```

MDR_GETRESPPTR

Retrieves a pointer to the response body

NOTE: The main use of this is to handle data larger than 16mb. By working with it using pointer/length.

NOTE: this returns a direct pointer to the buffer, it does not make a copy. You should not dealloc this pointer or change its data or unpredictable results will occur.

- @param (input) handle = handle returned by MDR_newClient.
- @param (output) ptr = pointer, on output will bet set to the address of the response body
- @param (output) size = on output will be set to the size of the response body in bytes
- @info MDRFRAME8

ProtoType

```
dcl-pr MDR_getRespPtr extproc(*cwiden:'MDR_getRespPtr');
  handle      like(MDR_Handle_t);
  ptr         pointer;
  size        uns(20);
end-pr;
```

MDR_GETCLIENTERROR

Retrieve the last error message for a client connection.

- @param (input) handle = handle representing the current client session
- @param (output/optional) msgid = CHAR(7) message id (or *OMIT)
- @param (output/optional) msg = textual message (or *OMIT)
- @param (output/optional) sev = message severity (or *OMIT)
- @info MDRFRAME8

ProtoType

```
dcl-pr MDR_getClientError extproc(*cwiden:'MDR_getClientError') opdesc;
  handle      like(MDR_Handle_t);
  msgid       char(7)          options(*omit:*nopass);
  msg         varchar(32767:4) options(*omit:*varsize:*nopass);
  sev         int(10)          options(*omit:*nopass);
end-pr;
```

Json-Functions

MDR_SETFORMAT

Sets the format that the MDRest4i framework uses to send/receive documents

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) format = one of the MDR_FORMAT_xxx constants
- @info MDRFRAME

dcl-c MDR_FORMAT_JSON 0; dcl-c MDR_FORMAT_XML 1;

ProtoType

```
dcl-pr MDR_setFormat extproc(*cwiden:'MDR_setFormat');
  handle          like(MDR_handle_t);
  format          packed(7: 0) const;
end-pr;
```

MDR_GENPARSEOPTIONS

Set parsing/generating options for the MDRest4i DATA-INTO or DATA-GEN tools.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) options = (input) configuration options for either the parser or generator. See notes below:

These options can be used to configure how MDRFRAME(GENERATOR) (with DATA-GEN or MDR_DATAGEN) generates documents as well as how MDRFRAME(PARSER) (with DATA-INTO or MDR_DATAINTO) parses documents.

Options are specified as a string in the format option=value. Multiple options can be specified by separating them with spaces. If spaces are needed within a value, they can be specified by placing the value in double quote marks, such as option="value with spaces"

For use with DATA-INTO or MDR_DATAINTO

- value_null = the value written to a subfield when the corresponding JSON is null. Default = *NULL
- value_true = the value written to a subfield when the corresponding JSON is true. Default = '1'
- value_false = the value written to a subfield when the corresponding JSON is false. Default = '0'
- document_name = The name given to the document-level element of a JSON or XML document.
- datasubf = = XML only, the name of a subfield for a tag's inner value. Default = none
- req = yes/no whether data is automatically read from the request body of HTTP requests. Default=yes
- ns = XML only, namespace processing. keep=keep namespaces, remove=strip namespaces from tag names. Default=keep
- convts = yes/no whether timestamps are interpreted and ISO 8601 timestamps are converted to RPG/DB2 timestamps. Default=no

For DATA-GEN or MDR_DATAGEN

- datasubf = XML only, the name of a subfield for a tag's inner value. Default=none
- req = yes/no whether data is automaticaly written to the response body of HTTP requests. Default=yes
- sequenceType = array/object, controls how DATA-GEN sequences are generated. default=object
- includeEmptyArrays = yes/no. If yes, an array with 0 elements will be written as an empty array in the resulting document. If no, 0 elements will prevent the element from being written entirely. Default=no
- beautify = yes/no, whether the output contains tabbing and linefeeds to make it easier for a human to read. default=no
- format = XML or JSON = which format should data be generated in. Default=json unless the request body was in XML.
- convts = yes/no = should RPG/DB2 timestamps be converted into ISO 8601 format. default=no
- tstzoutput = ucs/local/none, timestamp timezone output. Should a timezone be appended? IF ucs, timestamps will be converted from the computer's current timezone to UCS, and a 'Z' (Z=Zulu, which is the ISO 8601 method of denoting UTC/GMT) appeneded. If local, the computers current timezone will be appended to the timestamp. If none (default) no timezone is appended. Default=none

For MDR_DATAINTO or MDR_DATAGEN

- trace = Specify an IFS path, and a trace file will be generated to assist with debugging the operation. Default=none. Example: trace="/tmp/trace.txt"
- trim = all/none. Whether blanks are trimmed from character strings. default=all
- countPrefix = A prefix used for counting JSON or XML elements when parsing, or controlling the number of output elements when writing. Same a
- renamePrefix = A prefix used by MDR_DATAGEN to control the names of outout elements. Same as the "renameprefix" option for RPG's DATA-GEN opcode. Default=none
- req = yes/no Whether data is automatically read from the request body or written to the response body of HTTP requests. Default=yes
- ccsid = CCSID of document. Values are ucs2, utf16, utf8, job or a ccsid number. Default=utf8 when generating, job ccsid when reading from a variable, utf8 when reading from a request body.
- document_name = the name of the document-level element. Used by MDR_DATAGEN for generating XML documents. default=none
- doc = string/file, controls where the JSON is read from or written to. If set to string, the Document parameter is interpreted as the JSON or XML document itself. If set to file, the Document parameter is interpreted as the path to a file in the IFS. Default=string
- output = in MDR_DATAGEN sequences, controls how the output is treated. clear=output is cleared before generating, append=output is added to the end of an existing file/string, or continue=output continued from the last sequence operation. Default=clear for start of sequences, continue otherwise.

Example

```
MDR_genParseOptions(handle: 'document_name=inputmodel +
req=yes +
tsconv=yes +
trace="/tmp/debug trace.txt"')
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_genParseOptions extproc(*cwiden:'MDR_genParseOptions')
                           opdesc;
  handle      like(MDR_handle_t);
  options     varchar(32764:4)    const options(*varsize:*omit);
end-pr;
```

MDR_DATAINTO

The MDRest4i version of DATA-INTO.

Note: It's typically considered better to use RPG's DATA-INTO opcode if possible, but this works similarly, and can be used on V7R2.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) format = (input) a JSON document representing the format of variable in 'resultVar'.
- @param (input) resultVar = (output) a variable (typically a DS) that will contain the result of parsing the JSON/XML.
- @param (input) document = (input) JSON/XML document to parse. If doc=file is specified, this should contain the IFS pathname where the data is read from. Otherwise this should contain the data. This parameter is ignored (and may be *OMIT) unless req=no is specified in the options.
- @param (input) options = (input) configuration options for the parser
- @info MDRFRAME6

ProtoType

```
dcl-pr MDR_DATAINTO extproc('MDR_DATAINTO') opdesc;
  handle      like(MDR_handle_t);
  format      varchar(10000:4)    const options(*varsize:*omit);
  resultVar   char(16773100)      options(*varsize:*omit);
  document    varchar(16773100:4) options(*varsize:*omit);
```

```
  options     varchar(10000:4)    const options(*varsize:*omit);
end-pr;
```

MDR_GETCOUNT

Get the count of the outermost number of elements parsed by MDR_DATAINTO. (For RPG's DATA-INTO, this is in the PSDS, do not use this procedure.)

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @return the count of the outermost element in the document
- @info MDRFRAME6

ProtoType

```
dcl-pr MDR_getCount uns(10) extproc('MDR_getCount');
  handle      like(MDR_handle_t);
end-pr;
```

MDR_DATAGEN

The MDRest4i version of DATA-GEN

Note: It's typically considered better to use RPG's DATA-GEN opcode if possible, but this works similarly, and can be used on V7R2.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) format = (input) a JSON document representing the format of variable in 'sourceVar'. or may be START or END for sequences.
- @param (input) sourceVar = (input) a variable (typically a DS) that contains the data that will be placed in the JSON/XML
- @param (input) document = (i/o) JSON/XML document. If doc=file is specified this is input-only, and should contain the pathname to where the document goes. If doc=string, this is output-only and is where the parsed document goes may be omitted if the document is only to be sent via HTTP (i.e. you don't need a copy in your program.)

options = (input) configuration options for the generator - @info MDRFRAME6

ProtoType

```
dcl-pr MDR_DATAGEN extproc('MDR_DATAGEN') opdesc;
  handle      like(MDR_handle_t);
  format      varchar(10000:4)    const options(*varsize);
  sourceVar   char(16773100)      options(*varsize: *omit);
  document    varchar(16773100:4) options(*varsize:*omit);
  options     varchar(10000:4)    const options(*varsize);
end-pr;
```

MDR_TREE_PARSE

Parse a JSON document into a tree structure

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) stmf = IFS stream file containing JSON document to parse or *OMIT if not reading from a file.
- @param (input) input = String containing JSON document to parse in UTF-8 or *OMIT if not reading from a string
- @param (input) errMsg = Error message returned if JSON does not parse successfully (or *OMIT)
- @return the pointer to the tree structure's document node or *NULL upon failure
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_tree_parse pointer extproc(*cwiden: 'MDR_tree_parse') opdesc;
  handle      like(MDR_handle_t);
  stmf        varchar(5000:4)     const options(*varsize: *omit);
  input       varchar(5242880:4)  const options(*varsize: *omit)
                                  ccsid(*utf8);
  input       varchar(5242880:4)  const options(*varsize: *omit);
  errMsg      varchar(4096:4)     options(*varsize:*omit);
end-pr;
```

MDR_TREE_FREE

Free up memory used by a JSON tree structure

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) docNode = pointer to the document node (returned by the MDR_tree_parse procedure)
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_tree_free extproc(*cwiden:'MDR_tree_free');
  handle      like(MDR_handle_t);
  docNode     pointer value;
end-pr;
```

MDR_BUF_JSONPATHV

Retrieve a string representation of a given JSON path. Output to buffer.

MDR_BUF_JSONPATHU:

Same as above, but output is uppercase

MDR_BUF_JSONPATHL:

Same as above, but output is lowercase

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) dst = pointer to the destination buffer to populate (data will be in job CCSID)
- @param (input) size = size of destination buffer (in bytes)
- @param (input) path = character string representing a JSON path
- @param (input) isNull = Returns 1 if value is NULL, 0 otherwise (you may *OMIT isNull)
- @return the length of data placed in dst (in bytes)
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_buf_jsonPathV uns(10) extproc(*cwiden:'MDR_buf_jsonPathV');
  handle      like(MDR_handle_t);
  dst         pointer value;
  size        uns(10) value;
  path        pointer value options(*string);
  isNull      int(10) options(*omit);
end-pr;
```

MDR_BUF_JSONPATHU

Retrieve a string representation of a given JSON path. Output to buffer in upper case.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) dst = pointer to the destination buffer to populate (data will be in job CCSID)
- @param (input) size = size of destination buffer (in bytes)
- @param (input) path = character string representing a JSON path
- @param (input) isNull = Returns 1 if value is NULL, 0 otherwise (you may *OMIT isNull)
- @return the length of data placed in dst (in bytes)
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_buf_jsonPathU uns(10) extproc(*cwiden:'MDR_buf_jsonPathU');
  handle      like(MDR_handle_t);
  dst         pointer value;
  size        uns(10) value;
  path        pointer value options(*string);
```

```
  isNull      int(10) options(*omit);
end-pr;
```

MDR_BUF_JSONPATHL

Retrieve a string representation of a given JSON path. Output to buffer in lower case.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) dst = pointer to the destination buffer to populate (data will be in job CCSID)
- @param (input) size = size of destination buffer (in bytes)
- @param (input) path = character string representing a JSON path
- @param (input) isNull = Returns 1 if value is NULL, 0 otherwise (you may *OMIT isNull)
- @return the length of data placed in dst (in bytes)
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_buf_jsonPathL uns(10) extproc(*cwiden:'MDR_buf_jsonPathL');
  handle      like(MDR_handle_t);
  dst         pointer value;
  size        uns(10) value;
  path        pointer value options(*string);
  isNull      int(10) options(*omit);
end-pr;
```

MDR_JSONPATHV

Retrieve a string representation of a given JSON path. Output to string.

MDR_JSONPATHU: SAME AS ABOVE, BUT OUTPUT IS UPPERCASE

MDR_JSONPATHL: SAVE AS ABOVE, BUT OUTPUT IS LOWERCASE

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) path = character string representing a JSON path
- @param (input/optional) isNull = Returns ON if value is NULL, OFF otherwise (you may *OMIT isNull)
- @return value (in string format) of the given JSON path
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_jsonPathV varchar(10240:4)
                     extproc(*cwiden:'MDR_jsonPathV')
                     rtnparm opdesc;
  handle      like(MDR_handle_t);
  path        varchar(2048:4) const options(*varsize);
  isNull      ind options(*omit:*nopass);
end-pr;
```

MDR_JSONPATHU

Retrieve a string representation of a given JSON path. Output to string in upper case.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) path = character string representing a JSON path
- @param (input/optional) isNull = Returns ON if value is NULL, OFF otherwise (you may *OMIT isNull)
- @return value (in upper case string format) of the given JSON path
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_jsonPathU varchar(10240:4)
                     extproc(*cwiden:'MDR_jsonPathU')
                     rtnparm opdesc;
  handle      like(MDR_handle_t);
  path        varchar(2048:4) const options(*varsize);
```

```
  isNull      ind options(*omit:*nopass);
end-pr;
```

MDR_JSONPATHL

Retrieve a string representation of a given JSON path. Output to string in lower case.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) path = character string representing a JSON path
- @param (input/optional) isNull = Returns ON if value is NULL, OFF otherwise (you may *OMIT isNull)
- @return value (in lower case string format) of the given JSON path
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_jsonPathL varchar(10240:4)
                     extproc(*cwiden:'MDR_jsonPathL')
                     rtnparm opdesc;
  handle      like(MDR_handle_t);
  path        varchar(2048:4) const options(*varsize);
  isNull      ind options(*omit:*nopass);
end-pr;
```

MDR_JSONPATHN

Retrieve the numeric representation of a given JSON path.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) path = character string representing a JSON path
- @param (input/optional) isNull = Returns ON if value is NULL, OFF otherwise (you may *OMIT isNull)
- @return the numeric representation as packed(30: 9) or 0 if it cannot be represented as a number
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_jsonPathN packed(30: 9)
                     extproc(*cwiden:'MDR_jsonPathN') opdesc;
  handle      like(MDR_handle_t);
  path        varchar(2048:4) const options(*varsize);
  isNull      ind options(*omit:*nopass);
end-pr;
```

MDR_JSONPATHF

Retrieve a numeric representation of a given JSON path in floating point format

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) path = character string representing a JSON path
- @param (input/optional) isNull = Returns ON if value is NULL, OFF otherwise (you may *OMIT isNull)
- @return the numeric representation as float(8) or 0 if it cannot be represented as a number
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_jsonPathF float(8)
                     extproc(*cwiden:'MDR_jsonPathF') opdesc;
  handle      like(MDR_handle_t);
  path        varchar(2048:4) const options(*varsize);
  isNull      ind options(*omit:*nopass);
end-pr;
```

MDR_GETARRAYSIZE

Returns the number of elements in a JSON array given the path to the array element.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) path = path to the JSON array node (can be CHAR, CHARZ or VARCHAR4)
- if this is empty/blank/omitted the root node is assumed.
- @return the number of array elements, or -1 upon error
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_getArraySize int(10)
                        extproc(*cwiden:'MDR_getArraySize')
                        opdesc;
  handle      like(MDR_handle_t);
  path        varchar(2048:4) const options(*varsize:*omit);
end-pr;
```

MDR_JSONPATHZ

This is like MDR_jsonPathV except that it returns a timestamp field. If the data matches the ISO 8601 format for a timestamp, it will be converted to an RPG timestamp automatically. If it matches an RPG timestamp, that will be returned instead.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) path = character string representing a JSON path
- @param (input/optional) isNull = Returns ON if value is NULL, OFF otherwise (you may *OMIT isNull)
- @return value (in timestamp format) of the JSON data, or z'0001-01-01-00.00.00.000000' upon error.
- @info MDRFRAME7

ProtoType

```
dcl-pr MDR_jsonPathZ timestamp
       extproc(*cwiden:'MDR_jsonPathZ') opdesc;
  path        varchar(2048:4) const options(*varsize);
  isNull      ind options(*omit:*nopass);
end-pr;
```

Attachment-Functions

MDR_GETPARTCOUNT

Retrieves the number of parts in a multipart input document

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @return the number of parts if successful, -1 upon failure
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getPartCount int(10)
       extproc(*cwiden:'MDR_getPartCount') opdesc;
  handle like(MDR_Handle_t);
end-pr;
```

MDR_GETPARTNAME

Retrieves the name of a part in a multi part input document

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @param (input) partNo = the part number to get the name of
- @return the name
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getPartName varchar(2048:4)
       extproc(*cwiden:'MDR_getPartName') rtnparm opdesc;
  handle like(MDR_Handle_t);
  partNo int(10) const;
end-pr;
```

MDR_GETPARTFILENAME

Retrieves the filename of a part in a multipart input document

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @param (input) partNo = the part number to get the file name of
- @return the filename
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getPartFileName varchar(2048:4)
       extproc(*cwiden:'MDR_getPartFileName') rtnparm opdesc;
  handle like(MDR_Handle_t);
  partNo int(10) const;
end-pr;
```

MDR_GETPARTTYPE

Retrieves the content-type of a part in a multipart input document

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @param (input) partNo = the part number to get the type of
- @return the content-type
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getPartType varchar(2048:4)
       extproc(*cwiden:'MDR_getPartType') rtnparm opdesc;
  handle like(MDR_Handle_t);
  partNo int(10) const;
end-pr;
```

MDR_GETPARTBUF

Copy attachment body to a memory buffer identified by a pointer

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @param (input) partNo = the part number to get the type of
- @param (output) buf = pointer to the memory buffer to store body in
- @param (input) size = size of the memory buffer
- @return the length of data written to the memory buffer or -1 upon failure
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getPartBuf int(20)
       extproc(*cwiden:'MDR_getPartBuf');
  handle like(MDR_Handle_t);
  partNo int(10) const;
  buf    pointer value;
  size   uns(20) const;
end-pr;
```

MDR_GETPART

Copy attachment body to a variable

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @param (input) partNo = the part number to get the type of
- @param (output) var = character variable to store data into
- @return the length of data written to the memory buffer or -1 upon failure
- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getPart int(20)
       extproc(*cwiden:'MDR_getPart') opdesc;
  handle like(MDR_Handle_t);
  partNo int(10) const;
  var    varchar(16000000:4) options(*varsize);
end-pr;
```

Content-Transfer-Encoding values for multipart messages

Constant	Description
MDR_CTE_NONE	no content-transfer-encoding assigned
MDR_CTE_BINARY	data should be interpreted as binary values
MDR_CTE_8BIT	data should be interpreted as 8-bit char
MDR_CTE_BASE64	data should be base64 encoded/decoded

Content-Transfer-Encoding Constants

```
dcl-c MDR_CTE_NONE 0;
dcl-c MDR_CTE_BINARY 536870912;
dcl-c MDR_CTE_8BIT   536870913;
dcl-c MDR_CTE_BASE64 536870915;
```

MDR_ADDPART

Add a new attachment to a multipart output

- @param (input) handle = handle provided by MDR_newClient or MDRAPI
- @param (input) contentType = content-type of newly added part
- @param (input) xferEnc = content-transfer-encoding of part
- @param (input) attname = attachment filename sent over network
- @param (input) data = string representing the data to store in the attachment. If this starts with stmf: the data will be read from a file.
- @param (input/output/optional) name = name of this message part.
- @param (input/optional) Disposition of this part. Should be one of the following:

Constant	Description
MDR_MSGDISP_NONE	do not provide a disposition
MDR_MSGDISP_ATTACH (default)	part is an attachment
MDR_MSGDISP_INLINE	part should be considered inline
MDR_MSGDISP_FORMDATA	part is a variable in a form

- @return the length of data written to the memory buffer or -1 upon failure
- @info MDRFRAME

Constants for setting Attachment Disposition Types

```
dcl-c MDR_MSGDISP_NONE     x'00000000';
dcl-c MDR_MSGDISP_INLINE   x'10000001';
dcl-c MDR_MSGDISP_ATTACH   x'10000002';
dcl-c MDR_MSGDISP_FORMDATA x'10000003';
```

ProtoType

```
dcl-pr MDR_addPart int(10)
       extproc(*cwiden:'MDR_addPart') opdesc;
  handle like(MDR_Handle_t);
  contentType varchar(2048:4)     const options(*omit);
  xferEnc     int(10)             const options(*omit);
  attName     varchar(2048:4)     const options(*omit);
  data        varchar(16000000:4) const options(*varsize:*omit);
  name        varchar(2048:4)     const options(*varsize:*nopass:*omit);
  disposition int(10)             const options(*omit:*nopass);
end-pr;
```

MDR_COPYTOCLOB

Copies data from a VARCHAR(4) (or fixed-length char if called from Cobol or CL) to a DB2 CLOB field.

