User & Reference Guide

MDRest4i

REST Framework & SDK
from
Midrange Dynamics

Version 12

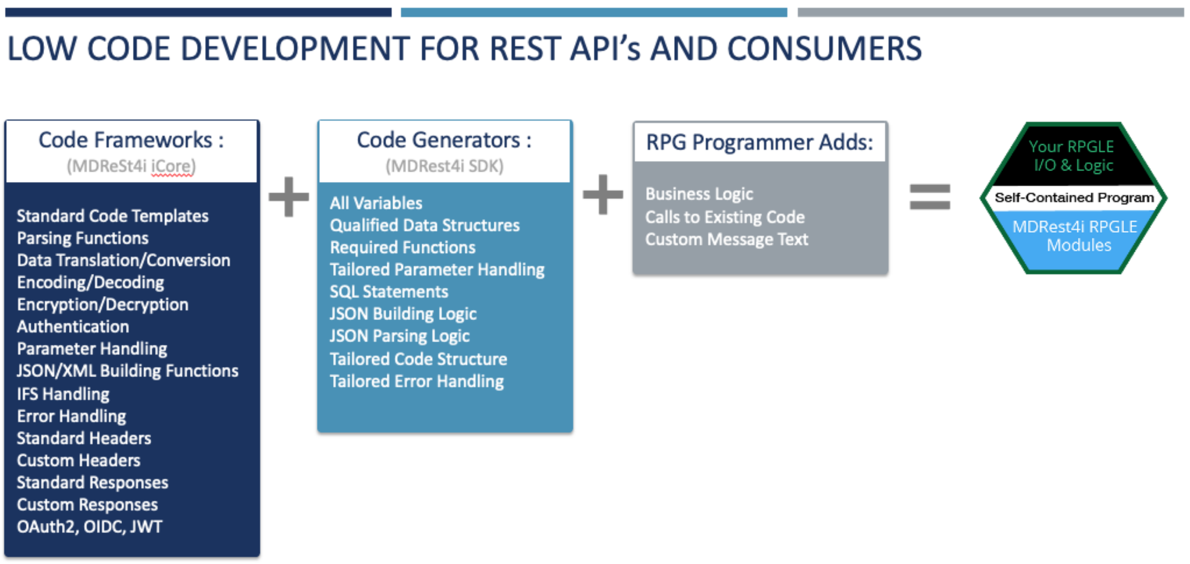
Published April 24, 2024

Table of Contents

1 Product Overview

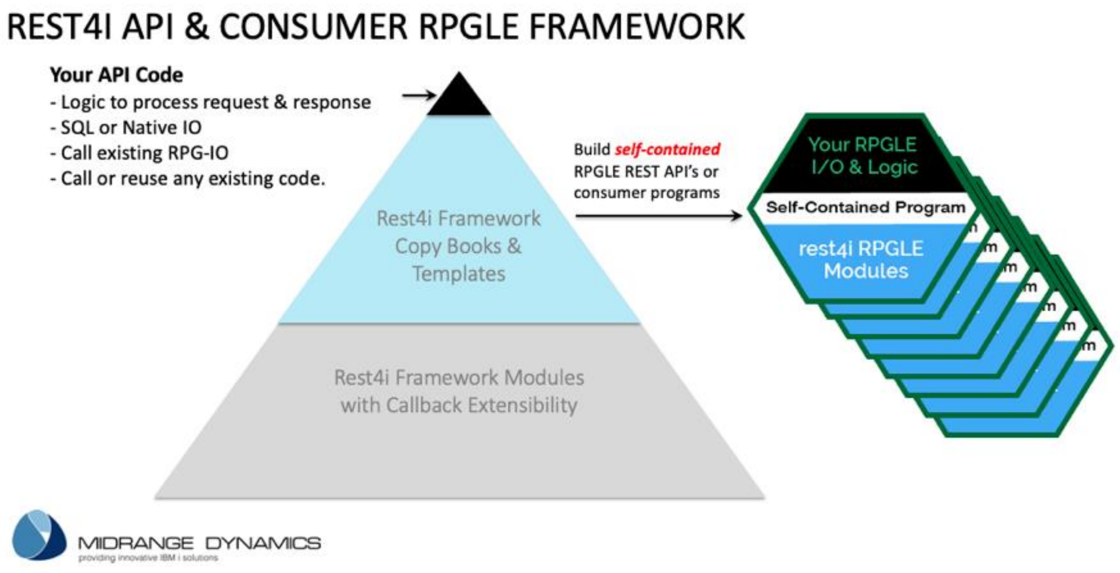
MDRest4i uses a direct communication protocol to an Apache / IBM Http Web Server enabling your IBMi to become a powerful application server using the latest industry standards.

MDRest4i is comprised of a series of RPG modules combined using ILE Binding directories by use case, into a cohesive framework, for building RPG consumer and producers for both REST and Web Services over HTTP and SOAP.



Standard SOAP and REST templates for Both Client as well as Server in RPG are provided with the framework to kick-start web services development. The Tutorial Document shows how to build a service using the supplied templates.

2 Architecture


REST4I API & CONSUMER RPGLE FRAMEWORK
Your API Code
- Logic to process request & response
- SQL or Native IO
- Call existing RPG-IO
- Call or reuse any existing code.
Rest4i Framework Copy Books & Templates
Rest4i Framework Modules with Callback Extensibility
Build self-contained RPGLE REST API's or consumer programs
Your RPGLE I/O & Logic
Self-Contained Program
rest4i RPGLE Modules
MIDRANGE DYNAMICS
providing innovative IBM i solutions

MDRest4i uses a set of interlocking engines that perform a variety of functions. These engines are implemented as Modules bound in to programs using binding directories, with prototypes exported using copybooks. Thus making the engines easier to install and use.

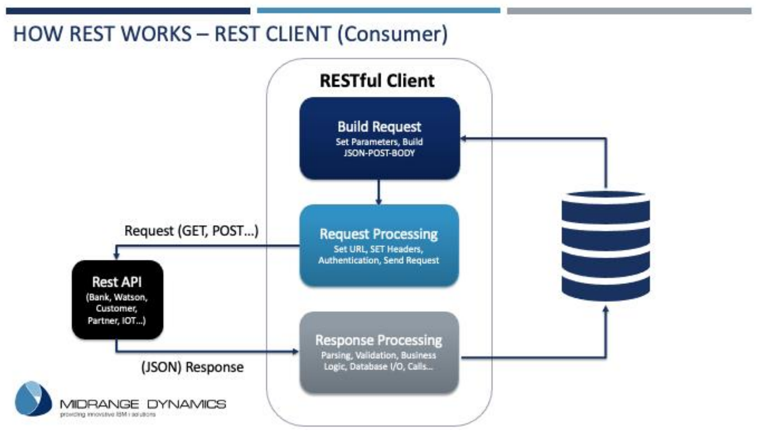
2.1 Master Templates

To simplify building of Consumers and Producers, two master templates are provided in MDRest4i. The SDK generators use these to create the programs.

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

2.1.1 Consumer (Client) Template

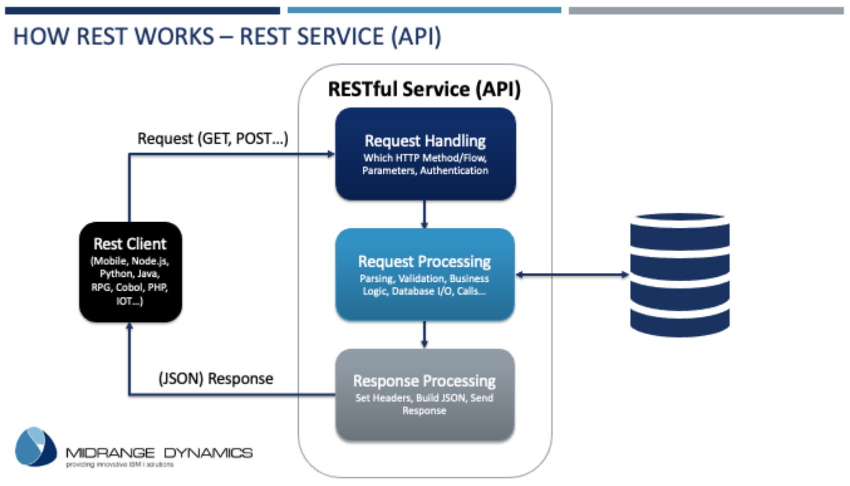
The generic flow of a REST consumer is handled by module MDRCNSM. The diagram below shows the generic flow of a REST Consumer/Client.


HOW REST WORKS – REST CLIENT (Consumer)
RESTful Client
Build Request
Set Parameters, Build JSON-POST-BODY
Request (GET, POST...)
Request Processing
Set URL, SET Headers, Authentication, Send Request
Rest API
(Bank, Watson, Customer, Partner, IOT...)
Response Processing
Parsing, Validation, Business Logic, Database I/O, Calls...
(JSON) Response
MIDRANGE DYNAMICS

For the full details of a consumer template see - REST Consumer Handling below.

2.1.2 Producer (API) Template

The generic flow of a REST producer is collectively handled by the copy books LXRRESTC, MDRESTDFN, LXRRESTP. The diagram below shows the generic flow of a REST Service.


HOW REST WORKS – REST SERVICE (API)
RESTful Service (API)
Request (GET, POST...)
Request Handling
Which HTTP Method/Flow, Parameters, Authentication
Rest Client
(Mobile, Node.js, Python, Java, RPG, Cobol, PHP, IOT...)
Request Processing
Parsing, Validation, Business Logic, Database I/O, Calls...
(JSON) Response
Response Processing
Set Headers, Build JSON, Send Response
MIDRANGE DYNAMICS

For the full details of a producer template see - REST Provider/Service Handling below.

2.2 MDRest4i Modules

MDRest4i has set of ILE modules exposed as exported procedure Interfaces – the details of which are documented in the ***MDRest4i Function/Variable Reference*** in this document.

LXRBITS	Bit Level Functions
LXRCSV	CSV Functions
LXRGLOBAL	Global Functions
LXRHTML	Common HTML Functions
LXRIFS	IFS Functions
MDRJSONY	JSON Functions
LXRLANG	Language Functions
MDRCNSM	REST Master Consumer Module
LXRXML	XML Functions

All the engines can go in the same binding directory. The details of each function contained in the MDRest4i Engines can be found in the ***MDRest4i Function Reference Guide*.**

2.3 Copybooks

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

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

2.4 MDRest4i RPGLE Header Specifications

MDREST4i uses header specifications to bind in the necessary framework modules, set activation groups, increase payload size and specify decimal separators.

2.4.1 MDRest4i System Binding Directories

This will include the MDRest4i System service program.

This will also include APIs that are used by the MDRest4i system:

- CGI APIs
- IP Socket APIs
- MDRest4i System APIs
- IBM DCM APi’s
- MDRest4i global functions
- MDRest4i database functions

One or more statements are added by the generators to include the necessary binding directories depending on the required operations in the REST service or consumer. Here are examples of a provider and consumer generated by MDREST4i:

MDRGENPRD:Provider/API

```
h bnddir('MDRUSERBND')
h bnddir('LXRGLOBAL')
```

MDRGENCNS:Consumer

```
h bnddir('LXRGLOBAL': 'BNDZIP')
```

2.4.2 Client Specific Binding Directory

This binding directory is for anything pertinent to a specific client. The empty binding directory MDRUSERBND is supplied with the product and is added to all the programs/consumer code generated via the SDK. It can be used to add the common modules and service programs. Please refer section “Custom binding directory” for more details.

2.4.3 Activation Groups

The REST service generally uses multiple modules and therefore DFTACTGRP(*NO) and ACTGRP(*NEW) are added by the SDK at the top of the program. You may change it to named or *CALLER activation group if required.

```
H dftactgrp(*no) actgrp(*new)
```

You will also see the following commented out code:

```
// h decedit(',')
```

This code when uncommented will switch the system to use a , instead of a . as the decimal separator.

2.4.4 Large Attachments and Payloads

If you are developing the REST API or consumer which is expected to receive large attachments (i.e. more than 5MB in size), you will have to add below statement in H-spec. Please refer “Downloading attachments in REST service” section in this document for further technical insight on large attachments.

```
h alloc(*teraspace)
```

3 MDRest4i SDK Generator Commands

The MDRest4i SDK Generator commands build and compile consumer and service(producers) RPGLE stubs based upon parameters provided by the user. MDRest4i iCore has two simple generators which can be invoked as commands. MDRest4i SDK web UI/SWGGER editor has more advanced generators that can be invoked as REST API’s. In both cases, once generated the developer can then add all the necessary business logic to make the service/consumer fully functional.

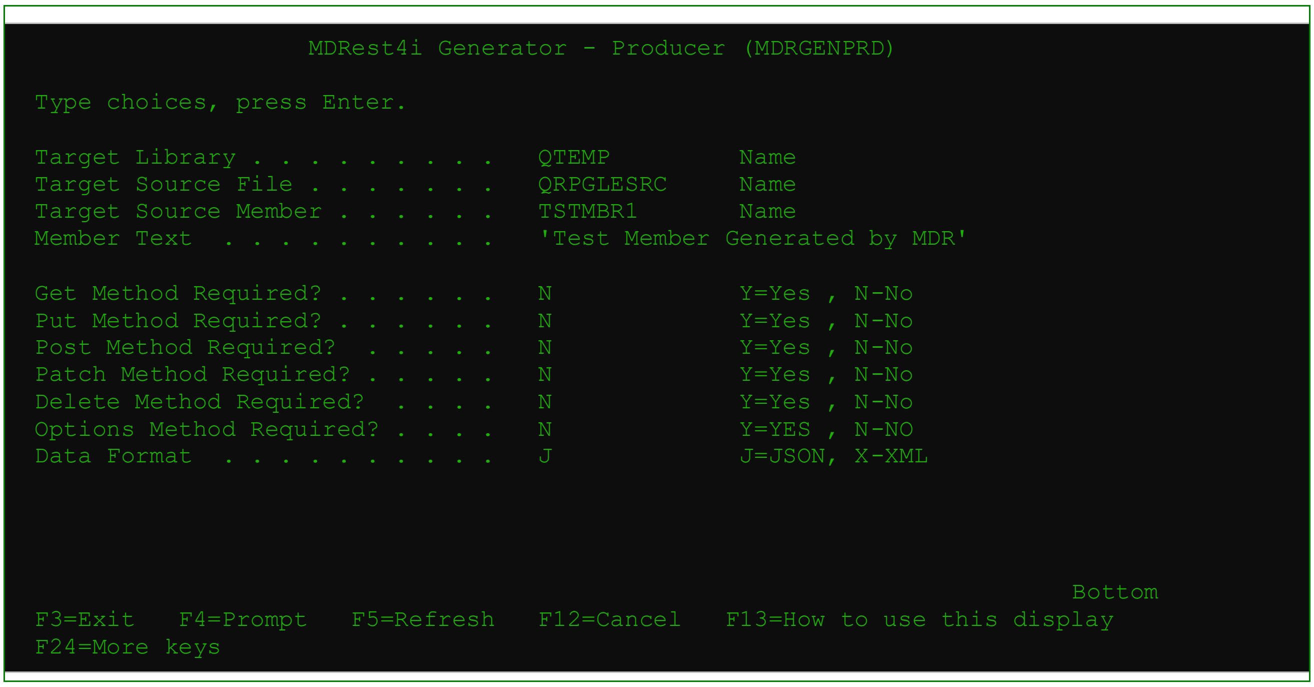
3.1 MDRGENPRD – Generate Producer Stub

From the IBMi command line execute the following:

- ADDLIBLE MDRST *& press enter*

- MDRGENPRD & press F4

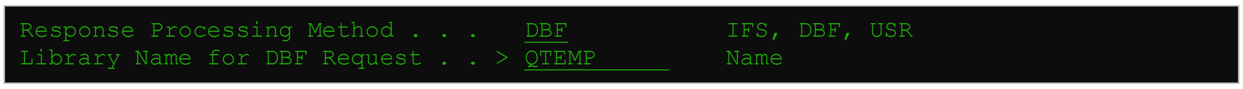
The following screen will appear:



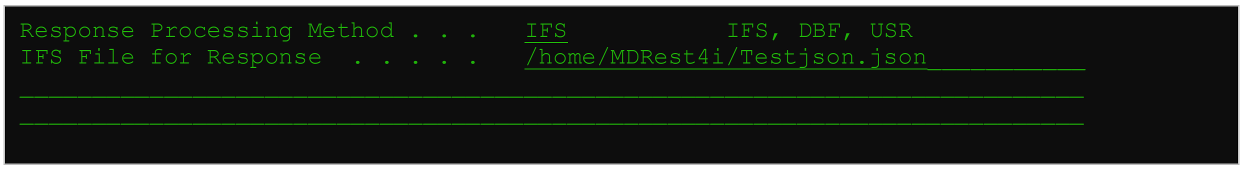
```
                     MDRest4i Generator - Producer (MDRGENPRD)

 Type choices, press Enter.

 Target Library . . . . . . . . .   QTEMP         Name
 Target Source File . . . . . . .   QRPGLESRC     Name
 Target Source Member . . . . . .   TSTMBR1       Name
 Member Text  . . . . . . . . . .   'Test Member Generated by MDR'

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



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

Figure 6 – MDRest4i Generator for Producer/Service/API

There are some additional parameters which are shown on the basis of the selected service type. If "Y" is entered on either of "Get", "Post", "Put", "Patch", "Delete", below parameter is displayed. You can use this parameter to control about how you want to send the response. The value "USR" means you have inline logic to write the response (e.g. use of addchar, adddeci, beginobject, endobject etc functions). The value "DBF" means the response to be written is pre-loaded in MDRJSONF file and the API is expected to read the data and build the JSON out of it. The value "IFS" means the response to be sent is available in JSON form in an IFS file as entered on the next parameter and the API should read that content and send the response. When "DBF" is entered, it opens another parameter where you can specify the library name for "MDRJSONF" file. Likewise, when "IFS" is entered, the new parameter is opened to enter the IFS file name with complete path.

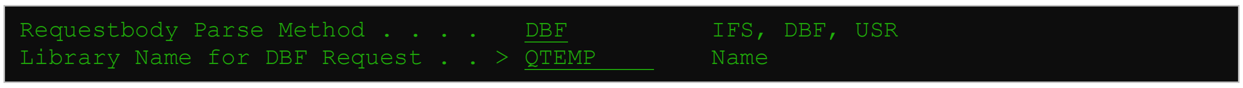


```
Response Processing Method . . .    DBF           IFS, DBF, USR
Library Name for DBF Request . . >  QTEMP         Name
```

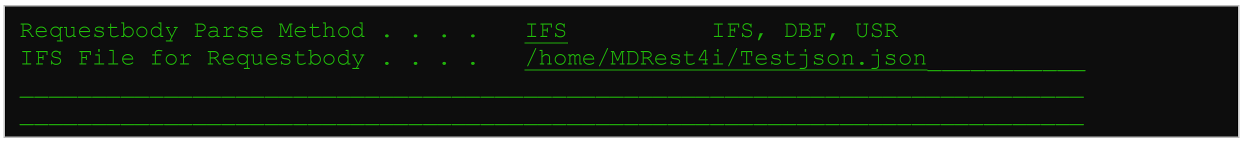


```
Response Processing Method . . .    IFS           IFS, DBF, USR
IFS File for Response  . . . . .    /home/MDRest4i/Testjson.json___________
__________________________________________________________________________
__________________________________________________________________________
```

Likewise, if "Y" is entered on the request type which take request body (i.e. either of "Post", "Put", "Patch" parameters), below parameter is displayed. This means whether you want to automatically load the response in an IFS file (i.e. value "IFS") or load the formatted content in DB file (when the value is "DBF"). The default value "USR" means continue with the standard parsing so that you can use the "JPathv", "JPathN" etc functions to find the value of specific elements. When "DBF" is entered, it opens another parameter where you can specify the library name for "MDRJSONF" file. Similarly, when "IFS" is entered, the new parameter is opened for entering the IFS file including complete path.

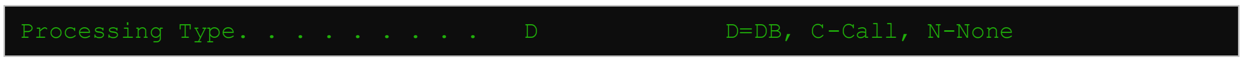


```
Requestbody Parse Method . . . .    DBF          IFS, DBF, USR
Library Name for DBF Request . . >  QTEMP        Name
```



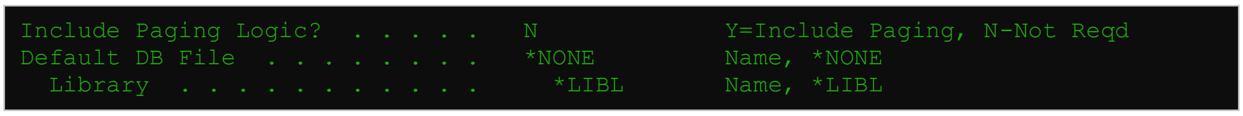
```
Requestbody Parse Method . . . .   IFS           IFS, DBF, USR
IFS File for Requestbody . . . .   /home/MDRest4i/Testjson.json__________
__________________________________________________________________________
__________________________________________________________________________
```

The parameter “Processing Type” is shown when “Y” is typed on “Get Method Required” or “Post Method Required” or “Put Method Required”. The next parameter is “Include Paging Logic” which is applicable for “GET” method when “Processing Type” is selected as “D”. When “Include Paging Logic” is selected as “Y”, the command generates the previous and next page processing logic.



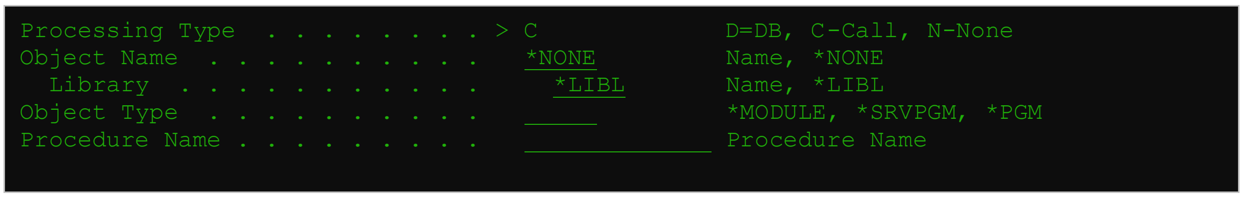
```
Processing Type. . . . . . . . .   D              D=DB, C-Call, N-None
```

If “D” is entered in “Processing Type” parameter, it will prompt for additional information. The parameter “Default DB File” and “Library” is for specifying the database file from where the information is to be sent back as the response (“GET” and/or “POST” request) or the request body will be processed for “POST”/”PUT”requests. The generator will add the relevant logic for this processing.



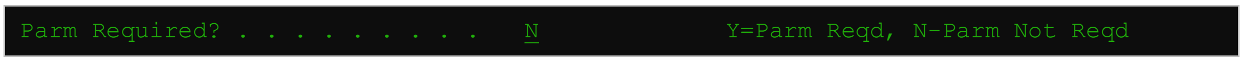
```
Include Paging Logic?  . . . . .   N              Y=Include Paging, N-Not Reqd
Default DB File  . . . . . . . .   *NONE          Name, *NONE
  Library  . . . . . . . . . . .     *LIBL        Name, *LIBL
```

If “C” is entered in “Processing Type” parameter, it means the request is to call an external program or a procedure of the specified module/service program. The additional parameters are therefore presented to enter the object name, object type and library. The parameter “Object Name” and “Library” is for specifying the name of the *MODULE, *PGM or *SRVPGM object and its library. When *MODULE or *SRVPGM is entered in “Object Type”, another parameter is displayed to enter the procedure name. The generator will add the relevant logic for this processing.



```
Processing Type  . . . . . . . . > C              D=DB, C-Call, N-None
Object Name  . . . . . . . . . .   *NONE          Name, *NONE
  Library  . . . . . . . . . . .     *LIBL        Name, *LIBL
Object Type  . . . . . . . . . .   _____          *MODULE, *SRVPGM, *PGM
Procedure Name . . . . . . . . .   _____________  Procedure Name
```

The next entry is “Parm Required?”. When you select “Processing Type” as “N” or “D”, it gets displayed like below and it only asks whether the query parameters are required in REST service or not.



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

However, when you enter “Processing Type” as “N”, it gets displayed like below.

```
Processing Type  . . . . . . . . > N              D=DB, C-Call, N-None
Parm Required? . . . . . . . . .   N              P=Call, Q-Query & Call, N-None
```

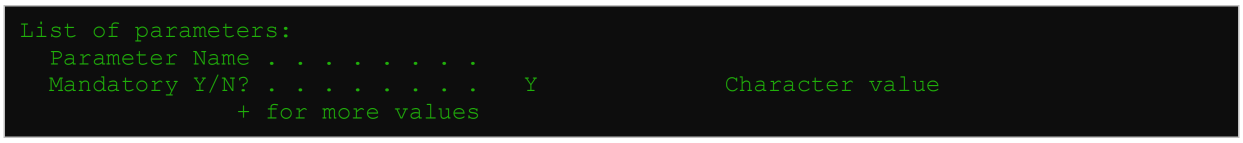
If you choose “N”, means query parameters are not required, when you select “P” it would assume you directly want to call the program with the value received from query parameter and therefore it will declare "p_" prefixed variable name for the entered parameter. The code is generated like below

```
  // Write query parameter definitions
d  p_cuid         s               5A
d  n_cuid         s                n


   // ==================================================================
   //    Override Get Method
   // ==================================================================
   Begsr z_ProcGET;

     custmnt1(p_cuid);

     Exsr Z_PrcSndRsp;

   Endsr;
```

When you enter “Q” as the value on “Parameter Required?”, it considers that you want to perform some intermediate calculation on the query parameter received and therefore, it declares another variable with “w_” prefix and assigns the value of “P_” prefixed before the call and does the reverse after the call. Below is the code generated for this type of request:

```
d  w_cuid         s               5A

  // Write query parameter definitions
d  p_cuid         s               5A
d  n_cuid         s                n

   // ==================================================================
   //    Override Get Method
   // ==================================================================
   Begsr z_ProcGET;

     w_cuid = p_cuid;
     custmnt1(w_cuid);
     p_cuid = w_cuid;

     Exsr Z_PrcSndRsp;

   Endsr;
   // ==================================================================
```

If “Processing Type” is “D” and “Parameter Required?” is set to “Y”, it asks you to enter the parameter names. Here, you have to only enter the parameter name which should be a field from DB field and specify Y/N on “Mandatory Y/N” to mark create a program with mandatory or optional query parameter. You can enter multiple parameters. When the API will be generated, it will have entered parameters as the mandatory or optional query parameters.



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

If "Processing Type" is "N" and "Parameter Required?" is set to "Y" or "Processing Type" is "C" and "Parameter Required?" is set to "P" or "Q", it asks you to enter the parameter names along with their data types. The parameter name and Mandatory entries are same as previous type but here, you have to also enter the data types of the parameters since there is reference from database.

Parameter	Description
`Target Library`	Where the source file/member and compiled program will be placed by the generator
`Target Source File`	The source file where the program source will be saved
`Target Source Member`	Name of the program source member and the name of the generated program object
`Member Text`	Description assigned to the source and object text

`Get Method Required?`
`Put Method Required?`
`Post Method Required?`
`Patch Method Required?`
`Delete Method Required?`
`Options Method Required?`
	Y will insert a subroutine for the selected HTTP method in the API/Producer

`Data Format`	J – for JSON format and X - for XML format, default is J
`Requestbody Parse Method/ Library Name for DBF Request`	This parameter controls on how you want to process the response. The value "IFS" means load the response in an IFS file, the value "DBF" loads the formatted content in DB file. The default value "USR" means continue with the standard parsing so that you can use the "JPathv", "JPathN" etc functions to find the value of specific elements. *When this entry is "DBF", you can specify the library name in the next parameter.*
`Response Processing Method/ Library Name for DBF Request`	This parameter controls about how to send the response. The value "USR" means response is being sent via inline logic to write the response (e.g. use of addchar, adddeci, beginobject, endobject etc functions). The value "DBF" means the response to be written is pre-loaded in MDRJSONF file and the API is expected to read the data and build the JSON out of it. The value "IFS" means the response to be sent is available in JSON form in an IFS file and you want to read that content and send

	the response. *When this entry is "DBF", you can specify the library name in the next parameter.*
`Processing Type`	This is applicable for GET, PUT and POST requests. You can specify whether the target API is expected to perform DB I/O on specified file or it would call a program/procedure. The logic will accordingly be written in the generated member.
`Include Paging Logic?`	This option writes the logic in the necessary subroutines dedicated to process the next and previous page request. Please refer "MDRest4i Paging implementation" section in this document for more details.

`Default DB File`
 	
`Library`
	This parameter is prompted when "Processing Type" is requested as "D". The name of database file from where the API should pick the fields for the output
	
	Library where DB file exists

`Object Name`
 	
`Library`
 	
`Object Type`
 	
`Procedure Name`
	This parameter is prompted when "Processing Type" is requested as "C". The name of Program/module or procedure can be entered.
	
	Library where *PGM or *MODULE or *SRVPGM object exists
	
	Object Type (i.e. *MODULE, *SRVPGM, *PGM)
	
	Name of the procedure for *MODULE or *SRVPGM

`Parm Required?`
 	
`List of parameters:`
`Parameter Name`
`Mandatory Y/N?`
 	
`Length`
`Data Type`
`Decimal Positions`
	Prompts the user if the query parameters are required in the API.
	
	If Parm Required has been selected as "Y", below subroutines are added to the API with the standard logic and the developer can make the necessary adjustments according to the requirement:
	
	If the Processing type is not "D" and "Parm Required?" is selected some value other than "N", the length, data type and decimal position entries (i.e. attributes of the specific parameter are prompted so that the parameter can be defined accordingly in the generated program.
	
	z_setMethod: This subroutine sends the error when required parameters are not received.

	z_setParms: This subroutine is to set the parameter names (case sensitive) and mark them mandatory/optional. z_evalParms: This subroutine extracts the parameters from the query in url and sets them in standard parameters array. z_checkParms: This subroutine sets the parameter values in "P_" prefixed parameter variable names and sets the corresponding "n_" prefixed variable set to *On (i.e. we can check this indicator later if the parameter has been received or not). The developer can write his own logic of parameter handling if required. The last set "List of parameters" is to provide the parameter name and whether the parameter is mandatory or optional. The appropriate code would be added in the generated service to process and validate these parameters. One point to be noted is that, if the value is populated for "Default DB File", the parameters must come from the file. They could be short field names or long field names driven by the field text or column heading (if field text is not found)

3.1.1 Custom copybook logic for specific subroutines (z_ProcGet etc):

There may be a requirement to provide a fixed set of statements in the different subroutines being generated using this command. An empty copybook per generated subroutine has been provided in QRPGLESRC source file under MDRST for this purpose.

```
MDRCPYGET  - Contains the statements which will be included in z_ProcGet subroutine
MDRCPYOPT  - Contains the statements which will be included in z_ProcOptions subroutine
MDRCPYPCH  - Contains the statements which will be included in z_ProcPatch subroutine
MDRCPYPOST - Contains the statements which will be included in z_ProcPost subroutine
MDRCPYPUT  - Contains the statements which will be included in z_ProcPut subroutine
```

The statements of each member above will then be copied by the MDREST4i generators (not the compiler) into the relevant subroutines from (*LIBL/QRPGLESRC) when API/Producers are generated either by **MDRGENRPD** or MDREST4i provider API - **MDRGENXAPI**.

It is advised therefore that a copy of these is made into a custom library which is then added above MDRST in the *LIBL whenever MDRGENRPD or MDRGENXAPI are used to build an API.

3.1.2 Custom copybook logic for generated Providers

Copy book *LIBL/MDRUSERCPY is added by **MDRGENRPD** or MDREST4i provider API – **MDRGENXAPI** when a provider is generated. An empty copybook MDRST/QRPGLESRC.MDRUSERCPY has been provided in source file under MDRST for this purpose.

3.1.3 Custom binding directory

If you would like to have all the generated APIs include some specific service programs or modules, MDRGENPRD command and MDRGENXAPI interface adds below statement in all the generated members.

```
h bnddir('MDRUSERBND')
```

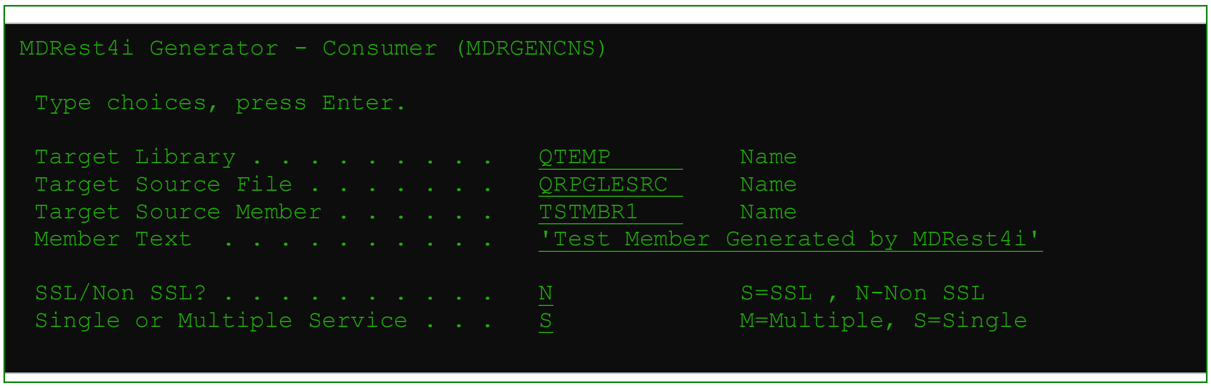
The empty binding directory “MDRUSERBND” is provided in MDRST Library. You can copy this binding directory in custom library kept above MDRST library and add the necessary modules and services programs.

3.2 MDRGENCNS – Generate Consumer Stub

From the IBMi command line execute the following:

- ADDLIBLE MDRST *& press enter*
- MDRGENCNS & press F4

The following screen will appear:



```
MDRest4i Generator - Consumer (MDRGENCNS)

 Type choices, press Enter.

 Target Library . . . . . . . . .   QTEMP         Name
 Target Source File . . . . . . .   QRPGLESRC     Name
 Target Source Member . . . . . .   TSTMBR1       Name
 Member Text  . . . . . . . . . .   'Test Member Generated by MDRest4i'

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

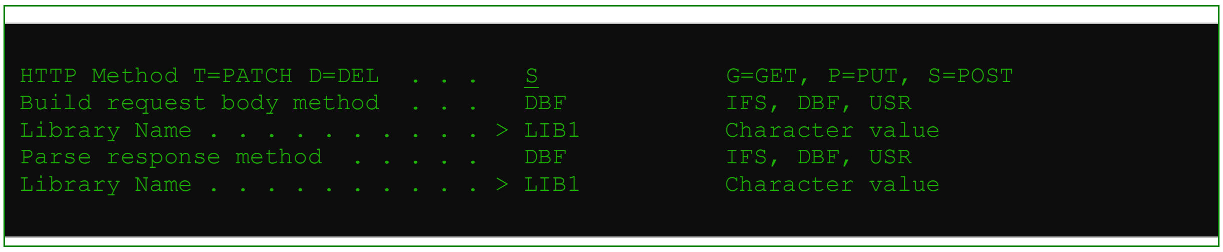
Figure 3 – MDRest4i Consumer Generator MDRGENCNS Screen 1

This example using the default values will build a RESTful consumer/client program in RPG.

Parameter	Description
`Target Library`	Where the source and compiled program will be placed by the generator
`Target Source File`	The source file where the program source will be saved
`Target Source Member`	Name of the program source member and the name of the generated program object
`Member Text`	Description assigned to the source and object text
`SSL/Non SSL`	***N*** to generate a non-secure consumer ***Y*** to copy on the appropriate SSL modules for handling encryption and SSL
`Single or Multiple Service`	***S*** for a program that contains a single request/call to an API/Service ***M*** for a program that is able top call multiple API’s/Services in a single program

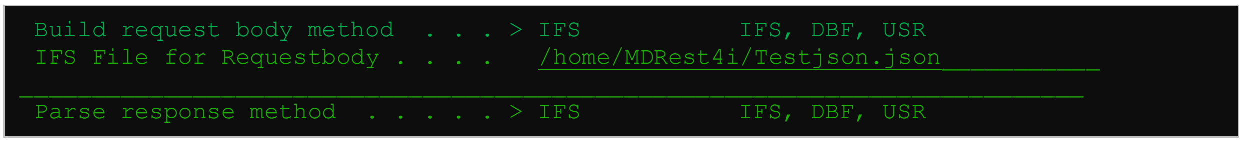
`Build request body method/ Library name`	***IFS*** *for reading the JSON content available in an IFS file and then send it as the request body. When "IFS' specified we need to specify the complete IFS path (including file name) in the next parameter.* ***DBF*** *for sending the content from the DB file "MDRJSONF" and build the JSON. When this entry is "DBF", you can specify the library name in the next parameter.* ***USR*** *means the request body will be created using addChar, beginObject etc functions manually by the developer.*
`Parse response method / Library name`	***IFS*** *for writing the request body received in JSON form to an IFS file. When "IFS" is specified, we need to provide complete IFS path (including file name) in the next parameter.* ***DBF*** *for loading the JSON request body in DB file "MDRJSONF". When this entry is "DBF", you can specify the library name in the next parameter.* ***USR*** *would cause default JSON parsing to happen automatically and the developer can use JPathv, JPathN etc functions to load specific JSON elements.*

- *Press enter*



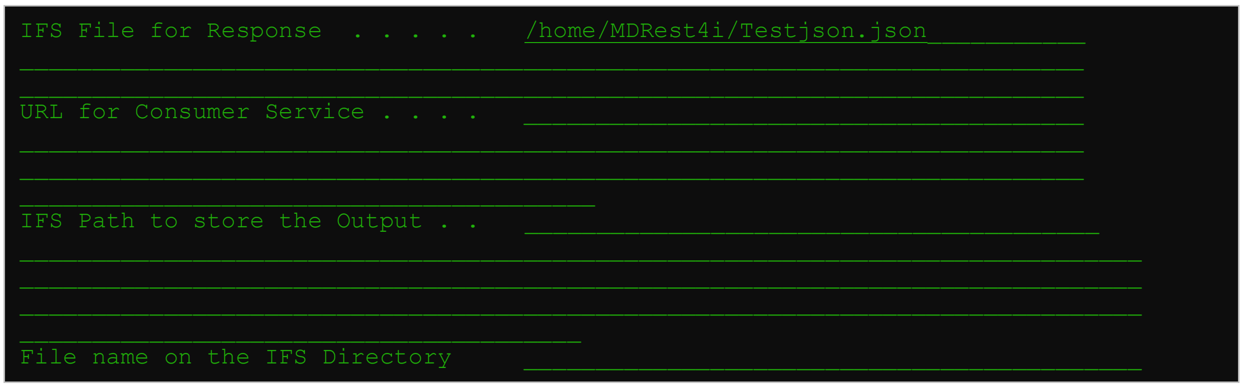
```
HTTP Method T=PATCH D=DEL  . . .   S              G=GET, P=PUT, S=POST
Build request body method  . . .   DBF            IFS, DBF, USR
Library Name . . . . . . . . . . > LIB1           Character value
Parse response method  . . . . .   DBF            IFS, DBF, USR
Library Name . . . . . . . . . . > LIB1           Character value
```

Figure 4 – MDRest4i Consumer Generator MDRGENCNS Screen 2



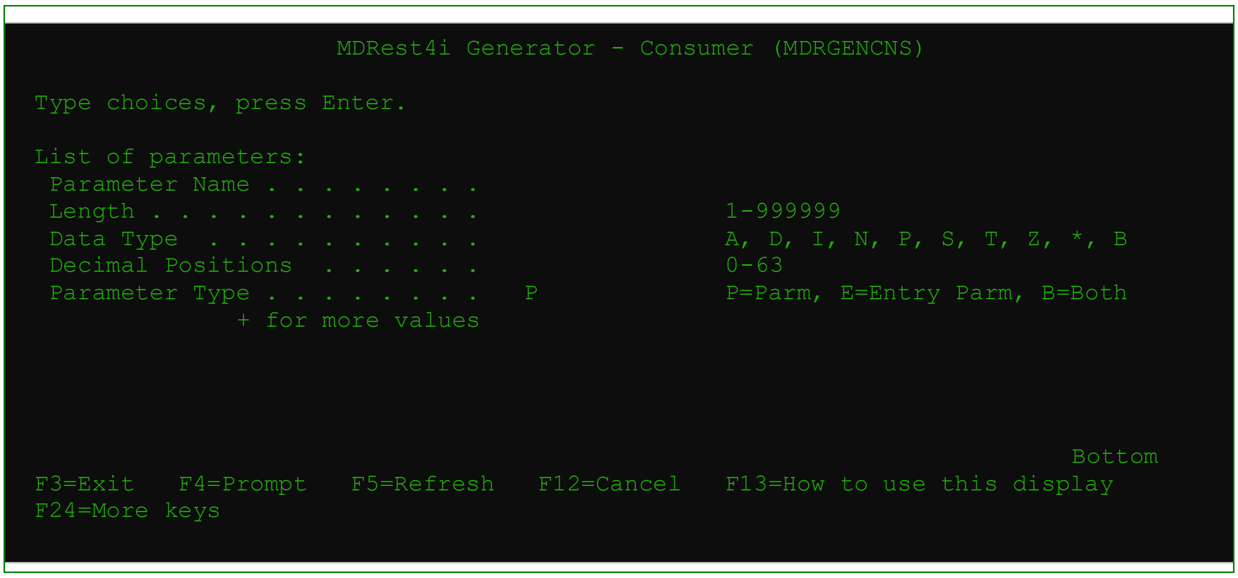
```
 Build request body method  . . . > IFS            IFS, DBF, USR
 IFS File for Requestbody . . . .   /home/MDRest4i/Testjson.json__________
____________________________________________________________________________
 Parse response method  . . . . . > IFS            IFS, DBF, USR
```

1. Page down to see the remaining parameters.



```
IFS File for Response  . . . . .   /home/MDRest4i/Testjson.json__________
____________________________________________________________________________
____________________________________________________________________________
URL for Consumer Service . . . .   ________________________________________
____________________________________________________________________________
____________________________________________________________________________
_________________________________________
IFS Path to store the Output . .   _________________________________________
________________________________________________________________________________
________________________________________________________________________________
________________________________________________________________________________
________________________________________
File name on the IFS Directory     ____________________________________________
```

- Page down to see the remaining parameters.



```
                     MDRest4i Generator - Consumer (MDRGENCNS)

 Type choices, press Enter.

 List of parameters:
  Parameter Name . . . . . . . .
  Length . . . . . . . . . . . .                 1-999999
  Data Type  . . . . . . . . . .                 A, D, I, N, P, S, T, Z, *, B
  Decimal Positions  . . . . . .                 0-63
  Parameter Type . . . . . . . .    P            P=Parm, E=Entry Parm, B=Both
              + for more values




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

Figure 5 – MDRest4i Consumer Generator MDRGENCNS Screen 3

Parameter	Description
`HTTP Method`	**G** for HTTP GET, **P** for HTTP PUT, **S** for HTTP POST, **T** for HTTP PATCH and **D** for HTTP DELETE
`Build request body method`	Request body method (IFS, DBF, USR)
`Library Name`	Library name if requested method is "DBF".
`IFS File for Requestbody`	IFS file name if request method is 'IFS"
`Parse response method`	Response parse method (IFS, DBF, USR)
`IFS File for Response`	IFS file name if reesponse method is 'IFS".
`Library Name`	Library name response method is "DBF"
`URL for Consumer Service`	THE URL provided by the provider
`IFS Path to store the Output`	Output to save log file from transaction
`File name…`	Name of the Log file
`List of Parameters`	Parameters and their values. Use a + to add more parameters up to 20 Parameters are available in the generator command.

The generated code can be viewed in the LIBRARY/FILE specified in the command.

4 MDRest4i SDK Generator REST APIs

The MDRest4i SDK Generators build and compile consumer and service(producers) RPGLE stubs based upon SWAGGER definitions submitted to the API's using a POST method, with the SWAGGER definitions in the request Body. Once generated the developer can then add all the necessary business logic to make the service/consumer fully functional.

The SWAGGER definitions can be edited at a very fine-grained level of detail in areas such as:

- Using response field names that are different from column names(less cryptic) of the underlying database file,
- Selecting a subset of fields from a database file

The SWAGGER definitions can be edited in the MDRest4i SDK Console UI, SWAGGERHub, SWAGGER.Editor, and submitted via any POST capable UI such as SOAP-UI, POSTMAN, and off course MDRest4i SDK Console itself.

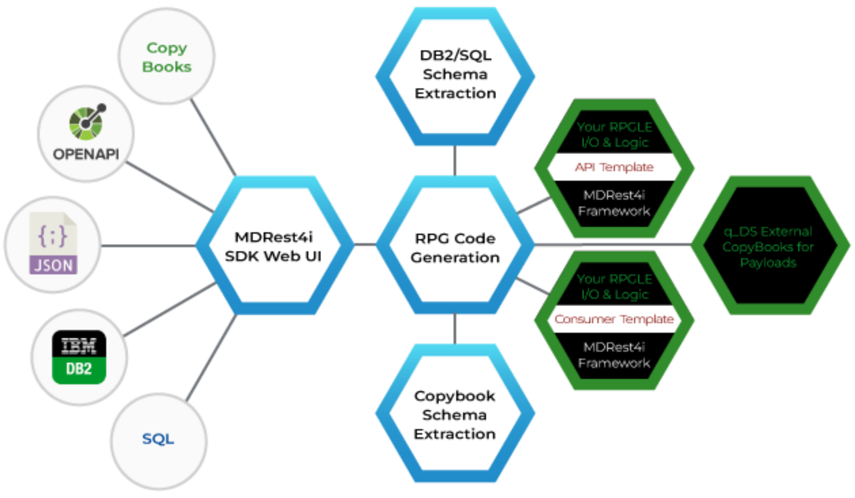


Figure 1 – MDRest4i Accelerator API Generator

4.1 MDRGENXAPI – Generate MDRest4i REST service or consumer from swagger

This API is for generating a REST service or consumer for GET/POST/PUT/PATCH/DELETE methods from a swagger/OpenAPI specification. The API creates and compiles, the source member in the requested source file/library combination as supplied in the swagger under x-lxrgen section. If the supplied JSON has "callType" set to "path", the swagger JSON is loaded from the IFS file specified for the "path" label.

4.1.1 MDREST4i SWAGGER/Open APi MDRGENXAPI generator extensions

4.1.1.1 x-lxrgen extension

The request and processing type is controlled by "x-lxrgen" section which could have below details:

```
"x-lxrgen": {
      "library": "MDRDEMOD",
      "srcfile": "QRPGLESRC",
      "srclibrary": "MDRDEMOD",
      "object": "CREDITLMT1",
      "x-ioaction": "read",
      "x-rspaction": "assign",
      "reqtype":"consumer",
      "ssncns": "Y",
      "lxrpath": http://mddev.mdcms.ch:2513/skdemo/tstmbr1,
      "multi-ccsid": "Y",
      "jsonreqmethod": "USR",
      "reqdblib": "MDRDEMOD",
      "reqifspath": "/home/MDRest4i/TSTIFSPRD.json",
      "jsonrespmethod": "DBF",
      "respdblib": "MDRDEMOD",
      "rspifspath": "/home/MDRest4i/TSTIFSPRD.json",
     "allownotfound": true,
     "missingentryhandling": false,
     "externalizemainds": "Y",
     "rootds": "Y"
    }
```

Entry Name	Purpose/Meaning
library	Library where object will be created during compilation. If this parameter is not found, the srclibrary is used as the object library.
srcfile	Name of the source file where source member will be created. If this entry is blank, the source file is determined from the first 10 bytes of "LXRSRCDTL" data area from MDRST library.
srclibrary	Library name for the source file/member. If this entry is not found, it fetches the value from 11th to 20th byte position of "LXRSRCDTL" data area from MDRST library.
object	Source member and object is created using this name. If this entry is not provided, it checks the "operationId" field to see if its length is less than 10 chars. If that's the case, it uses operationId for the source/object name. Otherwise, it retrieves the value of data area "LXRSRCDTL" from MDRSTxx library. The first 10 bytes contain source file and next 10 bytes contain the library name. It then tries to fetch the last source member starting with "LXOA" followed by six digit number in this source file and library. If the member found, it uses the next sequence, otherwise it creates LXOA000001 as the first object.
x-ioaction	This entry is applicable for "POST" request and it can be set to "*none", "read", "insert". The value "*none" means don't write the logic to load the data structure from the request body. The value "read" means consider the POST service as GET and load the request body parameters from the database using SQL with the DB mapping information provided under the request body. The value "insert" means write the logic to perform insert operation in the database file as per the database mapping information provided in the request body.
reqtype	It must be "consumer" for consumer generation, otherwise, this is ignored

ssncns	It is applicable when “reqtype” is set to “consumer”. The expected values are “Y” or “N”. When “Y” is specified, it creates SSL consumer, otherwise, non-SSL
jwt	“Y” or “N”. this sets on JWT handling and implements the ValidateToken() procedure in the relevant z_proc[methodxxx] subroutine. It extracts the “authorization: Bearer …..” header value
lxrpath	It is again applicable when “reqtype” is set to “consumer” and this is the path of the REST service that will be called from the consumer.
multi-ccsid	This parameter can be set to “Y” if the REST service you are generating is expected to be compiled and executed in different CCSID environments. In this case, it doesn’t write the actual characters for the curly/square braces, instead it uses Unicode constant.
jsonreqmethod	This parameter controls building JSON. For the consumer, the JSON is built for the request body and for the REST service, JSON is created for the response. If you want to automatically send the JSON from an IFS file, the value should be“IFS”. “DBF” means the JSON parsed content is stored in MDRJSONF file and JSON should be built from the data in this file. The default value “USR” means continue with the standard way of building JSON using addChar, beginObject etc functions.
reqdblib	This parameter is only used when “jsonreqmethod” is set to “DBF” and it tells from where to pick the MDRJSONF file.
reqifspath	When “jsonreqmethod” is set to “IFS”, this parameter is used to pick the IFS file where the final JSON (i.e. the request body received for the REST service) would be written. When used with “reqtype” set as “consumer”, the JSON content available in this IFS file will be read and written as the request body while sending the request.
jsonrespmethod	This parameter controls on how to parse the incoming JSON. For the consumer, the incoming JSON is the response from the REST service and that for the REST service is the request body. Use the value “IFS” to load the JSON in an IFS file, whereas, the value “DBF” can be used to load the json labels, values in the DB file named “MDRJSONF”. The default value “USR” means continue with the standard way of parsing JSON and load the required values using JPathN, JPathV etc functions.
respdblib	This parameter is only used when “jsonrespmethod” is set to “DBF” and it tells from where to pick the MDRJSONF file.
rspifspath	When “jsonrespmethod” is set to “IFS”, this parameter tells the IFS file where the final JSON (i.e. the response received from the REST service in consumer) would be written. When used with REST service, the JSON content available in this IFS file will be read and written as the response to the REST service.
allownotfound	The JSON values for the specific JSON label at the specific path are extracted using “JPathv” function. If the requested label/path is not found, the function returns “*notFound”. However, if you expect “*notFound” to be something possibly coming in JSON, set this parameter to “true” and then “JpathE” function will be used to extract the value of JSON path which returns blank if the entry is not found.
missingentryhandling	This parameter is used to apply condition before setting the value to avoid setting the variable when the value is not found. This is explained below using code example.

externalizemainds	This parameter to use declaring the data structure's definition in the external copybook. For this we need to set this as "externalizemainds":"Y". This parameter can be use in MDRGNAPI. So we can use this when gentype = "Inline" or "external".
rootds	This parameter uses for ignoring the rootds name so when we will set is as rootds to 'N' it will ignore not define data structure and add only standalone variables. This parameter can be use in MDRGNAPI. SO it can be use when gentype is set as "Inline" or "external".
x-rspaction	This parameter is only meaningful in post type request and it can contain the value "assign" which means do not write the response.

4.1.1.2 Schema parameter attribute specifications:

type	String, integer, number, boolean
x-ibmitype	Possible values are "zoned" or "packed" and this parameter is only considered when type is "number" or "integer"
multipleOf	This parameter tells the domain of the values but here it is used to identify the number of decimal positions. As an example, if the value is "multipleOf": 0.0001 and the data type is "number", that means the decimal positions are 4.
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.
minimum	This parameter is similar to "maximum" above and is used to identify the max decimal precision when the variable is being declared in the program.
format	The possible values for this entry could be "int32" or int64" for "integer" type 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 x-ibmitype 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 "x-ibmitype".

Below is the example of schema using above parameters:

```
"country": {
      "type": "array",
      "items": {
            "type": "object",
            "properties": {
                  "name": {
                        "type": "string",
                        "description": "Country Name",
                        "maxLength": 20
                  },
                  "countrycode": {
                        "type": "number",
                        "x-ibmitype": "zoned",
                        "description": "Country Code",
                        "multipleOf": 8.12345,
                        "maximum": 524567895678.1567876542
                  },
                  "population": {
                        "type": "integer",
                        "format": "int32",
                        "description": "Country Population"
                  },
            }
      }
}
```

The generated code will have different data structures declared based on the request body and response section in the swagger. The relevant subroutines e.g. z_ProcPost for "POST", z_PROCGET" for "GET" and "Z_PROCPUT" for "PUT" will have the logic to fetch the data structure subfield values from the request body using "JPathV" function and appropriate type-casting.

The advantage of this API is that, it can be used to generate the REST service as well as consumer from the same swagger. If you would like to generate the consumer for the service generated from the same swagger, just add "reqtype":"consumer" as explained above. If the consumer is expected to interact with the SSL REST service, the parameter "sslcns" should be set to "Y", otherwise, it should be set to "N". In consumer generation, the "BuildRequest" procedure will have the logic to prepare the request body from the variables declared corresponding to the request body. If the parameter "lxrpath" is written, it will set the exported variables "wg_servicepath", "wg_portNumber" and "wg_hostname" from this path. Otherwise, these three variables will be set to default/dummy value and the developer has to edit the source to specify the correct values. If you are generating SSL consumer, you need to check and fix the values of DCM application and SSL authentication in respective variables under "Initialize" procedure of the generated member.

When the parameter "missingEntryHandling" is set to true or not specified in "x-lxrgen" block, the retrieval of variable values from the request body like below:

```
w_str2 = jpathv('actor.type');
If w_str2 <> '*notFound';
  d_BitBucketReq.s_actor.s_type = w_str2;
Endif;
```

However, when "missingEntryHandling" is set to false in "x-lxrgen" block, it will not add the conditional block and the statement will look like below:

```
d_BitBucketReq.s_actor.s_type = jpathv('actor.type');
```

If the swagger contains multiple paths, it will generate different programs (one program for each path) in single call to MDRGENXAPI. In case the specific path contains multiple methods (e.g. GET, PUT, POST etc.), the program will have the logic for all those methods generated in different subroutines.

The information specified in the "requestbody" or "response" section is used to define the data structures at the top of the generated program. If the same JSON structure is found in requestbody and response sections, the data structure or standalone variables (as applicable) are defined only once and they are used for both loading of the request body and writing the response.

4.1.2 Numeric field format mapping between SWAGGER and RPGLE

Using this standard as a guide line,

https://swagger.io/docs/specification/data-models/data-types/

we have implemented this in the MDRGENXAPI generator in the following manner

type	format	multipleOf	maximum specified	default attribute	zoned specified	packed specified
number			Y	N/A	n s,2	n p,2
number			N	9b,2	10s,2	10p,2
number	float		Y	N/A	n s,2	n p,2
number	float		N	9b,5	10s,5	10p,5
number	double		Y	N/A	n s,2	n p,2
number	double		N	9b,5	16s,11	16p,11
integer			Y	N/A	n s, 0	n p, 0
integer			N	10i, 0	10s, 0	10p, 0
integer	int32		Y	N/A	n s, 0	n p, 0
integer	int32		N	10i, 0	10s, 0	10p, 0
integer	int64		Y	N/A	n s, 0	n p, 0
integer	int64		N	20i, 0	20s, 0	20p, 0

Note: If MultipleOf is given, it will have higher priority regarding the digits after decimal point and in that case, the format will be ignored

For the moment you must add format and multipleof in the swagger itself. The form UI ability in is being implemented V12

4.2 MDRGNAPI – Generate API or Client programs with schema-based procedures

REST API called from the MDRest4i SDK UI when "inline" or "external" is defined in the "Generation Type" drop down, in the Path Tab of the MDRest4i SDK UI . It uses SWAGGER input to generate API or Client programs with schema-based procedures. However, the parameters X-rspaction, x-ioaction, missingentryhandling, allownotfound, rspifspath, respdblib, jsonrespmethod, jsonreqmethod, reqifspath, reqdblib, multi-ccdid are not applicable. This API externalizes the JSON loading and response writing logic into sub-procedures but there are some pre-requisites. This API has some extra parameters listed below:

copymember	Copybook member name for the definition of data structure and prototypes externalized in separate module.
copysrcf	Source file for copybook member. If not specified, “srcfile” parameter is used.
copylib	Library for the copybook member
prcmodule	Module name for the exported procedure definitions.
modsrcf	Source file for the module
modslib	Library for the module

The procedure names are driven from the schema name and “Fetch” is prefixed to the schema name for the procedure that is supposed to read the JSON (i.e. request body in case of REST service and API response in case of consumer). Likewise, “Write” is prefixed to schema name for the procedure which is expected to build the JSON by using addChar, beginObje**c**t etc functions (this happens while processing the response in REST service or while building the request body in “BuildRequest” procedure of the consumer.

If the Copy member and procedure module is not found, the procedures are created inline within the same member (i.e. REST API or consumer).

4.3 MDRSCHEMA – Generate JSON schema from JSON

This API is used to generate the JSON schema from the sample of JSON response (extracted from the request body or the response section). There are three optional entries “lxrschemaname”, “lxrinputpath”, “lxroutputpath” which are supposed to be specified at the root of JSON as required. Below are the rules on how these entries make the difference in processing.

“**lxrinputpath**”: This entry is expected to contain the full IFS path (including the file name) to the JSON file. The content from this file will be read and the schema will get extracted from this JSON. These rules are applicable only when the entry “lxrschemaname” is specified at the root of the JSON, otherwise, “lxrinputpath” will be considered as any other label for schema generation and therefore it will become a field name in the extracted JSON schema.

“**lxroutputpath**”: This entry is expected to contain either the full IFS path (including the file name with last four characters as “.json”) or it can be the IFS folder name (e.g. “/home/MDRest4i/” or “/home/MDRest4i”) and in this case, the value of “lxrschemaname” suffixed with “.json” will be appended to make the full IFS path (e.g. “/home/MDRest4i/mySchema.json” assuming “mySchema” was the value of “lxrschemaname”). This schema will be extracted and written to this IFS file. These rules are applicable only when the entry “lxrschemaname” is specified at the root of the JSON, otherwise, “lxroutputpath” will be considered as any other label for schema generation and therefore it will become a field name in the extracted JSON schema.

“**lxrschemaname**”: If this entry is specified, the label “lxrjson” will be searched in the supplied JSON (if “lxrinputpath” entry is not present at the root of the schema or in the JSON available in the IFS file when “lxrinputpath” has the IFS file name). If “lxrjson” is not present and output path is specified, the schema extractor will report an error. If “lxrschemaname” is not present, the whole JSON is considered for schema generation (including “lxrinputpath” and “lxroutputpath” if supplied) and it gets labelled with “generic_schema” at the root of JSON, otherwise, the schema will be labelled with “lxrjson”.

When the output goes to the IFS file, the API response will provide two entries “lxrschemaname”, “lxroutputpath” along with the message “schema extracted successfully”. If the “lxrinputpath” has also been specified, the same will be available in the response too.

When the output is not in the IFS file, the response will be the entire schema created from the supplied JSON using above rules.

Example 1: Below is the JSON supplied as the request body where neither of the three (i.e. “mdrschemaname”, “lxrinputpath” or “lxroutputpath”) are blank.