- @param (output) dstClob = the clob to copy to
- @param (input) srcData = the source VARCHAR4 string.
- @param (input) size = the %SIZE of the CLOB column
- @info MDRFRAME5

ProtoType

```
dcl-pr MDR_copyToClob extproc(*cwiden:'MDR_copyToClob') opdesc;
  dstClob char(16000000)            options(*varsize);
/if defined(*V7R2M0)
  srcData varchar(16000000:4) const options(*varsize) ccsid(*utf8);
/else
  srcData varchar(16000000:4) const options(*varsize);
/endif
  size    uns(10)             const;
end-pr;
```

Ifs-Functions

MDRF_DIRNAME

Extract the directory portion of a path. For example, if the path

- @param (input) path = name to extract the directory from
- @return the directory, or '' upon error
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_dirname varchar(5000:4) rtnparm
       extproc(*cwiden:'MDRF_dirname') opdesc;
  path varchar(5000:4) const options(*varsize);
end-pr;
```

MDRF_BASENAME

Extract the base (object name) portion of a path. For example if the path is one/two/three, the base is "three"

- @param (input) path = name to extract the base name from
- @return the base name, or '' upon error
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_basename varchar(5000:4) rtnparm
       extproc(*cwiden:'MDRF_basename') opdesc;
  path varchar(5000:4) const options(*varsize);
end-pr;
```

MDRF_EXISTS

check if an object exists in the IFS

- @param (input) path = name of object to check
- @return '1' if it exists or '0' otherwise
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_exists ind
       extproc(*cwiden:'MDRF_exists') opdesc;
  path varchar(5000:4) const options(*varsize);
end-pr;
```

MDRF_MKDIR

Create a new directory ("folder")

- @param (input) path = name of directory to create
- @param (input/optional) opts = options to control how directory is created

Options are a space-separated name=value listing. For example they could be "owner=rwx group=rx public=x"

options are: owner=rwx - owner permissions group=rwx - group permissions public=rwx - public permissions rstdrnmunl=Y/N - enable restricted ren/unlink support intermed=Y/N - should intermediate directories be created if they are, they will be given rwxrwxrwx permissions (modified by the umask.)

- @return '1' if successful (or exists) and '0' otherwise
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_mkdir ind
       extproc(*cwiden:'MDRF_mkdir') opdesc;
  path varchar(5000:4) const options(*varsize);
  opts varchar(256: 4) const options(*varsize:*omit:*nopass);
end-pr;
```

MDRF_RMDIR

Remove/delete a directory (optionally including its contents)

- @param (input) path = name of directory to delete
- @param (input/optional) subtree = ON to delete any contents, or OFF only remove directory if empty.
- @return 0 for success, -1 for error
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_rmdir int(10)
       extproc(*cwiden:'MDRF_rmdir') opdesc;
  path    varchar(5000:4) const options(*varsize);
  subtree ind const options(*omit:*nopass);
end-pr;
```

MDRF_DELETE

Delete a file or other non-directory object

- @param (input) path name of object to delete
- @return 0 for success, -1 for error
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_delete int(10)
       extproc(*cwiden:'MDRF_delete') opdesc;
  path    varchar(5000:4) const options(*varsize);
end-pr;
```

MDRF_OPEN

Open a stream (IFS) file

- @param (input) path = IFS path of the file to open
- @param (input/optional) opts = options string
- @param (output/optional) errmsg = error message

OPTIONS STRING Format is XXXX, prop=val prop=val prop=val

XXXX = flags:

- **R**=read,
- **W**=write,
- **A**=append,
- **B**=binary,
- **N**=create new

prop=val = list of space-separated property=value options from the list below

- **owner**=permissions (rwx, rw, rx, wx, w, x)
- **group**=permissions (rwx, rw, rx, wx, w, x)
- **public**=permissions (rwx, rw, rx, wx, w, x) (default if none of the permissions set = inherit from directory)
- **intermediate**=Y/N should intermediate directories be created (short form intermed=Y/N or int=Y/N also works)
- **ccsid**=XXXX the CCSID of the file when creating (default = 1208 aka UTF-8) (fileccsid is an alias for ccsid)
- **pgmccsid**=XXXX the CCSID of the data in your program (default = job ccsid. localccsid is an alias for pgmccsid)
- **buffersize**=XXXX the size of the internal read buffer used by the file access routines. (default=65536 -- aka 64k)
- **eol**=XXX end of line sequence. crlf, cr, lf or a 1 or 2 byte hex code (default=CRLF, for UTF-8 use eol=0d0a for unix use eol=lf or eol=0a)
- **replace**=Y/N, if opening with W=write or N=new, replace an existing file if found.

Example

```
fh = MDRF_open('/path/to/file'
: 'N,ccsid=819 owner=RW group=RW public=R'
: err );

- @return a handle to the open file
or NULL upon error (and errmsg is set)
```

- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_open pointer
       extproc(*cwiden:'MDRF_open') opdesc;
  path   varchar(5000:4)  const options(*varsize);
  opts   varchar(500: 4)  const options(*varsize: *nopass: *omit);
  errmsg varchar(32767:4) options(*varsize: *nopass: *omit);
end-pr;
```

MDRF_READ

Read data from a stream (IFS) file

- @param (input) handle = file handle returned by MDR_open
- @param (output) buf = buffer to read data into
- @return the length of the data read or -1 upon failure
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_read int(10)
       extproc(*cwiden:'MDRF_read') opdesc;
  handle pointer value;
  buf    varchar(16000000: 4) options(*varsize) ccsid(*hex);
end-pr;
```

MDRF_LEN_READ

Same as MDRF_read, except reads into pointer/length

- @param (input) handle = file handle returned by MDR_open
- @param (output) buf = pointer to buffer to read data into
- @param (input) bufsiz = space available in buffer
- @return the length of the data read or -1 upon failure
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_len_read int(10)
       extproc(*cwiden:'MDRF_len_read') opdesc;
  handle pointer value;
  buf    pointer value;
  bufsiz uns(10) value;
end-pr;
```

MDRF_READLN

Read a line of text from a stream (IFS) file

- @param (input) handle = file handle returned by MDR_open
- @param (output) line = variable to read the line into
- @return the length of the data read or -1 upon failure
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_readln int(10)
       extproc(*cwiden:'MDRF_readln') opdesc;
  handle pointer value;
  line   varchar(16000000: 4) options(*varsize) ccsid(*hex);
end-pr;
```

MDRF_LEN_READLN

Same as MDRF_readln except it reads into a buffer addressed via a pointer and length

- @param (input) handle = file handle returned by MDR_open
- @param (output) linebuf = pointer to buffer to read line into
- @param (input) bufsiz = space available in buffer
- @return the length of the data read or -1 upon failure
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_len_readln int(10)
       extproc(*cwiden:'MDRF_len_readln') opdesc;
  handle  pointer value;
  linebuf pointer value;
  bufsiz  uns(10) value;
end-pr;
```

MDRF_WRITE

Write data to a stream (IFS) file

- @param (input) handle = file handle returned by MDR_open
- @param (input) data = variable containing data to be written
- @return the length of the data written or 0 upon failure
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_write int(10)
       extproc(*cwiden:'MDRF_write') opdesc;
  handle pointer value;
  data   varchar(16000000: 4) const options(*varsize) ccsid(*hex);
end-pr;
```

MDRF_LEN_WRITE

Same as MDRF_write except data is written from a pointer/length

- @param (input) handle = file handle returned by MDR_open
- @param (input) buffer = pointer to buffer containing data to write
- @param (input) length = length of data to write from the buffer
- @return the length of the data written or 0 upon failure
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_len_write int(10)
       extproc(*cwiden:'MDRF_len_write') opdesc;
  handle pointer value;
  buffer pointer value;
  length uns(10) value;
end-pr;
```

MDRF_WRITELN

Write data to a stream (IFS) file as a series of lines

- @param (input) handle = file handle returned by MDR_open
- @param (input) line = data to be written to disk. An EOL sequence (as set on MDRF_open) will be appended to the data.
- @return the length of the line written or 0 upon failure
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_writeln int(10)
       extproc(*cwiden:'MDRF_writeln') opdesc;
  handle pointer value;
  line   varchar(16000000: 4) const options(*varsize) ccsid(*hex);
end-pr;
```

MDRF_LEN_WRITELN

Same a MDRF_writeln except data is written from a pointer/length

- @param (input) handle = file handle returned by MDR_open
- @param (input) buffer = pointer to buffer containing the data to be written to disk. An EOL sequence (as set on MDRF_open) will be appended to the data.
- @param (input) length = length of buffer (in bytes)
- @return the length of the line written or 0 upon failure
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_len_writeln int(10)
       extproc(*cwiden:'MDRF_len_writeln') opdesc;
  handle pointer value;
  buffer pointer value;
  length uns(10) value;
end-pr;
```

MDRF_CLOSE

Close a file that was opened by MDR_open

- @param (input) handle = file handle returned by MDR_open
- @return 0 if the file was successfully closed -1 otherwise
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_close int(10)
       extproc(*cwiden:'MDRF_close') opdesc;
  handle pointer value;
end-pr;
```

MDRF_OBJINFO_T

Structure for information about an object in a directory

mode = file mode (flags identifying auths, etc) uid = numeric userid gid = numeric group id ccsid = ccsid of the data in the file isDirectory = ON means its a directory, OFF otherwise isFile = ON means its a regular obj, OFF otherwise size = size of the data in the file (bytes) allocSize = amount of disk space reserved for file (in bytes) timeAccessed = the last time the file was accessed (in any manner, including read-only) timeAttrChange = the last time the file's attributes were changed (file name, permissions, ccsid, etc) timeDataChange = the last time the data in the file was changed. objType = IBM i object type owner = owning user profile name group = assigned group profile name

Definition

```
dcl-ds MDRF_objInfo_t qualified template;
  mode           uns(10);
  uid            uns(10);
  gid            uns(10);
  ccsid          uns(5);
  isDirectory    ind;
  isFile         ind;
  size           uns(20);
  allocSize      uns(20);
  timeAccessed   timestamp;
  timeAttrChange timestamp;
  timeDataChange timestamp;
  timeCreated    timestamp;
  objType        char(10);
  owner          char(10);
  group          char(10);
end-ds;
```

MDRF_OPENDIR

Open a directory so you can read the files within it

- @param (input) path = path name to directory to open
- @param (input/optional) pattern = pattern of filenames to return, this allows for *, ? or [a-z] wildcards. Default = return all filenames
- @return a handle to an open directory. It can be read with MDRF_readDir, and must be closed with MDRF_closeDir. or *NULL upon error
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_openDir pointer
       extproc(*cwiden:'MDRF_openDir') opdesc;
  path    varchar(5000:4) const options(*varsize);
  pattern varchar(5000:4) const options(*varsize: *omit: *nopass);
end-pr;
```

MDRF_READDIR

Read the next filename from a directory (previously opened with MDRF_openDir)

- @param (input) dirh = handle to the open directory
- @param (output) filename = filename returned
- @param (output/optional) fileinfo = information about the returned file.

NOTE: Gathering the object information takes extra time, so don't pass the 3rd parameter if you don't need it.

- @return ON is returned to indicate that a filename was read or OFF is returned to indicate that all files in the directory (that match your pattern) have been read.
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_readDir ind
       extproc(*cwiden:'MDRF_readDir') opdesc;
  dirh     pointer         const;
  filename varchar(5000:4) options(*varsize);
  fileinfo likeds(MDRF_objInfo_t) options(*nopass: *omit);
end-pr;
```

MDRF_CLOSEDIR

Close an open directory handle (that was opened with MDRF_open)

- @param (input) dirh = directory handle to close
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_closeDir
       extproc(*cwiden:'MDRF_closeDir');
  dirh     pointer         value;
end-pr;
```

MDRF_LEN_READALL

Read the entire contents of a file into a buffer

- @param (input) path = pathname to file to read from
- @param (output) buf = buffer to place file data into
- @param (input) bufsize = size of buf
- @param (output) errmsg = buffer to place an error message into
- @param (input) errsize = size of errmsg
- @return the number of bytes read from the file or -1 upon error (with errmsg set)
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_len_readAll int(20)
       extproc(*cwiden:'MDRF_len_readAll');
  path    pointer value options(*string);
  buf     pointer value;
  bufsize uns(20) value;
  errmsg  pointer value;
  errsize uns(10) value;
end-pr;
```

MDRF_READALL

Read the entire contents of a file into a string

- @param (input) path = pathname to file to read from
- @param (output) string = string to place file contents into
- @param (output/optional) errmsg = string to place error message into
- @return the number of bytes read from the file or -1 upon error (with errmsg set)
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_readAll int(20)
       extproc(*cwiden:'MDRF_readAll') opdesc;
  path    varchar(5000:4)     options(*varsize) const;
  string  varchar(16000000:4) options(*varsize);
  errmsg  varchar(1000:4)     options(*varsize:*omit:*nopass);
end-pr;
```

MDRF_LEN_WRITEALL

Write an entire file from one buffer

- @param (input) path = pathname to file to write to
- @param (input) buf = buffer to write file data from
- @param (input) bufsize = length of data in buf to write
- @param (output) errmsg = buffer to place an error message into
- @param (input) errsize = size of errmsg
- @return the number of bytes written to the file or -1 upon error (with errmsg set)
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_len_writeAll int(20)
       extproc(*cwiden:'MDRF_len_writeAll');
  path    pointer value options(*string);
  buf     pointer value;
  bufsize uns(20) value;
  errmsg  pointer value;
  errsize uns(10) value;
end-pr;
```

MDRF_WRITEALL

Write an entire file from one string. If the file already exists, its contents are replaced.

- @param (input) path = pathname to file to write to
- @param (input) buf = string to write file data from
- @param (output/optional) errmsg buffer to place an error message into
- @return the number of bytes written to the file or -1 upon error (with errmsg set)
- @info MDRFRAME12

ProtoType

```
dcl-pr MDRF_writeAll int(20)
       extproc(*cwiden:'MDRF_writeAll') opdesc;
  path    varchar(5000:4)     options(*varsize) const;
  buf     varchar(16000000:4) options(*varsize) const;
  errmsg  varchar(1000:4)     options(*varsize:*omit:*nopass);
end-pr;
```

MDR_PATH2OBJ

Given an IFS-style path name, calculate the system library, object, member, IASP, etc.

- @param (input) inpath = ifs-style path name
- @param (output) qsysobj = MDR_sysobj_t structure containing the system names. If the resulting object is an IFS-only object, the isStmf indicator will be on. Otherwise, the member, object, library and iasp will be set. the Path is always set.
- @return 1 if the output is a stream file, or 0 if it is a library/obj.

Example

```
dcl-s  path varchar(5000) inz('/qsys.lib/qgpl.lib/proof.pgm);
dcl-ds sys  likeds(MDR_sysobj_t);

if MDRSDK_path2Obj(path: sys) = 1;
cmd = 'CALL PGM(' + %trim(sys.library)
+ '/' + %trim(sys.object) + ')';
QCMDEXC(cmd: %len(cmd));
endif;
```

- @info MDRFRAME5

ProtoType

```
dcl-pr MDR_path2obj int(10)
       extproc(*cwiden:'MDR_path2obj') opdesc;
  path    varchar(5000:4) const options(*varsize);
  qsysobj likeds(MDR_sysobj_t);
end-pr;
```

The data structure used by MDR_path2obj and MDR_path2src to format the IBMi object name from IFS path format to library naming

isStmf= *On = path is an IFS stream file path = IFS format path of IBMi object member = source member name object = object name library = library name iasp = IASP name

Definition

```
dcl-ds MDR_sysobj_t qualified template;
  isStmf  ind;
  path    char(5000);
  member  char(10);
  object  char(10);
  library char(10);
  iasp    char(10);
end-ds;
```

MDR_CASECONVERT

This simulates RPG's case=convert option for transforming a name from external document representation (XML or JSON) to a valid variable name.

The process works like this: 1) All accented letters are converted to non-accented versions. 2) any remaining non-letter/number values in the name are converted to the symbol specified in 'sym' (typically, an underscore) 3) Any consecutive symbols are compressed into one symbol. 4) Any symbols at the start of the name are removed 5) If the first letter of a variable is a number, the name is prefixed with N

- @param = (input) EBCDIC string representing the input name
- @param = (input) symbol to replace punctuation characters with.
- @return the converted name

Example

```
var = MDR_caseConvert('TEST!!!1TIME': '_');
returns TEST_1TIME
```

```
var = MDR_caseConvert('1TIME@@TEST': '_');
returns N1TIME_TEST
```

- @info MDRFRAME6

ProtoType

```
dcl-pr MDR_caseConvert varchar(4096:4) rtnparm
                       extproc(*cwiden: 'MDR_caseConvertOD') opdesc;
  src varchar(4096:4) const options(*varsize);
  sym varchar(1:4)    const;
end-pr;
```

Advanced-Consumer-Functions

Note: This section is for more advanced usage and will not be included in the customer program unless they define MDR_ADVANCED_OPTIONS

if defined(MDR_ADVANCED_OPTIONS)

dcl-c MAX_HEADERS 100; dcl-c MAX_PATHVARS 100; dcl-c MAX_QUERYVARS 100; dcl-c MAX_COOKIES 100;

MDR_RPG_HEADERS_T

This is the structure where incoming HTTP headers are stored inside the request.

count = number of headers header(x).name = name of header header(x).value = value of header

Note: Cookies are not provided in the header list.

Definition

```
dcl-ds MDR_RPG_Headers_t qualified template;
  count uns(10) inz(0);
  dcl-ds header dim(MAX_HEADERS);
    name  varchar(200);
    *n    char(1) inz(x'00');
    value varchar(3072);
    *n    char(1) inz(x'00');
  end-ds;
end-ds;
```

MDR_RPG_PATHVARS_T

This is the structure where path variables are stored inside the request.

count = number of path vars pathvar(x).value = value of path

Note: Path parts are relative to MDRAPI, so pathvar(1).value is always 'mdrapi' pathvar(2).value is always 'api-name'

typically customers will use 3+ for their own data.

Example

```
http://server:port/mdrapi/customer/1

pathvar(1).value = 'mdrpapi'
pathvar(2).value = 'customer'
parmvar(3).value = '1'
```

Definition

```
dcl-ds MDR_RPG_PathVars_t qualified template inz;
  count uns(10) inz(0);
  dcl-ds pathvar dim(MAX_PATHVARS);
    value varchar(3072);
    *n    char(1) inz(x'00');
  end-ds;
end-ds;
```

MDR_RPG_COOKIES_T

This is the structure where cookies are stored inside the request.

count = number of cookies cookie(x).name = name of the cookie cookie(x).value = value of the cookie

Definition

```
dcl-ds MDR_RPG_Cookies_t qualified template inz;
  count uns(10) inz(0);
  dcl-ds cookie dim(MAX_COOKIES);
    name  varchar(200);
    *n    char(1) inz(x'00');
    value varchar(3072);
    *n    char(1) inz(x'00');
  end-ds;
end-ds;
```

MDR_RPG_QUERYVARS_T

This is the structure where query string variables are stored inside the request.

count = number of variables queryvar(x).name = name of the variable queryvar(x).value = value of the variable

Definition

```
dcl-ds MDR_RPG_QueryVars_t qualified template inz;
  count uns(10) inz(0);
  dcl-ds queryvar dim(MAX_QUERYVARS);
    name  varchar(200);
    *n    char(1) inz(x'00');
    value varchar(2048) inz(x'00');
    *n    char(1);
  end-ds;
end-ds;
```

MDR_HTTPBODY_T

This is how MDRFRAME internatlly stores the HTTP request body. I don't expect that it would typically be used by customers -- but it is here if needed.

size = amount of memory allocated for the body ccsid = CCSID that the body data is stored in length = length of the data in the body buffer = data in the body.

Note that the HTTP body is allocated in teraspace and can be as large as 4 GB. (We may expand this to to 2 TB if ever needed.) To access the full data, when this exceeds 16 MB, you'll need to use pointers instead of the 'buffer' variable.

Definition

```
dcl-ds MDR_httpBody_t qualified template;
  size     int(20) inz(0);
  ccsid    uns(5)  inz(0);
  *n       char(6) inz(*ALLx'00');
  length   int(20) inz(0);
/if defined(*V7R2M0)
  buffer   char(16000000) ccsid(*utf8);
/else
  buffer   char(16000000);
/endif
end-ds;
```

MDR_ERROR_T

This is how MDRFRAME internally stores error messages.

id = null-terminated msgid. statement = null-terminated statement number text = null-terminated error message text sev = message severity

Definition

```
dcl-ds MDR_Error_t qualified align(*full) template;
  id         char(8);
  statement  char(11);
  text       char(32768);
  sev        int(10);
end-ds;
```

MDR_LOGOPTS_T

Details of which parts of the requests are logged

reqHdr = 1=log request header, 0=dont reqUri = 1=log request URI, 0=dont reqBody = 1=log request body, 0=dont respHdr = 1=log response header, 0=dont respBody = 1=log response body, 0=dont

Definition

```
dcl-ds MDR_logOpts_t qualified align(*full) template;
  reqHdr int(10);
  reqUri int(10);
  reqBody int(10);
  respHdr int(10);
  respBody int(10);
end-ds;
```

MDR_REQUEST_T:

This is the internal data structure that MDRest4i stores data about the current request (or transaction) in.

Sub-Field	Description
headers	structure containing the header information
cookies	structure containing the cookie information
queryvars	structure containing the query variable info
pathvars	structure containing the path variable info
outgoing_headers	The headers to use on output
outgoing_cookies	The cookies to use on output
request	teraspace pointer to the HTTP body of the input. (see MDR_httpBody_t)
response	teraspace pointer to the HTTP body of the output. (see MDR_httpBody_t)
contentType	pointer to C-style string containing the content type received from the caller
method	pointer to C-style string containing the HTTP method
uri	pointer to C-style string containing the URI
utf2job	code for mapping network/job CCSIDs in_multi = multipart document for input
out_multi	multipart document for output
status	the HTTP status code to return
isHTTPS	1 if using SSL/TLS, or 0 otherwise
format	format that MDRest4i is using for this transaction. See the MDR_FORMAT_xxx constants.
binary	1 if the Apache server called us in binary mode, 0 otherwise.
error	the last error that occurred in the framework

Definition

```
dcl-ds MDR_Request_t qualified template inz align(*full);
  headers          likeds(MDR_RPG_Headers_t);
  cookies          likeds(MDR_RPG_Cookies_t);
  queryvars        likeds(MDR_RPG_QueryVars_t);
  pathvars         likeds(MDR_RPG_PathVars_t);
  outgoing_headers likeds(MDR_RPG_Headers_t);
  outgoing_cookies likeds(MDR_RPG_Cookies_t);
  request          pointer;
  response         pointer;
  contentType      pointer;
  method           pointer;
  uri              pointer;
  utf2job          pointer;
  in_multi         pointer;
  out_multi        pointer;
  status           int(10);
  isHTTPS          int(10);
  format           int(10);
  binary           int(10);
  error            likeds(MDR_Error_t);
```

```
  logOpts          likeds(MDR_logOpts_t);
end-ds;
```

MDR_GETREQUEST

This retrieves the (internal) request structure that the MDRest4i framework uses to store information about this transaction.

- @param (input) handle = context handle, identifies which MDRest4i session is currently running.
- @param (input) request = returns a pointer to the request structure which should be in the format of MDR_Request_t, above.
- @return 0 always

Example

```
dcl-ds req likeds(MDR_Request_t) based(p_req);
MDR_getRequest(handle: p_req);

if req.format = MDR_FORMAT_XML;
xml-specific code here.
endif;
```

- @info MDRFRAME

ProtoType

```
dcl-pr MDR_getRequest int(10) extproc(*cwiden: 'MDR_getRequest');
  handle  like(MDR_handle_t);
  request pointer;
end-pr;
```

1.5.4 Token Handling

Overview

MDRest4i supports creating, refreshing and validating Java Web Tokens (JWT) using HS256 and RS256 Algorithms. The standards used follow the details and examples found on the jwt.io website.

The encryption and signature of MDRest4i JWT tokens use either an IBM i DCM application, a custom IBM i Key Store or a pre-supplied public and private key.

The signature of a token can also be verified using the MDR_verifySign() procedure.

MDRest4i also supports creating, validating Personal Access Tokens (PAT). It uses the ILE-bindable API, CEERAN0 to encode this.

Token Handling Procedures

All token handling procedures in MDRest4i require the copybook MDRST/QRPGLERSC.MDRTOKEN. The MDRTOKEN *SRVPGM is automatically bound in via the MDRFRAME binding directory.

Creating Tokens

MDRTOKEN is supplied with procedures MDR_CreateToken and MDR_create_JWT_RS256_DCM The main parameter in this procedure is the creds data structure.