```
{
      "lxrschemaname": "myjsonschema",
      "lxrinputpath": "/MDREST4i/logs/schemainput1.json",
      "lxroutputpath": "/MDREST4i/logs/outputfile1.json"
}
```

The content of the “schemainput1.json” file:

```
{
   "repository": {
        "website": "www.mysite.com",
        "scm": "git",
        "name": "bitbucket-hooks-examples",
        "uuid": "{1e8809a8-35a9-4d85-99c2-64d16afc8634}",
        "full_name": "gableroux/bitbucket-hooks-examples",
        "lxrjson": {
            "username": "gableroux",
            "type": "user",
            "display_name": "Gabriel Le Breton"
           }
    }


}
```

Below is the response in SOAPUI or the consumer:

```
{
   "status": "Schema Extracted Successfully",
   "inputfile": "Processed Input json
file:/MDREST4i/logs/schemainput1.json",
   "outputfile": "Schema output written at:/MDREST4i/logs/outputfile1.json"
}
```

Below is the output in IFS file named “outputfile1.json”

```
{
  "lxrjson": {
    "type": "object",
    "properties": {
      "username": {
        "description": "username_desc",
        "type": "string",
        "examples": [
"gableroux"
]
      },
      "type": {
        "description": "type_desc",
        "type": "string",
        "examples": [
"user"
]
      },
      "display_name": {
        "description": "display_name_desc",
        "type": "string",
        "examples": [
"Gabriel Le Breton   "
]
      }
    }
  }
}
```

Example 2: Below is the example where neither of the three (i.e. “lxrschemaname”, “lxrinputpath” or “lxroutputpath”) are provided.

```
{
    "repository": {
        "website": "www.mysite.com",
        "scm": "git",
        "name": "bitbucket-hooks-examples",
        "uuid": "{1e8809a8-35a9-4d85-99c2-64d16afc8634}",
        "full_name": "gableroux/bitbucket-hooks-examples",
        "owner": {
            "username": "gableroux",
            "type": "user",
            "display_name": "Gabriel Le Breton"
        }
    }

}
```

In this case, the schema generator doesn’t search for “lxrjson” label. Below is the output:

```
{
 "generic_schema": {
      "repository": {
        "type": "object",
        "properties": {
            "website": {
               "description": "website_desc",
               "type": "string",
               "examples": ["www.mysite.com"]
          },
            "scm": {
               "description": "scm_desc",
               "type": "string",
               "examples": ["git"]
      },
            "name": {
               "description": "name_desc",
               "type": "string",
               "examples": ["bitbucket-hooks-examples"]
       },
            "uuid": {
               "description": "uuid_desc",
               "type": "string",
               "examples": ["\"{1e8809a8-35a9-4d85-99c2-64d16afc8634}\""]
      },
            "full_name": {
               "description": "full_name_desc",
                "type": "string",
                "examples": ["gableroux/bitbucket-hooks-examples"]
              },
             "owner": {
                "type": "object",
                "properties": {
       "username": {
             "description": "username_desc",
             "type": "string",
             "examples": ["gableroux"]
                    },
       "type": {
             "description": "type_desc",
             "type": "string",
             "examples": ["user"]
                },
       "display_name": {
            "description": "display_name_desc",
             "type": "string",
             "examples": ["Gabriel Le Breton\t"]
                        }
               }
            }
        }
                }
         }
}
```

4.4 MDRFIELDS – Return the database schema of specified file

This API returns the JSON schema of the requested database file. The API expects three query parameters. Whereas “library and “file” are mandatory and third parameter is “tmpfile” is optional. If we want to remove the file after processing, please use third parameter with “tmpfile=Y”. In this way API will remove the file after sending the fields details in JSON.. The API identifies the field details of the file object in the specified library. Below is the example:

http://yourserver.com/mdrst/mdrfields?library=XAN4CDEM&file=CUSTS

or

http://yourserver.com/mdrst/mdrfields?library=XAN4CDEM&file=CUSTS&tmpfile=Y

4.5 DSPSRCMBR – Download the specified source member

This API returns the IFS file's source and source code of the source member from the specified source file/library. So if we want to get back source of ifs path, we need to execute this API with one query parameter "ifspath" and if we want to read source code of source member the we need to execute this API with three parameters are "srclib", "srcfil" and "srcmbr". As an example, below API returns the source code of the source member JWTTKN in QRPGLESRC source file under LXRDEVDV library:

http://yourserver.com/mdrst/dspsrcmbr?srclib=lxrdevdv&srcfil=qrpglesrc&srcmbr=JWTTKN

Below example for reading IFS path

http://yourserver.com/mdrst/dspsrcmbr?ifspath=/home/MDRest4i/testjson.json

4.6 MDRCHK – Check an object's existence

This API checks the object existence and returns the success or failure response. The API has three parameters. These parameters are "objName", "objType" and "libName". As an example, below API checks if the object "LXRPRDDA" of type *DTAARA exists in MDRST library or not:

http://yourserver.com/devend1/mdrchk?objName=LXRPRDDA&objType=*DTAARA&libName=MDRST

4.7 Generated SWAGGER Examples

Swagger Name	Program Name	Description
GETPRODUCEREXAMPLEONTICLIENTFILE.JSON	GETCLNTPGM	This is GET service generated from the accelerator using swagger and this API brings the data from TICLIENT file on the basis of the supplied parameter.
GETCONSUMEREXAMPLEONTICLIENTFILE.JSON	GETCLNTPGC	This is a REST consumer program generated from the accelerator using swagger. It calls the GET service (i.e. GETCLNTPGM API) and writes the received records in the TICLIENT file.
GETPRODUCEREXAMPLEONTIPOLICYFILE.JSON	GETPLCYPGM	This is GET service generated from the accelerator using swagger and this API brings the data from TIPOLICY file on the basis of the supplied parameter.
GETCONSUMEREXAMPLEONTIPOLICYFILE.JSON	GETPLCYPGC	This is a REST consumer program generated from the accelerator using swagger. It calls the GET service (i.e. GETPLCYPGM API) and

		writes the received records in the TIPOLICY file.
POSTPRODUCEREXAMPLEONTICLIENTFILE.JSON	POSTCLNTPG	This is POST service generated from Accelerator using swagger. This API writes the records in TICLIENT file as per the information received in POST request body
POSTCONSUMEREXAMPLEONTICLIENTFILE.JSON	POSTCLNTPC	This is a REST consumer program generated from the accelerator using swagger. It calls the POST service (i.e. POSTCLNTPG API) and supplies the data as part of the request body.
POSTPRODUCEREXAMPLEONTIPOLICYTFILE.JSON	POSTPLCYPG	This is POST service generated from Accelerator using swagger. This API writes the records in TIPOLICY file as per the information received in POST request body
POSTCONSUMEREXAMPLEONTIPOLICYFILE.JSON	POSTPLCYPC	This is a REST consumer program generated from the accelerator using swagger. It calls the POST service (i.e. POSTPLCYPG API) and supplies the data as part of the request body.
PUTPRODUCEREXAMPLEONTICLIENTFILE.JSON	PUTCLNTPG	This is PUT service generated from Accelerator using swagger. This API writes/Update the records on the basis of the supplied parameter in TICLIENT file as per the information received in POST request body.
PUTCONSUMEREXAMPLEONTICLIENTFILE.JSON	PUTCLNTPGC	This is a REST consumer program generated from the accelerator using swagger. It calls the PUT service (i.e. PUTCLNTPG API) and supplies the data as part of the POST request body.
PUTPRODUCEREXAMPLEONTIPOLICYFILE.JSON	PUTPLCYPG	This is PUT service generated from Accelerator using swagger. This API writes/Update the records on the basis of the supplied parameter in TIPOLICY file as per the information received in POST request body.
PUTCONSUMEREXAMPLEONTICLIENTFILE.JSON	PUTPLCYPC	This is a REST consumer program generated from the accelerator using swagger. It calls the PUT service (i.e. PUTPLCYPG API) and

		supplies the data as part of the POST request body.
PATCHPRODUCEREXAMPLEONTICLIENTFILE.JSON	PCHCLNTPG	This is PATCH service generated from Accelerator using swagger. This API Update the records because of the supplied parameter in TICLIENT file as per the information received in POST request body.
PATCHCONSUMEREXAMPLEONTICLIENTFILE.JSON	PCHCLNTPC	This is a REST consumer program generated from the accelerator using swagger. It calls the PATCH service (i.e. PCHCLNTPG API) and supplies the data as part of the POST request body.
PATCHPRODUCEREXAMPLEONTIPOLICYTFILE.JSON	PCHPLCYPG	This is PATCH service generated from Accelerator using swagger. This API Update the records on the basis of the supplied parameter in TIPOLICY file as per the information received in POST request body.
PATCHCONSUMEREXAMPLEONTIPOLICYTFILE.JSON	PCHPLCYPC	This is a REST consumer program generated from the accelerator using swagger. It calls the PATCH service (i.e. PCHPLCYPG API) and supplies the data as part of the request body.
DELETEPRODUCEREXAMPLEONTICLIENTFILE.JSON	DELCLNTPG	This is DELETE service generated from Accelerator using swagger. This API deletes the records on the basis of the supplied parameter in TICLIENT file.
DELETECONSUMEREXAMPLEONTICLIENTFILE.JSON	DELCLNTPC	This is a REST consumer program generated from the accelerator using swagger. It calls the DELETE service (i.e. DELCLNTPG API).
DELETEPRODUCEREXAMPLEONTIPOLICYFILE.JSON	DELPLCYPG	This is DELETE service generated from Accelerator using swagger. This API deletes the records on the basis of the supplied parameter in TIPOLICY file.
DELETECONSUMEREXAMPLEONTIPOLICYFILE.JSON	DELPLCYPC	This is a REST consumer program generated from the accelerator using swagger. It calls the DELETE service (i.e. DELPLCYPG API).
POSTREADRECORDEXAMPLE.JSON	POSTREADPG	This is POST service generated from Accelerator using swagger. This API

		reads the records from the file (i.e. TICLIENT) as per the parameter values received in POST request body.
POSTWRITEANDRESPONSE.JSON	POSTWRTRSP	This is POST service generated from Accelerator using swagger. This API writes the records in TICLIENT file as per the information received in POST request body and returns the response.
POSTLOADWITHOUTFILE.JSON	POSTLOADVL	This is POST service generated from Accelerator using swagger. This API only loads the variable values received in POST request body.

5 MDRest4i Paging implementation

The MDRest4i copybooks for generating the REST services and the modules utilizing these services have the provision of implementing paging. In order to be able to use page forward and backward, the changes are required at three places:

5.1 Additional pagination headers in HTTP server config

The current implementation requires the four custom headers defined below to be added to the HTTP server config

These HTTP headers are automatically added when building an HTTP server instance using command MDRGENSRV

LXRPBGNKEY: Page begin key value. This is the begin key of the current page to be returned from the service back to the consumer so that the consumer can send this key to the service for loading the previous page. This key contains the database file key field values and each value is separated by “&&”.

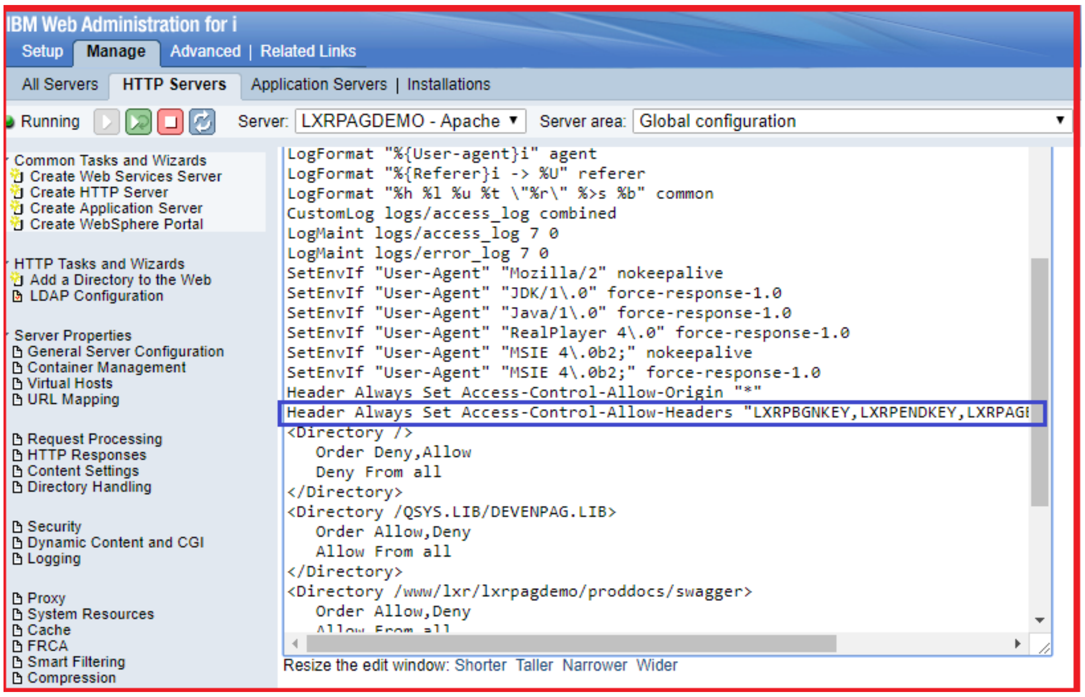
LXRPENDKEY: Page end key value. This is the end key of the current page to be returned from the service back to the consumer so that the consumer can send this key to the service for loading the next page. This key contains the database file key field values and each value is separated by “&&”.

LXRPAGETYP: One character value which should be blank for the first request, “N” for next and “P” for the previous page request

LXRNBRRCD: Number of records requested. This value should be supplied in each request (initial loading, previous page and next page). If this value is not sent in custom header, the service will return first 100 rows or end of file whichever occurs before.

To allow these custom header values, open the http administration page and make sure that below entries are allowed in headers.

"LXRPBGNKEY,LXRPENDKEY,LXRPAGETYP,LXRNBRRCD"



```
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 7 0
LogMaint logs/error_log 7 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
Header Always Set Access-Control-Allow-Origin "*"
Header Always Set Access-Control-Allow-Headers "LXRPBGNKEY,LXRPENDKEY,LXRPAGE
<Directory />
   Order Deny,Allow
   Deny From all
</Directory>
<Directory /QSYS.LIB/DEVENPAG.LIB>
   Order Allow,Deny
   Allow From all
</Directory>
<Directory /www/lxr/lxrpagdemo/proddocs/swagger>
   Order Allow,Deny
   Allow From all
```

5.2 Pagination logic in the MDRest4i service copybooks

The changes have been made in LXRRESTC, MDRESTDFN and LXRRESTP members for paging implementation. Below are the details":

LXRRESTC: When the request is of type "GET", the first check is done to retrieve the custom header values. These are LXRPBGNKEY, LXRPENDKEY, LXRPAGETYP and LXRNBRRCD. If the requested page type is not "N" or "P", the request is normally processed, otherwise, it is routed to appropriate routine for the page request. In page forward or backward request, the query parameters in the request URL are ignored because the processing is done based on the received values of begin and end keys. In case of the previous page request, the "begin key" is loaded in s_Value field of qualified data structure array named "d_parm". In next page request, the key field values in "end key" is loaded in s_value field of qualified data structure array. The control is then sent to Z_PrcNxtPage or Z_PrcPrvPage subroutines depending on the type of request. Blank subroutine bodies of these two subroutines have been provided in LXRREST and these are expected to be overridden in the program implementing the "GET" service with page forward and backward capabilities. The default inclusion of subroutine z_PrcNxtPage is subject to the compiler directive LXR_NxtPage not being defined using define compiler directive command and that for z_PrcPrvPage is subject to the compiler directive LXR_PrvPage. The loading of the key values happens in Z_LoadKeys subroutine and if you would like to change the default processing, use the compiler directive LXR_LoadKeys and define the subroutine Z_LoadKeys in your program that uses LXRRESTC/D/P copybooks.

MDRESTDFN: Below variables have been defined in this copybook for paging purpose:

```
0235.00        * Page processing variables
0236.00       d w_wrkkeys        s            1024
0237.00       d w_pageEndkey     s            1024
0238.00       d w_pageBgnkey     s            1024
0239.00       d w_pageReqType    s               2
0240.00       d w_nbrRcd         s             10i 0
```

```
0241.00       d n_BgnKey         s                 n
```

LXRRESTP: The only change in regard is in LXRPush procedure to send these custom headers.

Example: The program CLM_DETPGS has been provided in QEXAMPLES source file and this example depicts the usage of paging. This example works on our development machine and can be used. If you would like to use this on your system, copy the file LXCLAIM from MDRSTxx to the library of your CGI job. Alternatively, change the source to have the library qualification for LXCLAIM file in SQL query of this program. Alternatively, MDRGENPRD command can be used to create an example depicting page processing.

5.3 Logic in the REST Consumer module for pagination

If your REST service has been developed by MDRest4i and it has the pagination logic implemented through the custom headers LXRPBGNKEY, LXRPENDKEY, LXRPAGETYP and LXRNBRRCD, you can handle paging in REST consumer using same headers.

In order to do this, below variable definition should be added in the consumer program:

```
d w_BgnKey        s           1024
d w_EndKey        s           1024
d w_NbrRcdc       s           1024
d w_PageTyp       s              1
d w_nbrrcd        s              5s 0 Inz(6)
```

After the control returns from GskConsume procedure, you have to load the values of custom header, followed by setting “w_pagetyp” = ‘N’ (for next page) and “P” (for previous page) and again call GskConsume procedure.

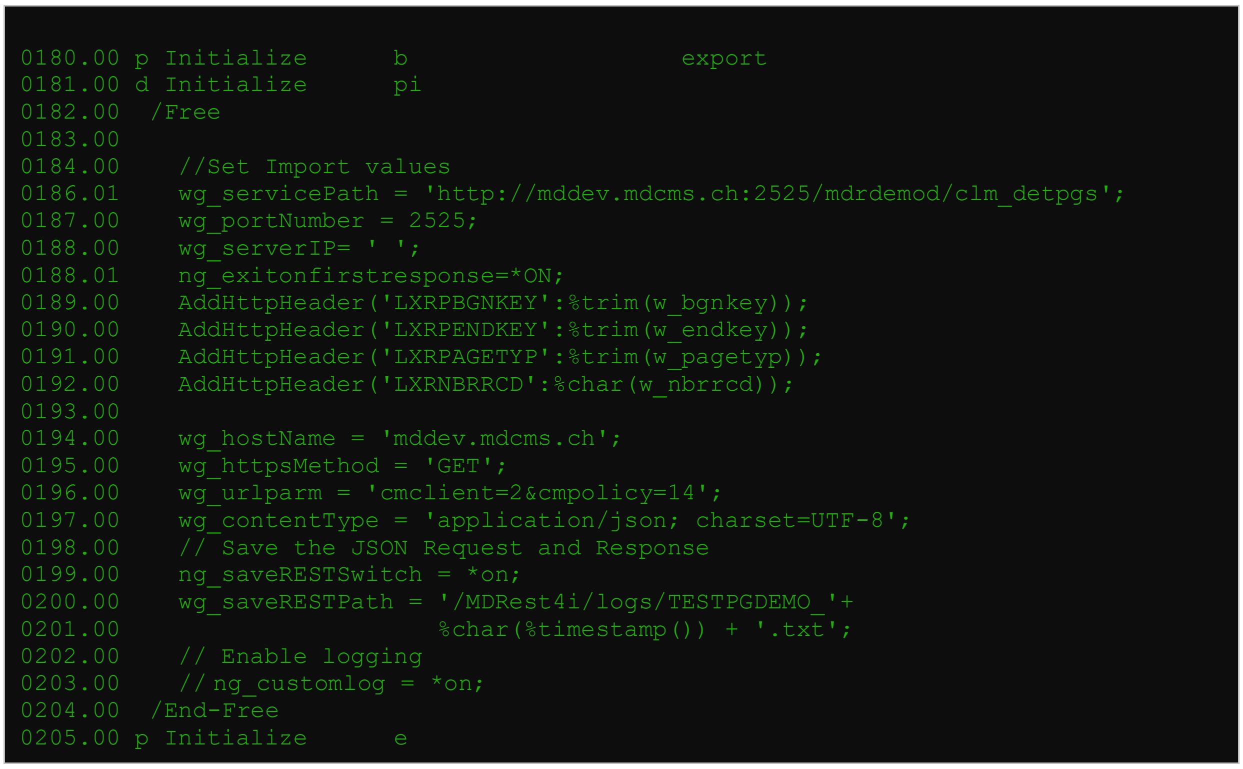
Example: The sample depicting the use of consumer to load initial, next and previous page is provided in CLM_DETPGC source member in QEXAMPLES. The mainline section of this program has below statements:

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

GskConsume();

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

// Process the next request for previous page
Exsr Z_LoadCustomHdr;
```

```
w_pagetyp = 'P';
GskConsume();
*Inlr = *On;
```

In this example, the initial GskConsume call is to load the begin page and then subroutine z_LoadCustomHdr is executed to load the begin and end key values from the custom headers as received from the REST service and the same gets sent to the subsequent request. The variable w_pageTyp is set to “N” for next page request and the third call is with w_pageTyp=”P” for loading the previous page.

The last difference is in Initialize procedure. We have the call to “AddHttpHeader” procedure to send these custom headers to the next http GET Request.



```
p Initialize      b                    export
d Initialize      pi
 /Free

   //Set Import values
   wg_servicePath = 'http://mddev.mdcms.ch:2525/mdrdemod/clm_detpgs';
   wg_portNumber = 2525;
   wg_serverIP= ' ';
   ng_exitonfirstresponse=*ON;
   AddHttpHeader('LXRPBGNKEY':%trim(w_bgnkey));
   AddHttpHeader('LXRPENDKEY':%trim(w_endkey));
   AddHttpHeader('LXRPAGETYP':%trim(w_pagetyp));
   AddHttpHeader('LXRNBRRCD':%char(w_nbrrcd));

   wg_hostName = 'mddev.mdcms.ch';
   wg_httpsMethod = 'GET';
   wg_urlparm = 'cmclient=2&cmpolicy=14';
   wg_contentType = 'application/json; charset=UTF-8';
   // Save the JSON Request and Response
   ng_saveRESTSwitch = *on;
   wg_saveRESTPath = '/MDRest4i/logs/TESTPGDEMO_'+
                     %char(%timestamp()) + '.txt';
   // Enable logging
   //ng_customlog = *on;
 /End-Free
p Initialize      e
```

6 V12 Token Management

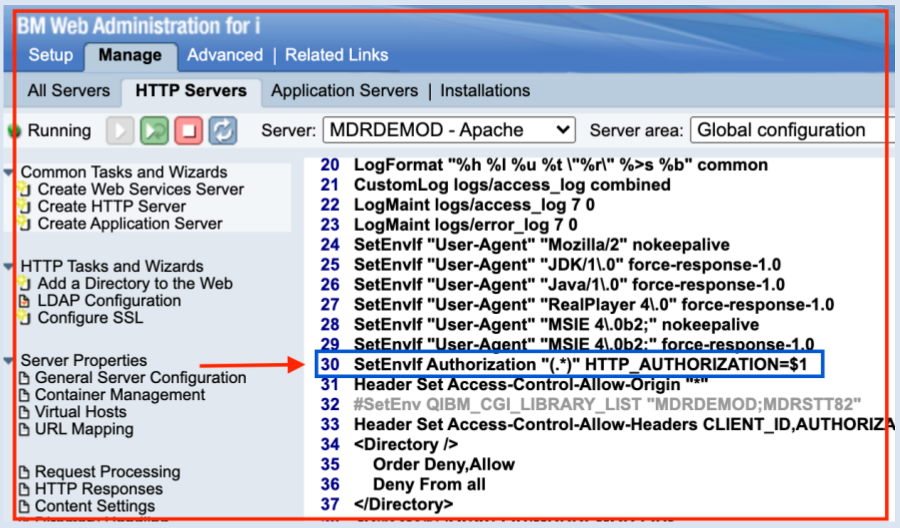
From Version 12 of MDRest4i onwards, the JWT tokens are now being stored/managed with a new file and set of REST API's, RPGLE modules.

Type	Object
PF/TABLE	MRDCREDX – MDRest4i SDK Credential Store
RPGLE Procedure	ValidateToken() – Validate Token (Found in copybook QRPGLESRC/LXRRESTP)
RPGLE MODULE	MRDENCRX – MDRest4i SDK Credentials Store I/O Module
RPGLE MODULE	MRDCREDXR – MDRest4i Token Management Module
REST API	MRDCREDXA – MDRest4i SDK Credentials Store API *(Supports GET/POST/PATCH/DELETE methods)*
REST API	MRDCREDXB – MDRest4i SDK Credentials Store API – Validate Tkn *(Supports GET/POST methods)*

Token Validation for an API built with MDREST4i itself, is handled by the **ValidateToken()** procedure from LXRRESTP copybook of the REST producer.

When this procedure call is added to producer in the z_proc[method] it extracts the specific http headers (i.e. "CLIENTID", "APPID") and extracts the "AUTHORIZATION" where the value is expected as "Bearer [token]". It then validates the token against the "***MRDCREDX*** - Token Credentials Store" detailed below.

In order to make sure the API is able to get the Authorization header from the request, you must add below highlighted setting in the HTTP server configuration.



```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log combined
LogMaint logs/access_log 7 0
LogMaint logs/error_log 7 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
SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1
Header Set Access-Control-Allow-Origin "*"
#SetEnv QIBM_CGI_LIBRARY_LIST "MDRDEMOD;MDRSTT82"
Header Set Access-Control-Allow-Headers CLIENT_ID,AUTHORIZA
<Directory />
   Order Deny,Allow
   Deny From all
</Directory>
```

Parameter	Attributes	Description
Client Id	36A	This is the first part of the composite key which is the unique identifier to issue and store the token in MRDCRED file
Application Id	20A	This is the second part of the composite key which is the unique identifier to issue and store the token in MRDCRED file
Usage Type	1A	Usage type of the token (A – API, C – Consumer, B – Both, O – Other)
Credential Type	1A	Credential Type (B – Basic, T – Bearer, R – Remote)
Extended Option	5A	

Algorithm	20A	Depending on the type of token being generated, the value should be sent accordingly. The possible values are:
		UUID = Random 36 byte value
		JWT-HS256 = Secret and SHA256 Encryption for Signature
		JWT-RS256-DCM = RS256 using DCM App
		JWT-RS256-KST = RS256 using PF and Lib Key Store
		JWT-RS256-PVT = RS256 using Private & Public Keys
		JWS-HS256 = Secret and SHA256 Encryption for Signature
		JWS-RS256-DCM = RS256 using DCM App
		JWS-RS256-KST = RS256 using PF and Lib Key Store
		JWS-RS256-PVT = RS256 using Private & Public Keys
		PAT = Personal Access Token

Authentication Token	8000A	Authentication Token
User Name	256A	User name associated with the token (not used now)
Password	1024A	Password corresponding to the user profile above
Payload	2048A	JSON payload for the token generation. It must have “exp” for the token validity duration in seconds starting from the token creation timestamp. If “iat” is there, it will be replaced with the current epoch timestamp value. Refer https://www.epochconverter.com/
Client Secret	4096A	Some string that can be used as the secret to generate/hash the token
Public Key	4096A	Public Key for RSA token
Private Key	4096A	Private Key for RSA token
Auth Server	128A	Authentication Server
Auth Port	5P,0	Authentication Port
Token Server	128A	Token Server
Token Port	5P,0	Token Port
Refresh Method	10A	Refresh Method (e.g. *AUTO, *MANUAL)
Refresh Token	8000A	Refresh Token

Auth Token Expiry	11P,0	Auth token expiry period(how long before token expires in Expiry Unit(s)
Refresh Token Expiry	11P,0	Refresh token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
Expiry Unit	10A	Time Unit of measure (i.e. the duration in which the token expiry and refresh expiry should be calculated e.g. *SECONDS)
Token Issuer	36A	Auto populated by the API generating the token and it contains the user profile which was used to create the token
Issue Time	26Z	Token issue timestamp
Issue Log	64A	Issue user and job details
Update Time	26Z	Token update timestamp
Update Log	64A	Updated by user and job information

6.1 *MRDCREDX* - Token Credentials Store

The V12 physical file that stores all token credentials and tokens is:

"MRDCREDX - MDRest4i SDK Credential Store"

By default, this is stored in the MDRSTxxxx library. Below is the structure of this file:

*Note: The sensitive data (i.e. the fields Auth Token, Refresh Token, Password, Secret, public key, private key) are stored in encrypted form. The encryption is done using HS256 algorithm(in Procedure mrdencrxs below) taking the secret from the data area "MRDCREDDA" if present in library list, otherwise, a hard-coded string is used to encrypt/decrypt this data.

6.2 MRDENCRX – MDRest4i SDK Credentials Store I/O Module

If you want to access the data in this file, include the copybook MRDENCRXP in your program for the relevant definitions.

```
/Copy MRDERNCXP
```

The copybook "MRDENCRXP" has the prototype definition of the main procedure in MRDENCRX module (which is available in LXRGLOBAL binding directory). The data structure being used in this prototype has all the fields of MRDCREDX file and it also has an "option" field with the meanings listed below. The table MRDCREDX file stores some of the sensitive information in encrypted form but the program MRDCREDR returns the decrypted data.

"G" – Retrieve the record for the supplied Client ID and App ID

"U" – Update limited fields (e.g. payload, token, refresh token, refresh token expiry etc.)

"R" – Replace all the columns of the record

"C" – Create the record

“D” – Delete the record

During update operation (i.e. “U” actions, the field which you do not want to update should be supplied with the hard-coded value '*Blank'.

In order to retrieve the specific token or Client/App Id record, set the value of “option” field of the MRDCREDDS data structure to 'G' and optionally set the “client Id” and “app Id” values for which you want to access the auth token etc information. Once the data structure “mrdcredds” is set, call the procedure “mrdencrxs” coming from the module “MRDENCRX”. This procedure will return the complete data structure loaded with the token and all other details relevant to that specific record.

6.3 MRDCREDXR – Token Management Module

The module MRDCREDXR has various procedures for handling the HS256 or RS256 tokens. Below is the list of available procedures. The prototype definitions of the same can be found in MRDCREDXC copybook member available in QRPGLESRC source file of the product library. The module MRDCREDXR is accessible via the LXRGLOBAL binding directory.

6.3.1 CrtHS256Token ()

This procedure is used to create HS256 token. Below is the list of parameters and their meanings as used in this procedure.

Please note that first seven parameters are mandatory, rest are optional

Parameter	**Attributes**	**Purpose**
Action	1A	This parameter is not used at present
Payload	2048A	This is expected to have the JSON payload for creating token
Secret	4096A	Secret String (must be greater than 32 bytes in size)
Authtoken	2048A	Returned authentication token
RefreshToken	2048A	Returned refresh token
Algorithm	20A	This refers to Encryption Algorithm & Token Creation Method
Status	50A	Details of successful token creation or error message
ClientId	36A	This is the first part of possible composite key with which you may like store token
ApplicationId	20A	This is the second part of possible composite key with which you may like store token
UsageType	1A	This refers to the usage of token (A-API, C-Consumer, B-Both, O-Other)
CredType	1A	Credential type (B-Basic authentication, T-Bearer token, R-Remote)

JsonNamVal	1780A	This is a data structure containing two separate arrays (i.e. label & value) of 10 dim each. The definition of this data structure is available in MRDCREDXC copybook `d d_JsonNamVal     DS                   qualified inz` `d   s_Name                        50a    dim(10)` `d   s_Value                      128a    dim(10)`
refexp	11,0	Refresh token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
expunit	10A	Expiry unit (e.g. *SECONDS, *MINUTES) - Not in use now
username	256A	User name that you want to set while creating the token
password	1024A	Password of the user profile
refreshmth	10A	Refresh method (e.g. *AUTO, *MANUAL) - Not in use now
issuer	36A	Issuer of the token (if you want to set some value other than the user profile)

6.3.2 CrtPAToken ()

This procedure is used to create personal access token. Below is the list of parameters and their meanings as used in this procedure.

Please note that all the parameters are mandatory

Parameter	Attributes	Purpose
Authtoken	2048A	Returned authentication token
RefreshToken	2048A	Returned refresh token
TokenLength	3,0	Supply the token length (How much length's token wants to create).
Status	50A	Details of successful token creation or error message
ClientId	36A	This is the first part of possible composite key with which you may like store token
ApplicationId	20A	This is the second part of possible composite key with which you may like store token
UsageType	1A	This refers to the usage of token (A-API, C-Consumer, B-Both, O-Other)
CredType	1A	Credential type (B-Basic authentication, T-Bearer token, R-Remote)

authTokenExpiry	11,0	Auth token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
RefreshTokenExpiry	11,0	Refresh token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
expTimestamp	26A	Token Expiry (timestamp) – Output parameter

6.3.3 CrtRS256Token()

This procedure is used to create RS256 token. Below is the list of parameters and their meanings as used in this procedure. When algorithm is used as “JWT-RS256-DCM”, it uses DCM application parameter to create the token, however, when the value is “JWT-RS256-KST”, it means use key file, library and key label parameters to find the RS256 key. The generated token is returned in the second parameter:

Parameter	**Attributes**	**Purpose**
Action	1A	This parameter used as “S” would use signing algorithm, otherwise with “E”, it will use encryption
Authtoken	5242880A	Returned authentication token
Payload	5242880A	This is expected to have the JSON payload for creating token

Algorithm	20A	This parameter expects the algorithm to use
		JWT-RS256-DCM = RS256 using DCM App
		JWT-RS256-KST = RS256 using PF and Lib Key Store

DcmApp	20A	DCM application to be used to create RS256 token
ClientId	36A	This is the first part of possible composite key with which you may like store token
ApplicationId	20A	This is the second part of possible composite key with which you may like store token
UsageType	1A	This refers to the usage of token (A-API, C-Consumer, B-Both, O-Other)
CredType	1A	Credential type (B-Basic authentication, T-Bearer token, R-Remote)
refexp	11,0	Refresh token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
expunit	10A	Expiry unit (e.g. *SECONDS, *MINUTES) - Not in use now
Key file	10A	The file that contains RS256 key
Key lib	10A	The library that contains RS256 key file
Key label	32A	Label within the key file

6.3.4 RefHS256Token()

This procedure is used to refresh the HS256 token. Below is the list of parameters and their meanings as used in this procedure. Please note that first four parameters are mandatory, rest are optional. You can supply blank in second parameter if you are supplying the client Id and application Id parameters, otherwise you have to supply the secret too for decrypting the token. If you want to receive the Json name and value pairs from the payload which was used to create the supplied token, you can supply the last parameter:

Parameter	**Attributes**	**Purpose**
RefreshToken	2048A	Supply the refresh token in this parameter
Secret	4096A	Secret String (required if client Id and application Id not supplied)
Authtoken	2048A	Returned authentication token
Status	50A	Details of successful token creation or error message
ClientId	36A	This is the first part of possible composite key with which you may like store token
ApplicationId	20A	This is the second part of possible composite key with which you may like store token

JsonNamVal	1780A	This is a data structure containing two separate arrays (i.e. label & value) of 10 dim each
		`d d_JsonNamVal     DS                    qualified inz`
		`d   s_Name                         50a   dim(10)`
		`d   s_Value                       128a   dim(10)`

6.3.5 ValidateToken ()

This procedure is used to validate an HS256 or RS256 token. Below is the list of parameters and their meanings as used in this procedure. This procedure works only on the auth token information supplied. If you have also supplied the values in client Id and Application Id, it would use them as well to locate the record for identifying whether its RS256 or HS256 token. The third parameter will return the JSON label and value arrays as part of the data structure after extracting the payload from the token:

Parameter	**Attributes**	**Purpose**
AuthToken	2048A	Supply the refresh token in this parameter
Status	50A	Details of successful token creation or error message
JsonNamVal	1780A	This is a data structure containing two separate arrays (i.e. label & value) of 10 dim each

		`d d_JsonNamVal     DS                  qualified inz`
		`d   s_Name                       50a    dim(10)`
		`d   s_Value                     128a    dim(10)`

Secret	4096A	Secret String (required if client Id and application Id not supplied)
ClientId	36A	This is the first part of possible composite key with which you may like store token
ApplicationId	20A	This is the second part of possible composite key with which you may like store token
TokenType	1A	This parameter is for internal use when the procedure is called for token refresh. You can supply blank.

6.3.6 getCNSToken()

This procedure is used to retrieve the token from the MRDCREDX file and refresh it if token has expired. This function has five parameters. Where first two parameters are input parameters and last three parameters are output parameters. We need to supply application id in first parameter and supply client id as second parameter. Third parameter to return Token, fourth parameter to send message severity and fifth last parameter to send message. If token exist and not expired, will return the token. If token does not exist, return error message, and send message severity as 30. If token expired, refresh it, and write the refreshed token to MRDCREDX and return the token to consumer and send message severity as 10 and message (Token refreshed successfully). If any error at a time of token refreshing, return the message (‘Error occurred during token refresh') and send message severity 30.

Parameter	**Attributes**	**Purpose**
ApplicationId	20A	This is the first part of possible composite key with which you may like store token.
ClientId	36A	This is the second part of possible composite key with which you may like store token.
Token	8000A	Returned The token.
Message Severity	2S 0	Return message severity (00), when token valid and not expired. If token expired set message severity (10). If token does not exist, not valid or any error occurred during token refreshing it will return as (30).
Message	50A	If token exist and valid it will not return any massage. If token expired, token does not exist or token is not valid, return the error message to consumer.

6.4 Token Management REST API’s

There are two REST APIs MRDCRECXA and MRDCREDXB which provide the options to create, refresh, update validate and delete the token entries in the MDRest4i Credentials Store

6.4.1 MRDCREDXA - Issue a new token using HS256 algorithm

To issue a token, use MRDCREDXA with “POST” method. We have to supply below information as part of the request body:

Parameter	Description
clientId	This is the first part of the composite key which is the unique identifier to issue and store the token in MRDCRED file
applicationId	This is the second part of the composite key which is the unique identifier to issue and store the token in MRDCRED file
usageTyp	Usage type of the token (A – API, C – Consumer, B – Both, O – Other)
credTyp	Credential Type (B – Basic, T – Bearer, R – Remote)
alg_keys	Depending on the type of token being generated, the value should be sent accordingly. You need to supply the value “JWT-HS256” in this field.
username	User name associated with the token (not used now)
password	Password corresponding to the user profile above
payload	JSON payload for the token generation. The payload should be escaped as its JSON inside JSON. It must have “exp” for the token validity duration in seconds starting from the token creation timestamp. If “iat” is there, it will be replaced with the current epoch timestamp value. Refer https://www.epochconverter.com/
clientSecret	Some string that can be used as the secret to generate/hash the token
refreshMethod	Refresh method (e.g. *AUTO, *MANUAL) - Not in use now
refreshTokenExpiry	**An optional value:** Refresh token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
timeUnitOfMeasure	The duration in which the token expiry and refresh expiry should be calculated

As an example, below is the URL and the request body to this program (Please note that, the payload should be escaped, otherwise, it won’t be a valid JSON):

https://dev.mdcms.ch/mdrstt11/mrdcredxa

```
{
    "clientId":"POSTMANREQ",
    "applicationId":"MDREST4IAPP",
    "usageType":"C",
    "credType":"T",
    "alg_keys":"JWT-HS256",
    "userName":"postmanuser",
    "password":"postmanpwd",
    "payload":"{\"sub\": \"1234567890\", \"name\": \"John Doe\", \"iat\": 1716239022}",
    "clientSecret":"Birds play significant role in human lifecycle",
    "refreshMethod":"*MANUAL",
    "refreshTokenExpiry":80000,
    "timeUnitOfMeasure":"*seconds"
}
```

Below would be the response:

```
{
    "message": "Token Created",
    "authToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNzE2MjM5MDIyfQ.QU8z5oIVdSQjpwACzps81wS2Mq4wm4a1rhxYru0T7sM",
    "refreshToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNzE2MjM5MDIyLCJyZWZyZXNoX3ZhbHVlIjoiUmVmcmVzaF9WYWwifQ.nzaoRjsDD9_ytjw0maE6eQBqYnfeuRcLPsB
dh35886o"
}
```

6.4.2 MRDCREDXA - Issue a new token using RS256 algorithm

To issue a token, use MRDCREDXA with "POST" method. We have to supply below information as part of the request body:

Parameter	**Description**
clientId	This is the first part of the composite key which is the unique identifier to issue and store the token in MRDCRED file
applicationId	This is the second part of the composite key which is the unique identifier to issue and store the token in MRDCRED file
usageTyp	Usage type of the token (A – API, C – Consumer, B – Both, O – Other)
credTyp	Credential Type (B – Basic, T – Bearer, R – Remote)
alg_keys	Depending on the type of token being generated, the value should be sent accordingly. You need to supply the value "JWT-RS256-DCM" in this field.
username	User name associated with the token (not used now)
password	Password corresponding to the user profile above
payload	JSON payload for the token generation. The payload should be escaped as its JSON inside JSON. It must have "exp" for the token validity duration in seconds starting from the token creation timestamp. If "iat" is there, it will be replaced

	with the current epoch timestamp value. Refer https://www.epochconverter.com/
dcmApp	DCM application name which will be used for RS256 algorithm
refreshMethod	Refresh method (e.g. *AUTO, *MANUAL) - Not in use now
refreshTokenExpiry	**An optional value:** Refresh token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
timeUnitOfMeasure	The duration in which the token expiry and refresh expiry should be calculated

As an example, below is the URL and the request body to this program (Please note that, the payload should be escaped, otherwise, it won't be a valid JSON):

https://yourserver/mdrst/mrdcredxa

```
{
    "clientId":"POSTMANREQRS256",
    "applicationId":"MDREST4IAPPRS256",
    "usageType":"C",
    "credType":"T",
    "alg_keys":"JWT-RS256-DCM",
    "userName":"postmanuser",
    "password":"postmanpwd",
    "payload":"{\"sub\": \"1234567890\", \"name\": \"John Doe\", \"iat\": 1716239022}",
    "dcmApp":"MDREST4I_SIGN",
    "refreshMethod":"*MANUAL",
    "refreshTokenExpiry":0,
    "timeUnitOfMeasure":"*seconds"
}
```

Below would be the response:

```
{
    "message": "Token Created",
    "authToken":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNzE2MjM5MDIyfQ.vgokcwJAABhE3kDXXXmZwbiseRV2G7iJ_yd_cO-i86J4N0h1iMY3yMp6rZQRMK5YWV-
AMp44CPOaATg77jbUl8zvy3T685mJbF5MUKrhMyuQIhG2L2NaPTlB-
pkjdEIaquS0XNH0wSeD51B4Y3dKzyk0I15EDJtcsaMcricDIMUh0gCDKjZGBjOJATiMUwpjE_IKT1KsklV8IwtlagZ324J
9EywcxeQ3MSJXAiJVur1gaiG-
X6IbTUq1C7CKenjmhDDCfAbpNMXn_2Q4is4B2RUjCpTLIvUYTl_N3Bcf6mp8VAeVLtRa4R0OCfzdwLqiNewTyGIpQTtXVI
yAiapRlAFowxZc1Ih8MF_3TatDh3tqWpICHypCrjyWcCMRog0noxpmOo4i_jBXSFuIeEo9xv_i6FgAcMEGOF_Ay17dV41l
ZLezpFA_VpMzKYiYOPQRU0JH0a6BqQXROFowyaOJo88r1vjA1q_UHcziexhmKaVX_a-
cfk6AGp0WCoJ4rLtYqwFqCrwltS_GrUQQAjwVEB3YZB2poT-
bPpOmzZQBCkdVx5VoAlVwRuURoASdrFiW4YFeUB4BhXU1B6QyBbFbcQCjDYifVt1lvLb3Z0Fho6JtCLxtEbwSOgqi6_6HP
PTeaFmZeQDOIO2n8568I7zffbyTt-UCYOQT4tmxrPg8i5A"
}
```

6.4.3 MRDCREDXA - Issue a personal access token

To issue a personal access token, use MRDCREDXA with "POST" method. We have to supply below information as part of the request body:

Parameter	Description
clientId	This is the first part of the composite key which is the unique identifier to issue and store the token in MRDCRED file
applicationId	This is the second part of the composite key which is the unique identifier to issue and store the token in MRDCRED file
authToken	You may like to create a personal access token of your choice and in that case, you can supply the string in this parameter. The token creation API will store this as the personal access token. If this parameter is not supplied or supplied as blank, it will generate the 32 byte string as the auth token.
refreshToken	You may like to create a refresh token of your choice and in that case, you can supply the string in this parameter. The token creation API will store this as the personal access token. If this parameter is not supplied or supplied as blank, it will generate the 32 byte string as the auth token.
usageTyp	Usage type of the token (A – API, C – Consumer, B – Both, O – Other)
credTyp	Credential Type (B – Basic, T – Bearer, R – Remote)
alg_keys	Supply the value "PAT" in this parameter
authTokenExpiry	**An optional value:** Auth token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
RefreshTokenExpiry	**An optional value:** Refresh token expiry period(how long before token expires in Expiry Unit(s) If not supplied, this is set to 30 days by default
timeUnitOfMeasure	The duration in which the token expiry and refresh expiry should be calculated

As an example, below is the URL and the request body to this program:

https://dev.mdcms.ch/mdrstt11/mrdcredxa

```
{
    "clientId":"POSTMANREQ8",
    "applicationId":"MDCMSAPP1",
    "authToken":"StuartMilliganPersonalToken20210617",
    "refreshToken": "StuartMilliganRefreshToken20210617",
    "usageType":"C",
    "credType":"T",
    "alg_keys":"PAT",
    "timeUnitOfMeasure":"*seconds",
    "authTokenExpiry": 3600,
    "refreshTokenExpiry": 15552000
}
```

Below would be the response:

```
{
    "message": "Token Created",
    "authToken": "StuartMilliganPersonalToken20210617",
    "expiry": "2023-01-02-10.18.26.630000",
    "refreshToken": "StuartMilliganRefreshToken20210617",
    "refreshTokenExpiry": "2023-07-01-09.18.26.630000"
}
```

However, if we supply below request body (i.e. not supplying auth token):

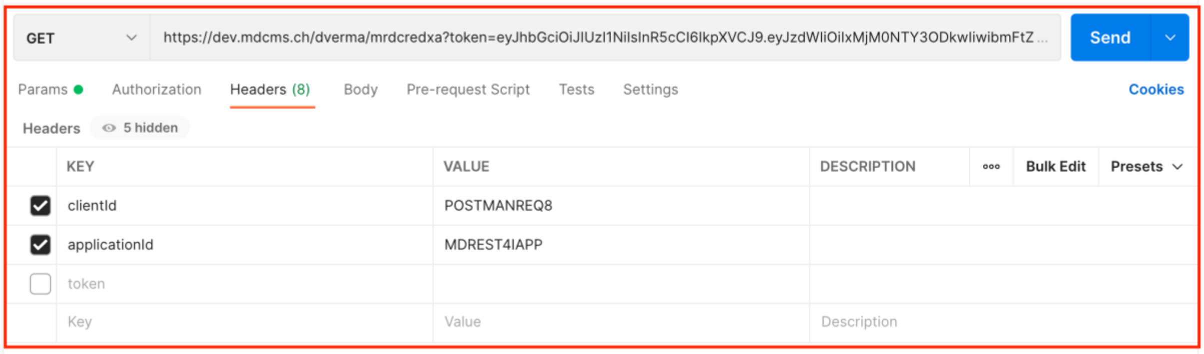
```
{
    "clientId":"POSTMANREQ8",
    "applicationId":"MDCMSAPP1",
    "usageType":"C",
    "credType":"T",
    "alg_keys":"PAT",
    "timeUnitOfMeasure":"*seconds",
    "authTokenExpiry": 3600,
    "refreshTokenExpiry": 15552000
}
```

We get below response:

```
{
    "message": "Token Created",
    "authToken": "MzIwOTEzMDgwNzEyMjk3ODc0Mjc5NDg2",
    "expiry": "2023-01-02-10.49.34.098000",
    "refreshToken": "MzE0Nzg4NzMwMzE2ODk3MDM4MDU4MzM2",
    "refreshTokenExpiry": "2023-07-01-09.49.34.098000"
}
```

6.4.4 MRDCREDXA - Validate the token using client Id and application ID

There are two ways we can validate a token through MRDCREDXA. The first one is using "GET" method and second by "POST" method. In "GET" method, we have to supply the auth token and optionally the client Id and application ID either as query parameters or as http headers. This is shown in below screenshot where for example the auth token is supplied as query parameter and remaining two fields have been supplied via http headers:



GET https://dev.mdcms.ch/dverma/mrdcredxa?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZ ...

Headers (8)

KEY	VALUE	DESCRIPTION
☑ clientId	POSTMANREQ8	
☑ applicationId	MDREST4IAPP	
☐ token		

You may also supply the client Id and application Id in exceptional case when you have created two entries of the same token for different client Id and application Id. If supplied, the combination of token with the client and application Id will be searched, otherwise, only the token will be searched in DB file.

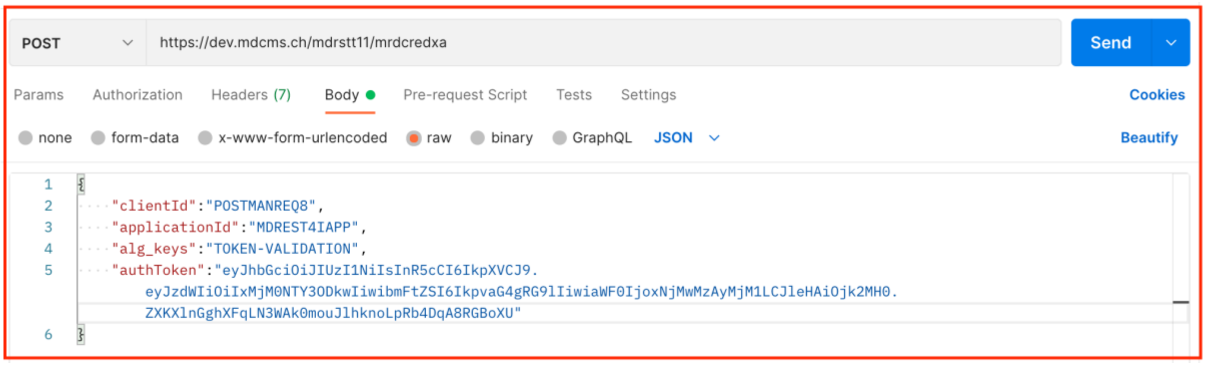
Below is the response to this request. The other responses could be "Token Invalid" or "Token Expired".

```
{
    "message": "Token is valid",
}
```

Below are the other possible error messages. When these messages appear, the response code is set to 403.

```
Error: Invalid Authentication
Error: Invalid Token
Error: Token Expired
```

You may also use POST request to validate the token and in that case, you have to supply "alg_keys" as "TOKEN-VALIDATION" and "authToken". You may supply clientId and applicationId if there is possibility of more than one entries existing with the same token value.



POST https://dev.mdcms.ch/mdrstt11/mrdcredxa

Body: raw JSON

```
{
    "clientId":"POSTMANREQ8",
    "applicationId":"MDREST4IAPP",
    "alg_keys":"TOKEN-VALIDATION",
    "authToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNjMwMzAyMjM1LCJleHAiOjk2MH0.
        ZXKXlnGghXFqLN3WAk0mouJlhknoLpRb4DqA8RGBoXU"
}
```

The response will be same as in case of GET.

6.4.5 MRDCREDXA - Update JWT information

We can use the program MRDCREDXA with "PATCH" method. We have to supply the values to be updated along with the Client Id/Application Id or auth token. The first priority is given to client Id and Application Id but if either of them is blank, the next search is made with the token.

Below is an example URL to this request:

https://yourserver/mdrst/mrdcredxa

Here is the example requestBody payload:

```
{
    "clientId":"POSTMANREQ6",
    "applicationId":"MDREST4IAPP",
    "authtoken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNjIxODY2NTQyLCJleHAiOjk2MH0.WpsKn_5FVZrGPUSR8LHe-RA2NJ-g5Axa9XNNwVadr08",
    "usageType":"C",
    "credType":"T",
    "alg_keys":"JWT-HS256",
    "userName":"postmanuser1",
    "password":"postmanpwd2",
    "authServer":"",
    "authPortNumber":0,
    "tokenServer":"",
    "tokenPortNumber":0,
    "refreshMethod":"*AUTO",
    "refreshExpiry":600,
    "timeUnitOfMeasure":"*seconds"
}
```

Below would be the response:

```
{
    "message": "Entry Updated successfully"
}
```

The other possible error message is below:

```
"message": "Entry not found for supplied client/app Id or token"
```

6.4.6 MRDCREDXA - Delete Credentials Store Entry

We can use the program MRDCREDXA with "DELETE" method. We have to supply the token as the query parameter. We can also supply the client Id and application Id. The first priority is given to client Id and Application Id but if either of them is blank, the next search is made with the token which is a mandatory query parameter.

Below is the URL to this request:

```
https://dev.mdcms.ch/dverma/mrdcredxa?clientId=POSTMANREQ6&applicationId=MDREST4IAPP&token=eyJ
hbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ijox
NjIxODY2NTQyLCJleHAiOjk2MH0.WpsKn_5FVZrGPUSR8LHe-RA2NJ-g5Axa9XNNwVadr08
```

Below would be the response:

```
{
    "message": "Token deleted from database successfully"
}
```

The other possible error message is below:

```
"message": "Token not found"
```

6.4.7 MRDCREDXB - Retrieve the token using client Id and application ID

We can use the REST API MRDCREDXB with “GET” method with two query parameters (i.e. clientId and applicationId).

Below is the URL to this request:

https://yourserver/mdrst/mrdcredxb?clientId=POSTMANREQ6&applicationId=MDREST4IAPP

Below is the response:

```
{
    "authToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNjIxODg3NDkyLCJleHAiOjk2MH0.UhExAux7LX7mLttk2mJomGgaS6cDFoN-gE53DN8fBKs",
    "refreshToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNjIxODg3NDkyLCJleHAiOjk2MCwicmVmcmVzaF92YWx1ZSI6IlJlZnJlc2hfVmFsIn0.DEpnbVPfrIvfbjgACJkWyp
8kv_lrO5XjixOzvjqjRQw"
}
```

6.4.8 MRDCREDXB - Refresh JWT token

We can use the REST API MRDCREDXB with “POST” method. We have to supply the values to be updated along with the Client Id/Application Id or auth token. The first priority is given to client Id and Application Id but if either of them is blank, the next search is made with the token.

This API will update the Credentials Store record for this token by using the current date as the start date or “iat” in the payload and then update the token and refresh tokens using the stored expiry period and expiry unit from the Credential Store accordingly

Below is an example URL to this request:

https://yourserver/mdrst/mrdcredxb

requestBody payload:

```
{
    "clientId":"POSTMANREQ6",
    "applicationId":"MDREST4IAPP",
    "usageType":"C",
    "credType":"T",
"refreshToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNzE2MjM5MDIyLCJyZWZyZXNoX3ZhbHVlIjoiUmVmcmVzaF9WYWwifQ.nzaoRjsDD9_ytjw0maE6
eQBqYnfeuRcLPsBdh35886o"
}
```

Below would be the response:

```
{
    "message": "Token Refresh Successful",
    "authToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNjIxODg4NzA0fQ.i_-Mq2M4NvsNgkgqW2y5Y6URVyhMsqd5g3KdVldZlCg",
    "refreshToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNzE2MjM5MDIyLCJyZWZyZXNoX3ZhbHVlIjoiUmVmcmVzaF9WYWwifQ.nzaoRjsDD9_ytjw0maE6eQBqYnfeuRcLPsB
dh35886o"
}
```

If the refresh token is invalid or expired, relevant messages will be displayed.

6.4.9 MRDCREDXM - Retrieve all fields from MRDCREDX

We can use the REST API MRDCREDXM with "GET" method to get all fields from MRDCREDX file. We can also use this with two query parameters (i.e. clientId and applicationId).

Below is the URL to this request:

https://yourserver/mdrst/mrdcredxm

or

https://yourserver/mdrst/mrdcredxm?clientId=POSTMANREQ7&applicationId=MDREST4IAPP

6.4.10 MRDCREDXM – Insert or update credentials into MRDCREDX

We can use MRDCREDXM API with "PUT" HTTP method to add or insert token details as per supplied clientId and applicationId in the request body. If supplied clientId and applicationId will be available in the MRDCREDX (Token Credential store) file, it will update the record of MRDCREDX file. If supplied clientId and applicationId will not be available in MRDCREDX file, API will create new record in the MRDCREDX (Token Credential store) file.

Below is an example URL to this request:

https://yourserver/mdrst/mrdcredxm

requestBody payload:

```
{
      "clientId": "POSTMANREQ7",
      "applicationId": "MDREST4IAPP",
      "usageType": "A",
      "credType": "T",
      "extendedOpt": "S",
      "alg_keys": "MAT",
      "authToken": "StuartMilliganManualaccesstoken20221114",
      "userName": "Jhon",
      "password": "jhonpassword",
      "payload": " eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IllvdXJOYW1lIiwiaWF0IjoxNj
Y0NTI2MTEwLCJleHAiOjM2MDB9",
      "clientSecret": "This is my test secret for manual access token",
      "publicKey": "test public key",
      "privateKey": "test privatekey",
      "authServer": "MDRDEMOD",
      "authPortNumber":2525,
      "tokenServer": "MDRDEMOD",
      "tokenPortNumber": 111,
      "refreshMethod": "A",
      "refreshToken": "Testrefresh token",
      "authTokenExpiry": 3600,
      "refreshTokenExpiry": 3600,
      "timeUnitOfMeasure": "*seconds",
      "tokenIssuer": "Stuart Milligan",
      "issueUserjob": "MDRDEMOD/QTMHHTTP/510173",
      "updateUserjob": "MDRDEMOD/QTMHHTTP/510173"
}
```

Below would be the response:

When supplied clientId and applicationId will not been available in the MRDCREDX file:

```
{
    "message": "Token Created",
    "authToken": "StuartMilliganManualaccesstoken20221114"
}
```

When clientId and applicationId will be available:

```
{
    "message": "Entry Updated successfully"
}
```

6.4.11 MRDCREDXM - Delete Credentials Store

We can use MRDCREDXM REST API with "DELETE" HTTP method. We have to supply the clientId and applicationId as the query parameters for deleting the entry from the MRDCREDX (Token Credential store) file.