Validating Tokens

The validation of a JWT or PAT token by MDREst4i works as follows:

- Tokens created by MDRest4i are stored in the table MDRST/MDRDCRED.
- A token is passed to the REST API request either as an Authorization header Bearer Token, or as a custom HTTP header
- The procedures MDR_ValidateToken or MDR_ValidateAuthToken are used to check the existence, and expiry status of the supplied token.
- For JWT Tokens, the JWT Claims data structure is returned by MDR_ValidateToken, MDR_ValidateAuthToken, and MDR_getClaims procedures from the service program MDRTOKEN

Refreshing Tokens

Token Handling Components

The MDRest4i Token handling is supplied with these development and runtime components:

Name	Type	Descriptio
MDRTOKEN	*SRVPGM	This service program is bound in to all REST Providers via Binding Directory MDRFRAME. It contains the ILE module MDRTOKEN with all of the token handling procedures exported.
MDRST/ QRPGLESRC.MDRTOKEN	RPGLE COPYBOOK	This copybook contains the procedures and documentation fpr the MDRTOKEN procedures
MDRST/MDRDCRED	DB2 TABLE/ PF	A Physical file containing saved tokens.
MDRST/ EXAMPLE.MDRTKNAPI	REST API	A REST API that demonstrates the creation of JWT tokens via REST.
MDRST/ EXAMPLE.MDRTKNREF	REST API	A Rest API that demonstrates the refresh of a JWT token using the refresh token issued by MDRest4i via REST.
MDRST/ EXAMPLE.MDRTKNTST	REST API	A Rest API that demonstrates the refresh of a JWT token using the refresh token issued by MDRest4i via REST.

MDRDCRED TOKEN STORE

Tokens (and their configuration) created by MDRest4i functions are stored in table MDRST/MDRDCRED. If REST API/Provider uses MDR_ValidateToken, it checks the existence of the provided token in this TABLE:

```
CREATE or replace TABLE ++OBJLIB++/MDRDCRED (
        -- Client ID: can be any value used to identify this credential
        CLIENTID VARCHAR(36) NOT NULL DEFAULT '' ,
        -- Application ID: can be any value used to identify this credential
        APPID VARCHAR(20) NOT NULL DEFAULT '' ,
        -- Usage Type: A=API, C=Consumer, B=Both, O=Other
        USETYP CHAR(1) NOT NULL DEFAULT '' ,
        -- Credential Type: B=Basic, T=Bearer(Token) R=Remote
        CREDTYP CHAR(1) NOT NULL DEFAULT '' ,
        -- tkntype: Token Type :
        --  PAT= Personal Access Token,
        --  JWT=JSON Web Token,
        --  JWS=JSON Web Signature
        TKNTYP CHAR(3) NOT NULL DEFAULT '' ,
        -- algor: Encryption/Decryption Algorithm & Key type
        --  UUID          = Random 36 byte value
        --  HS256     = Secret and SHA256 Encryption for Signature
        --  RS256-DCM = RS256 using DCM App
        --  RS256-KST = RS256 using PF and Lib Key Store
        --  RS256-PVT = RS256 using Private & Public Keys
        ALGOR VARCHAR(20) NOT NULL DEFAULT '' ,
        -- Authentication Token
        AUTHTOKEN VARCHAR(4096) NOT NULL DEFAULT '' ,
        -- User Name - usually used in consumers to obtain token
        USERNAME VARCHAR(256) NOT NULL DEFAULT '',
        -- Password - usually used in consumers to obtain token
        PASSWD VARCHAR(1024) NOT NULL DEFAULT '' ,
        -- small JSON payload for JWT Claims or small JWS body as JSON string
        CLAIMS VARCHAR(1024) NOT NULL DEFAULT '' ,
        -- DCM Application used for signatures
        DCMAPP VARCHAR(50) NOT NULL DEFAULT '' ,
        -- Path to keystore
        KEYSTORE VARCHAR(256) NOT NULL DEFAULT '' ,
        -- Keystore user
        KEYSTUSER VARCHAR(256) NOT NULL DEFAULT '' ,
        -- Keystore password
        KEYSTPWD VARCHAR(256) NOT NULL DEFAULT '' ,
        -- Client Secret
        CLISECRET VARCHAR(2048) NOT NULL DEFAULT '' ,
        -- Public key used for algor=RS256-PVT
        PUBLICKEY VARCHAR(2048) NOT NULL DEFAULT '' ,
        -- Private key used for algor=RS256-PVT
        PRIVATEKEY VARCHAR(2048) NOT NULL DEFAULT '' ,
        -- URI used to validate token
        AUTSVR VARCHAR(2048)  NOT NULL DEFAULT '' ,
        -- URI used to get new, or refresh token
        TOKSVR VARCHAR(2048)  NOT NULL DEFAULT '' ,
```

```
        -- Refresh Method: Automatic or Manual
        REFRESHMTH CHAR(10) NOT NULL DEFAULT '' ,
        -- Token used to request refresh of new token after expiry of current one
        REFRESHTKN VARCHAR(4096)  NOT NULL DEFAULT '' ,
        -- Token Expiry period
        ATOKENEXP INTEGER NOT NULL DEFAULT 0 ,
        RTOKENEXP INTEGER NOT NULL DEFAULT 0 ,
        -- Unit of measure used for Auth/Refresh token expiry
        EXPIRYUNIT CHAR(10) NOT NULL DEFAULT '' ,
        -- issued at time in epoch timestamp format
        CLAIMSIAT BIGINT NOT NULL DEFAULT 0 ,
        -- expiry time in epoch timestamp format
        CLAIMSEXP BIGINT NOT NULL DEFAULT 0 ,
        -- Token Issuer
        ISSUER CHAR(36) NOT NULL DEFAULT '' ,
        ISSUETIME TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
        -- Issue User/Jobname/jobnumber
        ISSUELOG CHAR(64) NOT NULL DEFAULT '' ,
        UPDATETIME TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
        -- Issue User/Jobname/jobnumber
        UPDATELOG CHAR(64) NOT NULL DEFAULT '' ,
        PRIMARY KEY( CLIENTID , APPID, AUTHTOKEN ) )   RCDFMT MDRDCREDF;

LABEL ON TABLE ++OBJLIB++/MDRDCRED
        IS 'MDRest4i - Credential Store' ;

LABEL ON COLUMN ++OBJLIB++/MDRDCRED
( CLIENTID IS 'Client ID' ,
        APPID IS 'Application ID' ,
        USETYP IS 'Usage Type' ,
        CREDTYP IS 'CredTyp' ,
        algor IS 'Alg-Keys' ,
        AUTHTOKEN IS 'Auth Token' ,
        USERNAME IS 'User Name' ,
        PASSWD IS 'Password' ,
        CLAIMS IS 'JSON Payload/Claims' ,
        DCMAPP IS 'DCM Application',
        KEYSTORE IS 'Keystore Path',
        KEYSTUSER IS 'Keystore User',
        KEYSTPWD IS 'Keystore Pwd',
        CLISECRET IS 'Client Secret' ,
        PUBLICKEY IS 'Public Key' ,
        PRIVATEKEY IS 'Private Key' ,
        AUTSVR IS 'Auth Server' ,
        TOKSVR IS 'Token Server' ,
        REFRESHMTH IS 'Refresh Method' ,
        REFRESHTKN IS 'Refresh Token' ,
        ATOKENEXP IS 'Auth Token Expiry' ,
        RTOKENEXP IS 'Refresh Token Expiry' ,
        EXPIRYUNIT IS 'Time Unit of Measure' ,
        CLAIMSIAT IS 'IssuedAt time(epoch)',
        CLAIMSEXP IS 'Expiry Timestamp(epoch)',
        ISSUER IS 'Token Issuer' ,
        ISSUETIME IS 'Issue Timestamp' ,
        ISSUELOG IS 'Issue User/Job' ,
        UPDATETIME IS 'Update Timestamp' ,
        UPDATELOG IS 'Update User/Job' ) ;
```

Default & Mandatory Values

Use the following **creds** subfields as applicable:

Creating an RS256 JWT Token Using DCM

creds.claims - JWT claims payload

At least one of the "iat" or "exp" name value pairs must be supplied in the claims JSON. These are used to determine the expiry of the token. **iat** = token issue timestamp in Epoch format. This is moved to creds.claimsiat. Along with the Example:

```
{
    "department":"Customer Care",
    "country":"US",
    "area":"Area1",
    "iat": 1720514756
}
```

Item	Subfields & Notes
creds.claims JWT claims payload	At least one of the "iat" or "exp" name value pairs must be supplied in the claims JSON. **exp** = token expiry timestamp in Epoch format. This is moved to creds.claimsexp
For an RS256 Token	**dcmApp** is required if a DCM application is being used for the certs. If a custom KeyStore us being used: **keystore**, **keystUser** and **keystPwd** must be supplied.
For an HS256 token	**cliSecret** (Client Secret - which is a max 32 byte string) is mandatory.
For any token	Below are the other mandatory entries in data structure "creds": **algor** **clientId** **appId** **claims** **atokenExp** **rtokenExp** **expiryUnit** For more details on each, see the MDRDCRED_t description above

MDRTKNEXP Data Area

The data area MDRTKNEXP contains the following default values, in the event these are not supplied in the creds subfields when calling procedure MDR_createToken. These values are used by procedure MDR_createToken, to create the token expiry value - ie creds.expclaims.

Pos	Value Description
1-10	Valid duration period of the authorization token
11-20	The expiry units used to calculate the actually expiry timestamp. seconds, minutes etc.
21-30	Valid duration period of the refresh token

Please refer refer to MDRST/EXAMPLE.MDRTKNAPI example to see how this data area is being used when the necessary header parameters for auth token expiry, refresh token exp and expiry unit are not supplied.

If the MDRTKNEXP values are blank, atokenExp is set to 180 and the expiry unit is then set as *seconds. Likewise, if the value in "rtokenExp" is 0, this is set to the epoch time equivalent of 24 hours.

If a valid expiryunit is neither supplied nor found in the data area "MDRTKNEXP", it is set as *seconds by default.

The value "atokenexp" will be used while refreshing the token to set the "claimsExp" as the current timestamp + atokenExp in specified expiry units.

The value "rtokenexp" is used to check the expiry of the refresh token. The calculation of refresh token expiry as an Epoch timestamp is: "rtokenExp" + "issueTime". This value is compared with the current timestamp to see if refresh token is expired

MDR_CREATETOKEN

For RS256 Token, either the "dcmApp" or the "keystore", "keystUser" and keystPwd" must be supplied. For HS256 token, cliSecret (which is a 32 byte string) is mandatory. Below are the other mandatory entries in data strcture "creds":

"algor", "clientId", "appId", "claims", "atokenExp", "rtokenExp", "expiryUnit"

The data area "MDRTKNEXP" contains default token expiry "duration" from the position 1-10, the expiry units SECONDS, MINUTES etc. are on 11-20 and the default refresh token expiry "duration" from 21-30. You can use them in the API if the requester hasn't supplied token expiry or refresh token expiry via header parms. refer "MDRTKNAPI" example to see how this data area is being used when the necessary header parameters for auth token expiry, refresh token exp and expiry unit are not supplied. If "atokenExp" is still having the value 0, it is set to 180 and the expiry unit is then set as "SECONDS". Likewise, if the value in

"rtokenExp" is 0, this is set to the epoch time equivalent of 24 hours. If the valid "expiryunit" is neither supplied and nor found in the data area "MDRTKNESP", it is set as "SECONDS" by default

The value "atokenexp" will be used while refreshing the token to set the "claimsExp" as the current timestamp + atokenExp in specified expiry units.

The value "rtokenexp" is used to check the expiry of the refresh token. The calculation of refresh token expiry is being done from "rtokenExp" added with "issueTime" and this value is compared with the current timestamp to see if refresh token is expired.

In the new API, at least one of the "iat" or "exp" must be supplied, otherwise give an error to requester

1.5.5 Log File Creation With Wildcards

The following wildcards/format-names can be used in the path for your logfiles as defined in a consumer or provider program. MDRest4i will then replace these with the appropriate value as described below.

> **Warning**
>
> This naming substitution only applies to IFS logs, not the DATAQ and PF options available in MDRSTCFG

Sub-folder creation

If the last directory in the path (immediately before the file name -- 2024-10-01 in the example below) doesn't exist it will created automatically before the log is added to it. This can be used with hardoced values AND replacement wildcards.

Examples:

```
MDR_setClientOpt(handle: MDR_LOG_PATH: 'ifs:/logs/%F/APICNS-%E.TXT');
```

Creates log file:

/logs/**2024-10-03**/APICNS-**2024-10-03-08.29.46.469000000000**.TXT (creating directory "2024-10-03" in the process)

```
MDR_setClientOpt(handle: MDR_LOG_PATH: 'ifs:/logs/createme/APICNS-%E.TXT');
```

Creates log file:

/logs/**createme**/APICNS-**2024-10-03-08.29.46.469000000000**.TXT (creating directory "createme" in the process)

Format	Description
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%C	Century number [00-99], the year divided by 100 and truncated to an integer
%d	Day of the month [01-31]
%F	ISO Date Format, same as %Y-%m-%d
%E	Adds a unique timestamp such as `2024-10-01-08.45.22.134660000001`
%H	Hour in 24-hour format [00-23]
%I	Hour in 12-hour format [01-12]
%j	Day of the year [001-366]
%m	Month [01-12]
%M	Minute [00-59]
%p	AM or PM string
%R	24-hour time format without seconds, same as %H:%M
%S	Second [00-61]. The range for seconds allows for a leap second and a double leap second
%T	24-hour time format with seconds, same as %H:%M:%S
%u	Weekday [1,7]. Monday is 1 and Sunday is 7
%U	Week number of the year [00-53]. Sunday is the first day of the week
%w	Weekday [0,6], Sunday is 0
%W	Week number of the year [00-53]. Monday is the first day of the week
%y	2 digit year [00,99]
%Y	4-digit year. Can be negative
%z	UTC offset. Output is a string with format +HHMM/-HHMM where: + indicates east of GMT, - indicates west of GMT, HH indicates the number of hours from GMT, MM indicates the number of minutes from GMT
%Z	Time zone name
%%	% character

1.5.6 Generating COBOL in MDRest4i

Overview

From version 14 upwards, MDRest4i generates COBOL programs.

Creating Provider and Consumer specifications is exactly the same, but the generated COBOL code architecture is slightly different. This difference is to account for the way the COBOL ILE compilers work on IBM i, but is on keeping with good COBOL coding practices.

COBOL Copybooks

COBOL copy books for using the ILE procedures in MDRFRAME service program can be found in MDRST/QLBLSRC.

Unlike RPGLE compilers, COBOL-ILE compilers increase the compiled object size based upon the size of every variable defined in the program and its copybooks. To optimize the size of the generated program objects, MDRest4i copybooks are split into two types.

DEFAULT COPYBOOKS

The most commonly used variables and their typical size definitions are defined in the default set of copybooks below. Teh generators in MDREst4i SDK use these copybooks when generating COBOL

Copybook	Description
MDRCBLLSC	MDRest4i COBOL Consumer Linkage Section
MDRCBLPRC	MDRest4i MDRFRAME Prodedures Sections
MDRCBLSPC	MDRest4i MDRFRAME Special Names Division. The linkage type definitions for the MDRest4i procs are defined in this copy book in order to ensure the called procedures in the MDRFRAME service program can accurately interpret the attributes and values of the passed parameters.
MDRCBLWSC	MDRest4i MDRFRAME COBOL Working Storage
MDRCBLLSAC	MDRest4i MDRFRAME for Advance Options
MDRCBLWSVS	MD Rest Variables small
MDRHTTPWSC	MDRest4i MDRFRAME for HTTP body with 16 MB

MDR_DATAGEN and MDR_DATAINTO

OVERVIEW

The RPGLE compiler determines the layout of the JSON structure at compile-time for RPG DATA-INTO and DATA-GEN functions.

This is only available from V7R2 upwards, and is not available at all in COBOL or other languages.

MDR_DATAGEN and MDR_DATAINTO are MDRFRAME functions that do the equivalent of IBM's DATA-INTO and DATA-GEN BIF's.

A schema is required to map between the JSON structure and the program DATA structures for MDR_DATAGEN and MDR_DATAINTO.

> **Please Note**
>
> The section below "JSON Schema Format" is for reference purposes. Both The MDRCBLCPY command, and the MDRest4i SDK, create copybook source automatically with these schema formats.

JSON SCHEMA FORMAT

The purpose of this format (or “schema") is to represent the layout of the data structure for a program that uses MDR_DATAINTO or MDR_DATAGEN.

```
{
"variable name": { "=type": "XXX", "=len": xx },
"variable name": { "=type": "XXX", "=len": xx },
"variable name": { "=type": "XXX", "=len": xx },
"structure name": {
  "=dim": xx,
  "variable": { "=type": "XXX", "=len": xx, "=dec": xx },
  "array variable": { "=type": "XXX", "=len": yyy, "=dim": xx },
  },
  "=sym": "X"
}
```

Names

Names of variables should match the JSON – except they should be converted to valid variables according to the “case=convert" rules specified for RPG’s DATA-INTO. There is a procedure called MDR_caseConvert that you can call to convert the name.

- Any accented characters are converted to unaccented variants.
- Anything that remains that is not a letter or number is converted to the “symbol" (=sym) character.
- Any consecutive symbols are merged into a single symbol.
- If there is a symbol at the start of a variable name, it is removed.
- If the variable name begins with a numeric digit, it is prefixed with the letter N.

For example: consider the following JSON key name: "21!!ALó"

- The accented character is replaced with an unaccented one. "21!!ALO"
- Symbols are replaced with an underscore. “21__ALO"
- Consecutive symbols are merged. “21_ALO"
- If the variable name begins with a digit, an N is added as a prefix “N21_ALO"
- (or N21-ALO for COBOL)

Attributes

Attributes describe a variable. Attribute names begin with the = (equal sign) sign to distinguish them from variable names. The equal sign was chosen because it exists in all character sets, but is not normally used as part of a variable name in programming languages.

- **=type** – (REQUIRED FOR ALL FIELDS) data type of variable. Can be:
- **char** = fixed length character
- **charz** = C-style, zero-terminated variable length character string. The length will indicate the maximum length of the string, including the zero-terminator.
- **varchar(2)** = RPG/SQL varchar data type, the first 2 bytes are an unsigned integer representing the current length, this is followed by up to =len characters (=len indicates the maximum)
- **varchar(4)** = same as varchar(2), except the length prefix is 4 bytes long. This is required for fields that are larger than 65535.
- **ucs2** = fixed-length UCS-2 characters. These are double-byte characters in Unicode.
- **varucs2(2)** = variable-length UCS-2. Like VARCHAR(2), except that the characters are double byte characters in UCS-2 rather than single byte in EBCDIC. The length specifies the number of characters (not the number of bytes.)
- **varucs2(4)** = same as varucs2(2), except with a 4-byte length prefix.
- **ind** = indicator. =len must be 1. Value must be either ‘0’ or ‘1’
- **zoned** = zoned decimal type. =len will indicate the total number of digits (including fractions) and =dec will indicate the number of fractional digits.
- packed = packed decimal type. =len will indicate the total number of digits (including fractions) and =dec will indicate the number of fractional digits.
- **binary** = this is meant to be used for RPG’s “bindec" (fixed format type B) fields. This should be avoided where possible. For =len of 1-4, a 2 byte binary is generated. For =len 5-9 a 4-byte is generated. (Same as RPG.) =dec indicates the number of decimal places.
- **int** = integer. (RPG fixed format type I, free format “int"). The =len value must be 3 for a 1 byte field, 5 for a 2 byte, 10 for 4 bytes, or 20 for 8 bytes. (same as RPG). Integers may not have decimal places, the =dec attribute is ignored.
- **uns** = unsigned integer. Same as “int", but allows larger numbers and cannot be negative. Lengths are the same as “int". Cannot have decimals.
- **float** = floating point. =len must be 8 for a double-precision floating point (8 bytes), or 4 for a single precision (4 bytes.) Since the number of decimal places can vary, the =dec attribute is not used during calculations, however, =dec will be used to control the number of decimal places generated when added to a JSON/XML document. For example, if you had the number 1.5, and =dec is 2, when added to JSON/XML document, it would be formatted as 1.50. NOTE: Floating point fields are imprecise, and not recommended.
- **date** = date field. The format is assumed to match the format of the customer data, no conversions are done.
- **time** = time field. The format is assumed to match the format of the customer data, no conversions are done
- **timestamp** = timestamp field. The format is assumed to match the format of the customer data, no conversions are done.
- **=dim** – (ARRAYS ONLY) array dimensions (only include if the fieldis an array or data structure array.)
- **=len** – (REQUIRED FOR ALL FIELDS) length of field indigits/characters.
- **=dec** – (NUMERIC ONLY) number of decimal positions.
- **=sym** – replacement symbol, generally would be _ (underscore) for most languages, or – (dash) for Cobol. This should be specified only once, at the top level, the same symbol applies to the entire format (it is not specific to a subfield.)

Countprefix fields

If the customer uses the “countprefix" option the fields MUST also be included in the the above JSON. (Even though they will not be included in the payload JSON/XML.) Countprefix fields must always be “=type": “int", “=len": 10

renameprefix fields

If the customer uses the “renameprefix" option, the field MUST also be included in the above JSON. (Even though they will not be included in the payload JSON/XML.) A renameprefix field must be coded with “=type" set to char, charz, varchar(2) or varchar(4), and should have “=len" set to a number between 1 and 4096.

Example

Suppose that you wanted a COBOL program to use MDR_DATAGEN to create a response in the following JSON format:

```
{
  "Stock": [
    {
      "Department": "STRING",
      "Category": {
        "MainCategory": "STRING",
        "Sub-Category": "STRING"
      },
      "Sizes": [ 55, 65, 70 ],
      "Colours": [ "White", "Black", "Green", "Red", "Blue" ]
    }
  ]
}
```

Note

note that all of the arrays are variable-length. for the sake of this example, we will say that there is a maximum of 10 elements in any array.

In order to process this data in COBOL, you want to generate from the following data structure:

```
01  WS-RESP.
  10 WS-NUM-STOCK PIC S9(8) USAGE BINARY VALUE 0.
  10 WS-RESP-STOCK OCCURS 10.
    15 WS-STOCK-DEPARTMENT PIC X(10).
    15 WS-STOCK-CATEGORY.
        20 WS-STOCK-CAT-MAINCATEGORY PIC X(10).
        20 WS-STOCK-CAT-NAME-SUB-CATEGORY PIC X(20).
        20 WS-STOCK-CAT-SUB-CATEGORY PIC X(10).
    15 WS-STOCK-NUM-SIZES   PIC S9(8) USAGE BINARY VALUE 0.
    15 WS-STOCK-SIZES       PIC S9(8) USAGE BINARY OCCURS 10.
    15 WS-STOCK-NUM-COLOURS PIC S9(8) USAGE BINARY VALUE 0.
    15 WS-STOCK-COLOURS     PIC X(20) OCCURS 10.
```

The **counprefix** feature will be used to control the number of array elements for each of the arrays. The caller will specify **countprefix=num_** so that any field that begins with “num_" is the count of a corresponding element.

The **renameprefix** feature will be used to allow the “Sub-Category" field to have a variable name.