Below is the URL to this request:

https://yourserver /mdrst/mrdcredxm?clientid=MDREST4IAPP&appid=MDREST4IAPP

Below would be the response:

```
{
      "message": "Token deleted from database successfully"
}
```

The other possible error message is below:

```
{
      "message": "Token not found"
}
```

7 MDRest4i Coding Standards

In order to make it easy to understand various things in MDRest4i, the product has predefined coding standards which have been used across both the product and the demo application. You will find these coding standards in MDRest4i sources (i.e. templates, copybooks and programs) under MDRST library as well as in the TIDEMO library. Standards make the code easier to read and understand, by differentiating between local and global variables, between different types of variables and declarations. This prevents any ambiguity in the situations where the global variables are imported from one of the MDRest4i modules/service programs. Below are the details about the different types of objects/variables and the pattern of their definition:

Type of Entity	Global Scope (within module)	Local procedure scope	Global Scope (Import/Export)	Comments
Work Field	W_	WL_	WG_	
Array	R_	RL_	RG_	
Terminal Index	I_	IL_	IG_	
Indicators	N_	NL_	NG_	
Constants	C_	CL_	CG_	
Data Structures	D_	DL_	DG_	Could be normal DS or array of DS

DS Subfields	**S_**	**SL_**	**SG_**	**Could be normal subfield or an array**
Parameters	**P_**	**PL_**	**N/A**	**Could be normal variable, data structure or an array**
Pointers	**T_**	**TL_**	**TG_**	
Subroutine Names	**Z_**	**ZL_**		
Procedure names	**Any name except starting with Z**			
Copybooks	**Any logical name**			
DSPF File Field(DDS)	**DD**			
Printer File Field(DDS)	**DP**			
RPG Programs	**LXR**			
CL Programs	**LXC**			
Data Areas	**LXD**			
Physical File Name	**LX**			
Logical File Name	**Ending with "L"**			
Record Format	**FileName+F or similar**			
Physical/Logical file fields	**Any two char prefix uniform to that file based on file content**			

8 MDRest4i Standard Logging REST API using a Physical File (MDRLOGS)

In this section we describe how to log the request and response details from a REST service in a database file - MDRLOGS.

8.1 MDRLOGS – MDRest4i Logging Database File

By default, this file stored in MDRSTXXXX library. It is updated during the execution of the API or consumer. To use a different library ensure a version of the file is above the MDRST in the library list when executing the MDRREADDQ program or Consumer.

Below is the structure of this file:

Parameter	Attributes	Description
ENTSEQ	INTEGER	This is the first field for maintaining the sequence no.
ENTUID	CHAR(36)	Entry Unique ID. This value is created at the beginning of the execution of a REST API or Consumer program. It is created using the following algorithm: “SRV” if its server “CNS” if it’s Consumer + jobno + time stamp when API/consumer starts executing
OBJNAME	CHAR(10)	Here we maintain the object name.
REQTYPE	CHAR(1)	In this field we maintain the request type. “S” for REST services and “C” for consumer programs.
ACTION	CHAR(10)	In this field we maintain the action. (ReqHDR, ReqBDY, RspHDR, RspBDY, ReqQRY, REQHDRCUST).
TIMESTAMP	TIMESTAMP	In this field we are adding the current timestamp value.
DATA	VARCHAR(32500)	In this field stored the data of request headers, request body, response headers, response body and query parameters.
CONTN	CHAR(1)	In this field we maintain the continuation. If data is greater than (32500) bytes, we write here “Y” that’s mean pending data available in the next sequence and if data is less than 32500 here we will set the value as “N”.

8.2 Logging flow for a REST API Program

To reduce performance degradation of a REST API that logs to MDRLOGS database file, logging details are captured and written in two separate programs.

If the indicator ng_Logfile = *On, the logging details(as specified in data area MDRST/ MDRLOGFLAG) are gathered during execution, and written to a dataQ (if not specified in the z_Custominit subroutine, it uses MDRST/MDRLOGDQ by default) as a single string.

A separate batch program “MDRST/MDRREADDQ” reads the dataQ and writes the records out to the MDRLOGS file in the library list of the batch job.

8.3 Components

These components are all pre-supplied in library MDRST, but be created and used specifically depending on requirements

Component	Description
MDRLOGS	**Physical File** – used to hold individual logging elements for REST APIs and REST Client/Consumer programs. See definition above
MDRLOGDQ	**DataQ** - used to hold logging details from REST API execution. Pre-Supplied in MDRST Default Creation command CRTDTAQ DTAQ(MDRST/MDRLOGDQ) MAXLEN(16000) SEQ(*KEYED) KEYLEN(52)

	SIZE(*MAX2GB 10) AUTORCL(*YES) TEXT('MDRest4i API Logging Data Queue') The two mandatory settings for creation of this dataQ are: **SEQ(*KEYED) KEYLEN(52)** – they are hardcoded into the MDRest4i logic and cannot be changed The key used for each message in the dataQ is made up of three parts: • **pos 1-36** – key that will be used in the MDRLOGSF made up of: SRV + Job number + Timestamp • **pos 37 –** hardcoded dash "-" • **pos 38-47** – name of REST API program used as object name in MDRLOGSF • **pos 48**-52 – counter to indicate a single record or multiple records for a API log detail * if all data is in one dtaq record, then this is set as 00000 * if the logging string exceeded the dtaq max record size, this = 00001 upwards * pos 6-100 = Alpha numeric value which is the unique ID number: SRV + Job number + Timestamp. for example: "SRV254397_2022-06-10-12.57.56.490000-BIKEAPIL 00000"

MDRLOGFLAG	**Data Area** – used to specify what details are saved in MDRLOGS file
	
	Here are the options as defined by this data area
	`d s_reqpath 1 1`
	`d s_reqmth 2 2`
	`d s_qryparm 3 3`
	`d s_reqbody 4 4`
	`d s_reqtohost 5 5`
	`d s_reqfclient 6 6`
	`d s_remoteuser 7 7`
	`d s_srvaddr 8 8`
	`d s_srvname 9 9`
	`d s_srvport 10 10`
	`d s_rsphdr 11 11`
	`d s_rspdta 12 12`
	`d s_rspbody 13 13`
	`d s_reqcusthdr 13 13`
	
	Here are some example commands that can be used to set the data area:
	
	* Most Typical
	`CHGDTAARA DTAARA(MDRST/MDRLOGFLAG (1 13)) VALUE(YYYYNNNNNNYYY)`
	
	* No JSON bodies
	`CHGDTAARA DTAARA(MDRST/MDRLOGFLAG (1 13)) VALUE(YYYNNNNNNNYYN)`
	
	* No JSON request bodies
	`CHGDTAARA DTAARA(MDRST/MDRLOGFLAG (1 13)) VALUE(YYYNNNNNNNYYY)`
	
	* No JSON RESPONSE bodies
	`CHGDTAARA DTAARA(MDRST/MDRLOGFLAG (1 13)) VALUE(YYYYNNNNNNYYN)`
	
	* Maximum setting for API log all aspects of API
	`CHGDTAARA DTAARA(MDRST/MDRLOGFLAG (1 13)) VALUE(YYYYYYYYYYYYY)`

MDRREADDQ	**RPG Program –** Program used to read dataQ entries created by REST API and write as individual records in MDRLOGS file Parameters: DATAQ – 10 *CHAR – (**MANDATORY**) dataQ to be read LIB – 10 *CHAR – (**MANDATORY**) library where the above dataQ is found WAIT – Number – (**OPTIONAL**) specify number of seconds it will wait before checking if above dataQ has messages on it. If not specified the default wait time is 5 seconds **Notes:** • This program can be submitted multiple times in the same batch Subsystem to speed up dataQ reads • The source for this program can be found in MDRST/QCUSTOMSRC • PF MDRLOGS **MUST** be in *LIBL when this program runs

8.4 Variables used to enable logging to MDRLOGS

These variables are all pre-supplied and set with default values in MDRST/QRPGLESRC.MDRESTDFN copy book.
If used they must be set in the z_Custominit subroutine of the REST API program

Variable	Description
ng_logFile	**Indicator** - Sets logging the request and response detail for the API into the MDRLOGS file
w_dqname	10A – dataQ logs will be written to If not specified defaults to MDRLOGDQ
w_dqlib	10A – library containing above dataQ If not specified defaults to *LIBL
w_dqLen	5P 0 – Maximum length of each message. If the string created by logging exceeds this value then it wraps. Notes: • If not specified it defaults to 1600 • If set to a number lower than the size of the string of gathered logging detail, it will wrap and write multiple messages to the dataQ using a counter in the dataQ key to differentiate records • If multiple messages are used to write a single execution log details on the dataQ, the counter in the key starts with 00001. Otherwise a default value of 00000 is used where a single dataQ message was used to write the details for a specific API execution

8.5 Procedures

Procedure	Description
logDqNow()	BY default the dataQ is written to at the end of the execution of the rest API. To add a logging entry to the end of the log entries currently loaded into memory, **AND** force the instant writing to the dataQ of the entries already loaded into memory use this procedure.

It accepts two parameters:
Action – 10A – This appears in the MDRLOGS.ACTION column
Data – 1024 – This appears in the MDRLOGS.DATA column

This procedure can be found in copybook MDRST/QRPGLESRC.LXRRESTP and is added at t end of all REST API's

A typical usecase is to add a timestamp after all parsing is complete before business logic is called in a z_ProcPOST subroutine. For example:

```
logDqNow('reqTime': %char(%timestamp()));
```

produces this rest in MDRLOGS:

```
S      ReqBDY   2022-07-28-18.52.37.339000  {"shipmentGid":"GUES
T.0100
S      reqTime  2022-07-28-18.52.37.339000  2022-07-28-18.52.37.
337060
S      RspHDR   2022-07-28-18.52.37.342000  Content-Type:applica
tion/j
```

writeLogsF()

This writes a logging entry for MDRLOGSF into memory. Each subsequent call to this procedure adds an entry at the end of the previous entry into a pointer in memory(tg_logdta).

It accepts three parameters:
Action – 10A – This appears in the MDRLOGS.ACTION column
Data – 1024A – This appears in the MDRLOGS.DATA column
DataLength – 10i 0 – Length of the value in Data parameter above

This is used by MDRest4i to add the values specified in data area MDRLOGFLAG. For example here is the logic used in MDRST/QRPGLESRC.LXRRESTC to write the POST request body entry:

```
if ng_Logfile =  *on and s_reqbody = 'Y';
  WriteLogsF('ReqBDY':w_readStr:w_rc);
endif;
```

This procedure is in module LXRGLOBAL and bound into all REST API's and Consumers. Here is an example of the data written for a POST method REST API:

```
HDR@@@@ReqPTH@@@@HDR/BIKEAPIL/12345HDR@@@@ReqHDR@@@@HDRRequest
Method:POSTHDR@@@@ReqQRY@@@@HDRdept=carsHDR@@@@ReqBDY@@@@HDR{"s
hipmentGid":"GUEST.01008","transactionNumber":4872089,"tenderTy
pe":"Ordinary","tenderedAction":"A","servprovGid":"GUEST","prog
ressStatus":"C"}HDR@@@@RspHDR@@@@HDRContent-Type:application/js
on; charset=UTF-8HDR@@@@RSPSTS@@@@HDR200-OKHDR@@@@RspBDY@@@@HDR
{    "shipmentGid": "GUEST.01008",     "transactionNumber": 1234
5,    "tenderType": "Ordinary",    "tenderedAction": "A",    "s
ervprovGid": "GUEST",    "progressStatus": "C"}
```

PrcLogDta()	This procedure writes the contents tg_logDta to the dataQ. It has no parameters.

8.6 Useful data queue utilities

8.6.1 Display current messages on a data queue

```
    SELECT CURRENT_MESSAGES FROM QSYS2.DATA_QUEUE_INFO
    WHERE DATA_QUEUE_LIBRARY = 'MDRST' AND DATA_QUEUE_NAME = 'MDRLOGDQ';
```

8.6.2 Display the Attributes of a data queue

```
    SELECT DATA_QUEUE_LIBRARY,
        DATA_QUEUE_NAME,
        MAXIMUM_MESSAGE_LENGTH,
        SEQUENCE,
        KEY_LENGTH,
        INCLUDE_SENDER_ID,
        INITIAL_MESSAGE_ALLOCATION,
        SPECIFIED_MAXIMUM_MESSAGES,
        FORCE,
        AUTOMATIC_RECLAIM
        FROM QSYS2.DATA_QUEUE_INFO
        WHERE DATA_QUEUE_NAME = 'MDRLOGDQ'
            AND DATA_QUEUE_LIBRARY = 'MDRST'
            AND DATA_QUEUE_TYPE = 'STANDARD'
```

8.6.3 Display the messages on a data queue

```
SELECT
DATA_QUEUE,
ORDINAL_POSITION,
MESSAGE_ENQUEUE_TIMESTAMP,
CAST(MESSAGE_DATA AS VARCHAR(1600)),
CAST(KEY_DATA AS VARCHAR(60))
FROM TABLE( QSYS2.DATA_QUEUE_ENTRIES (
DATA_QUEUE => 'MDRLOGDQ',
DATA_QUEUE_LIBRARY => 'MDRST' )) A
ORDER BY ORDINAL_POSITION
```

8.6.4 Clear all messages in a data queue

```
CALL QCLRDTAQ ('MDRLOGDQ'  'MDRST')
```

9 Logging Consumer in the DB file (MDRLOGS)

If you would like to enable logging in database file, set the variable “ng_logfile” to *On in “Initialize” procedure of your consumer. When this indicator is set ON, the outgoing request as well as incoming responses are logged directly into the MDRLOGS file.

10 MDRest4i Standard Logging using the IFS

10.1 Logging the REST Service using the IFS

If you would like to log the incoming request and outgoing response from the API created using MDRest4i, set the variable “n_saveIFSSwitch” to *On in “z_CustomInit” subroutine of your program running as REST service. When this indicator is set ON, the incoming request as well as outgoing responses are logged. The default log directory is ‘/MDREST4i/logs/” but if you want to log separate APIs in some specific IFS folder, set the variable “w_saveIFSPath” with the folder name like below but it should have complete path including the file name:

```
w_saveIFSPath = ‘/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt’;
```

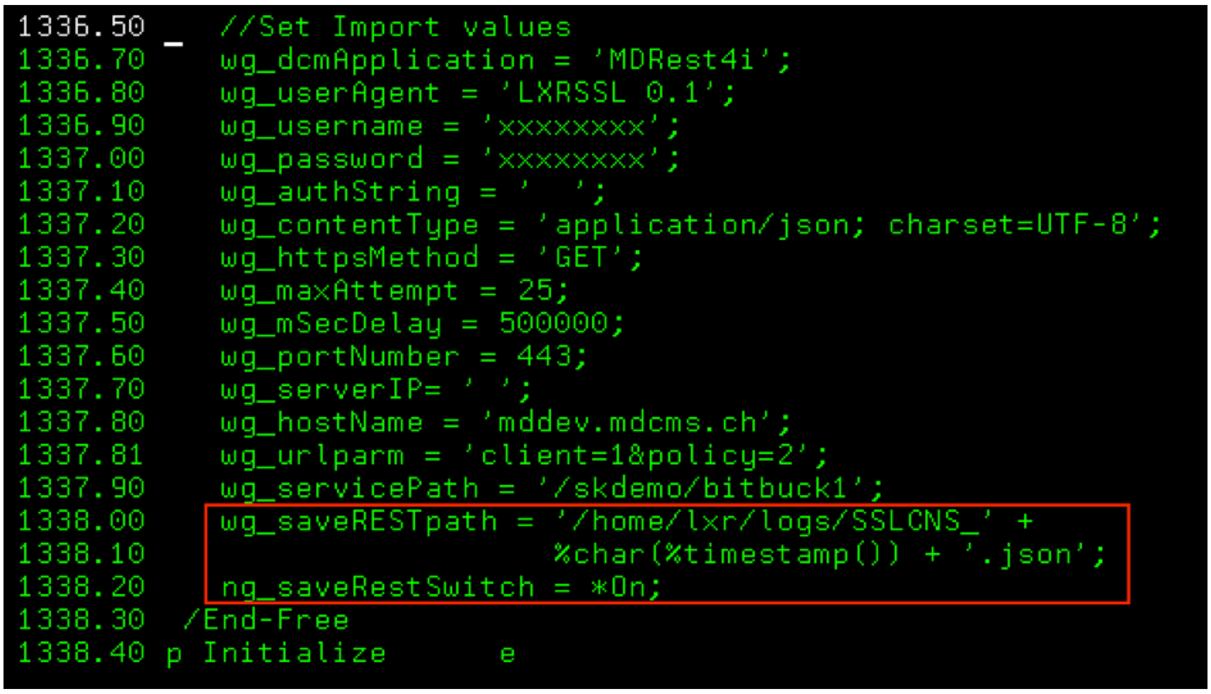
If “w_saveIFSPath” is not initialized, the logic picks the default path from the data area “DFTLOGDIR” under library list (i.e. from MDRST if it’s in library list). The data area is shipped with the value “/MDREST4i/logs/”. If for some reason, the variable “w_saveIFSPath” is blank and the data area DFTLOGDIR is not found in library list (e.g. MDRST is not in library list), the hard-coded directory “/home/” is considered to creates the IFS logs. The name of the log file is “LOGLOGS_” followed by timestamp followed by “.txt”. Below is an example of how the log file appears. The file name is “/MDREST4i/logs/LXRLOGS_2019-04-01-02.07.32.417000.txt”.

```
###########  REQUEST START  ################
 ###########  REQUEST HTTP-HEADER ################
 Request to Host Address= mddev.mdcms.ch:2517
 Request from client= 1.22.72.106
 Server Address= 10.15.1.34
 Server Name= mddev.mdcms.ch
 Server Port= 2517
 Request Method :POST
 Query Parameters :
 ##########  REQUEST BODY BEGIN  ##############
{
 "genmbrname": "CUSTSMM",
 "gensrcf": "QRPGLESRC", "genlib":"LXRDEMO201"
}
 ########## REQUEST BODY END ################
########### REQUEST END ################
########### RESPONSE BEGIN ################
###########  RESPONSE HEADER ################
Content-Type: application/json; charset=UTF-8
Status: 200 OK
LXRPBGNKEY:
LXRPENDKEY:
LXRPAGETYP:
LXRNBRRCD:0
 ########### RESPONSE HEADER END ################
 ########### RESPONSE DETAIL BEGIN ################
{
"SrcMbr": "CUSTSMM",
"SrcFile": "QRPGLESRC",
"SrcLib": "LXRDEMO201",
"ObjLib": "LXRDEMO201",
"Description": "A description of the program as it executes for th",
"Compilation": "Success"
}
```

```
 ########### RESPONSE END ################
```

10.2 Logging the request/response in REST Consumer

While working with the MDRest4i REST consumers (SSL or non SSL) bound via MDRCNSM, the configuration settings are performed in “INITIALIZE” procedure. This procedure is used in MDRCNSM and LXRSOAPCM for REST and SOAP services respectively. You can set the indicator “ng_saveRestSwitch” to *On and the variable “wg_saveRESTPath” should be set to the path of IFS file including the file name (e.g. below):



```
    //Set Import values
    wg_dcmApplication = 'MDRest4i';
    wg_userAgent = 'LXRSSL 0.1';
    wg_username = 'xxxxxxxx';
    wg_password = 'xxxxxxxx';
    wg_authString = '   ';
    wg_contentType = 'application/json; charset=UTF-8';
    wg_httpsMethod = 'GET';
    wg_maxAttempt = 25;
    wg_mSecDelay = 500000;
    wg_portNumber = 443;
    wg_serverIP= ' ';
    wg_hostName = 'mddev.mdcms.ch';
    wg_urlparm = 'client=1&policy=2';
    wg_servicePath = '/skdemo/bitbuck1';
    wg_saveRESTpath = '/home/lxr/logs/SSLCNS_' +
                      %char(%timestamp()) + '.json';
    ng_saveRestSwitch = *On;
  /End-Free
p Initialize      e
```

Below is the output in IFS file:

```
###REQUEST##START###

Attempting Non-SSL connection to the target
###REQUEST##HTTP-HEADER###

Request Length = 220

 ###REQUEST##DETAILS###

GET http://mddev.mdcms.ch:2517/devend1/CLM_DET?client=1&policy=1&claim=4
HTTP/1.1

Accept-Encoding: deflate

Host: mddev.mdcms.ch.com:2517

Connection: Keep-Alive

User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

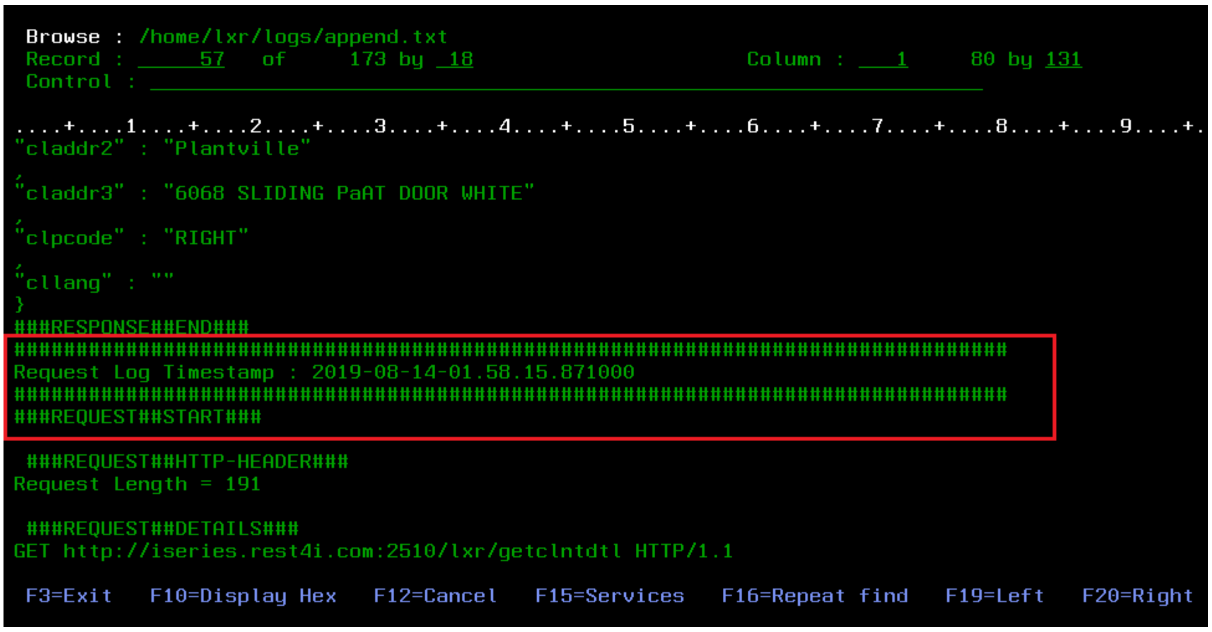
```
 ###REQUEST##END###

 ###RESPONSE##START###
Response Status = 200
Response Length = 715

 #RESPONSE#HTTP-HEADER#
HTTP/1.1 200 OK
Date: Mon, 01 Apr 2019 06:53:59 GMT
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: LXRPBGNKEY,LXRPENDKEY,LXRPAGETYP,LXRNBRRCD
Content-Length: 387
Keep-Alive: timeout=300, max=100
Connection: Keep-Alive
Content-Type: text/json; charset=ISO-8859-1



#RESPONSE#JSON-BODY#



{"claim": [
{
"client": "1",
"policy": "1",
"claimNumber": "4",
"claimRegistration": "AA01BB GP",
"claimIncidentDate": "2016-05-18",
"claimDescription": "Fit fallen thorn tree.",
"claimTyresDmg": "2",
"claimStatus": "O",
"claimAmount": "2300.00",
"claimAmountPaid": ".00",
"claimAuthDate": "2001-01-01",
"claimDatePaid": "2001-01-01",
"claimSupplier": "LS001",
"claimRepReason": ""
}
]}
```

```
###RESPONSE##END###
```

10.2.1 Appending to same IFS log file –

We have an exported indicator “ng_appendIfsF” which is by default switched to *Off. You can declare this with “import” keyword in the program calling “GskConsume” procedure. When this indicator is set to *On, the IFS file is opened in append mode and the consumer will then log all the requests in the same IFS file.

If append mode is requested, the first attempt is made to open the file in append mode but if the file doesn’t exist, append request is ignored and file is opened/created normally. Below is how the appended log looks like.



```
 Browse  : /home/lxr/logs/append.txt
 Record  :       57    of      173 by  18                       Column :    1      80 by 131
 Control :

....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+.
"claddr2" : "Plantville"
,
"claddr3" : "6068 SLIDING PaAT DOOR WHITE"
,
"clpcode" : "RIGHT"
,
"cllang" : ""
}
###RESPONSE##END###
################################################################################
Request Log Timestamp : 2019-08-14-01.58.15.871000
################################################################################
###REQUEST##START###

 ###REQUEST##HTTP-HEADER###
Request Length = 191

 ###REQUEST##DETAILS###
GET http://iseries.rest4i.com:2510/lxr/getclntdtl HTTP/1.1

 F3=Exit   F10=Display Hex   F12=Cancel   F15=Services   F16=Repeat find   F19=Left   F20=Right
```

10.3 Logging the request/response in SOAP Consumer

While working with the MDRest4i REST consumers, the configuration settings are done in “INITIALIZE” procedure like REST consumers but here the variables are different. The indicator to enable logging the request is “ng_saveXMLSwitch” and the variable for the IFS Path is “wg_saveXMLPath”. The processing of IFS logs along with its output remains the same as in above case of REST consumers explained above.

11 Custom Logging

MDRest4i has been built with logging and troubleshooting in mind and provides both a system as well as user controlled logging setup. This allows for the logging of the incoming service requests as well as the requests processed via consumer programs systematically by switch. This is easily extended according to your requirement by using the user controlled logging method. This section covers the user controlled logging for Consumers and Services built with MDRest4i

11.1 REST Consumer Logging

This is the custom logging in a DB2 PF for a REST Service

11.1.1 LXRCLTLOG File

The standard procedures used in the consumer programs generate the logs of the Outbound and Incoming requests in LXRCLTLOG file in MDRST library. This is enabled by setting the "ng_customlog" indicator to *ON. This indicator is import type declaration in the SOAP and REST consumer template based programs. Below is the content in this file:

Seq	Fields
1	Client Log ID
2	Client Log Occurrence timestamp
3	Client Log Application name
4	Client Log Destination Address
5	Client Log Service Type
6	Client Log Level --(1=CRIT,2=SEVR,3=ERR,4=WARN,5=INFO,6=DBG)
7	Client Log Message Details
8	Client Log Running Time

11.1.2 Field Details:

Client Log ID: This is running number incremented by 1 for next event.

Client Log occurrence timestamp: Timestamp of the event logged.

Client Log Application name: The name of consumer application program making service requests

Client Log Destination Address: The URL of the service request from the consumer.

Client Log Service Type: Type of the service e.g. REST (JSON) SOAL(XML).

Client Log Level: Logged message level/severity

Client Log Message Details: Details about the activity e.g. nnn bytes received, xxx error occurred.

Client Log Running time: Duration of consumer execution including service request/response.

11.2 REST Producer/Service Logging

This is the custom logging in a DB2 PF for a REST Service

11.2.1 LXRSRVLOG File

The standard procedures used in the consumer programs generate the logs of the Outbound and Incoming requests in LXRSRVLOG file in MDRST library. This is enabled by setting the "ng_customlog" indicator to *ON. This indicator is import type declaration in the SOAP and REST server template based programs. Below is the content in this file:

Seq	Fields
1	Server Log ID
2	Server Log Occurrence timestamp
3	Server Log Application name
4	Server Log Destination Address
5	Server Log Service Type
6	Server Log Level (1=Critical, 2=Severe, 3=Warning, 4=Info, 5=Debug)
7	Server Log Message Details
8	Server Log Running Time

11.2.2 Field Details:

Server Log ID: This is running number incremented by 1 for next event.

Server Log occurrence timestamp: Timestamp of the event logged.

Server Log Application name: The name of consumer application program making service requests

Server Log Destination Address: The URL of the service request from the consumer.

Server Log Service Type: Type of the service e.g. REST (JSON).

Server Log Level: Logged message level/severity

Server Log Message Details: Details about the activity e.g. nnn bytes received, xxx error occurred.

Server Log Running time: Duration of consumer execution including service request/response.

12 MDRest4i Exception Handling

12.1 Indicators and general Info

MDRest4i has inbuilt exception handling for everything. The exception handling is enabled by default but the indicators have been provided to control this. The copybook member LXRERRPRO has the prototype for the procedure “LogException” defined in LXRERRH module. This module is added to LXRGLOBAL binding directory and therefore no additional binding directory is required to use the exception handling. All the exceptions encountered either in the modules or in MDRest4i service copybook logic are logged in LXERRLOG file.

12.2 Exception Handling in Providers

By default, the exception handling is enabled in a provider generated using MDRest4i.

12.2.1 LXRRESTC Monitor Function

The LXRRESTC copybook (found in MDRST/QRPGLESRC) controls drives the flow of the MDRest4i Provider. It uses the RPG Monitor function two catch any exception lower down in the stack – provided the exception happens within the original boundary. For example any procedure bound in to the provider, that crashes with an exception, will return control to the Monitor statement in LXRRESTC, and the on-error function will be triggered. A called program will not be handled as it is outside the boundary of the Provider program.

1. If the on-error function is triggered, the following occurs:
2. The initial value of an error message with the exception type, number and data(if possible) is set,
3. The procedure LogCriticalError is called, passing this initial message part. LogCriticalError is customizable and is provided in MDRST/QCUSTOMSRC.
4. The error message is then written as a JSON object with a severity level of 30, as a response in JSON format using the CritErrJson() function.
5. The HTTP Status and Reason are set in LogCriticalError() procedure
6. The subroutine z_prereturn in LXRRESTC, then sends this with the JSON error message as a response to the client request.

From LXRRESTC:

```
Monitor;
  exsr lxrrestcSR;
on-error;
// Error details captured and sent to logging procedure
  critErrorMsg = s_sdsexctyp + s_sdsexcnbr + '-' + %trim(s_sdsexcdta)
  critErrorMsg = LogCriticalError(critErrorMsg);
  if n_startjson = *Off;
    StartJSON(*on:*on);
  endif;
  CritErrJson(critErrorMsg: 30);
  exsr z_prereturn;
  CleanUpMem();

  *Inlr = *On;
  Return;
endmon;
```

lxrrestcSR contains the entire execution stack of a Provider enabling the encapsulation of any exceptions within its boundary in the Monitor.

12.2.2 LogCriticalError Customizable Function

This customizable module/procedure is available in MDRST/QCUSTOMSRC. It does the following:

1. Extract Program and Job Info from PSDS
2. Create the complete Error message for the JSON response
3. Writes the error message extracted in copybook LXRRESTC , and any additional exception info to the LXERRLOG file, **irrespective of logging status**
4. Set the HTTP Status code and Reason text
5. Return control to the Provider(LXRRESTC)

12.2.3 Exception Log File

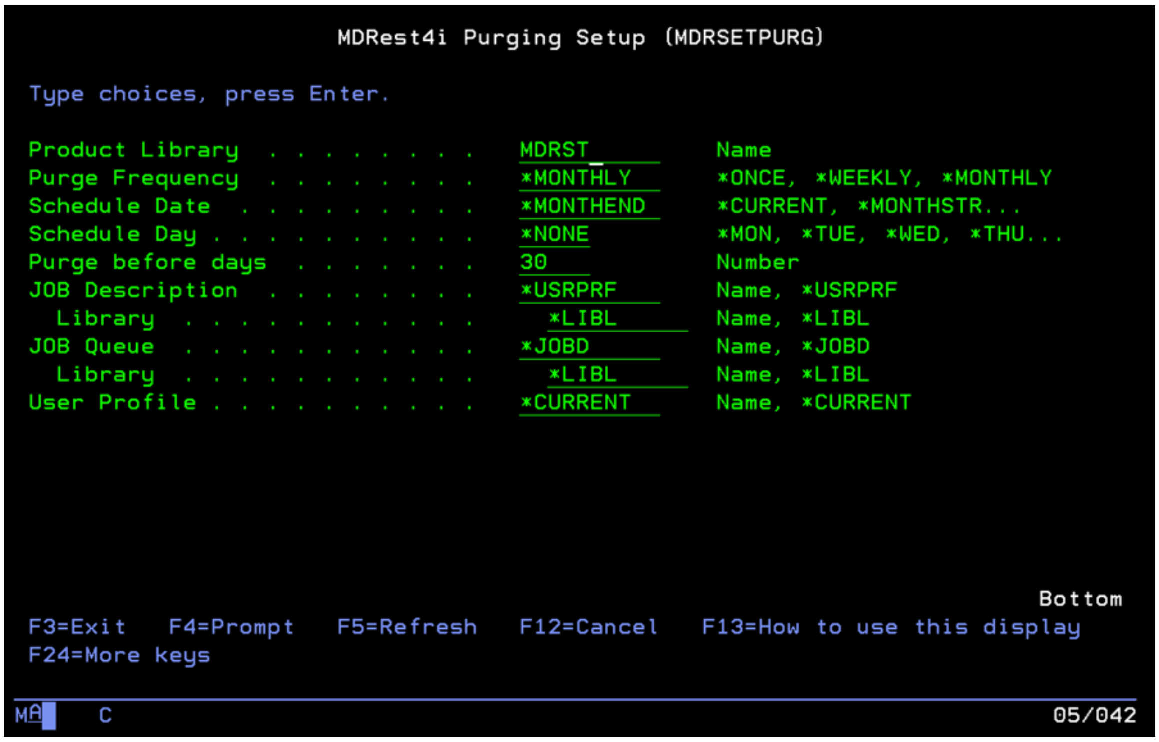
The exceptions are logged in “LXERRLOG” file in MDRST library. The module LXRERRH has this file in user open mode.

By default, the logging happens in MDRST library and the library name is fetched from “LXRPRDDA” data area under MDRST library.

The global variable “wg_Prddalib” is exported from LXRERRH module and it is available as “import” declaration in MDRESTDFN copybook. If you want the logs to be written to some non default MDRST library, just initialize this variable with the library name in “z_CustomInit” subroutine of your API. The file by default can contain up to 10020000 entries and if the entries are added after this count, it will throw an exception. The purging function (explained below) can be used to periodically clear the exception logs table.

12.3 Purging the Exception Logs (using IBM Job Schedular)

The command MDRSETPURG allows to schedule the purging of this file at regular intervals. This command adds a job in the IBM job scheduler such that it can run at specified frequency and removes the older records from LXERRLOG file. The number of records deleted in a specific execution instance are written to the same log file LXERRLOG.



```
                      MDRest4i Purging Setup (MDRSETPURG)

 Type choices, press Enter.

 Product Library . . . . . . . .   MDRST_        Name
 Purge Frequency . . . . . . . .   *MONTHLY      *ONCE, *WEEKLY, *MONTHLY
 Schedule Date . . . . . . . . .   *MONTHEND     *CURRENT, *MONTHSTR...
 Schedule Day . . . . . . . . . .  *NONE         *MON, *TUE, *WED, *THU...
 Purge before days . . . . . . .   30            Number
 JOB Description . . . . . . . .   *USRPRF       Name, *USRPRF
   Library . . . . . . . . . . .     *LIBL       Name, *LIBL
 JOB Queue . . . . . . . . . . .   *JOBD         Name, *JOBD
   Library . . . . . . . . . . .     *LIBL       Name, *LIBL
 User Profile . . . . . . . . . .  *CURRENT      Name, *CURRENT




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

MA      C                                                              05/042
```

The Product library is mandatory entry and should be the library name of the product e.g. MDRST. The purge frequency can be specified as monthly, weekly or once. Scheduled date could be *CURRENT, *MONTHSTR, *MONTHEND or *NONE. You can also schedule the job on required day of the week.The parameter “Purge before days” is set by default to 30 which means delete the exception log entries older than 30 days but it can be changed. Rest are the standard parameters.

13 HTTP Headers in MDRest4i

13.1 Using In-Bound Headers in the HTTP Request for a Provider

The procedure “GetHdr” can be called to retrieve the value of the required HTTP header. The prototype of this procedure is defined in MDRESTDFN copybook. More details can be found in sub-section “REST Provider/Handling” under the section “MDRest4i Function/Variable reference”.

13.2 Adding HTTP Headers to a MDRest4i API Provider Response

The custom headers can be added by calling “AddHdr” procedure supplying first parameter as the header name and second parameter the header value. More details can be found in sub-section “REST Provider/Handling” under the section “MDRest4i Function/Variable reference”.

14 MDRest4i Data Area settings

The MDRest4i product makes use of some data areas for the settings/behaviour which would vary at different customer site depending on the requirement. As an example, the MDRest4i generators which generate the source member using the swagger need to know the source file and library where the source members should be created. Likewise, the IFS log directory would be different for different customers. Such aspects are therefore controlled by data areas and the user can make the relevant changes after going through this part of the documentation.

14.1 LXRSRCDTL

Library: MDRST

Description: This data area holds the source file and library name where open API generator should produce the API requested from the swagger API generator as the post http request.

15 Upgrade V8 and V11 to V12 Programs

There are three approaches to upgrading existing code that can be adopted:

- Rebuild the API or consumer in V12 and manually copy in your business logic code
- Manually edit your existing code to the V12 standards (copy books, binding directories etc) and recompile over latest version of MDRST
- Use the MDRUPDVER command to automate the changes required for V12 plus any manual edits required after the automated upgrade processes (MDRUPDVER and MDRFIXIDX commands) have been run

In general the following structural changes have been made:

1. The module LXRSMSTGJ is changed to MDRCNSM and the copybook LXRSMSTGJP is renamed to MDRCNSP. This affects the existing consumer programs where "LXRSMSTGJP" copybook has to be replaced with MDRCNSP copybook. The entry of module LXRSMSTGJ has been replaced by MDRCNSM in the binding directory and therefore no action with regards to the module name changes.

2. Binding directories are combined together to form only two binding directories. LXRGLOBAL contains everything except the modules specific to consumer processing and BNDZIP contains the modules specific to consumer processing. You are therefore required to add lesser number of binding directories in the REST services or consumer programs. The existing binding directories are not changed and therefore your existing programs shouldn't have any impact of this change.

3. MDRest4i now uses YAJL and as a result the CallBack functionality has been deprecated in both Consumers and Producers. This means the CallBack subroutines and Procedures can now be deleted from your code. If you need CallBack type functionality please contact Midrange Dynamics and we would be happy to help you create new structures using YAJL Tree functions

4. In YAJL arrays start at 1 and not 0 when extracting values from JSON. So the "`for w_idx1 = 0 to ….`"loops in your code that use jPathvV jPathN, JPathE, JPathU, a fix command MDRFIXIDX has been written. See the section below on how to run this to automatically fix these statements in your code.

5. If in the REST APIs or consumer program added definition of w_idx1 index in D spec, please remove this. Because it is already defined in the MDRESTDFN copybook.

6. YAJL for JSON writing and parsing uses a service program: MDRJSONY which can be found in the library MDRST. Therefore, MDRST must be in the JOB Libl whenever you execute and API or Consumer program.
7. If the APIs and consumer programs using "JGet" to get the value from JSON, it will be replaced with 'JPathV'.

8. JPATHV no longer returns *notFound when a JSON key is not fond in the payload. It returns blanks

9. w_reqbody is no longer available as a variable in the REST Producer template. It has been replaced with a procedure:
   `getReqBody(yourVariable)`
   Its accepts three parms, but only the first is mandatory. It must be called in one of the method subroutines and the mandatory parameter must be defined as 5242880 AlphaNumeric.
   For more details on this function see the 18.4.8 GetReqBody section of this guide

15.1 Update REST APIs and consumer programs to V12

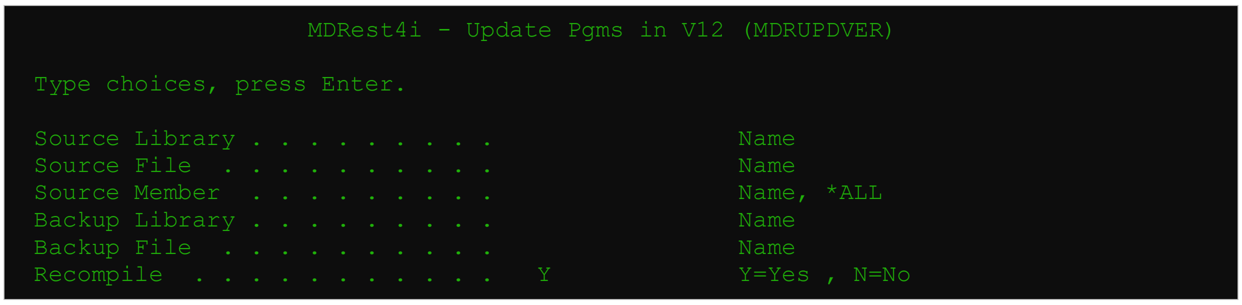
15.1.1 MDRUPDVER Command

We can use this command for updating the REST APIs and consumer programs to using V12 instead of V8 or V11. As per below:

From the IBMi command line execute the following:

- ADDLIBLE MDRST *& press enter*
- MDRUPDVER & press F4

The following screen will appear:



```
                    MDRest4i - Update Pgms in V12 (MDRUPDVER)

 Type choices, press Enter.

 Source Library . . . . . . . . .                  Name
 Source File  . . . . . . . . . .                  Name
 Source Member  . . . . . . . . .                  Name, *ALL
 Backup Library . . . . . . . . .                  Name
 Backup File  . . . . . . . . . .                  Name
 Recompile  . . . . . . . . . . .    Y             Y=Yes , N=No
```

Parameter	Description
Source Library	Where the REST APIs and consumer program available.
Source File	The source file where the program source is available.
Source Member	There are two functions If we want to update only one program, we need to provide here the program name. If we want to update all programs of given source file, provide the value as "*ALL".
Backup Library	Specify the library name for making a backup before updating in V12.

`Backup File`	Specify the Source file name for making the backup before updating in V12. It supplied source file not available then it will create.
`Recompile`	Here we can define Y or N. If we want to recompile the program after update, set the value Y and If we don't want to recompile the program, we need to set N. By default, set is Y.

To see the effects of this command, please have a look at sections 15.3 and 15.4 To see before and after changing the code of REST APIs and consumer programs.

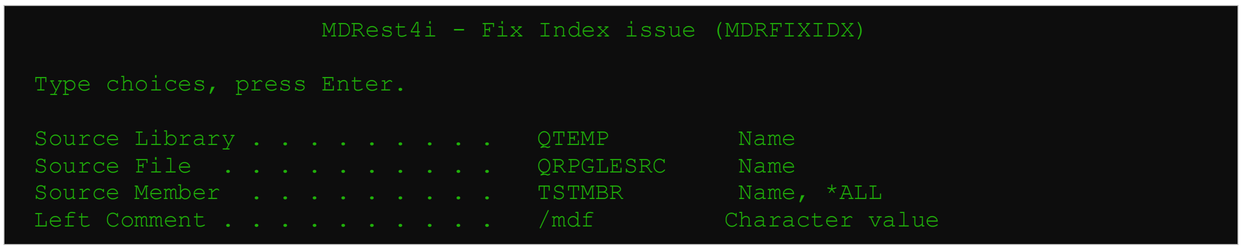
15.2 Fix "for" loops in when extracting JSON values with JPath…

15.2.1 MDRFIXIDX Command

From the IBMi command line execute the following:

- ADDLIBLE MDRST *& press enter*
- MDRFIXIDX & press F4

The following screen will appear:

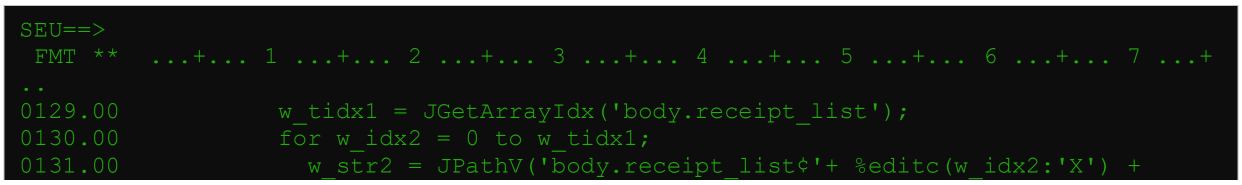


```
                         MDRest4i - Fix Index issue (MDRFIXIDX)

 Type choices, press Enter.

 Source Library . . . . . . . . .   QTEMP          Name
 Source File  . . . . . . . . . .   QRPGLESRC      Name
 Source Member  . . . . . . . . .   TSTMBR         Name, *ALL
 Left Comment . . . . . . . . . .   /mdf          Character value
```

Parameter	Description
`Source Library`	Where the REST APIs and consumer program available.
`Source File`	The source file where the program source is available.
`Source Member`	There are two functions If we want to fix only one program, we need to provide here the program name. If we want to fix all programs of given source file provide here value as "*ALL"
`Left Comment`	If we want to add left comment in program where this command will make the changes.

After running command, it will change program code where "for" loop will be executing with "w_idx1 = 0"and recompile. It will change code like below.

Code before change:



```
SEU==>
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+
..
                   w_tidx1 = JGetArrayIdx('body.receipt_list');
                   for w_idx2 = 0 to w_tidx1;
                     w_str2 = JPathV('body.receipt_list¢'+ %editc(w_idx2:'X') +
            '!.warehouse_code');
```

Code after changed by MDRFIXIDX command:

```
          w_tidx1 = JGetArrayIdx('body.receipt_list');
          // Changed by MDRFIXIDX at 2022-11-16-08.15.03.627000
/mdf      //for w_idx2 = 0 to w_tidx1;
/mdf      for w_idx2 = 1 to w_tidx1;
            w_str2 = JPathV('body.receipt_list¢'+ %editc(w_idx2:'X') +
            '!.warehouse_code');
```

15.3 V8 to V12 – Manual edits to existing code

15.3.1 Consumer Code

To upgrade the V8 Consumer programs to V12, please apply below changes

- The following binding directories will be replaced:

```
h bnddir('CGIBNDDIR':'QC2LE':'LXRGLOBAL': 'LXRJSON': 'LXRSSJ')
h bnddir('LXRLANG': 'LXRIFS': 'BNDZIP':'LXRXML')
```

With

```
h bnddir('LXRGLOBAL': 'BNDZIP')
```

- The following copybooks must be replaced:

```
 /copy qrpglesrc,lxrglobalc
 /copy qrpglesrc,lxrjsonc
 /copy qrpglesrc,lxrxmlc
 /copy qrpglesrc,lxrifspro
 /copy qrpglesrc,lxrvariant
 /copy qrpglesrc,lxrsmstgjp
```

With

```
 /copy qrpglesrc,mdrestdfn
```

- Prototype definition of CallBack should be removed:

```
d CallBack        pr
d p_event                       10a   const
d p_label                      200a   varying const
d p_value                     1024a   varying const
d p_stackLevel                  10i 0 const
d p_jsonPath                  1024a   varying const
```

This code that executes CallBack must also be removed.

```
   tg_CallbackPointer=%paddr(Callback);
```

This code:

```
w_str2 = JPathV('title');
if w_str2 <> '*notFound';
  d_model1.s_title = JPathV('title');
endif;
```

Is replaced with this code:

```
w_str2 = JPathV('title');
if w_str2 <> *Blanks;
```

```
  d_model1.s_title = JPathV('title');
endif;
```

This code:

```
   // Cleanup the memory allocated dynamically from heap
   JPathvClean();
```

Is replaced with this code:

```
   // Cleanup the memory allocated dynamically from heap
   cleantree();
```

The CallBack procedure code must be removed.

```
p CallBack        b                   export
d CallBack        pi
d  p_event                      10a   const
d  p_label                     200a   varying const
d  p_value                    1024a   varying const
d  p_stackLevel                 10i 0 const
d  p_JsonPath                 1024a   varying const

d wl_inspect      s           1024    varying
d wl_inspect1     s           1024    varying

 /Free

   wl_inspect = JRemoveX(p_JsonPath);
   select;
   //Start of JSON Object - "{"
   when p_event = 'startObj';
   //End  of JSON Object - "}"
   when p_event = 'endObj';
     if wl_inspect = 'xxxxxx/yyyyyy';
       //Write your processing logic here
     endif;
   //Start of JSON Array - "¢"
   when p_event = 'startArr';
   //End  of JSON Array - "!"
   when p_event = 'endArr';
   //Assignment of Value in JSON Pair - "LABEL": "VALUE"
   when p_event = 'assign';
   //Next Element in JSON Array - ","
   when p_event = 'next';
   endsl;

 /End-Free
p CallBack        e
```

Add the following code at the bottom part of Initialize procedure before /end-free statement of this procedure:

```
 // If you want to load config via MRCNSDTLF file, please uncomment
 // the below line. In this case you need to make sure you have loaded
 // (Host, Port, path etc.) in the MRCNSDTLF file with the key of This
 // Program name. Add below file in the 'F' spec in this program
 // mrcnsdtlf if    e            k disk    Usropn
 // /copy qrpglesrc,mrsetcon
```

Replace this line of code:

```
p InitVariant     b                   Export
```

With :

```
p InitVariant     b
```

15.3.2 Producer Code

To upgrade the V8 producer programs to V12, please apply below changes.

- The following binding directories will be replaced:

```
h bnddir('CGIBNDDIR':'QC2LE':'LXRGLOBAL': 'LXRJSON': 'LXRSSJ')
h bnddir('LXRLANG': 'LXRIFS': 'BNDZIP':'LXRXML')
```

With

```
h bnddir('LXRGLOBAL': 'BNDZIP')
```

First, we need to update the copybooks. The following copybooks **MUST** be deleted:

```
/copy qrpglesrc,LXRRESTD
/copy qrpglesrc,lxrcgipro
/copy qrpglesrc,lxrglobalc
/copy qrpglesrc,lxrapierr
/copy qrpglesrc,lxrusproto
/copy qrpglesrc,lxrifspro
/copy qrpglesrc,lxrsds
/copy qrpglesrc,lxrjsonc
/copy qrpglesrc,lxrlangc
/copy qrpglesrc,lxrxmlc
```

And replaced with this line, on the line before the lxrrestc copybook and AFTER the define statemenst as below:

```
/define LXR_CustomInit

/define LXR_ProcGET

/copy qrpglesrc,mdrestdfn
/copy qrpglesrc,lxrrestc

/undefine LXR_CustomInit

/undefine LXR_ProcGET
```

These line below will be replaced:

```
    // When activation group of the pgm is not *NEW, comment below stmt
    n_actgrpNew = *on;

    n_saveIFSSwitch = *off;
    ng_mdrJobHdr = *off;
    n_extractQryPrms = *off;
```

With this code:

```
    // Use this IFS switch to log the API data in an IFS Folder
    // If you set n_saveIFSSwitch to *on; Then set the log filepath e.g:
    // w_saveIFSPath='/MDREST4i/myfolder/mycustomlogs/LogFileddmmyyyy.txt'
    n_saveIFSSwitch = *off;

    // Add HTTP header x-mdrsrvjob: MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn
    // where nnnnnn is the CGI job number, and MYSERVERJOB is HTTP Instanc
```

```
       ng_mdrJobHdr = *off;

       // Extracts request query parameters to d_Qparm data structure
       n_extractQryPrms = *off;
```

If the REST APIs have query parameters, we need to made changes in the z_setMethod Subroutine’s code.

Example before changing:

```
begsr z_setMethod;
  // Required and Optional Parameters:
  w('{"Get": "Mandatory Parms: id"}');
  n_ParmError = *on;
Endsr;
```

After changing:

```
begsr z_setMethod;
  // Required and Optional Parameters:
  beginObject(' ');
  addChar('Get':'Mandatory Parms: id');
  endObject(' ');
  n_ParmError = *on;
Endsr;
```

This code:

```
w_str2 = JPathV('title');
if w_str2 <> '*notFound';
  d_model1.s_title = JPathV('title');
endif;
```

Is replaced with this code:

```
w_str2 = JPathV('title');
if w_str2 <> *Blanks;
  d_model1.s_title = JPathV('title');
endif;
```

The CallBack procedure can be removed:

```
    // ====================================================================
    //    Callback Procedure to bifurcate JSON input
    // ====================================================================

p Callback        b
d Callback        pi
d   pl_event                     10a   const
d   pl_label                           like(w_labelD) const
d   pl_value                           like(w_valueD) const
d   pl_stackLvl                  10i 0 const
d   pl_jsonPath                        like(w_jPathD) const
d wl_inspect      s              20
  /free
    wl_inspect = JRemoveX(pl_jsonPath);
    select;
    when pl_event = 'startObj';
    when pl_event = 'assign';
    when pl_event = 'endObj';
    when pl_event = 'next';
    when pl_event = 'startArr';
    when pl_event = 'endArr';
    endsl;
  /end-free
p Callback        e
```

Please also remove the following two lines:

```
/define LXR_Callback
```

and

```
/undefine LXR_Callback
```

In the Some producers or consumer programs have available bellow's definitions. So please remove them also.

```
 * Import the data elements from bound modules
d r_jsonPath      s            1024    import dim(16000)
d r_jsonValue     s            1024    import dim(16000)
d wg_lastlbl      s             200a   import
d wg_lastval      s            2048a   import
```

15.4 V11 to V12 – Manual edits to existing code

15.4.1 Consumer Code

To upgrade the V11 Consumer programs to V12, please apply below changes

First, we need to update the copybooks. The following copybooks **MUST** be deleted:

```
/copy qrpglesrc,lxrglobalc
/copy qrpglesrc,lxrjsonc
/copy qrpglesrc,lxrxmlc
/copy qrpglesrc,lxrifspro
/copy qrpglesrc,lxrvariant
/copy qrpglesrc,lxrsmstgjp
```

And replaced with:

```
/copy qrpglesrc,mdrestdfn
```

The following lines must be removed:

```
d CallBack        pr
d p_event                        10a   const
d p_label                       200a   varying const
d p_value                      1024a   varying const
d p_stackLevel                   10i 0 const
d p_jsonPath                   1024a   varying const
```

and:

```
  tg_CallbackPointer=%paddr(Callback);
```

This code:

```
w_str2 = JPathV('title');
if w_str2 <> '*notFound';
  d_model1.s_title = JPathV('title');
endif;
```

Is replaced with this code:

```
w_str2 = JPathV('title');
if w_str2 <> *Blanks;
```

```
  d_model1.s_title = JPathV('title');
endif;
```

The following code:

```
   // Cleanup the memory allocated dynamically from heap
   JPathvClean();
```

Is replaced with this code:

```
   // Cleanup the memory allocated dynamically from heap
   cleantree();
```

Below code will be removed from the Program:

```
p CallBack        b                   export
d CallBack        pi
d  p_event                      10a   const
d  p_label                     200a   varying const
d  p_value                    1024a   varying const
d  p_stackLevel                 10i 0 const
d  p_JsonPath                 1024a   varying const

d wl_inspect      s           1024    varying
d wl_inspect1     s           1024    varying

 /Free

   wl_inspect = JRemoveX(p_JsonPath);
   select;
   //Start of JSON Object - "{"
   when p_event = 'startObj';
   //End  of JSON Object - "}"
   when p_event = 'endObj';
     if wl_inspect = 'xxxxxx/yyyyyy';
       //Write your processing logic here
     endif;
   //Start of JSON Array - "¢"
   when p_event = 'startArr';
   //End  of JSON Array - "!"
   when p_event = 'endArr';
   //Assignment of Value in JSON Pair - "LABEL": "VALUE"
   when p_event = 'assign';
   //Next Element in JSON Array - ","
   when p_event = 'next';
   endsl;

 /End-Free
p CallBack        e
```

15.4.2 Producer Code

To upgrade the V11 producer programs to V12, please apply below changes:

First, we need to update the copybooks. The following copybooks can be deleted:

```
 /copy qrpglesrc,LXRRESTD
 /copy qrpglesrc,lxrcgipro
 /copy qrpglesrc,lxrglobalc
 /copy qrpglesrc,lxrapierr
 /copy qrpglesrc,lxrusproto
 /copy qrpglesrc,lxrifspro
 /copy qrpglesrc,lxrsds
 /copy qrpglesrc,lxrjsonc
```

```
/copy qrpglesrc,lxrlangc
/copy qrpglesrc,lxrxmlc
/copy qrpglesrc,lxrbitsc
```

And replaced with this line, on the line before the lxrrestc copybook and AFTER the define statemenst as below:

```
/define LXR_CustomInit

/define LXR_ProcGET

/copy qrpglesrc,mdrestdfn
/copy qrpglesrc,lxrrestc

/undefine LXR_CustomInit

/undefine LXR_ProcGET
```

If the REST APIs have query parameters, we need to made changes in the z_setMethod Subroutine.

Example before changing:

```
begsr z_setMethod;
  // Required and Optional Parameters:
  w('{"Get": "Mandatory Parms: id"}');
  n_ParmError = *on;
Endsr;
```

After changing:

```
begsr z_setMethod;
  // Required and Optional Parameters:
  beginObject(' ');
  addChar('Get':'Mandatory Parms: id');
  endObject(' ');
  n_ParmError = *on;
Endsr;
```

If the REST APIs using ‘JGet’ function to get the value from the JSON, we need to replace it with ‘JPathV’. As per below:

Before changing code:

```
d_s.lx_clidno = JGet('clidno');
```

After

```
d_s.lx_clname =JPathV ('clname');
```

This code:

```
w_str2 = JPathV('title');
if w_str2 <> '*notFound';
  d_model1.s_title = JPathV('title');
endif;
```

Is replaced with this code:

```
w_str2 = JPathV('title');
if w_str2 <> *Blanks;
  d_model1.s_title = JPathV('title');
endif;
```

Removed all of the CallBack procedure code below:

```
  // ====================================================================
   //    Callback Procedure to bifurcate JSON input
   // ====================================================================

p Callback        b
d Callback        pi
d   pl_event                    10a   const
d   pl_label                          like(w_labelD) const
d   pl_value                          like(w_valueD) const
d   pl_stackLvl                 10i 0 const
d   pl_jsonPath                       like(w_jPathD) const
d wl_inspect      s             20
 /free
   wl_inspect = JRemoveX(pl_jsonPath);
   select;
   when pl_event = 'startObj';
   when pl_event = 'assign';
   when pl_event = 'endObj';
   when pl_event = 'next';
   when pl_event = 'startArr';
   when pl_event = 'endArr';
   endsl;
 /end-free
p Callback        e
```

This must be removed:

```
 /define LXR_Callback
```

This line must also be removed:

```
 /undefine LXR_Callback
```

In the Some producers or consumer programs have available bellow's definitions. So please remove them also.