The schema should look as follows:

```
{
  "num_Stock": {"=type": "int","=len": 10 },
  "stock": { "=dim": 10,
    "Department": { "=type":"char","=len":10},
    "Category":{
      "MainCategory":{"=type":"char", "=len": 10 },
      "Name_Sub_Category": { "=type": "char", "=len": 20 },
      "Sub_Category": {"=type": "char", "=len": 10 }
     },
     "num_sizes": {"=type":"int","=len":10},
     "sizes":{"=type":"int","=len":10,"=dim":10},
     "num_Colours":{"=type":"int","=len":10},
     "Colours":{"=type":"char","=len": 20, "=dim": 10}
  }
}
```

These field names (aside from the num_ and name_ fields) must match those that will be generated into the resulting JSON or XML elements.

Notice that the actual Cobol names don’t have to match the schema(MDR_DATAGEN and MDR_DATAINTO have no way to know what the actual Cobol names are.) In this respect, it is different from RPG’s DATA-INTO and DATA-GEN, as the schema there is based on the actual names in the program.

The fact that the names don’t need to match is helpful, because in Cobol it’s often advantageous to prefix fields with something like “WS-RESP",such as “WS-RESP-STOCK", etc. (Though, this isn’t always ideal, as it prevents features like MOVE CORRESPONDING from working nicely.)

MDR_DATAGEN will create the JSON or XML payload names exactly the same as the names in the schema, except in the case where they are changed with a renameprefix. MDR_DATAGEN will use the schema’s lengths, data types and decimal positions to calculate the correct place in the caller’s memory, and then read the from memory at that position, and format it for the payload document based on the data type.

MDR_DATAINTO will load the fields in the JSON/XML payload by converting the payload field name using the case=convert methodology, then looking for the matching field in the schema. If found, it will calulate the correct place in memory based on the lengths specified in the schema,and will generate data at that location based on the data type, length,and decimal positions given in the schema.

Once the correct format (schema) is determined, the JSON should be“squished" (spaces, indents, carriage returns, etc removed) and itshould be placed in a variable in the copybook. MDR_DATAINTO andMDR_DATAGEN will detect whether your variable is fixed-lengthcharacter, C-style null-terminated character, or VARCHAR(4) – it must be provided in one of those formats. Typically, in Cobol the most efficient way is to generate a data structure where the first 4 bytes are thelength, and the remainder is the schema – this is equivalent to a VARCHAR(4) in RPG.

For Example:

```
  01  WS-RESP-FORMAT.
      10 WS-RESP-LEN PIC 9(8) USAGE BINARY VALUE 412.
      10 WS-RESP-001 PIC X(256) VALUE '{"num_Stock":{"=type":"int",
-        '"=len":10},"stock":{"=dim":10,"Department":{"=type":"cha
-        'r","=len":10},"Category":{"MainCategory":{"=type":"char"
-        ',"=len":10},"Name_Sub_Category":{"=type":"char","=len":2
-        '0},"Sub_Category":{"=type":"char","=len":10}},"num_sizes
-        '":{"'.
      10 WS-RESP-002 PIC X(156) VALUE '=type":"int","=len":10},"siz
-        'es":{"=type":"int","=len":10,"=dim":10},"num_Colours":{"
-        '=type":"int","=len":10},"Colours":{"=type":"char","=len"
-        ':20,"=dim":10}}}'.
```

The WS-RESP-FORMAT may now be passed to MDR_DATAGEN in the first parameter.

```
CALL PROCEDURE 'MDR_DATAGEN'
  USING WS-RESP-FORMAT
        WS-RESP
        OMITTED
        WS-RESP-OPT
END-CALL.
```

Options (the 4th parameter)

Both MDR_DATAGEN and MDR_DATAINTO provide options that can be used to control how they read/generate JSON or XML documents. The following are the properties they understand:

- **trace** = specifies a diagnostic trace file in the IFS. The value should start with FILE= or FILEAPPEND= followed by the filename. For example: trace=FILE=/tmp/trace.txt will create a file named /tmp/trace.txt and place the trace in it. If there is already a file with that name, it will be replaced. If FILEAPPEND is given, instead of replacing the existing file it will add the trace to the end of the file.
- **trim** = can be “all" (default) which causes leading/trailing blanks to be trimmed from character strings, or “none" which causes the blanks to be retained.
- **countprefix** = specifies a prefix to be placed before field names in the data structure that are to be used to control the number of elements for that item.
- **renameprefix** = (MDR_DATAGEN only) specifies that fields beginning with a given prefix will be used to dynamically set the name of the field being output.
- **req** = can be set to **no** to not process the HTTP request, or **yes** (default) to process the request. If yes, MDR_DATAINTO will read the request body sent via HTTP instead of reading the user’s payload variable. MDR_DATAGEN will write the output to the HTTP response if req=yes. **value_true** – when a boolean value in the JSON is true, this specifies the value to be written to the user’s variable. Default is ‘1’ (because it’s assumed that you will map booleans to indicators.)
- **value_false** – when a boolean value in the JSON is false, this specifies the value to be written to the user’s variable. Default is ‘0’ (because it’s assumed that you will map booleans to indicators.)
- **value_null** – when a JSON document contains a null element, this is the value that will be written to your variable. Default: *NULL.
- **ccsid** = may be ucs2, utf16, utf8 or job.
- **document_name** = the name of the outermost element of the payload document. When generating XML, this corresponds to the name of the outermost XML tag. When reading data, this corresponds to the name of the data structure in the program.
- **doc** = controls where the JSON comes from (MDR_DATAINTO) or goes to (MDR_DATAGEN) when using req=no. Default is doc=string, which means the third parameter to MDR_DATAGEN/MDR_DATAINTO will be used for the data for the payload document. Also supported is doc=file, which causes the third parameter to be used as an IFS pathname, and the document will be read/written to that IFS file.
- **datasubf** = when working with XML, this determines which subfield of an structure is used for the “data". Other subfields are treated as sub-element names or attributes. This parameter is ignored for JSON documents.
- **ns** = when reading XML, this determines what should happen with namespaces when reading XML. The default value ns=keep means that the variable name is expected to include the XML namespace. Specify ns=remove to strip the namespace from the name. This parameter is ignroed for JSON documents.
- **beautify** = when generating a payload document, this determines whether the generator adds indenting and linefeeds to make it easier for a human to read. Default is beautify=no. Specify beautify=yes to make it nicer for a person to read.
- **format** = when generating a payload, this can be used to override the format. Specify format=xml to cause the output to be xml, or format=json for json. By default, the format will be JSON. If the input is read from an HTTP request, the content type will be used to default the output format, so content-types text/xml or application/xml will cause the output to be treated as xml, anything else will cause the output to be json. The format parameter will have no effect on MDR_DATAINTO – instead, MDR_DATAINTO will attempt to determine the format from the content-type (if reading from HTTP) or the contents of the document (when not reading from HTTP.)

Options are specified in a string in the 4<supth</sup parameter to MDR_DATAGEN or MDR_DATAINTO, as a space separated list. For example:

'countprefix=num_ nameprefix=name_ format=json req=no'

A value after the equal-sign may be put in quotes if you need to include spaces in the value. For example:

'trace="FILE=/tmp/diagnostic log.txt" req=no'

MDRCBLCPY Create COBOL Copybook

Creates a COBOL copybook with schemas used by MDR_DATAGEN and MDR-DATAINTO functions from MDRFRAME

1.5.7 HTTP Content Types

This are used to set the "Content-Type" HTTP header when sending a request or response with attachments.

By default MDRest4i sets this as application/json.

The `Content-Type` need only be for attachments when adding multipart responses or a single file attachment

The most common types are:

Type application:

application/java-archive application/EDI-X12
application/EDIFACT
application/javascript (obsolete) application/octet-stream
application/ogg
application/pdf
application/xhtml+xml
application/x-shockwave-flash
application/json
application/ld+json
application/xml
application/zip
application/x-www-form-urlencoded

Type audio:

audio/mpeg
audio/x-ms-wma
audio/vnd.rn-realaudio
audio/x-wav

Type image:

image/gif
image/jpeg
image/png
image/tiff
image/vnd.microsoft.icon
image/x-icon
image/vnd.djvu
image/svg+xml

Type multipart:

multipart/mixed
multipart/alternative
multipart/related (using by MHTML (HTML mail).)
multipart/form-data

Type text:

text/css
text/csv text/event-stream
text/html
text/javascript

text/plain
text/xml

Type video:

video/mpeg
video/mp4
video/quicktime
video/x-ms-wmv
video/x-msvideo
video/x-flv
video/webm

Type vnd:

application/vnd.android.package-archive application/vnd.oasis.opendocument.text
application/vnd.oasis.opendocument.spreadsheet
application/vnd.oasis.opendocument.presentation
application/vnd.oasis.opendocument.graphics
application/vnd.ms-excel
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
application/vnd.ms-powerpoint
application/vnd.openxmlformats-officedocument.presentationml.presentation
application/msword
application/vnd.openxmlformats-officedocument.wordprocessingml.document
application/vnd.mozilla.xul+xml

For detailed reference on http Content-Type values see: Content-Type HTTP

1.6 MDRest4i Commands

MDRCBLCPY - Create COBOL Copybook

Creates a COBOL copybook with schemas used by MDR_DATA-GEN and MDR-DATA-INTO functions from MDRFRAME. See here for details of this command and its purpose.

MDRCVT12 - Copy the V12 Specs to V14

This command is the first step in a two stage process, used to import SWAGGER specs from V12 into v14

It copies V12 specs from the V12 SDK folders, into the V14 file MDRDOAPI.

After this command has been run, The SDK import function provided in the Site Admin section for importing these specs in bulk, and converting the MDRest4i OAPI extensions to V14 format.

MDRCVT12 Command

```
                MDRest4i Copy V12 specs in V14 (MDRCVT12)

Type choices, press Enter.

Instance Name  . . . . . . . . . . INSTANCE       *DFT
User Name  . . . . . . . . . . . . USERNM         *ALL
Sufix Value  . . . . . . . . . . . SUFIX          V12

...
                                                                Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Parameter Definitions:

Parameter	Description
INSTANCE	MDRest4i Instance name for the V12 SDK. For most this will be DFT, which means the MDRest4i library name for V12 is MDRST*
USERNM	Which user name in the V12 SDK registered users, should SWAGGER specifications be imported for.
SUFIX	This is a suffix that will be added to the imported specs

---

MDRHTTPAPI - Create API Server

To setup your own HTTP instances for development or production purposes, then prompt the **MDRHTTPAPI** command. The following screen will appear:

```
                 MDRest4i Create SDK Server (MDRHTTPAPI)

Type choices, press Enter.

HTTP Server Name . . . . . . . .   MDRSTAPI      Instance Name
HTTP Instance Root Path  . . . .   '/www/mdrstapi'
Port Number  . . . . . . . . . .   4546          TCP Port Number for Server
Program ASP  . . . . . . . . . .                 Leave blank for *SYSBASE
API Library Name/s . . . . . . .   YOURAPILIB    List of API Object Libs
              + for more values
Authority Profile  . . . . . . .   MDOWNER       Character value
Start TCP/IP Server  . . . . . .   Y             Y, N

                                                                  Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Parameter Definitions:

Parameter	Description
HTTP Server Name	Name of the HTTP server instance
HTTP Instance Root Path	Specify the value of IFS path used to store docs for this HTTP instance. For example, if server name is **MDRDEMO** then the value of Server Root parameter could be: **/www/mdrdemo**. It is easier to name them the same for auditing and admin purposes.
Port Number	Port Number for the HTTP Server (default value is 80) Please check with your networking, infrastructure admin if allocating a port other than 80. Also check if the port you are using is not already in use on the LPAR you are hosting the server instance being setup.
Program ASP	System ASP where the MDRST instance is installed.
API Library Name/s	Name the library that contains the API programs generated by MDRest4i SDK.
Authority Profile	This profile used to execute the CGI APIs and system APIs used when running REST CGi programs. By Default this is set as MDOWNER which was created during the MDREST4INS installation earlier.
Start TCP/IP Server	Start the new HTTP server instance. This will start the instance after it is setup by the command. To start the server manually later use command below (Considering the Server Name as MDRSTAPI) : `STRTCPSVR SERVER(*HTTP) HTTPSVR(MDRSTAPI)`

Once the command has successfully executed, you can see the HTTP server instance configuration stored in IFS path using below command (Considering the Server Root as “/www/mdrstapi):

```
Wrklnk '/www/mdrstapi/conf/httpd.conf'
```

To see the active jobs running for this server instance use this command on the IBM i:

```
WRKACTJOB SBS(QHTTPSVR) JOB(MDRSTAPI)
```

Where “MDRSTAPI” is the server name from the MDRHTTPAPI command above

Testing the API Server

Tip

If this is a first time installation, conduct these tests **after** the license installation.

Using a web browser, test the Helloworld REST API shipped with the MDREST4i product and found in library MDRST:

```
http://[your serverip or host]:[yourport]/mdrapi/mdrhello?firstName=Mike&lastName=Smith&Title=Mr
```

You should see the following response in the browser:

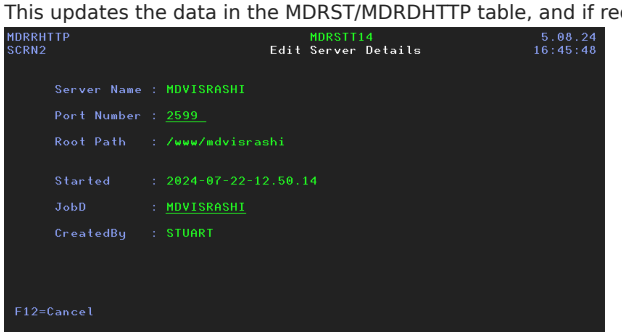
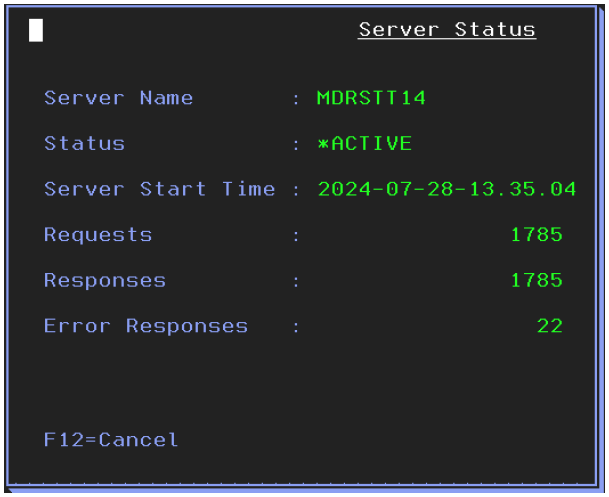
MDRRHTTP - Maintain HTTP Servers

This command allows a user to mange the variosu aspects of IBM i Apache HTTP Server instances, created with commands MDRSDKINS, MDRHTTPAPI.

Command MDRRHTTP

```
MDRRHTTP                          MDRSTT14                          5.08.24
                     MDRRHTTP - Maintain HTTP Servers              16:12:53

  Type options, press Enter.
   2=Edit   4=Delete   5=View   6=Start   7=Stop   9=Stats   L=Logs   J=JobD

Opt  Server Name  PortNo  Status       Started                JobD
___  MDRSTT14     2514    *ACTIVE      2024-02-26-18.02.21    MDRSTT14
___  MDRSTT14A    2515    *STOPPED     2024-05-28-17.16.50    MDRSTT14A
___  STUART13     2593    *ACTIVE      2024-04-05-15.24.10    STUART13
___  MDWEB        2566    *ACTIVE      2024-05-29-10.43.40    MDWEB
___  MDVISRASHI   2599    *STOPPED     2024-07-22-12.50.14    MDVISRASHI

                                                                    Bottom

 F6=Add Server     F3=Exit      F5=Refresh
```

Options

Option	Description
2=Edit	This updates the data in the MDRST/MDRDHTTP table, and if requested, the server instance httpd.conf. 
4=Delete	Deletes the server instance. Also deletes the file server ifs folder if selected on the conformation screen. 
5=View	Displays the server instance details view
6=Start	Start the server instance
7=Stop	Stop the server instance
9=Stats	If the server is active, it displays basic statistics since the the last start of the server. 
L=Logs	Opens the ifs folder for the HTTP server instance, as defined in the httpd.conf for that instance
J=JobD	Opens Work with Job Descriptions IBM i screen for the Job Description defined for that server. See the `HTTPStartJobDesc MDRSTT14/MDRSTT14` directive in the httpd.conf for that server.

MDRSDKINS - Install MDRest4i SDK

Details for this command are in the SDK installation page:

MDRSDKINS

MDRSDKTEMP - Init JSON Templates

This command is used to reset the SWAGGER/OAPI templates used by the SDK when creating a new Consumer or Provider specification, back to their shipped state.

MDRSDKUPD - Updating the SDK Web Application

> **Warning**
>
> To run this command, the MDRest4i SDK web application must be already installed for this instance. Run command MDRSDKINS to install the SDK web application

The MDRSDKUPD command has no parameters. It only executes if the SDK has already been installed. It backs up the existing config files for the MDRest4i SDK web UI, installs the new code base, and copies back the SDK web UI config files.

The SDK web UI is backed up to IFS folder: /mdrest4i/sdkbu-mdrst

> **Note**
>
> The backup path is based uon the instance name for MDRest4i. If the MDRest4i instance is 'V14' for example, the backup folder for te SDK update will be: /mdrest4i/sdkbu-mdrst

To run the command:

- `ADDLIBLE MDRST`
- `MDRSDKUPD` command and enter

MDRSDKUSR - Add/Update SDK User

This command is an alternative to the Site Admin>manage Users option in the SDK web UI.

```
                  MDRest4i- Add/Update SDK User (MDRSDKUSR)

Type choices, press Enter.

User Name  . . . . . . . . . . .                Name
Name . . . . . . . . . . . . . .   'REST4i USER'

Email address  . . . . . . . . .   'youremail@yourdomain.com'
Domain Name  . . . . . . . . . .   CONS          Name
User Class . . . . . . . . . . .   ADM
Activated  . . . . . . . . . . .   Y             Y, N
Default Object Library . . . . .   'YOURAPILIB'_____________________
_____________________________________________________________________
                                                                    ...
Default Source file  . . . . . .   '/QSYS.LIB/MDRST.LIB/EXAMPLE.FILE'
_____________________________________________________________________
```

```
                                                                ...
Default Server URL . . . . . .   'http://yourhost:yourport/yourlib'___
```

Variable	Description
USERNAME	A valid IBM i user name.
NAME	User's actual name
EMAIL	User's email address
DOMAIN	Valid Values: CONS for Console and Documenter DOCU for MDRest4i Portal ONLY
ACTIVATED	Users must be activated before they can login. Valid values are Y or N
DFTOBJL	Specify the default object library to use for new OAPI specs.
DFTSRCF	Specify the default source file to use for new OAPI specs.
DFTURL	Specify the default testing URL to use for new OAPI specs.

MDRSTCFG - API Path and Logging Configuration

MDRDCFG is a DB 2 table that holds the configuration details of API programs called by MDRAPI. It can be edited programmatically, by using command: MDRST/MDRSTCFG, or by MDCMS Data Group attributes.

It controls the following for each API program(Provider) called by MDRAPI:

- URI resource name to pgm object mapping
- Logging

To edit details in this file, invoke command `MDRST/MDRSCFG`

MDRSTCFG List

```
                     Edit MDRest4i Provider Configs

  2=Edit   4=Delete

                                                          Logging
  Opt  Root Resource                                    IFS  DTQ  PF
       /CLIENTS                                          Y    N    N

                                                               Bottom

F3=Exit F5=Refresh F6=Add
```

Selecting 2 to edit will bring up the following screen:

MDRSTCFG Edit Screen

```
                              Edit Config

     Root Resource:  /CLIENTS

           Program:  PGM001

           Library:

      Log to (Y/N):  IFS: Y               DATAQ: N              PF: N

          Log Name:
                IFS: /mylogs/pgm001-%F.txt
             DATAQ:
                PF:

  Log Details (Y/N):       For Request              For Response
                Path:            Y
             Headers:            Y                        Y
                Body:            Y                        Y
```

```
F12=Cancel
```

Variable	Description
Root Resource	This is user friendly resource name used to invoke the API. MDRAPI will resolve this and call the **Program** below
Program	Name of the API program called by MDRAPI
Library	Library name used in the call. No value will cause MDRAPI to use *LIBL to make the call
Log to	Possible values: **Y** or **N**. There are three logging mechanisms that MDREst4i MDRFRAME provides for a REST Provider. Each option will causes MDRAPI create a log using the details in the relevant **Log Name** below: **IFS** - Creates a log file in the IFS **DATAQ** - Creates a log in a DATAQ.
Log Name IFS	The format of these is as follows: /folder/subfolder/file-%F.txt The "**%F**" is substitute syntax handled by the MDRFRAME framework when creating the file. It is used to set date and time stamp values in the name. So the above example would look like this: /folder/subfolder/file-2024-02-11.txt for full detaisl on logging name options see LOGFILEOPTS
Log Name DATAQ	Use format LIBNAME/DTAQNAME. This creates a DATAQ of the specified library/name, and writes log entries to it.
Log Name PF	The format of these is LIBNAME/PFNAME. This makes a copy of MDRST/MDRLOGS into the LIBNAME/PFNAME specified, and logs requests there each time this Provider program is called
Log Details	These values specify the content that will appear in the log

1.7 MDRest4i Documenter

1.7.1 MDREST4i-Documenter-V14

Welcome to MDREST4i Documenter online Help. This wiki provides details on each of the screens available on the Web UI of the Documenter part of MDREST4i SDK

As the Documenter Application is launched, below in Figure-1 is the first screen for authentication of User. It has option for creating new user using **SIGNUP** button or **LOGIN** button for existing users