```
 * Import the data elements from bound modules
d r_jsonPath      s           1024    import dim(16000)
d r_jsonValue     s           1024    import dim(16000)
d wg_lastlbl      s            200a   import
d wg_lastval      s           2048a   import
```

Usage of w_reqBody must be replaced with a procedure call.

So the following code:

```
myVar = w_reqbody
```

Will be replaced with this:

```
d myVar        s        5242880A
……
getReqBody(myVar);
```

it MUST be placed in the boundary of the z_procPOST/PUT/PATCH subroutine/s

16 MDCMS Interface

16.1 Overview

An MDCMS Level corresponds to an MDREst4i Documenter Environment folder.

The MDCMS developer folder for an OAPI Specification corresponds to the MDrest4i SDK folder for that developer user.

MDRest4i produces two artefacts:

1. An OpenAPI specification, stored in the developer IFS folder of the MDRest4i SDK server instance. Each MDRest4i SDK user has their own folder for saving specs. For example a spec called "myapi" created by SDK user "laxadmin", would be stored here:
   /www/mdrst/specs/cons/lxradmin/myapi.json
   where "mdrst" is the instance of MDRest4i installed, and "lxradmin" is the user who created the OAPI spec.

2. The generated source code from the OAPI Spec, which is generated into the source file/library specified by the user in the "Paths" tab of SDK User Interface.

Once the user creates the OAPI spec, and then generates the RPG code from this in the SDK UI, the user can then submit an object request to MDCMS. The interface takes care of adding an object request for both the generated RPG source code AND the OpenAPI specification.

From then on MDCMS controls the promotion of both artefacts using the RFP.

The first promotion will remove the OAPI Specification from the SDK user's Developer folder, and move it to the SDK Documenter folder on the same server on the IBMi. Each MDRest4i Documenter environment has its own folder for saving OAPI specs and HTML documentation for that spec. For example a spec called "myapi" promoted to environment "RestTest" by MDCMS, would be stored here:

/www/mdrst/specs/docu/RestTest/myapi.json

where "mdrst" is the instance of MDRest4i installed, and RestTest is the MDRest4i Documenter environment key.

When the OPAI Spec and source code are promoted to the next MDCMS level by an RFP, the OPAI spec will be removed from the previous level and added to the next MDCMS level which will have an MDRest4i Documenter Environment folder level associated with it in the MDCMS Attribute

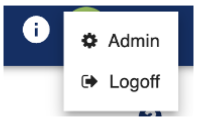
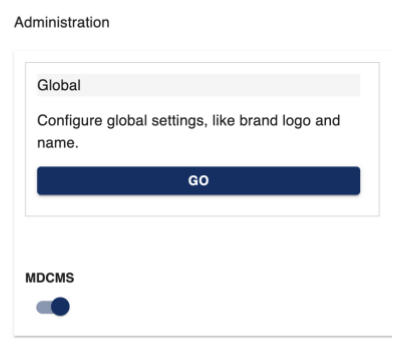
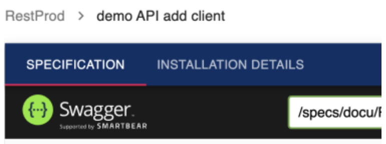
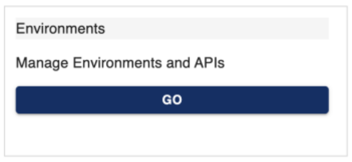
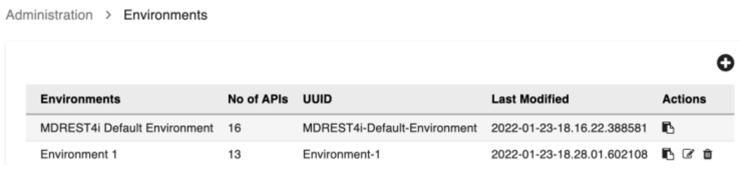
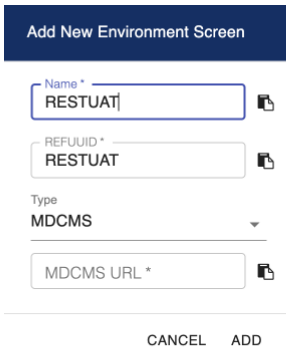
To modify an exsiting OPAI specification not in a SDK users development t folder, but in another Documenter Environment folder, the user can opt to "import from Documenter" option in the MDRest4i SDK. Ans so the cycle can begin again.

16.2 Setup Steps

1. Add library "MDRST" to any MDCMS level JobD where an API or Consumer will be compiled/promoted to
2. In SDK Documenter, create an MDRest4i Documenter Environment for each MDCMS level
3. Create an MDCMS attribute(for example OAPI) for type *IFS for each level/environment
4. Create a "Developer Library Naming Template" in MDCMS for the MDRest4i SDK console
5. Add the command MDRPROM as a command to each MDCMS lvl/attribute
6. IN SDK Console "Edit Site" dialogue, select "Object Required" from "MDCMS Values" section of dialogue

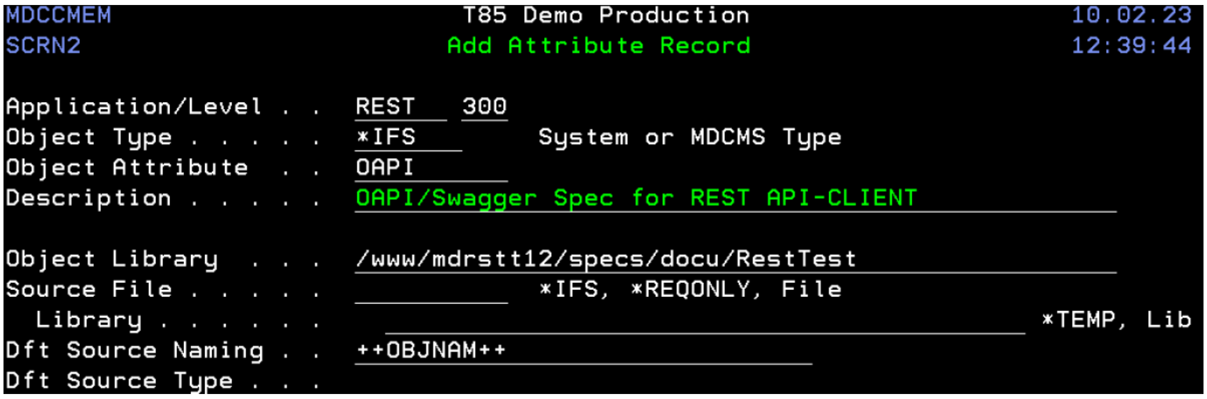
16.2.1 Create an Environment in SDK Documenter

Step	Comments

Login as an administrator to MDRest4i Documenter	http://youribmi:yourport/docu/
From clicking the user avatar on the top right-hand side, select Admin	
From the Administration screen switch on the MDCMS option	This makes the "Installation Details" tab appear for a Specification, which are added by MDCMS when the OSAPI is promoted. Administration Global Configure global settings, like brand logo and name. GO MDCMS RestProd > demo API add client SPECIFICATION INSTALLATION DETAILS Swagger /specs/docu/
From the Administration screen select the Environments "Go" button	Environments Manage Environments and APIs GO
From the Environments list screen, select the ⊕ icon on the top right hand side.	Administration > Environments Environments | No of APIs | UUID | Last Modified | Actions MDREST4i Default Environment | 16 | MDREST4i-Default-Environment | 2022-01-23-18.16.22.388581 Environment 1 | 13 | Environment-1 | 2022-01-23-18.28.01.602108
Create a Name for the environment. The **REFFUID** value should be noted/copied and saved, for the next step in MDMCS. Select Type as MDCMS. *MDCMS URL is for future use* Select Add to save the Environment details	Add New Environment Screen Name * RESTUAT REFUUID * RESTUAT Type MDCMS MDCMS URL * CANCEL ADD

16.2.2 Create MDCMS Attribute for OAPI Specifications

In MDCMS Add an Attribute



```
MDCCMEM                     T85 Demo Production                   10.02.23
SCRN2                       Add Attribute Record                  12:39:44

Application/Level . .  REST    300
Object Type . . . . .  *IFS         System or MDCMS Type
Object Attribute  . .  OAPI
Description . . . . .  OAPI/Swagger Spec for REST API-CLIENT

Object Library  . . .  /www/mdrstt12/specs/docu/RestTest
Source File . . . . .             *IFS, *REQONLY, File
  Library . . . . . .                                          *TEMP, Lib
Dft Source Naming . .  ++OBJNAM++
Dft Source Type . . .
```

Parameter	Description
Application/Level	As per your current setup
Object Type	*IFS
Object Attribute	Any name that seems appropriate
Description	Any suitable description
Object Library	This must correlate to the appropriate MDRest4i Documenter Environment folder copied or noted in setup steps above

16.2.3 Create “Developer Library Naming Template” in MDCMS for SDK Console

In MDCMS create a new “Developer Library Naming Template” for the appropriate application and level, according to your setup.

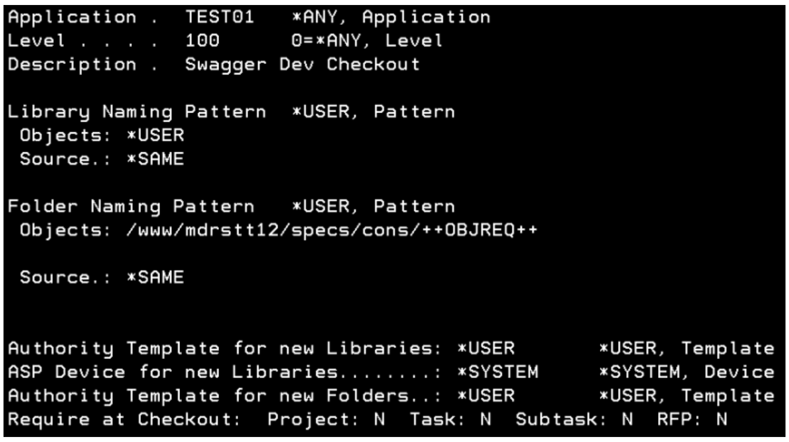
The only value required that isn’t part of a default setting weh adding a new template of this type, is:

“Folder Naming Pattern” for objects only. Everything else can be left as default

Use the following value:

/www/mdrst/specs/cons/++OBJREQ++

where “mdrst” is the instance of MDRest4i installed



```
Application .  TEST01    *ANY, Application
Level . . . .  100       0=*ANY, Level
Description .  Swagger Dev Checkout

Library Naming Pattern  *USER, Pattern
 Objects: *USER
 Source.: *SAME

Folder Naming Pattern   *USER, Pattern
 Objects: /www/mdrstt12/specs/cons/++OBJREQ++

 Source.: *SAME


Authority Template for new Libraries: *USER       *USER, Template
ASP Device for new Libraries........: *SYSTEM     *SYSTEM, Device
Authority Template for new Folders..: *USER       *USER, Template
Require at Checkout:  Project: N  Task: N  Subtask: N  RFP: N
```

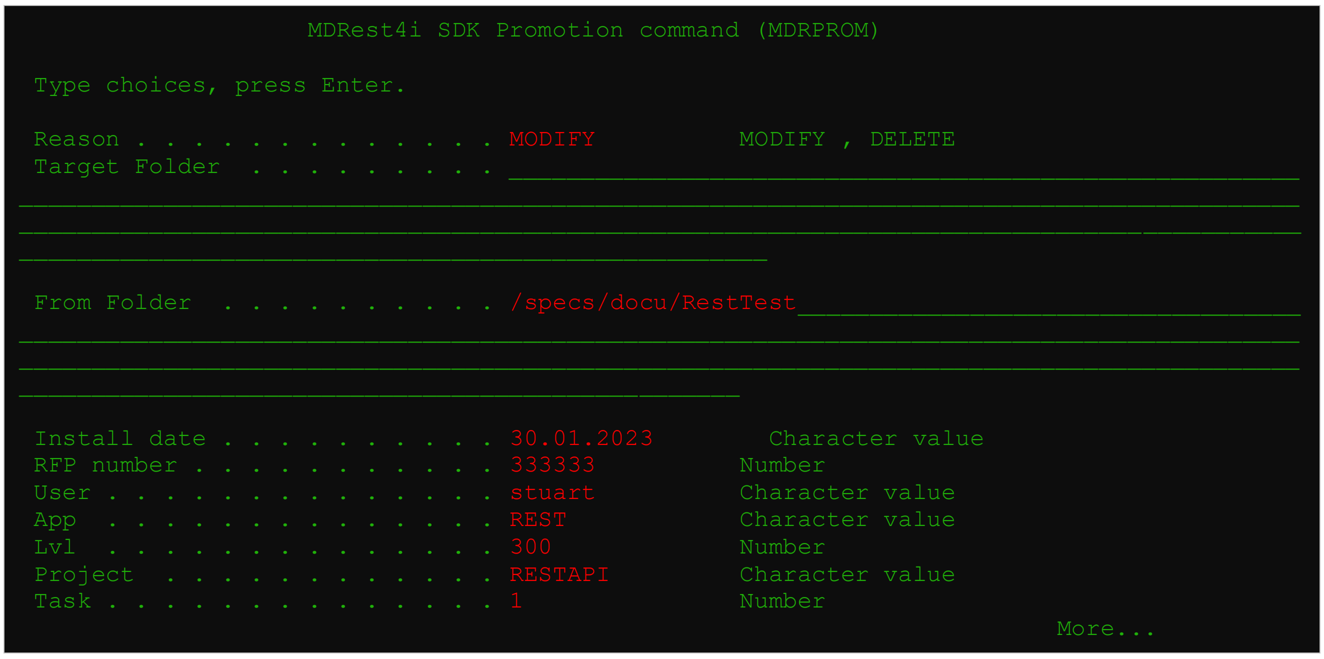
16.2.4 MDRPROM – MDRest4i SDK Promotion command

We can use this command to create object request in the MDCMS. As per below:

From the IBMi command line execute the following:

- ADDLIBLE MDRST *& press enter*
- MDRPROM & press F4

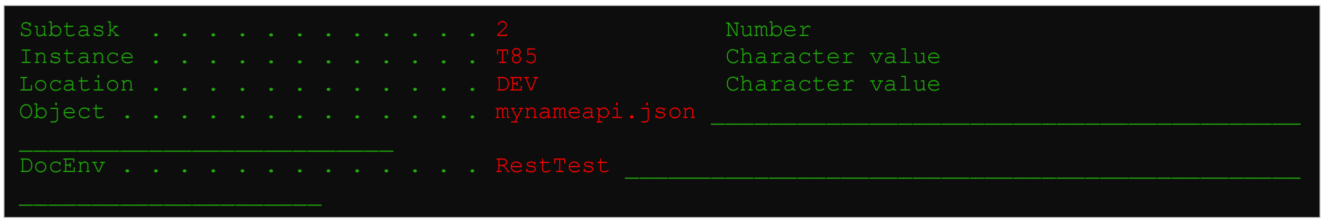
The following screen will appear:



```
                    MDRest4i SDK Promotion command (MDRPROM)

 Type choices, press Enter.

 Reason . . . . . . . . . . . . . . MODIFY          MODIFY , DELETE
 Target Folder  . . . . . . . . . . _______________________________________________________
__________________________________________________________________________________________
__________________________________________________________________________________________
____________________________________________________

 From Folder  . . . . . . . . . . . /specs/docu/RestTest___________________________________
__________________________________________________________________________________________
__________________________________________________________________________________________
__________________________________________________

 Install date . . . . . . . . . . . 30.01.2023        Character value
 RFP number . . . . . . . . . . . . 333333          Number
 User . . . . . . . . . . . . . . . stuart          Character value
 App  . . . . . . . . . . . . . . . REST            Character value
 Lvl  . . . . . . . . . . . . . . . 300             Number
 Project  . . . . . . . . . . . . . RESTAPI         Character value
 Task . . . . . . . . . . . . . . . 1               Number
                                                                        More...
```

Press the page down key.

Type choices, press Enter.



```
Subtask  . . . . . . . . . . . . . 2               Number
Instance . . . . . . . . . . . . . T85             Character value
Location . . . . . . . . . . . . . DEV             Character value
Object . . . . . . . . . . . . . . mynameapi.json ________________________________________

__________________________
DocEnv . . . . . . . . . . . . . . RestTest ______________________________________________

_____________________
```

- **Reason as DELETE –** When we will supply reason as DELETE, it will delete the record from SDK Documenter Repo for that from env/lvl/folder.
- **Reason as MODIFY**
  If the from folder is has “cons” init, (This is where it is still in the USERS development folder). In this case it will insert a new record in the SDK Documenter repo, based upon the USERS development record for that OAPI Specification. It will then create a copy of the OAPI .json file from the USERS Development foler, into the to-Level/Environment folder. After this, it will remove record from MDRDSPC table as per supplied object name as “ENTITY” and username, and remove the OAPI .json filer from the USER development folder. This API will no longer appear in the developers specification list in the SDK Console.
  If the from folder name is /docu/ - (This is where the OAPI specification is in an MDCMS environment/level folder) In this case it will insert a new record in the SDK Documenter repo, based upon the from env/folder record for that OAPI Specification. It will then create a copy of the OAPI .json file from the from env/lvl, into the to-Level/Environment folder. It does not remove the existing record from the from older in this scenario

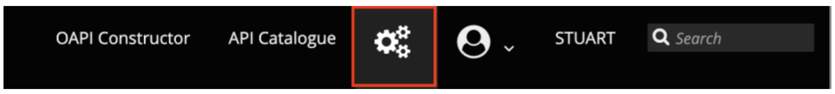
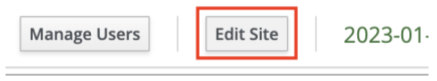
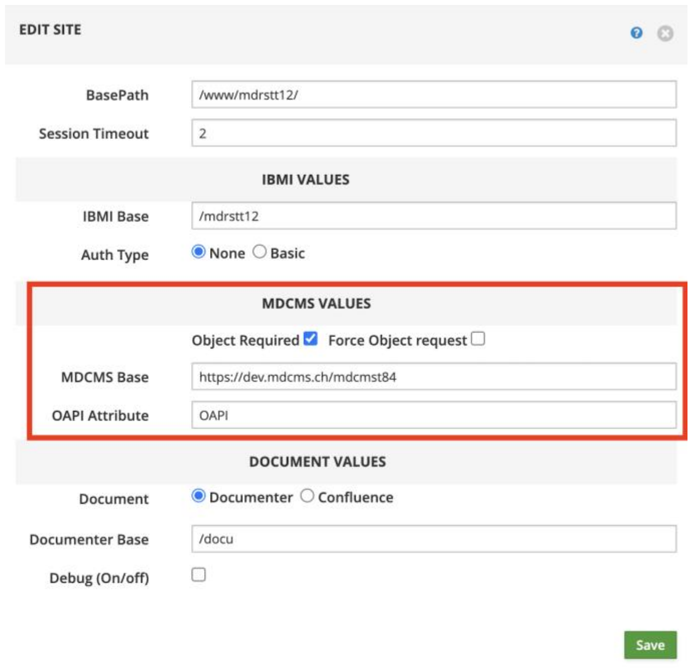
Parameter	Description
Reason	Provide the reason (MODIFY or DELETE) See comments above "Reason as Modify"
Target Folder	Provide the target folder name
From Folder	Provide the from folder name. When we will supply "/docu/RestTest", it will get envrefuuid as "RestTest". If we will supply "/cons/RestTest", it will insert or update the record into MDRDAPI table and also delete the record from MDRDSPC table as per supplied object name and as "ENTITY" and user.
Install date	Supply the date e.g. "30.01.2023"
RFP number	Supply the RFP number e.g. 333333
User	Supply the username e.g. stuart
App	Supply the app name e.g. "REST"
Level	Supply the Level number e.g. 300
Project	Supply the project name e.g. "RESTAPI"
Task	Supply the Task number e.g. 1
Subtask	Supply the SubTask number e.g. 2.
Instance	Supply the MDCMS instance name instance name e.g. T84 or *DFT if using the default MDMCS instance with no suffix.
Location	Supply the location name e.g. dev
Object	Supply the object name e.g. "mynameapi.json".
Document environment	Supply the Document environment name e.g. "RestTest".

The MDRPROM is added as a command to the appropriate attribute (as per example attribute OAPI in the previous section) therefore wildcards can be used. Here is an example:

```
MDRPROM REQRSN(##REQRSN##) trgflr(##OBJLIB##) fromflr(##FRMLIB##) INSTDT
('##RFPAPD##') RFPNBR(##RFPNBR##) APP(##APPLIC##) OBJ(##OBJNAM##) USER(#
#OBJREQ##) LVL(300) PROJ(##PROJID##) TASK(##TASKID##) STASK(##STSKID##)
INST(T84) LOCT(##SVFLOC##) DOCENV(RestTest)
```

16.2.5 SDK Console Setup for MDCMS Interface

then "Edit Site" dialogue, select "Object Required" from "MDCMS Values" section of dialogue

IN SDK Console UI select Console Admin	
Select "Edit Site" button	
select "Object Required" from "MDCMS Values"	 EDIT SITE BasePath: /www/mdrstt12/ Session Timeout: 2 IBMI VALUES IBMI Base: /mdrstt12 Auth Type: [✓ None] Basic MDCMS VALUES [✓ Object Required] Force Object request MDCMS Base: https://dev.mdcms.ch/mdcmst84 OAPI Attribute: OAPI DOCUMENT VALUES Document: [✓ Documenter] Confluence Documenter Base: /docu Debug (On/off): ☐

16.3 Promotion Steps

1. Create the API in SDK Web UI
2. Save the OAPI document/specification in SDK
3. Select "Submit Object Request" option in generate tab of SDK web UI
4. Specify options and submit. Object request will be created for API/Consumer source AND OAPI specification.
5. Promote as necessary and review promoted OAPI Specification in MDRest4i Documenter.
6. To produce the additional documentation, select the Publish button from Installation details tab in the MDRest4i Documenter

MDRest4i Function/Variable Reference

This section describes the important function/procedures and variable names defined in the MDRest4i copybooks or modules. Sometimes these variables are with global scope which means they are declared using “export” keyword in one module and imported in the other module for inter-module communication.

17 REST Consumer Handling

17.1 Consumer (REST Client) Process Flow

The consumer flow starts with initializing the standard variables followed by the call to “Initialize” procedure from the actual program where global variables controlling the consumer process flow are set. After receiving the control back from “Initialize” procedure, the memory is allocated from heap storage. The size of allocated memory is 5MB or the value in “wg_maxdtalen” whichever is higher.

IFS logging is initiated if “ng_saverestswitch” is set to *On. If the variable “wg_SaveIfsType” is set to ‘A’, the file is opened in append mode (if it already exists). Otherwise, the file is created and logging starts in that file.

The request string is then prepared with the http method, url, port and query string. If the indicator “ng_proxy” is set *On, the request is prepared for connecting via proxy, otherwise, sending directly to the target. More details on proxy routing can be seen in “Proxy Connection” section below.

The request string prepared so far is written to the output pointer following which, the standard headers “Accept-Encoding”, “Content-Type”, “Content-Length”, “Host”, “Connection” or “Proxy-Connection”, “User-Agent” are written.

If the indicator “ng_authsend” is set to *On, the value available in wg_authString is sent for “Authorization” header, otherwise the authentication string is generated using “wg_username” and “wg_password” values.

At this step, all the http headers added from the consumer program using “addhttpHeader” function are written in output string followed by a pair of newline and carriage return which means the end of request and begin of the payload for the request.

At this point, “BuildRequest” procedure is called, which again gives the control back to the consumer program for writing the request body using beginObject, endObject, addChar, addDeci etc functions. When the control returns from “BuildRequest” procedure, the http header “Content-Length” (which was written earlier while writing http headers) is updated to reflect the actual number of bytes written in “BuildRequest” procedure.

The final request (which contains headers and request body as applicable) is then logged in the IFS file if logging is requested and it is converted to UTF-8 form for sending the request to the target service. The resolution to the IP address of the target is then made first by using the value in “wg_serverIP” and then using “wg_hostname”. The rest of the processing is then handed over to GSKCSEC for the actual network communication by calling either the SSL connection procedure or non-SSL procedure depending on the type of the request.

When the response is received, if the transfer encoding is chunked, the different chunks are concatenated to create the complete response string. Likewise, if the response is in gzip form, it is unzipped and then the response is translated from UTF-8 back to host code. If the received content is “application/pdf” or “application/octet-stream” or “application/zip” or multipart-form-data”, another procedure is called to receive the attachments.

Finally, the procedure JSONSAX is called to perform SAX parsing on the response so that the values can be retrieved in the main consumer program using JPathN, JPathv etc functions. Complete details about what request has been send and what response has been received can be seen in the log file.

17.2 Consumer Functions

17.2.1.1 AddhttpHeader

This procedure can be used to instruct the consumer to write the custom HTTP header while sending the request to the target.

Arguments

• Header Name	100
• Header Value	8192

Returns

- Indicator (returns *on for success and *Off for failure)

17.2.2 BuildRequest

This procedure should contain the statements writing the request body for POST, PUT, DELETE, PATCH http service requests. The writing happens via “w” procedure. It doesn’t have any parameters.

17.2.3 CloseDown

Write any file closure, memory/resource clean-up instructions in this procedure. It doesn’t have any parameters.

17.2.4 FixReprocess

This procedure gets the control after the request has been sent and response received. The received data is available in “wg_rspdta” (size 5242880 bytes). You can scan the data and if anything is found to be wrong (e.g. authentication problems) and you would like to reprocess the request by changing one or the other variables which were in “Initialize” procedure, change them and set the variable WG_NEXTSTEP(10i,0) to 100. This will cause the same request to be processed again over the pre-existing socket/SSL connection without any delay in establishing the connection. It doesn’t have any parameters.

17.2.5 GetAttachments

This procedure can be called to receive the file names, length of the received file and the number of files received in last parameter. This can be called after GskConsume() procedure call. The purpose would be to check the file names and then use WriteIfsFile function to write the required file at required path (based on file name or file extension).

Arguments

• File_name	200A dim(20)
• File_Length	10i 0 dim(20)
• NumOfFiles	10i, 0

Returns

N/A

17.2.6 GetBody

This procedure can be called with variable declared of 5MB size (i.e. 5242880) to receive Only the JSON Part of the response. If you want to use this procedure, set the indicator "ng_cleanup" to *Off before call to GskConsume procedure in mainline section. Please make sure to call “MemCleanup” procedure before the program exits to cleanup the allocated memory.

Arguments

• Response_body	5242880A

Returns

N/A

Example

```
InitVariant();
tg_InitializePointer = %paddr(Initialize);
tg_BuildReqPointer = %paddr(BuildRequest);
tg_ClosedownPointer  = %paddr(Closedown);
ng_cleanup = *off;
GskConsume();

if wg_httpStatus = '200';
         (w_rspbody);
Endif;

// Cleanup the memory allocated dynamically from heap
cleantree();
Memcleanup();

*Inlr = *On;
```

17.2.7 GetErrorWarnings

This procedure can be called after the “GskConsume” procedure returns the control back to the program. This procedure returns an array qualified data structure with 99 elements. The first subfield of this data structure is message and second is type. In the second field, it will return W or E, where W means warning message and E means error message. If no errors or warnings were encountered while processing the request from the REST service, it will be empty.

Arguments

• ErrWarning	ds		qualified dim(99)
• message		100	
• type		1	

Returns

N/A

17.2.8 GetHdr

This procedure can be used to retrieve the value of http headers received in REST service response. The header name can be supplied in first parameter and it will return the corresponding value in the second parameter. If the header is not found, “*notFound” will be returned.

Arguments

• Header Name	100

• Header Value	2048

17.2.9 GetAllHdr

This procedure can be used to return all the headers received in response by the REST service.

Arguments

• Header_name	100A dim(99)
• Header_value	2048a dim(99)

Returns

N/A

17.2.10 GetReqBody

This procedure can be called from within BuildRequest procedure after complete request body is written. The purpose would be to get the complete content of the request body and maybe encrypt before sending. The parameter of 5MB size (i.e. 5242880) must be supplied to receive the content.

Arguments

• Request_body	5242880A

Returns

N/A

17.2.11 Initialize

This procedure contains the initialization of global variables (e.g. wg_servicepath, wg_PortNumber, wg_hostname/wg_serverIP, wg_httpsmeth, wg_urlparm etc.) which are used in GskConsume module for the REST service request. It doesn't have any parameters.

We can also load (e.g. wg_servicepath, wg_PortNumber, wg_hostname/wg_serverIP, wg_httpsmeth, wg_urlparm etc.) through the MRCNSDTLF file. For this we need to add "/copy qrpglesrc,mrsetcon" copy book

End of the Initialize procedure. This copy book will read MRCNSDTLF file as per key pgmname and load all variable's values of this procedure. And add MRCNSDTLF file in the F spec of the consumer program. For more details, please see user tutorial section 4.3.3.

17.2.12 LoadRspHdrToDB

This function can be used to process all the http headers/values (which are received as part of the response to the called REST service) and write them in MDRJSONF file. This function would generally be used after the call to GskConsume procedure has returned the control back in consumer program. The first parameter is the header name. You should ideally supply "*ALL" which means write all the headers. The second parameter is the entry type under which you want to write the http headers in DB file. The third parameter is optional and if not specified, the header and its value will be written to the DB file in library list. Below is the example.

Arguments

• Header Name	100a
• Entry Type	3a
• DBF Library	10a

Returns

N/A

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

LoadReqHdrToDB('OHD':'MDRDEMOD');

if wg_httpStatus = '200';

  // Change above condition to appropriate success status codes as
  // applicable to your REST service and add the processing logic here

  // You may also call "GetErrorWarnings" procedure with one parameter
  // of 1024a attribute to get any errors/warnings reported in execution

Endif;
```

17.2.13 LoadReqHdrFromDB

This function can be used to read all the headers and their values from MDRJSONF file and write them as part of the request body while sending the request to the REST service. The first parameter is the entry type under which you have headers written in DB file and you want to read them. The second parameter is the library name where MDRJSONF file exists and it is optional. If not specified, the header and its value will be read from the DB file in library list. Please refer the example provided in the next point below.