LOGIN SCREEN

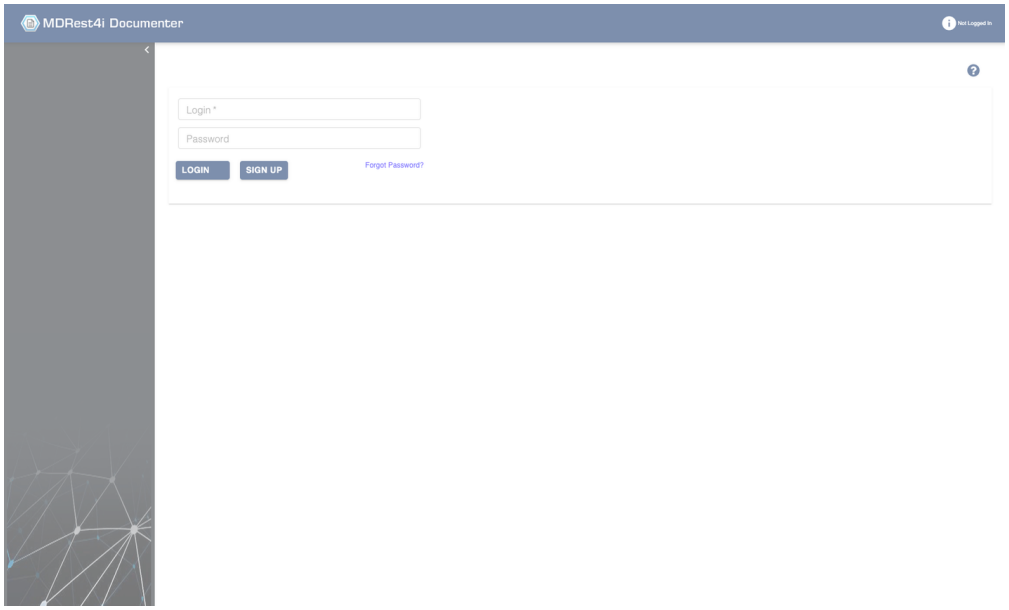


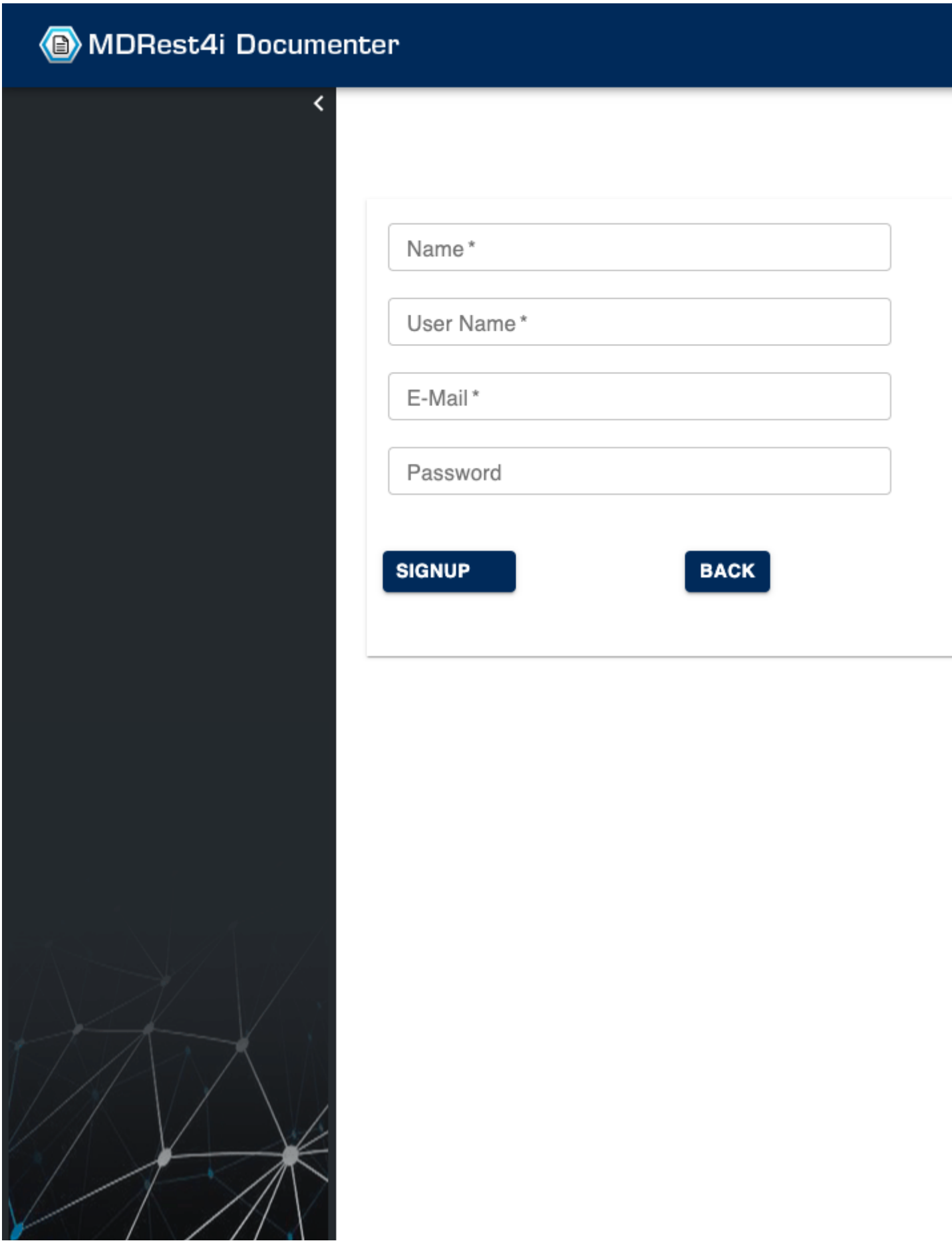
Figure-1

On successful login with the authentic user details, Below are the links to get details of the specific sections

Admin	It allows user to manage/edit some settings, and edit templates used in documenter
Documents	It allows user to view/edit documents published from the MDREST4i SDK Console application.

SIGNUP

The intial screen also have the option of **SIGN UP**. By clicking on it, we get the Screen as in Figure-2.



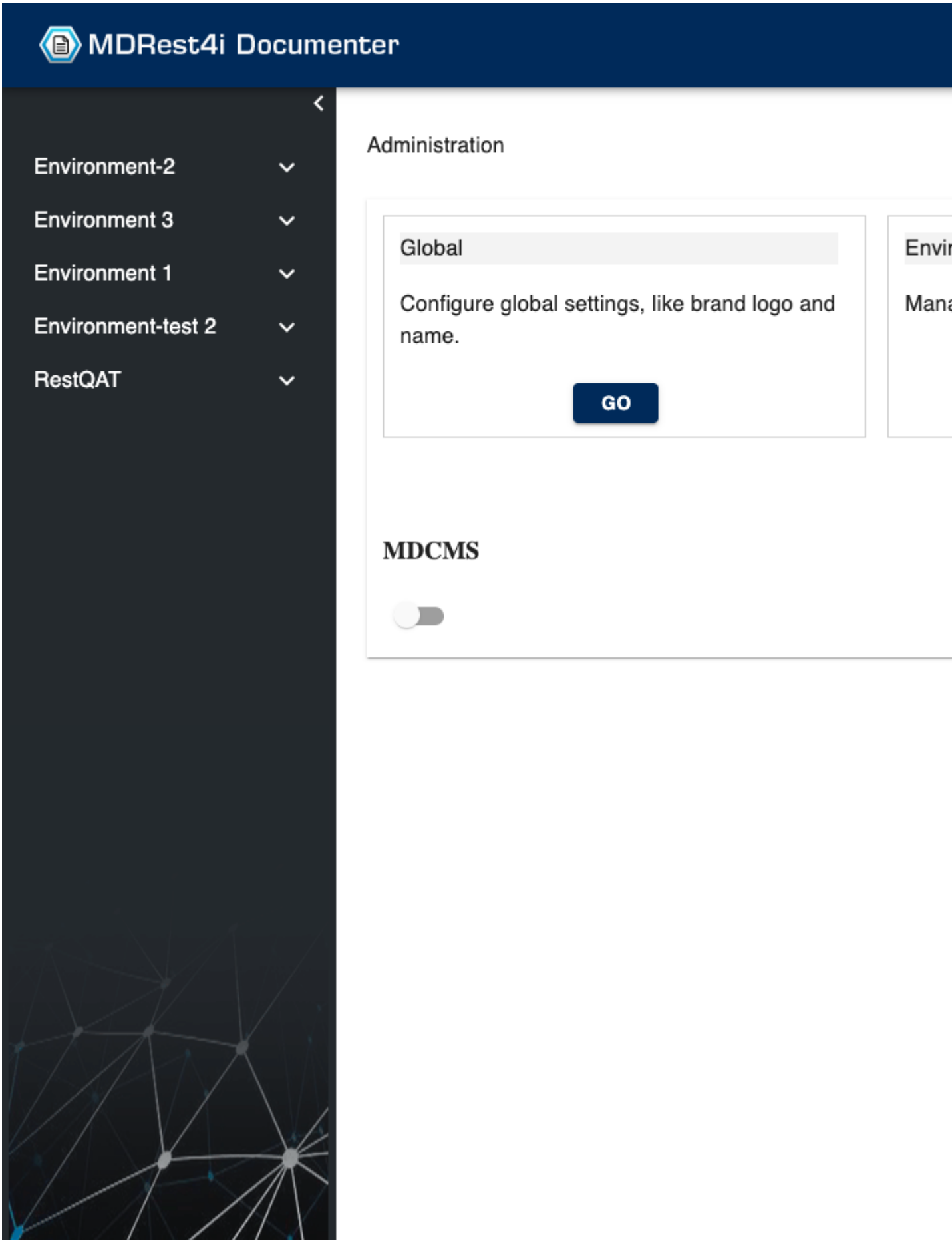
It asks for a valid username, Email Address and password to create a new User. Once User created it can be activated from the Console Application only.

1.7.2 Admin

Overview

ADMIN

Below in Figure-1 are the available options in Admin section. Only user with administrative privilege can use these options.



Fi

The details of each of the section can be seen by clicking respective Links

Global	Configure global settings, like brand logo and name.
Environments	Manage Environments and APIs

It has one MDCMS switch, ON mode allows to publish documents of MDCMS type of requests.

Global Settings

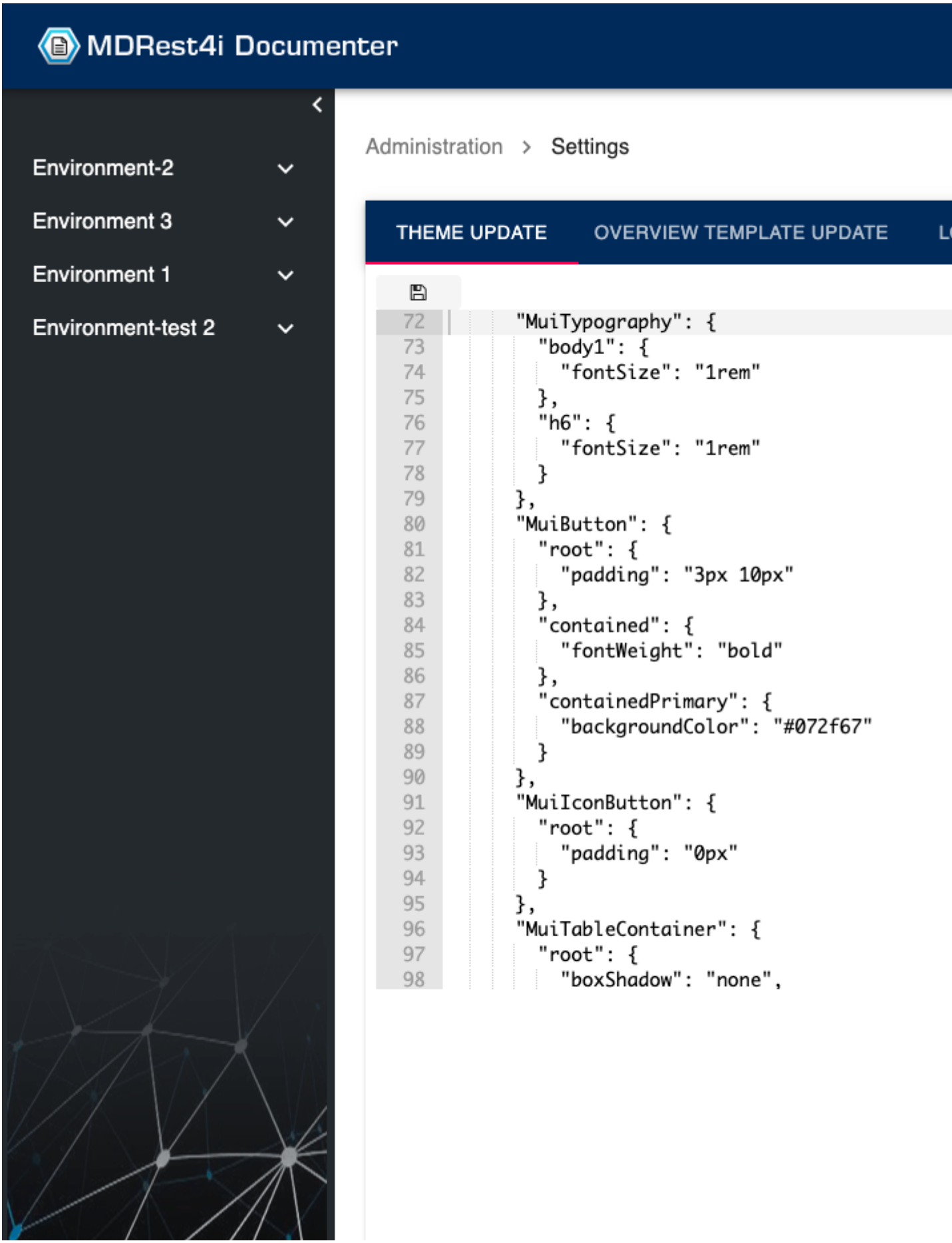
GLOBAL SETTINGS

The below Screen (Figure 1) appears when Global Setting option has been selected. It has 2 tabs namely :

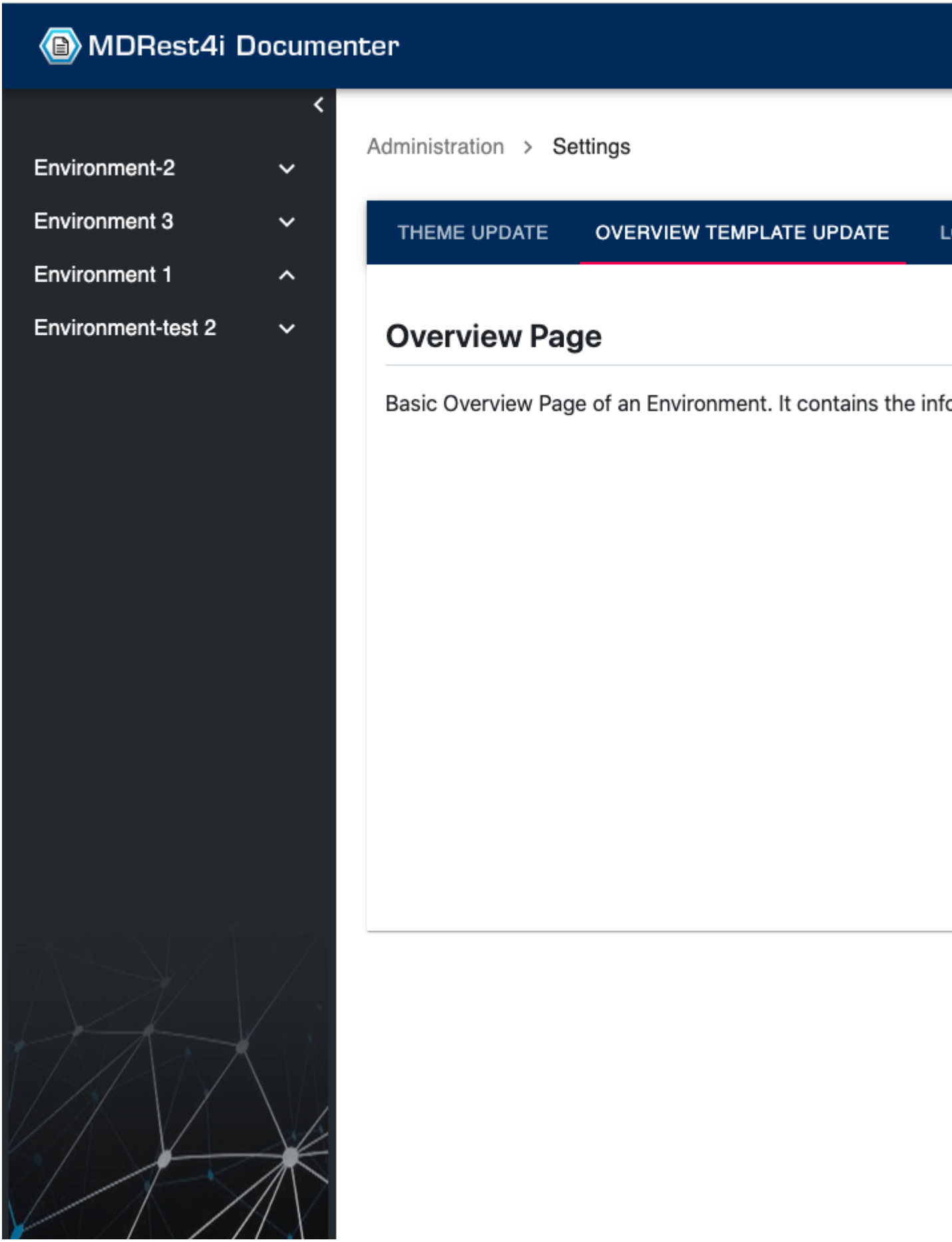
Theme Update	It gives option to edit theme.json file which contains the global css settings of the components used as seen in **Figure 1**.
Overview Template Update	It gives option to edit Overview Template as shown in **Figure 2**.
Logo Change	It gives option to edit the Logo of the Documenter Application as shown in **Figure 4**.

Theme Update

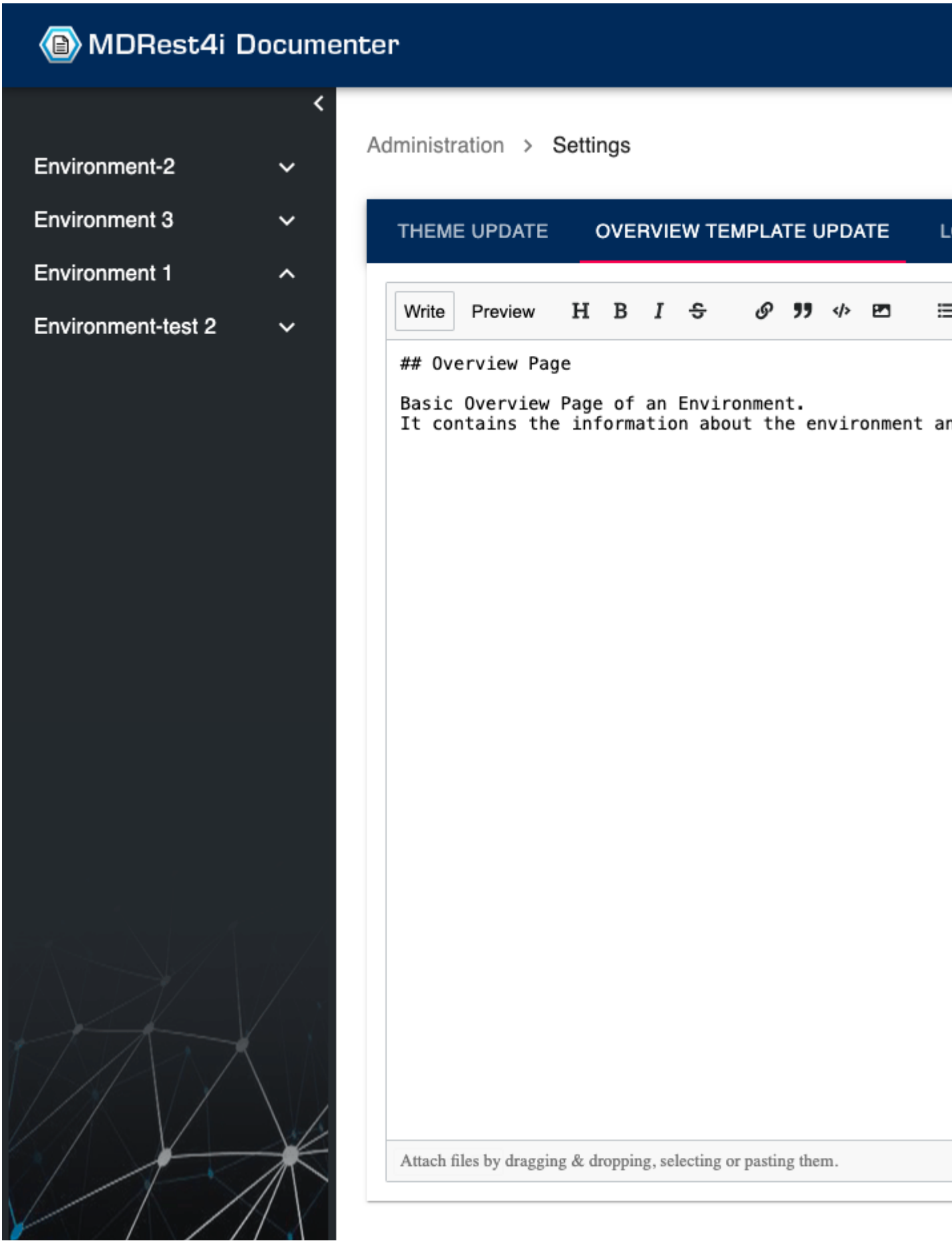
This file can be edited and by clicking the Save button, the new changes will take effect.



Overview Template Update

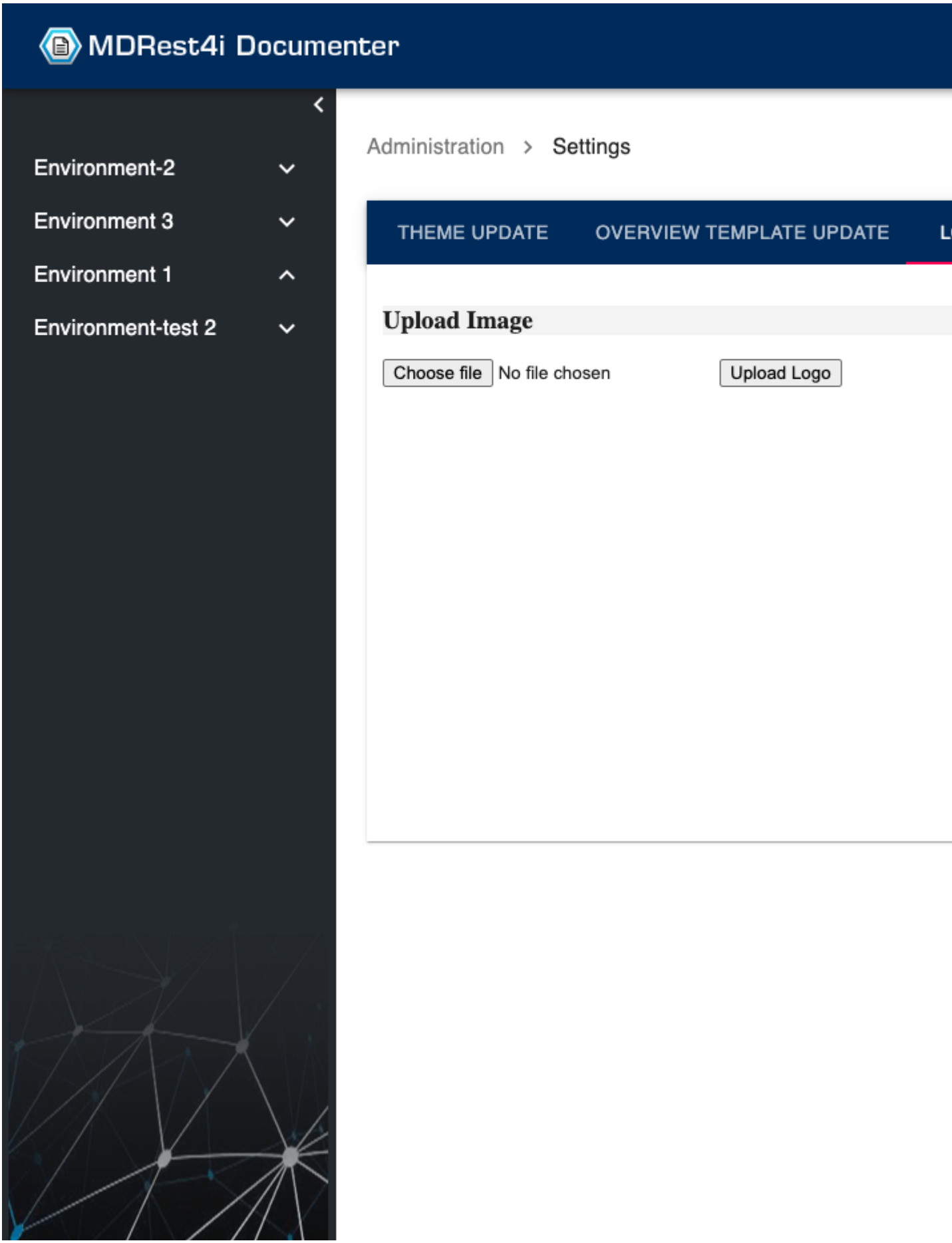


This file can be edited by opening it in the Edit Mode by clicking on pencil Icon. It will be open in the Edit mode as seen in Figure 3. Contents can be edited in a ShowDown editor and can be saved for further use.



Logo Change

By choosing new image file and uploading it, the Logo on the Left top position will be replaced by the new Logo.

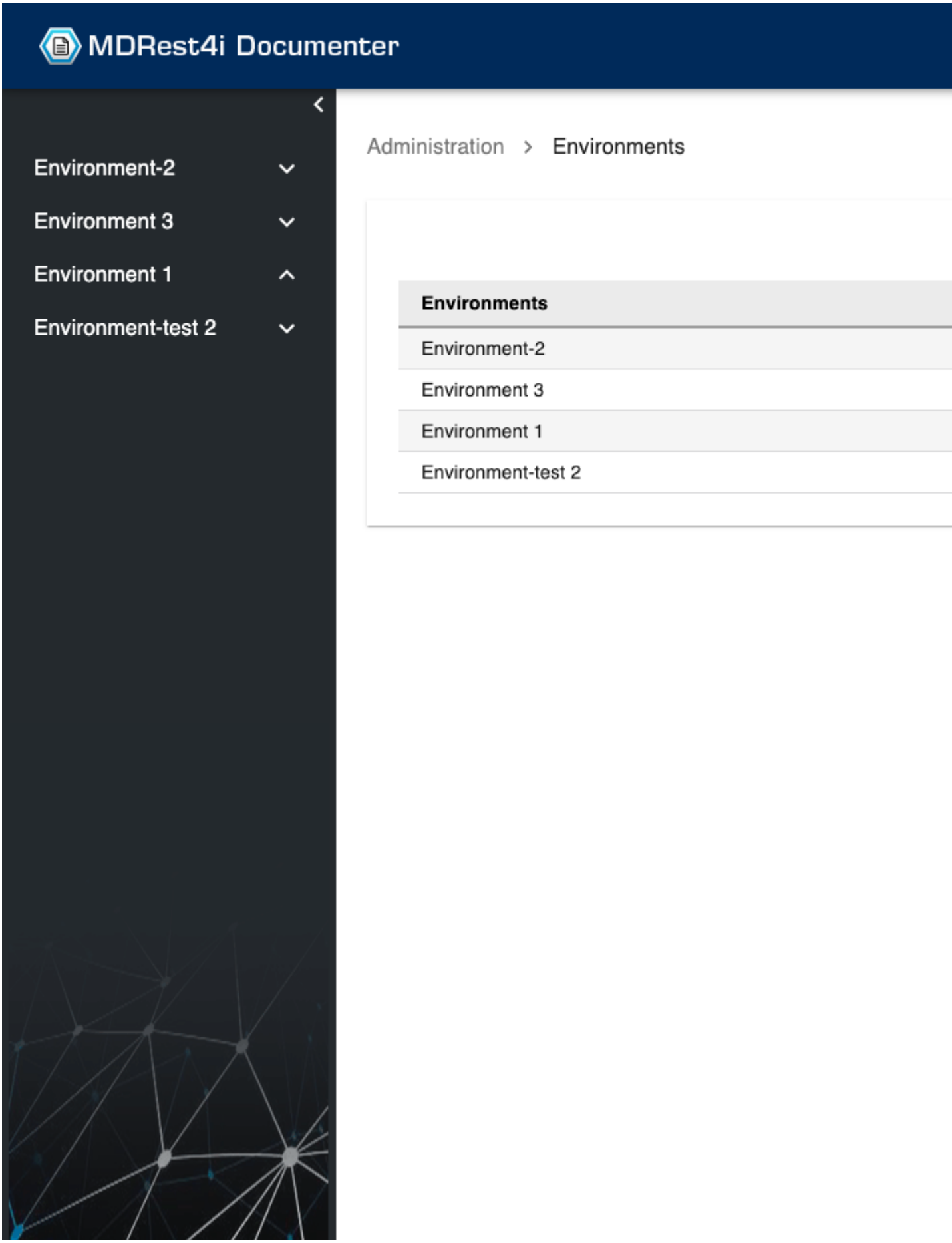


Environments

ENVIRONMENT SETTINGS

The below Screen (Figure 1) appears when Global Setting option has been selected. It has 2 tabs namely :

Displays list of available environments Figure-1. It allows to Edit any Environment Details or Delete specific Environment with all the data in it. There is also an option to Add new Environment +



Below are the screens for 3 available options on Environment Settings

Add Environment Screen

It allows the user to add a new Environment

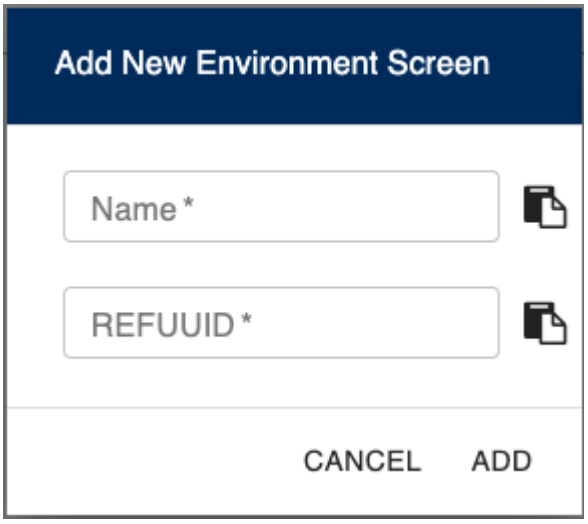


Figure-2

Edit Environment Screen

It allows the user to edit name of the selected Environment

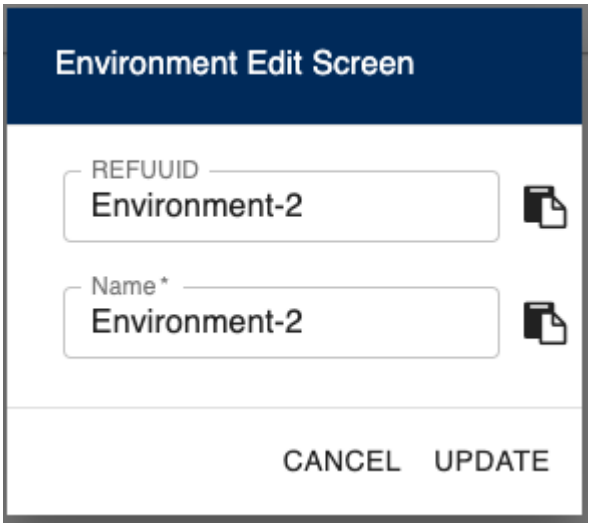


Figure-3

Delete Environment Screen

Confirmation Screen appears before deleting the selected Environment

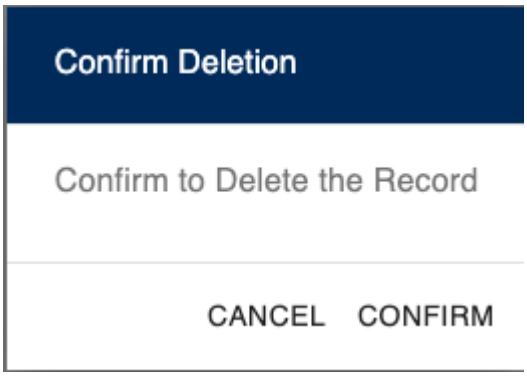
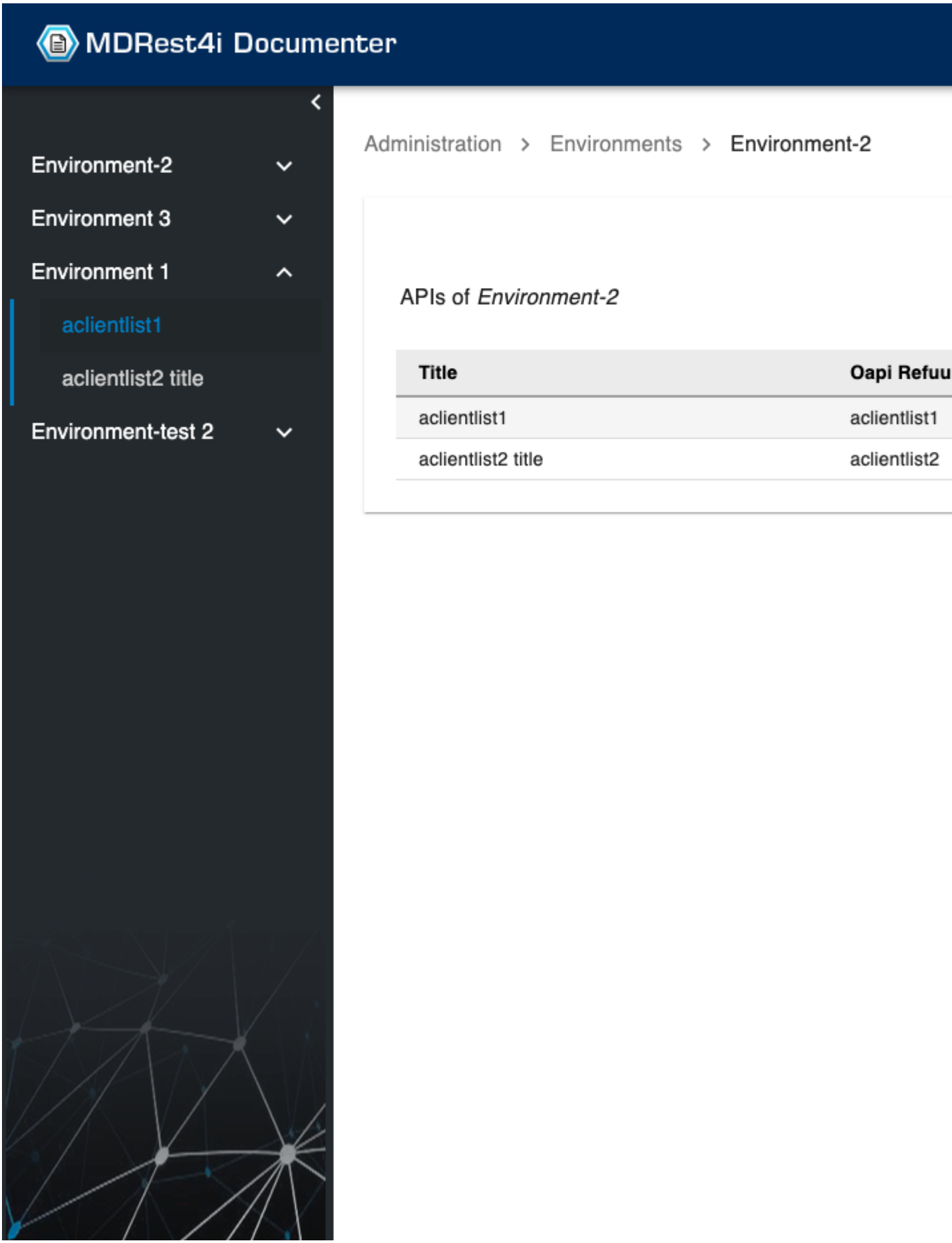


Figure-4

APIs Details

On clicking any Environment, it displays list of available APIs in that particular Environment Figure 5. It allows to Edit particular API Details, Delete the API or Copy API from one environment to another Environment.



Below are the different screens of available options in APIs screen

Edit Screen

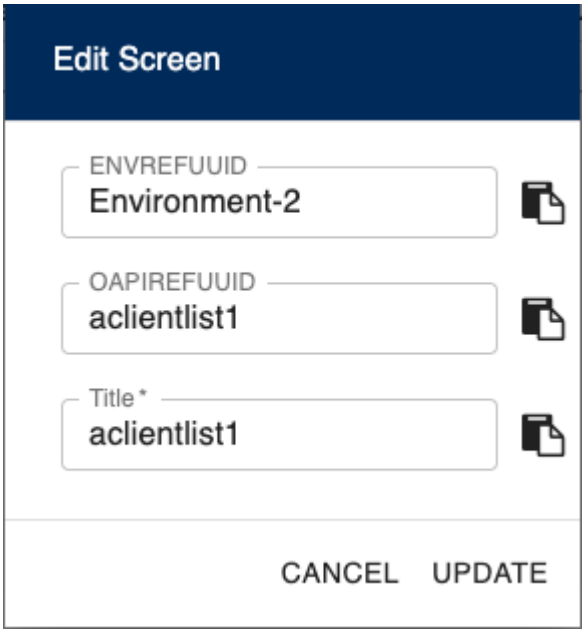


Figure-6

Delete Screen

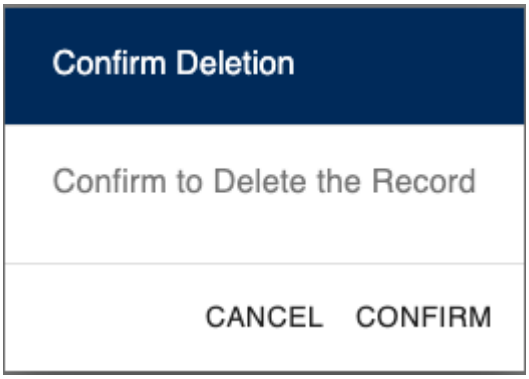


Figure-7

Copy API Specification

It asks for the destination Environment where the selected API needs to be copied

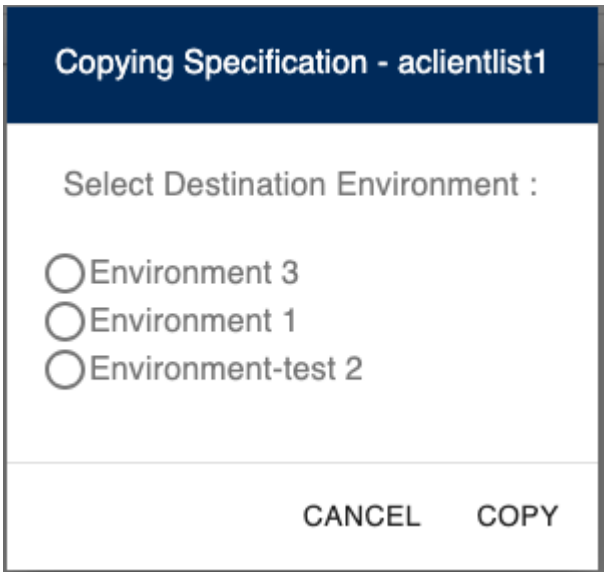


Figure-8

1.7.3 Managing Documents

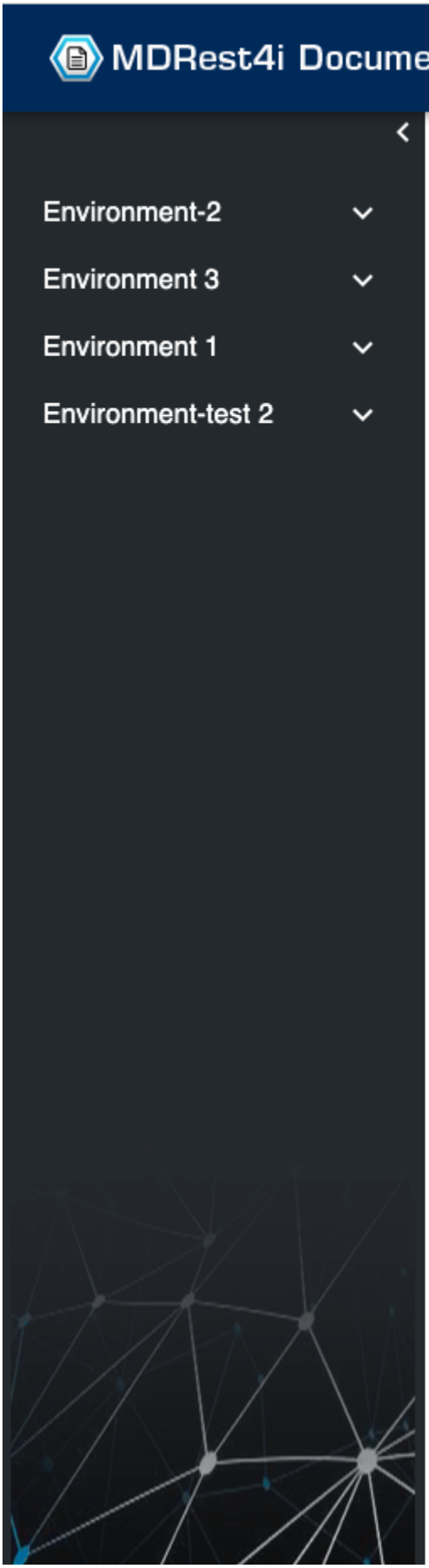
Overview

MANAGING DOCUMENTS

After successful login, Landing document will appear (Figure-1) containing information about the Documenter. It also lists the available Environments in the Application.

Landing Document

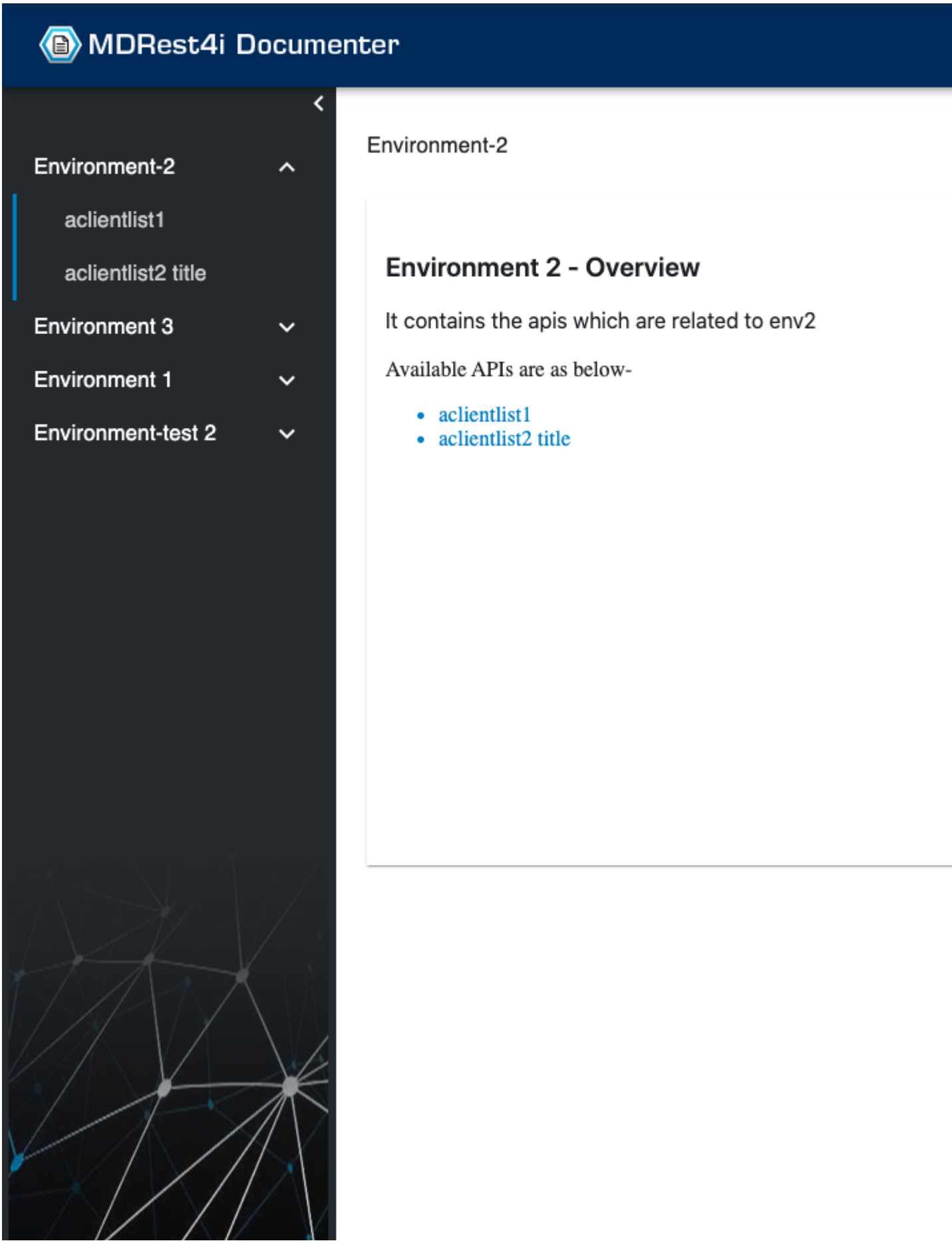
It contains the editable information about Documenter along with the list of available Environments.



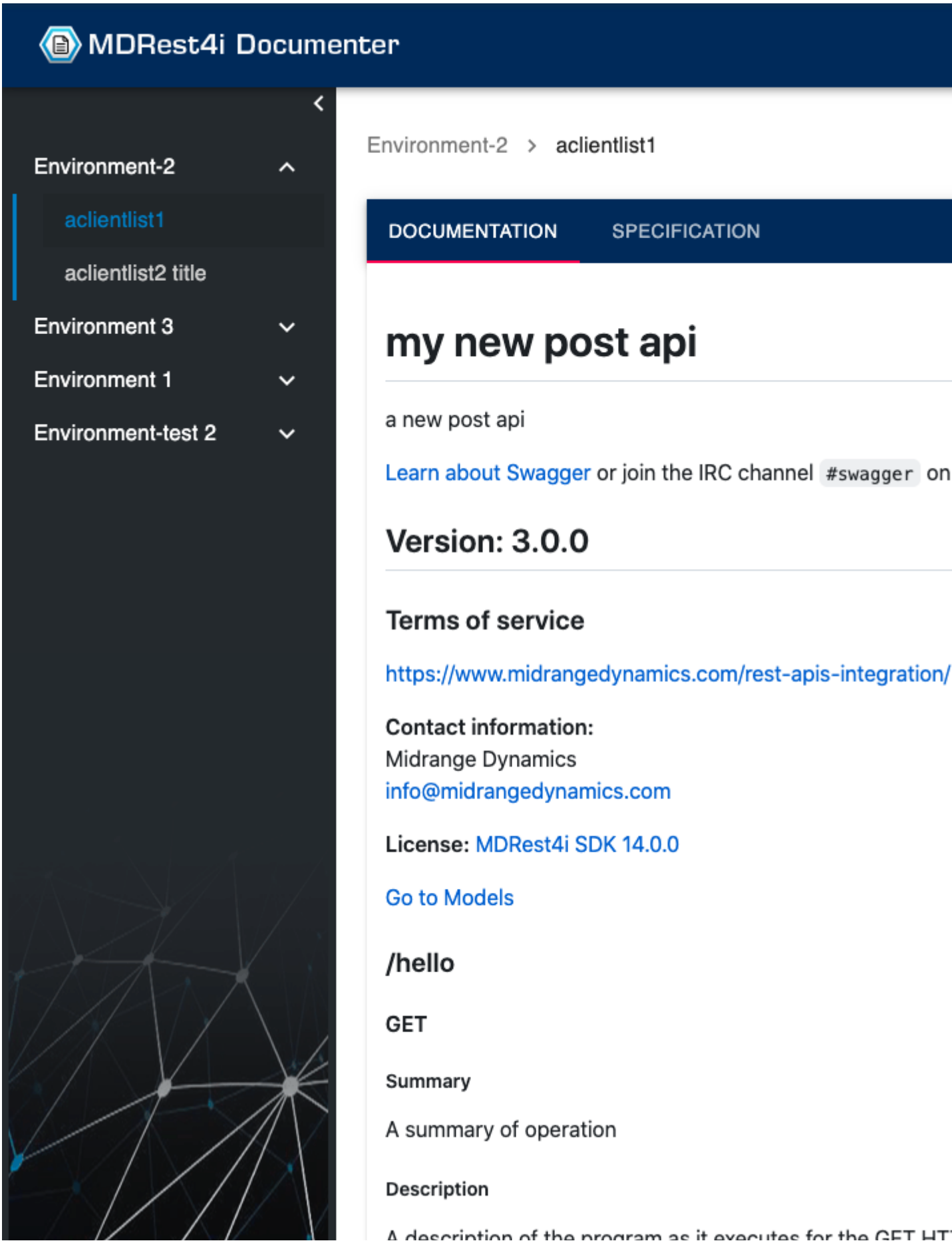
On clicking any of the Environment either from the Landing Page or from the Side menu, Overview document (Figure-2) of that particular Environment will appear containing list of the published API documents in that particular Environment.

Overview Document

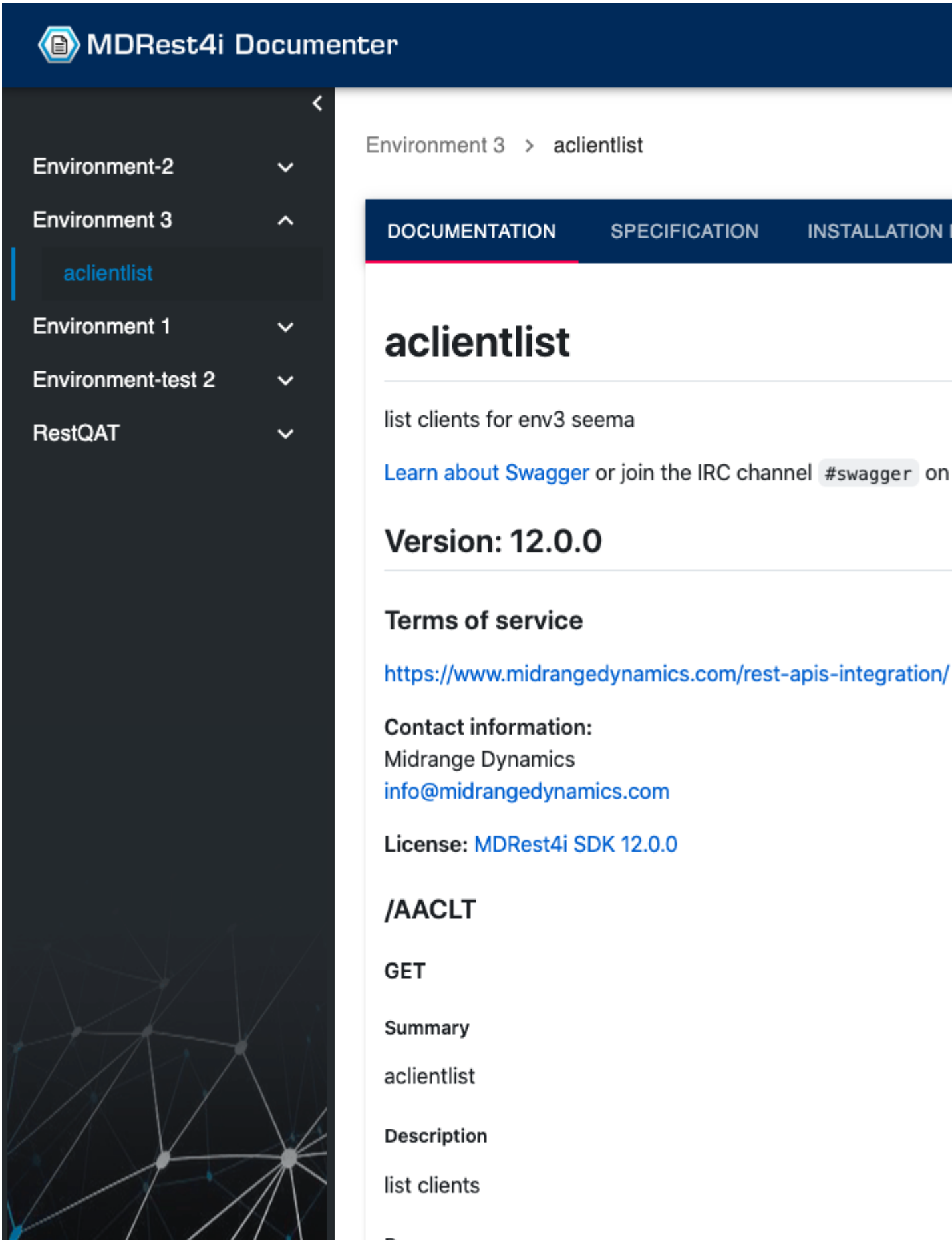
It contains the editable overview of the Environment along with the list of available APIs in it.



Further, On clicking any of the API link, it displays the information of that particular API in 2 different tabs namely **Documentation** and **Specification** for internal and external types of environment.

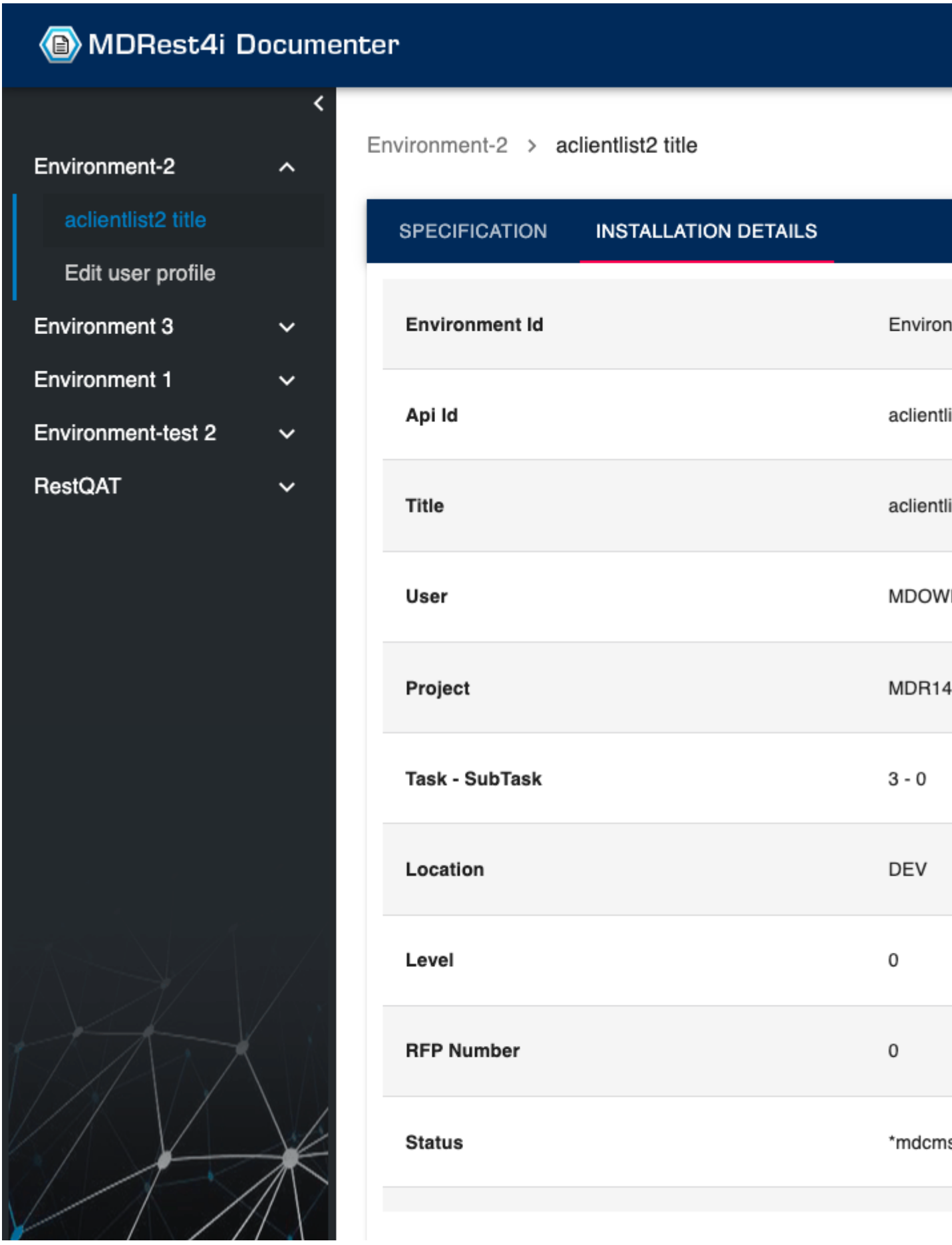


It has an additional tab named **Installtion Details**, if the selected API is a MDCMS Object Request and the API is published.



Fi

If the API is not published then it has **Installation Details** and **Specification** tabs and an icon at the top right to publish the API, provided the MDCMS switch is On.

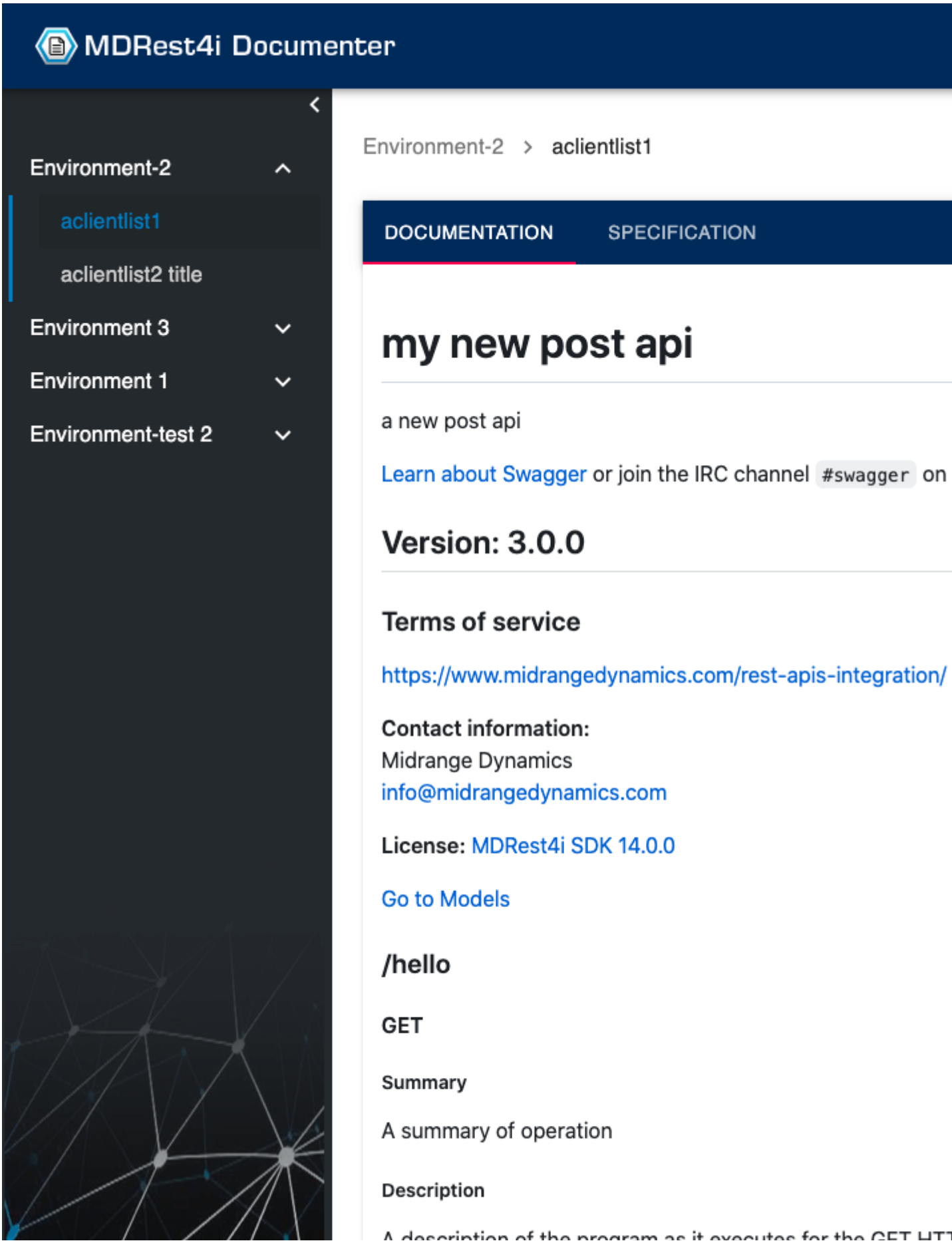


Fi	
Documentation	It displays the document which is published from MDREST4i SDK Console application
Specification	It displays the visual swagger definition of API in swagger editor
Installation Details	It displays the Installation Details of that particular API

Documentation

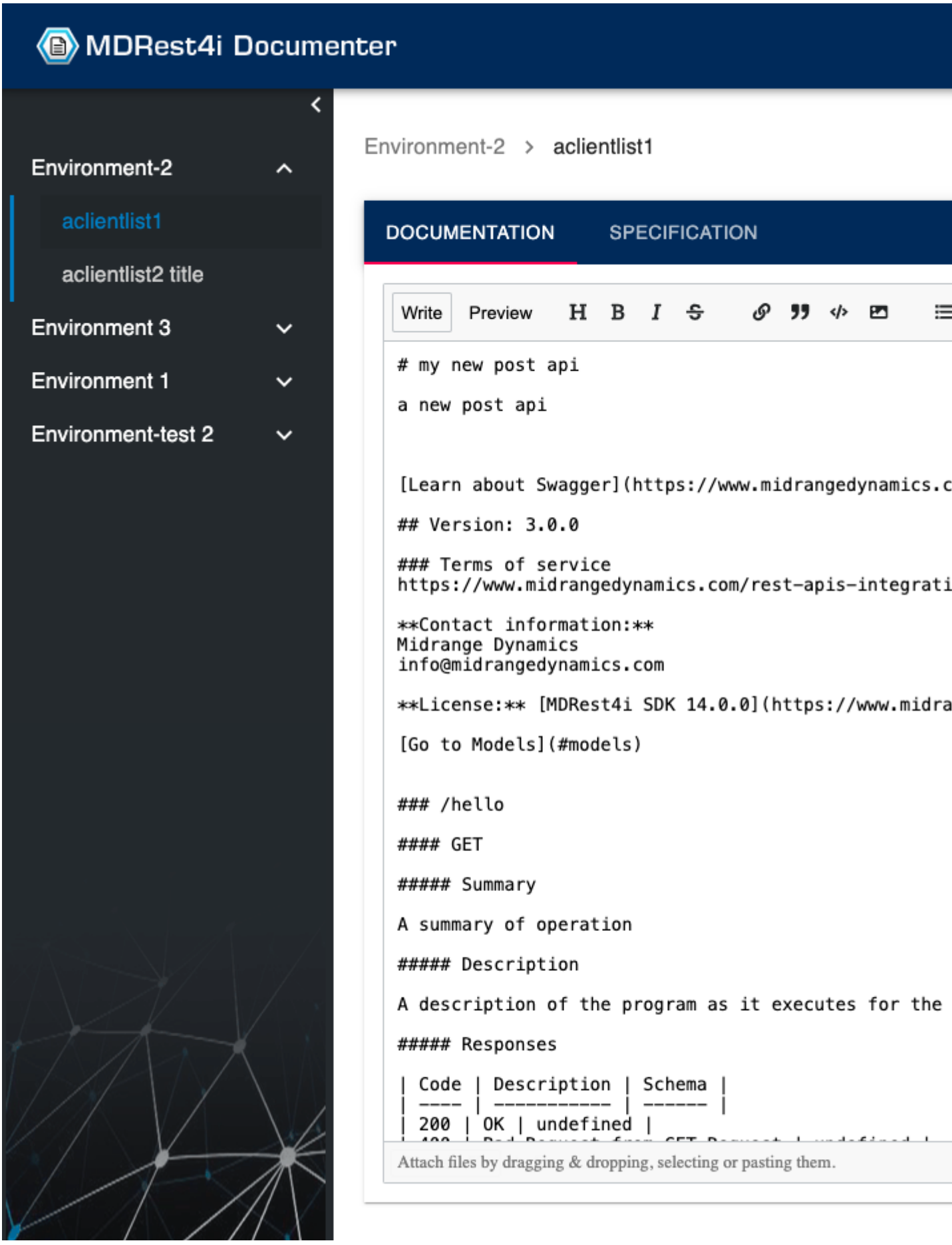
Documentation

It displays the document published from MDREST4i SDK Console application using the swagger definition.



Each document has the option to Edit it using the Edit button icon. Pressing the Edit button the document opens in the editor mode. Contents can be edited in a Showdown editor which is providing all powerful editing tools to edit as per requirement

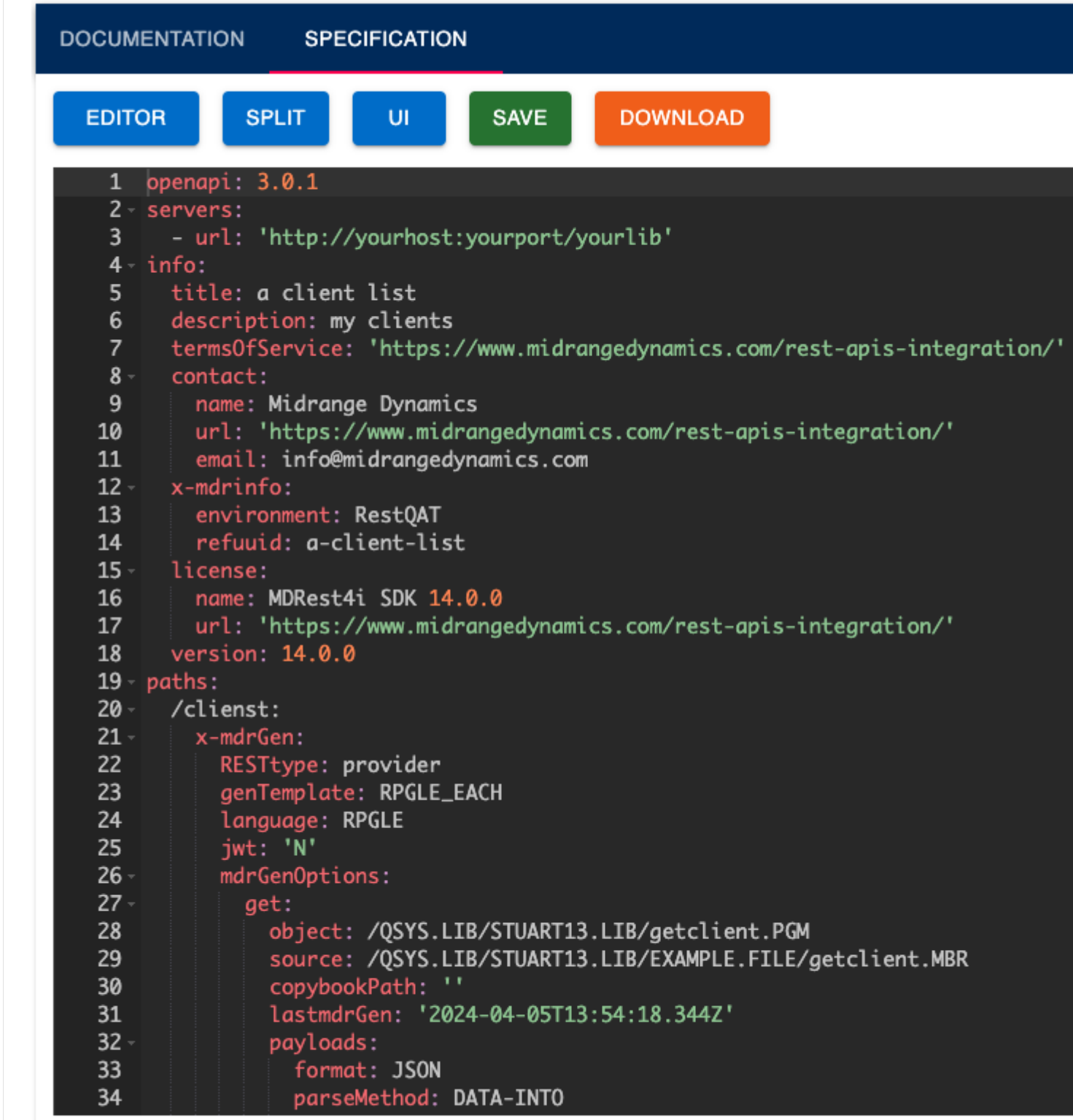
(Figure-2). Once edited it can be saved (save icon) for further use. Editing Option is allowed only to the user who has administrative privilege.



Specification

Specification

It displays the visual swagger definition of API in swagger editor with the option to edit it.



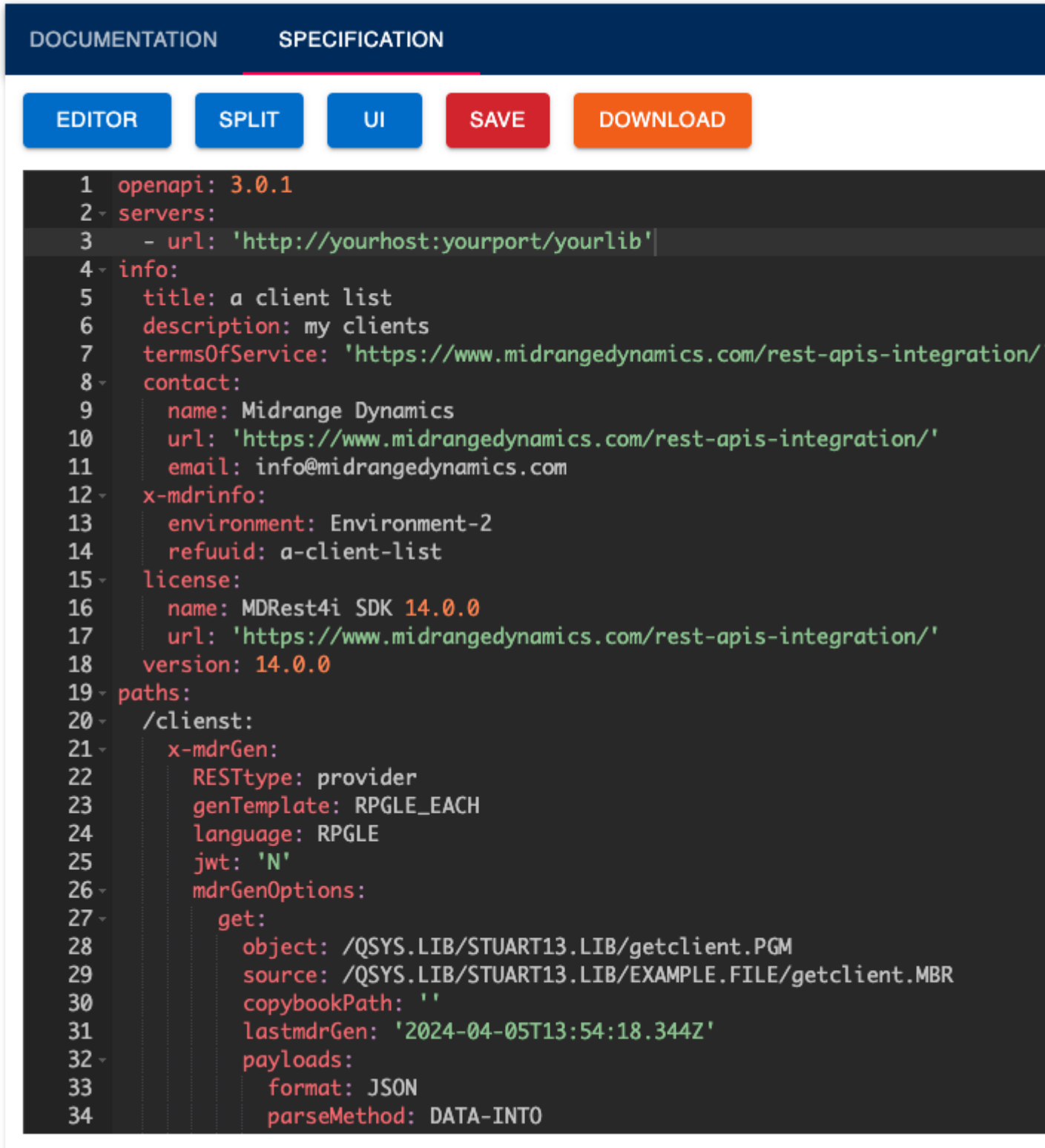
This page is split into two parts- the Editor and the UI section.

ONLY users with **Admin** or **Developer** class, can view the Editor section. Rest can only view the **UI Mode** as shown in Figure-4.

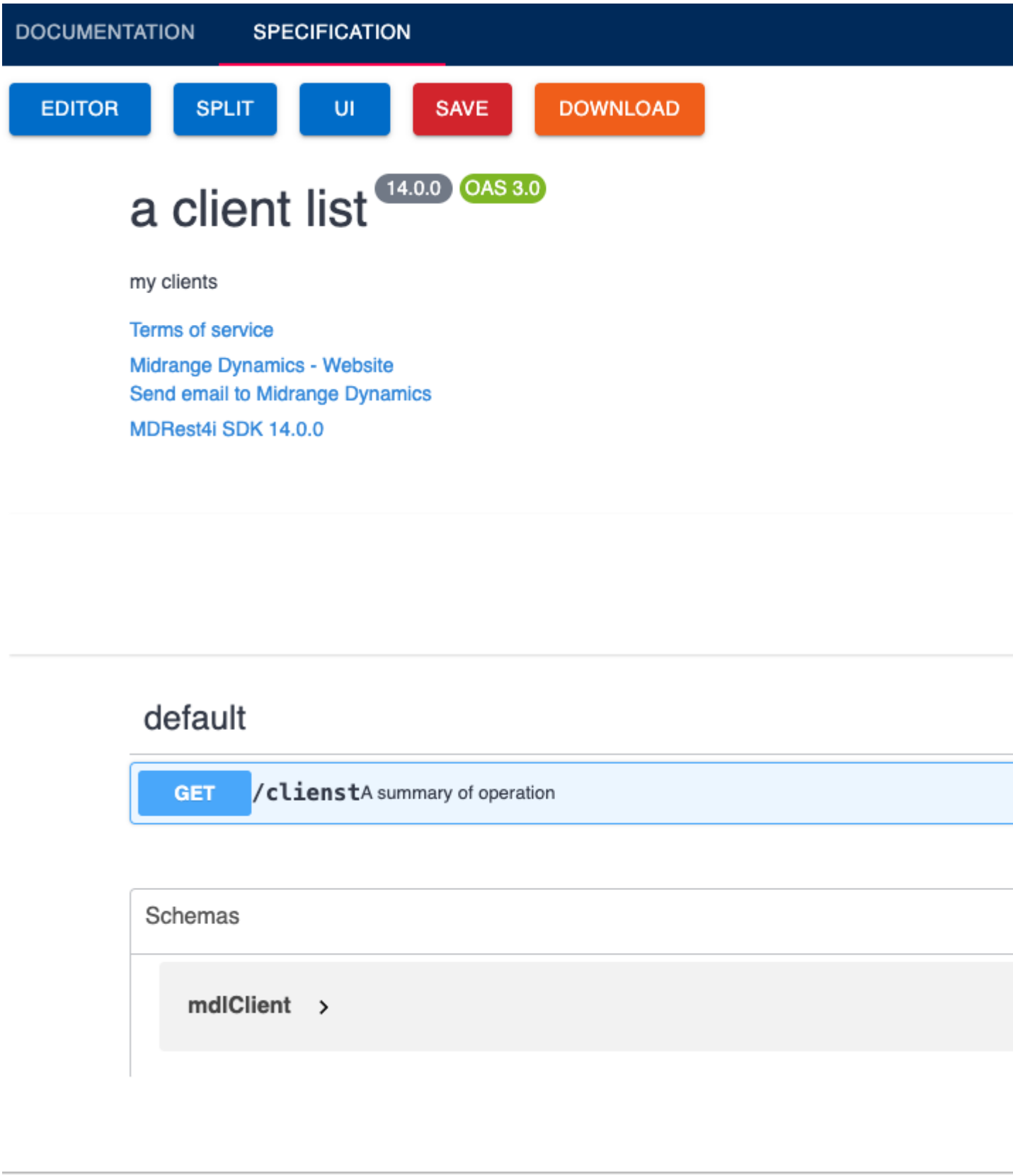
Respective buttons are provided to switch between different modes users with **Admin** or **Developer** class.

Editor	Swagger Editor is displayed in full screen (Figure-2)
Split	Swagger Editor and Swagger UI both are displayed (Figure-1)
UI	Swagger UI is displayed in full screen (Figure-3)

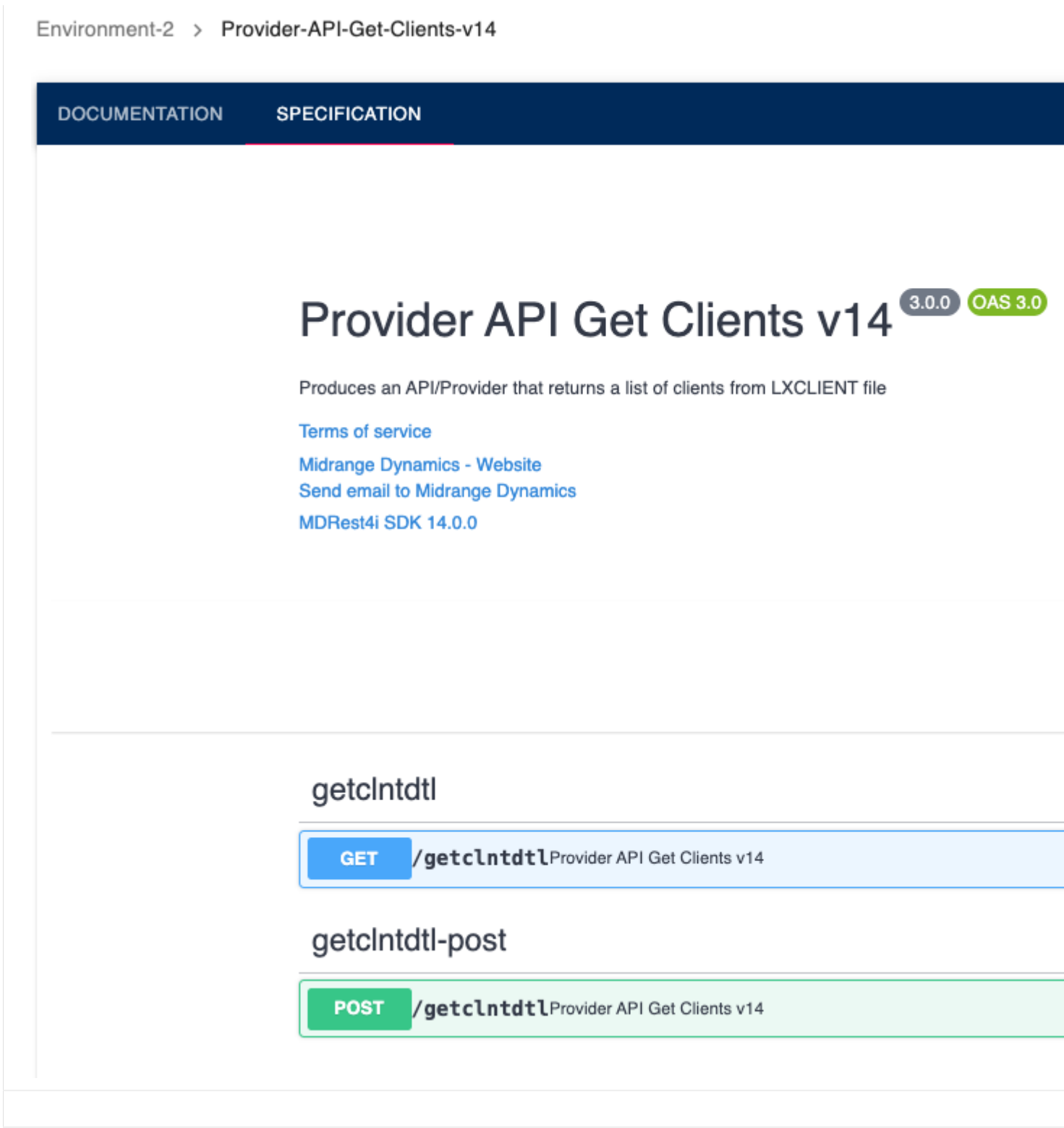
Editor Mode



UI Mode



UI Mode for User class



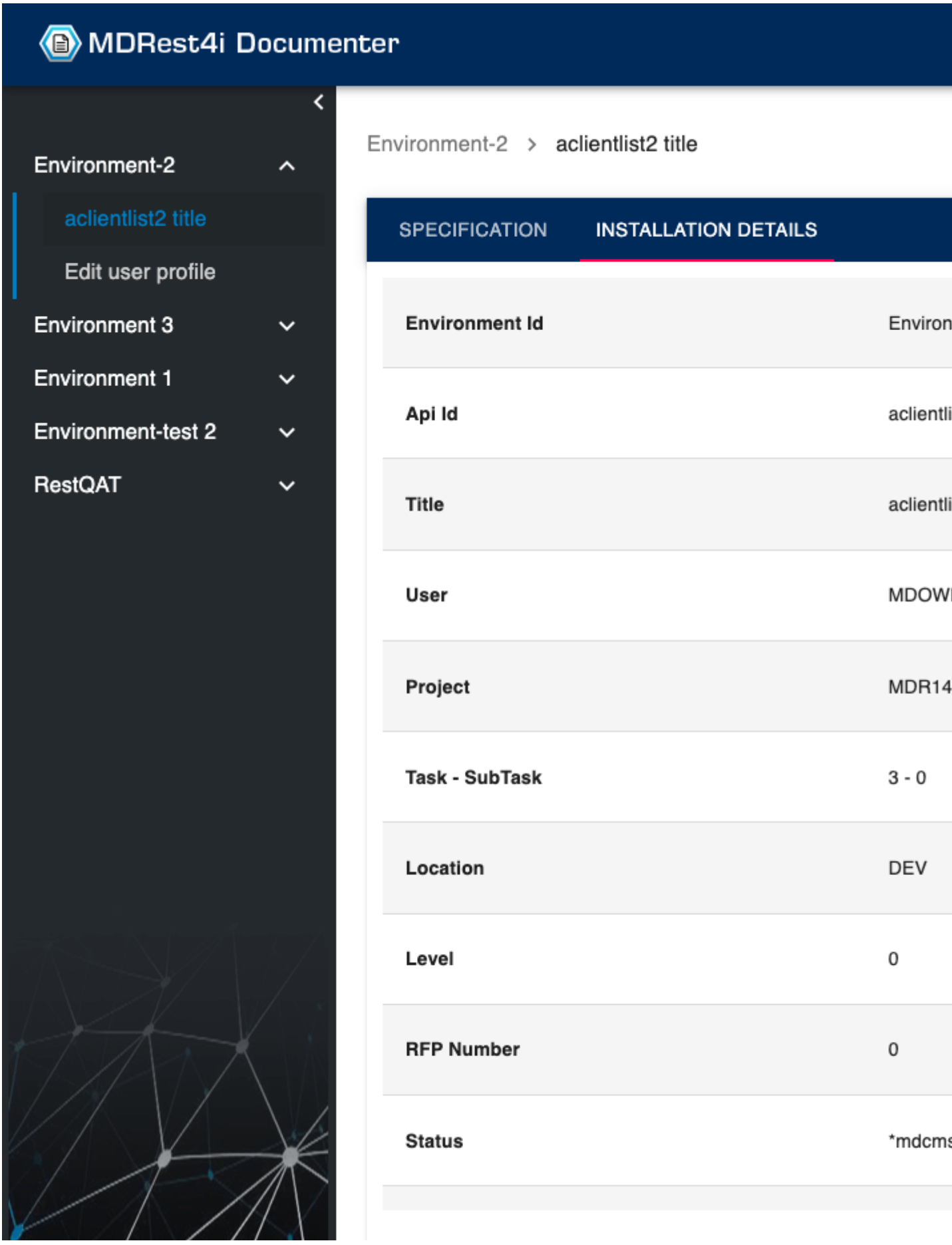
There are two additional buttons: SAVE and DOWNLOAD

Save	Appears Red if there are any unsaved changes made in the Editor, Green otherwise	Visible only to users with Admin or Developer class
Download	Used to download the swagger	Visible to all users

Installation Details

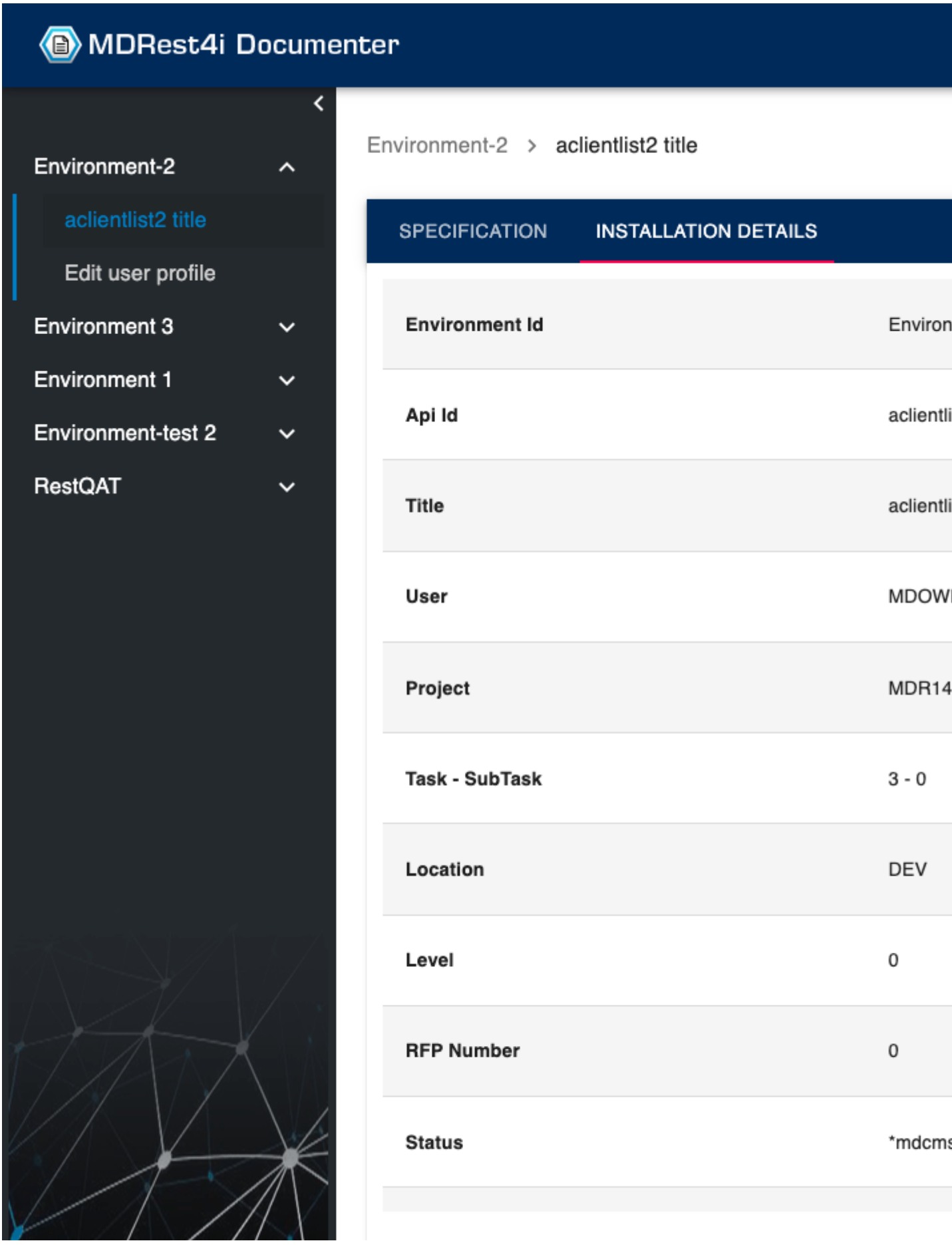
Installation Details

It displays the Installation Details of the MDCMS APIs.



Fi

If the API is not published then it has an icon at the top right to publish the API, provided the MDCMS switch is On.



Fi

Once the API is published an additional tab of documentation will appear containing the published document. (Figure-3)

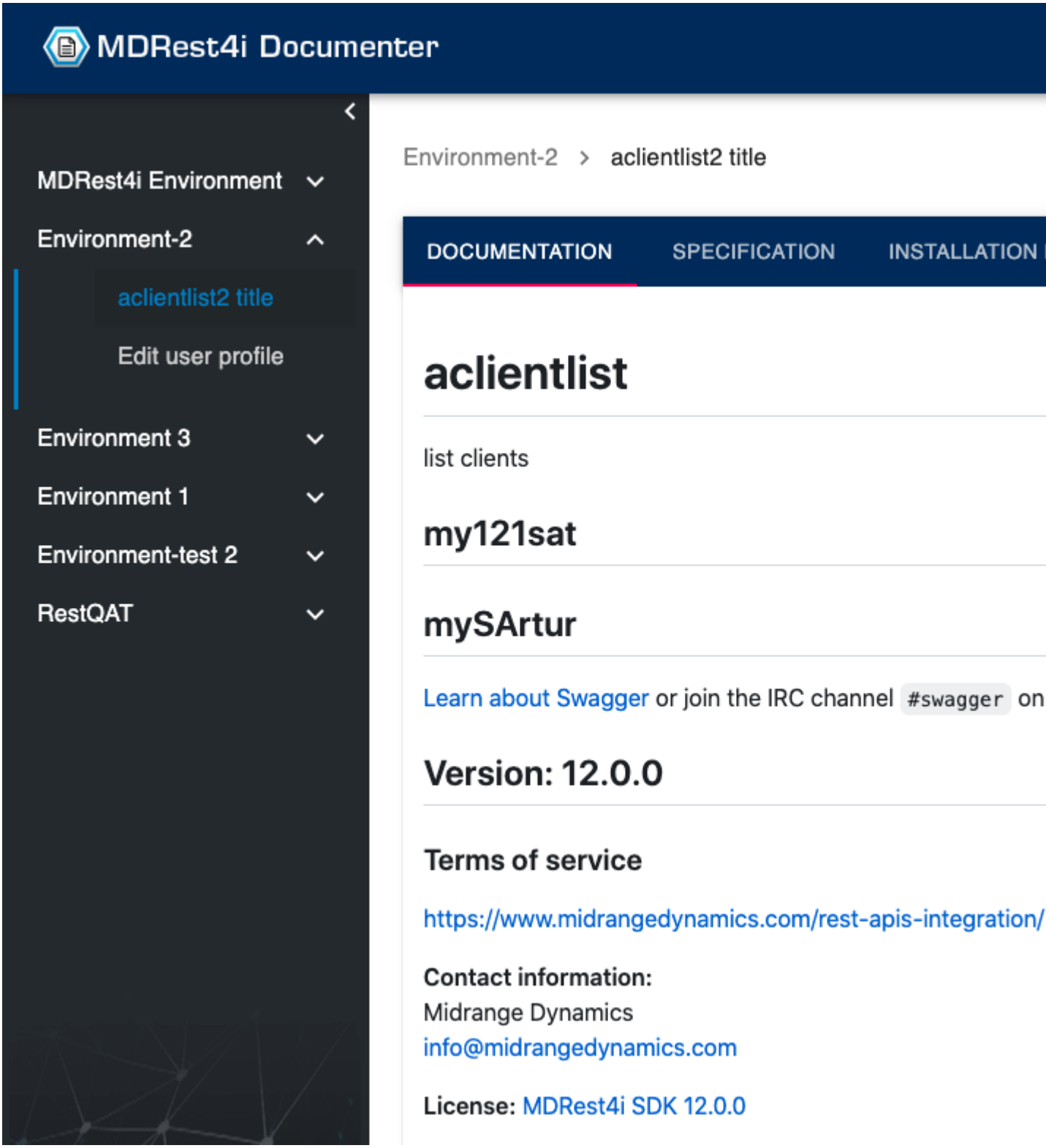


Figure-3