Arguments

1.	Entry Type	3a
2.	DBF Library	10a

Returns

N/A

17.2.14 LoadQryParmFromDB

This function can be used to read all the query parameters available in MDRJSONF file and write them to the “wg_urlparm” variable under Initialize procedure. The parameter “DBF Library” can be supplied to read the MDRJSONF file from there. If not specified, it will read the file from the library list. Below is the example

Arguments

• DBF Library	10a

Returns

N/A

```
p Initialize      b                   export
d Initialize      pi
 /Free

   // Set Import Values
   wg_contentType = 'application/json; charset=UTF-8';
   wg_httpsMethod = 'GET';

   wg_maxAttempt = 25;
```

```
  wg_mSecDelay = 500000;

  wg_hostName = 'dev.mdcms.ch';
  wg_serverIP= ' ';
  wg_portNumber = 2525;

  wg_servicePath = '/mdrdemod/HDRPARMFP';

  wg_urlParm = LoadqryParmFromDB('MDRDEMOD');

  // Set below indicator to *Off if you don't want to save logs in IFS
  ng_saveRestSwitch = *On;
  wg_saveRESTpath ='/MDREST4i/HDRPARMFPC.json';

  // Add headers from MDRJSONF
  LoadReqHdrFromDB('IHD':'MDRDEMOD');

  // The setting "ng_ssl=*On" enables SSL request. Otherwise, it's non SSL
  ng_ssl = *Off;
```

17.2.15 MemCleanup

This procedure can be called to run memory cleanup of the heap memory allocated in consumer module. This gets called automatically when GskConsumer procedure ends but if you have set “ng_cleanup” to *Off before the call to “GskConsume” in mainline section of the consumer program, you must call this procedure to return the heap storage back to the OS.

Arguments

N/A

Returns

N/A

17.2.16 SetReqBody

This procedure can be called from within BuildRequest procedure after complete request body is written usually when request body has been fetched using GetReqBody and encrypted. This procedure helps to replace the existing request body to this content. The first parameter of 5MB size (i.e. 5242880) should have the content to be written and the second parameter should have the length of data to be written from the first parameter to the target request body.

Arguments

• Request_body	5242880A
• Length	10i, 0

Returns

N/A

17.2.17 WriteIFSFile

This procedure can be used to write the received attachments at specified path. The path should contain folder names and not the file name. If you want to use this procedure, set the indicator "ng_cleanup" to *Off before call to GskConsume procedure in mainline section to instruct avoid running cleanup after consumer call returns.

Arguments

• File_name	200A
• IFS_Path	1024a

Returns

N/A

17.3 Build Request and Process Response via DB file

When the consumer has to supply the request body in JSON format and the JSON content is available in “MDRJSONF” file, you can use “JsonFromDB” procedure to build the request body from DB file. Similarly, if the consumer has received JSON response and you want to load that response in DB file, the same can be achieved via call to “JsonToDB” procedure. Please refer “Automated parsing and writing of JSON using DB File” section in this document for more details.

17.4 Useful Consumer Variables

Global Pointer Variables

tg_buildReqPointer

This pointer is required to hold the address of “BuildRequest” procedure in main program.

tg_closedownPointer

This pointer is required to hold the address of “Closedown” procedure in main program.

tg_initializePointer

This pointer is required to hold the address of “Initialize” procedure in main program.

tg_dtaptr

If for some reason, you want to receive the complete response (header as well as details), declare a variable of 5MB in the main program and assign the address of that variable to "tg_dtaptr" pointer. When the procedure “GskConsume” returns the control back to the main procedure, the variable will have the response data stored.

Global Work Variables

wg_servicePath

In the Initialize procedure of the program which is using GskConsume for consumer service, this variable should be assigned the url path of the REST service e.g.

```
wg_servicePath = 'http://mddev.mdcms.ch:2513/skdemo/clt_det';
```

wg_urlParm

If the http request contains the query parameters, this variable is assigned the parameter names and values e.g. below:

```
wg_urlparm = 'client=2&policy=14&claim=10';
```

wg_httpsMethod

This variable should be set to the http method for the service request being made.

```
wg_httpsMethod = 'GET';
```

wg_portNumber

This variable should be set to the port number where the REST service is available. When the port is not specified in the URL, still this variable should be set to the default http port number 80 for non-SSL and 443 for SSL.

```
wg_PortNumber = 4523;
```

wg_HostName:

If the URL contains host name, that should be assigned in this variable for mapping the IP address via DNS e.g.

```
wg_HostName = mddev.mdcms.ch';
```

wg_serverIP

If the URL contains IP address instead of the host name, in that case, wg_urlhost should be set to blank and wg_serverIP should be assigned the IP address of the host.

```
Wg_serverIP = ‘41.169.58.232’;
```

wg_urlParm

If the target REST service expects some parameters, set this variable to the parameter/values.

```
Wg_urlParm = ‘clientid=2&policy=5’;
```

wg_ContentType

This variable should be set to the type of content being supplied as part of http request.

```
wg_urlContent = 'text/html; charset=UTF-8';
```

wg_saveRESTPath

This variable contains the qualified IFS file name where REST service request and response should be logged. The logging will happen only if the indicator variable ng_saveRESTSwitch is set *on.

```
wg_saveRESTPath = ‘/MDREST4i/logs/TESTGEN_'+ %char(%timestamp()) + '.txt';
```

wg_saveIFSType

This variable can be set to “A” (append) if you want the logging of the outgoing request and incoming response should happen in the same IFS file.

wg_maxAttempt

This variable is initialized to 5 which means the consumer will make five attempts to read the response. It can be imported in the consumer program and the value changed as per the requirement in the Initialize procedure.

wg_MsecDelay

This variable is for delaying the job by specified “micro seconds” before trying for the next attempt if the response is not received. The variable is initialized to 50000 (i.e. 0.05 second). This variable can be overridden in the consumer program and value changed as required.

wg_dcmApplication

This variable is used to set the name of DCM application in digital certificate manager. It’s only used in SSL consumer. You should have one DCM application in digital certificate manager (it’s not necessary for that application to have some certificates attached). Assign the application name to this variable in “Initialize” procedure of the consumer.

wg_userAgent

Use this value to override the list of SSL

wg_authString

If the REST service being called requires authorization other than what you can provide through user ID and password, you can send the specific auth string instead of sending the basic authentication with user ID and pwd. In this case, assign the authentication string to this variable. E.g. wg_authstring = ‘Bearer <token name>’;

wg_userName/wg_password

If the REST service requires basic authentication via user ID and password, you can set the values in these variables. This would then be converted to UTF-8 followed by base64 and sent as the authorization string.

wg_spcerror

In some specific case, if the REST service used in consumer sends the 200/OK status but a specific message in response should be considered as an error, you can assign that error text in this variable, the special HTTP status code 801 will be returned and you can monitor the variable “wg_httpstatus” in your program.

wg_httpStatus

This variable holds the http status code (e.g. 200) received from the REST service. This can be used in consumer program to check the success/failure codes as applicable to your REST service.

wg_Reason

This variable holds the reason (e.g. OK) received from the REST service. This can be used in consumer program to check the success/failure codes as applicable to your REST service.

wg_maxdtalen

This variable is available for specifying the anticipated response length in consumer. By default 5MB (i.e. 5242880) bytes is taken (if not set already). If you specify length is more than 16MB (i.e. max allowed for a user space size), the statement "h alloc(*teraspace)" should be added to the consumer program. e.g. below statement sets the max data length in consumer program (before call to GskConsume procedure in mainline section) to 10MB (approx.).

```
wg_maxdtalen = 10000000;
```

wg_proxyhost

This variable should be set to the proxy host name (if the host name is not in IP address format) for calling the REST service via proxy.

wg_proxyIP

This variable should be set to the IP address of the proxy server for executing the REST service via proxy.

wg_proxyport

This variable should contain the port number of the proxy server when the REST service is being routed via proxy.

wg_proxyusr/wg_proxypwd

When the REST service is being run from the consumer via proxy and the proxy requires, authentication, these variables should be set to the user ID and pwd for basic authentication.

wg_maxredirect

When your REST service is expected to return the “redirect” type response (i.e. with the response code between 300 and 399 and the http header containing the “LOCATION” set to another URL where request should be redirected, you need to set the variable “wg_maxredirect” with the number of maximum possible redirections internally. For example, if the first level call returns the target URL, the variable should be set to 1 and if the first level call could return the URL which in turn provides the target URL, the variable should be set to 2 and so on.

wg_utf8reqbody/wg_utf8response

By default the consumer translates the UTF8 response to normal text and then does JSON parsing. Likewise, you use beginObject, addChar etc functions to create the request body. However, if you instead want to manage this in UTF8 form (e.g. the request body is coming from some DB file and the response is supposed to be written in DB file and both in UTF8 format), set the indicator “ng_utf8conv” to *off and in that case you have to load the request body in w_utf8reqbody variable in “BuildRequest” procedure. Likewise, you can get the UTF8 response in “w_utf8response” variable.

17.4.1 Consumer Timing Variables

These variables should be used after GSKCONSUME() function is called in consumer.

wg_fullreq

This variable tells the time taken in micro-seconds to process full consumer request. This includes all preparation and initialisation time

wg_reqPrep

This variable contains the time taken to build request after GSKCONSUME() is called but before start of making the actual request. Includes creation and conversion of the HTTP message including headers and requestBody prep

wg_reqRsp

Time taken to establish connection up until complete response is received. No conversion or data preparation occurs during this time: it’s the pure request/response time

wg_preSSL

This variable tells the pre-processing time in micro-seconds before any network communications are established for sending the request and receiving the response.

wg_jsonprc

This variable tells the time taken in JSON parsing of the received response.

wg_dtacvt

This variable tells the time taken in data translation (utf-8 to host code and gzip translation if applicable) after receiving the response.

17.4.2 Global Indicators

ng_customlog

If you want to enable logging the errors and warnings, set this indicator variable to *On. The logging happens in LXRCLTLOG file under MDRST library via call to procedure wlogclt internally.

ng_saveRESTSwitch

If you want to log the service request in the IFS directory at the qualified file name available in wg_saveRESTPath, set this indicator to *On.

ng_ExitOnFirstResponse

This indicator is initialized to *On and the purpose is to exit out of the loop once the response is received instead of iterating further. You can set it to *Off if the response is expected to be large and coming in multiple chunks.

ng_LogErrSRCM

This indicator is initialized to *On and the purpose is to log all the exceptions encountered in consumer processing. You can set it to *Off if you would like to disable the default exception handling to write the logs.

ng_authsend

This indicator is set *On by default and is used in SSL reqest. If you don't want to send the authorization in the specific consumer program, set the value to *Off before calling GskConsume procedure. When this indicator is set *On, the logic checks the value in wg_authstring and if it's not blank, this value is sent, otherwise, the value from wg_userName and wg_password is used to sent as "Basic" authentication type.

ng_ssl

This indicator is used to decide whether the consumer is of SSL or non-SSL. If you want to use SSL request, set this indicator to *On, else *Off (which is the default).

ng_autowrite

This variable is set to *On by default and that means the IFS files received as attachments from the REST service will be written at the IFS path set in data area DFTUPLOADP. If the data area is not found, the files are downloaded in /MDRest4i/upload.

ng_cleanup

This variable is set to *On by default and that means the user spaces created to send/receive the data or intermediate language translation are deleted and any pointer memory allocated dynamically from the heap is also deallocated.

ng_proxy

This variable is for enabling the http proxy if the REST services you want to access have to be channelized via proxy. This indicator is initialized to *Off by default. You can set it to *On to enable sending the request via proxy which will use the information from the variables "wg_proxyhost", "wg_proxyIp", "wg_proxyport", "wg_proxyusr" and "wg_proxypwd".

ng_proxyssl

When the proxy is used to access the target REST service, this variable can be set to *On to enable communication to proxy over SSL channel and establish the connection with the target. This indicator is initialized to *Off by default.

ng_parse

This variable is set to *On by default and that means the JSON/XML parsing will be done by default after the response is received in the consumer. You can import this indicator in the consumer program and set it to *off to disable the parsing. You may later use GetBody procedure to receive the JSON or XML response.

ng_tracktime

This variable is set to *Off by default and the developer can set it to *On if there is some performance issue to check what time is being consumed by the different internal processing sections.

ng_UTF8Conv

This variable is set *On by default and that means the UTF8 to ASCII conversion will be performed on the response received from the REST service and likewise, the writing of the request body is being done using standard beginObject, addChar etc functions. However, in exceptional case when there is a need to directly accept UTF8 state of the REST service response, set this indicator to *Off in the "Initialize" procedure and you can access the response in "w_utf8response" variable. When this indicator is *Off, you have to send the request body in UTF8 format which is expected to be assigned in the variable "w_utf8reqbody".

ng_oldmethod

This variable is set *Off by default and that means the the writing of the request body is being done using standard beginObject, addChar etc functions. However, if you want to do it via old "w" function of MDRest4i, set this indicator to *On in "Initialize" procedure but then only data written by "w" function will be accepted and anything used via addChar, beginObject etc will be ignored.

ng_bypasscert

This variable is used when REST service is running in SSL mode and it is set *Off by default. You can set it to *On if you want to bypass certificate validity checking.

ng_logevents

This variable is used to log the detailed http actions/events which are being performed from the consumer to send the request and receive the response.

ng_logfile

This variable can be set *On if you want to log the http events in MDRLOGS database file.

ng_decodeutf

If you are expecting the Unicode constants (e.g. \u0034) in the response, set this indicator to *On to convert them to normal text.

ng_largefiles

By default, the maximum data size allocation is 5 MB. However, if you expect the attachments to be received and larger than 5MB, set this indicator to *On and it will then allocate 16MB size. If you expect to allocate any other size higher than 5MB, set the allocation size in "wg_maxdtalen" variable.

17.5 Controlling Authentication in Consumers

The standard headers used for authentication are added setting specific variable combinations in the "**Initialize**" procedure of the consumer. The table below describes these Authentication type:

Auth type	Request header requirement/result	Parameters to set
'Basic'	This algorithm Base64 Encodes the concatenated wg_userName and wg_password values, and adds the result to "Basic" + space as the value for the Header field "Authorization" resulting in: **Authorization:** Basic c3R1OnBhc3M	wg_userName / wg_password
"Bearer'	This algorithm concatenates "Bearer" + space to the value of wg_authString as the value of the "Authorization" header field resulting in: **Authorization:** Bearer ACd9d5e30cfeeff79ad8b25c046a69c95a Standard token (bearer) "Authorization: token"	wg_authString

You need to set “ng_authsend” to *On and then either set the value in “wg_authString” for non-basic type authentication, or keep “wg_authString” to blank as default and set the values in “wg_userName” and “wg_password”. The consumer module will take care of sending the authorization based on these variables.

17.6 Proxy Handling

When the indicator “ng_proxy” is set as *On, the connection to the target REST service is made via proxy. The socket is then opened after resolving the IP address first using “wg_proxyIP” and then by using “wg_proxyhost”. If the target is non-SSL, the connection is made to the proxy and complete URL including the host name, port, endpoint and query parameters (e.g. http://mddev.mdcms.ch:2515/skdemo/getclient?id=4) is sent. However, if the target is SSL, the request is sent to proxy using http “CONNECT” method. Proxy then establishes the connection and sends http status 200 with the response as “Connection Established”. At that point, the socket connection is upgraded to use various SSL parameters and the request is sent to the target over this upgraded SSL channel.

18 REST Producer (API) Handling

18.1 Producer Process Flow

The REST service program is composed of some custom logic written within the program itself and fixed processing section coming from the copybooks. The processing starts with the logic in LXRRESTC copybook. The variable definitions are in MDRESTDFN copybook. The main program is expected to have z_CustomInit subroutine to initialize some variables (e.g. expected memory allocation size, IFS logging indicator and IFS file name). Depending on the service types, the main program is expected to have subroutines (e.g. z_ProcGet, z_ProcPut, z_ProcPost, z_ProcDel, z_ProcPatch). If the REST service is expected to have some query parameters, the main program will also have z_CheckParms, z_SetParms and z_SetMethod subroutines.

The subroutine z_CustomInit is executed in the beginning and is the first control to the developer so that he can write this subroutine in main program and for necessary variable/memory initialization. Usually the variables “w_maxrsplen” and “w_maxrsplen” are set with the expected request and response size. The indicator “n_saveIFSSwitch” can be set for logging the request in an IFS file. The internal procedure is then called to extract the method type and query parameters.

The logging is then started if the indicator “n_saveIFSSwitch” is set to *On. If the variable “w_saveIFSPath” is non-blank, the logging happens in this file, otherwise, the IFS file name is built using the MDRest4i instance installed (e.g. /MDRest4i/logs/LXRLOGS_2019-07-31.10.34.25.300000.txt).

Based on the method type (e.g. GET, POST), the respective subroutine is called.

The expected names of the query parameter are set in z_SetParms subroutine and then their value is assigned using z_EvalParms subroutine. If any of the mandatory parameter is not supplied, the subroutine z_SetMethod is called to send the parameter error. When the request is of type “GET”, the page handling is done to check and process next/previous page (refer MDRest4i Paging implementation section for more details).

In request types “PUT”, “POST”, “PATCH”, the request body is processed by first getting the Content-Type and Content-Length. When the indicator “n_getreqsize” is set *On, instead of loading the body, just the payload size is sent to show the request body size. Otherwise, if the value of “w_maxreqlen” is greater than 5MB (i.e. 5242880), the memory is allocated from heap but when its less than 5MB, the user space of size 5MB is created in QTEMP library and the actual payload from the request body is loaded and translated to UTF-8.

If the attachments are received in the request body (i.e. content type is “application/pdf” or “multipart/form-data”), the files are downloaded at the default upload path if the indicator “n_autowrite” is set *On. Otherwise, you can later download the IFS files at non default location using the procedure “WriteIFSFile”. Refer Downloading Attachments in REST service for more details. The JSON content received in the request body is supplied to JSONSAX to perform SAX parsing.

After that, the control is given to specific subroutines z_ProcGet, z_ProcPost etc. where you can use JPathv, JPathN etc functions to extract the specific values from JSON body and use beginObject, endObject, addChar etc functions (refer JSON section for more details about these functions). These functions write the response at the internal pointer and not directly to the output.

Once complete response is written, it is sent to the requester followed by the call to z_CustomExit subroutine execution. This is last control to the program so that you can define this subroutine in your program and run some post processing memory cleanup or anything before REST service exits.

18.2 MDRest4i Producer Switches

These are the conditional compiler directives guiding the compiler whether to include the relevant source section or not (depending on which subroutine or procedure the user wants to override for changing the default behavior). The REST services generated using the MDRest4i product makes use of the standard copybooks MDRESTDFN, LXRRESTC and LXRRESTP. The copybook LXRRESTC contains the mainline part of the program and the subroutines and LXRRESTP contains the procedures used either by the subroutines in LXRRESTC or other procedures in LXRRESTP. Most of the subroutines in LXRRESTC and the procedures LXRRESTP are conditioned by a specific conditional compiler directive. In certain cases, the developer may like to make some alterations in the processing logic of these subroutines and/or procedures. In order to do that, the developer can define the appropriate switch using the "/Define" compiler directive before "/copy" statement of the relevant source member. This is to tell the compiler not to include that part of source code section conditioned by "/If Not defined" directive.

18.2.1 LXR_CheckParms

Source Type:	Copy Book
Source Member:	LXRRESTC
Source File:	QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine "z_checkParms" defined in LXRRESTC. If you are expecting the parameters in the REST service program, Add "/Define LXR_setParms" directive before the "copy" clause for including LXRRESTC member and write "z_setParms" subroutine with the necessary processing to validate the parameter values received in query parameter string.

18.2.2 LXR_CustomInit

Source Type:	Copy Book
Source Member:	LXRRESTC
Source File:	QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine "z_CustomInit" defined in LXRRESTC. If you want to initialize some variables (e.g. setting the indicator "n_saveRestSwitch", setting the logging file name in "w_saveIFSPath", setting "w_maxreqlen" and "w_maxrsplen" variables with the expected memory size for the request body and response output) or would like to have some pre-processing before any method gets processed, Add "/Define LXR_CustomInit" directive before the "copy" clause for including LXRRESTC member and write "z_CustomInit" subroutine with the necessary processing.

18.2.3 LXR_CustomExit

Source Type:	Copy Book
Source Member:	LXRRESTC
Source File:	QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine "z_CustomExit" defined in LXRRESTC. If you want to deallocate some memory or close some explicityly opened files, Add "/Define LXR_CustomExit" directive before the "copy" clause for including LXRRESTC member and write "z_CustomExit" subroutine with the necessary processing.

18.2.4 LXR_LogService

Source Type:	Copy Book
Source Member:	LXRRESTP
Source File:	QRPGLESRC

Description: Include procedure wlogSRV for standard service logging. If you want to customize this procedure, add "/Define LXR_setParms" directive before the "copy" clause for including LXRRESTP member and define "wlogsrv" procedure with the necessary processing.

Arguments*:*

1.	p_writeText	Text	Contains the text to write to the log	50a
2.	p_msgLvl	Message Level	The severity of the message	10i 0

Example

```
/define LXR_LogService
…
/copy LXRRESTP
…
/undefine LXR_LogService
```

18.2.5 LXR_ProcGet

Source Type: Copy Book

Source Member: LXRRESTC

Source File: QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine "z_ProcGet" defined in LXRRESTC. If you are writing the REST service program for "get" method, Add "/Define LXR_setParms" directive before the "copy" clause for including LXRRESTC member and write "z_setParms" subroutine with the necessary processing.

18.2.6 LXR_ProcPut

Source Type: Copy Book

Source Member: LXRRESTC

Source File: QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine "z_ProcPut" defined in LXRRESTC. If you are writing the REST service program for "put" method, Add "/Define LXR_setParms" directive before the "copy" clause for including LXRRESTC member and write "z_ProcPut" subroutine with the necessary processing.

18.2.7 LXR_ProcPost

Source Type:	Copy Book
Source Member:	LXRRESTC
Source File:	QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine “z_ProcPost” defined in LXRRESTC. If you are writing the REST service program for “post” method, Add “/Define LXR_ProcPost” directive before the “copy” clause for including LXRRESTC member and write “z_ProcPost” subroutine with the necessary processing.

18.2.8 LXR_ProcPatch

Source Type:	Copy Book
Source Member:	LXRRESTC
Source File:	QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine “z_ProcPatch” defined in LXRRESTC. If you are writing the REST service program for “patch” method, Add “/Define LXR_ProcPatch” directive before the “copy” clause for including LXRRESTC member and write “z_ProcPatch” subroutine with the necessary processing.

18.2.9 LXR_ProcDel

Source Type:	Copy Book
Source Member:	LXRRESTC
Source File:	QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine “z_ProcDel” defined in LXRRESTC. If you are writing the REST service program for “delete” method, Add “/Define LXR_ProcDel” directive before the “copy” clause for including LXRRESTC member and write “z_ProcDel” subroutine with the necessary processing.

18.2.10 LXR_SetMethod

Source Type:	Copy Book
Source Member:	LXRRESTC
Source File:	QRPGLESRC

Description: This switch causes the inclusion or exclusion of the “z_setmethod” subroutine defined in LXRRESTC. When the REST API program expects the parameters and any of the mandatory parameter is not supplied, this subroutine is executed to report an error mentioning the mandatory and optional parameter names. If you want to customize this subroutine, Add “/Define LXR_SetMethod” directive before the “copy” clause for including LXRRESTC member and write “z_SetMethod” subroutine with the necessary processing.

18.2.11 LXR_setParms

Source Type: Copy Book

Source Member: LXRRESTC

Source File: QRPGLESRC

Description: This switch causes the inclusion or exclusion of the empty copy of the subroutine “z_setParms” defined in LXRRESTC. If you are expecting the parameters in the REST service program, Add “/Define LXR_setParms” directive before the “copy” clause for including LXRRESTC member and define “z_setParms” subroutine with the necessary setting of the parameter names and mandatory/optional flags.

18.2.12 LXR_SendSchema

Source Type: Copy Book

Source Member: LXRRESTC

Source File: QRPGLESRC

Description: This switch causes the inclusion or exclusion of the “z_sendschema” subroutine defined in LXRRESTC. This subroutine is used for sending the expected schema when the REST service expects some payload as part of the request body but the payload is not received. Then, based on the value in “w_schemapath” variable, either subroutine “z_sendSchemaManual” is executed (when the variable “w_schemapath” is blank) or the schema from the IFS file is sent (when “w_schemapath” contains the valid IFS file). If you want to customize this subroutine, Add “/Define LXR_PreReturn” directive before the “copy” clause for including LXRRESTC member and write “z_PreReturn” subroutine with the necessary processing.

18.2.13 LXR_SendSchemaManual

Source Type: Copy Book

Source Member: LXRRESTC

Source File: QRPGLESRC

Description: This switch causes the inclusion or exclusion of the “z_sendschemaManual” subroutine defined in LXRRESTC. This subroutine is called from z_sendschema subroutine when the variable w_schemapath is blank or when the read to the schema file on IFS fails. You may like to customize this subroutine to have the specific fields, add “/Define LXR_SendSchemaManual” directive before the “copy” clause for including LXRRESTC member and write “z_sendschemaManual” subroutine with the necessary processing.

18.3 Important /Customizable subroutines in Producer Processing

z_SaveIFS – This subroutine contains the logic to fetch various environment variables for the http request. create the user space LXRREST in QTEMP library. Whenever the procedure “w” is used to write the response on standard output, the data is stored at the pointer t_sendpointer pointing to this user space. This data is later written to standard output using CgiStandardWrite1 procedure call.

18.4 Producer Functions

18.4.1 AddAttachment( )

This procedure can be used to send the attachments in the response to the REST service. As many attachments can be sent as required making sure the maximum response size doesn't exceed allowed for the REST service. The files with the extension jar, war, zip, pdf and json are supported. When there are more than one types of the files, the content type is set to "Content-Type: multipart/mixed; boundary=MDREST4I". With each file type, the "Content-Type" is set for that type. The name of the file is given in two headers "name" and "filename". With each next file, the boundary '--MDREST4I' is added after the exact end of the previous file content. At the end, '--MDREST4I--' is written and that means the end of the response.

Copy Book member:	LXRRESTP

Arguments

File Path	1024a

Returns

N/A

Example:

```
addAttachment('/home/stuart/afftest-invoice-clickwarp.pdf');
addAttachment('/home/stuart/Synon.pdf');
```

18.4.2 AddHdr( )

This procedure can be used to add the customer headers in REST http response.

Copy Book member:	LXRRESTP

Arguments

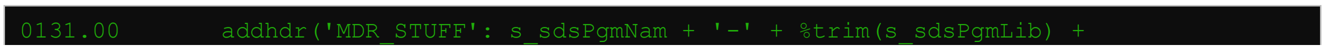
● Header Name	100
● Header Value	2048

Returns

None

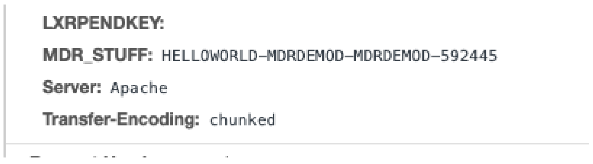
Example:

You can call the header field anything you like. In the example below s_sdsPgmNam comes from copybook MDRST/QRPGLESRC.LXRSRC which by default is copied into every program. Other values that can be used from there include s_sdsPgmLib, s_sdsJobNam, s_sdsJobNbr etc



```
         addhdr('MDR_STUFF': s_sdsPgmNam + '-' + %trim(s_sdsPgmLib) +
0132.00        '-' + %trim(s_sdsJobNam) + '-' + %trim(s_sdsJobNbr));
```

Gives this header in the response:



```
LXRPENDKEY:
MDR_STUFF: HELLOWORLD-MDRDEMOD-MDRDEMOD-592445
Server: Apache
Transfer-Encoding: chunked
```

Where MDRDEMOD is the HTTPSERVER instance and SERVER CGI job and the lib where the program is, and HELLOWORLD is the name of the program

18.4.3 CrtUsrSpc( )

This procedure creates the requested user space in QTEMP library and returns the pointer to user space. If the user space already exists, it won't be created and the pointer will be returned. In this procedure available three parameters where as two parameters is mandatory and third last parameter is optional for user space length. If we will use this procedure with three parameter it will create user space as supplied length in third parameter and If we will use this procedure with two parameters, it will create 5 MB user space.

Copy Book member: LXRRESTP

Arguments

1.	User Space name	10a	
2.	User Space pointer	*	
3.	User Space size	10i 0	options(*nopass)

Returns

N/A

18.4.4 extractQryPrms

This function can be called from the REST service to extract all query parameters and their values as received in the REST service. The parameters and their values are updated in d_Qparm data structure which is defined in the copybook MDRESTDFN as below.

```
d d_Qparm          DS                   qualified dim(50)
d   s_Name                              inz(*blanks) like(w_parm)
d   s_Value                             inz(*blanks) like(w_parmval)
```

You can therefore directly use the data structure after the function call to extract any query parameter:

```
extrctQryPrms();
for w_idx1 = 1 to %elem(d_qparm);
  if d_qparm(w_idx1).s_name = ‘MYHEADER’;
    w_hdrvalue = d_qparm(w_idx1).s_value;
    leave;
  endif;
endfor;
```

Arguments

N/A

Returns

N/A

18.4.5 GetAuth( )

This procedure can be used to return the authorization information in the REST service. The procedure has three parameters where first is the input parameter and remaining two are output parameters.

If the first parameter (i.e. authorization type) is supplied as 'BASIC', it will return the user name and password in second and third parameter respectively. If the authorization type is 'BEARER', it returns the token in the second parameter and third parameter will be blank.

In order to use this function in any rest service, make sure to have below configuration in your http server so that the Apache server sends the authorization info to the program.

```
SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1
```

Copy Book member: LXRRESTP

Arguments

1.	Auth Type	10a
2.	User Name	1024a
3.	Password	1024a

Returns

N/A

Example

```
GetAuth(w_type:w_User:w_pwd)
```

18.4.6 GetHdr( )

This procedure can be used to retrieve the value of custom headers received in REST http request. The second parameter returns the value of custom header supplied in the first parameter.

Copy Book member: LXRRESTP

Arguments

Header Name	100
Header Value	2048

Returns

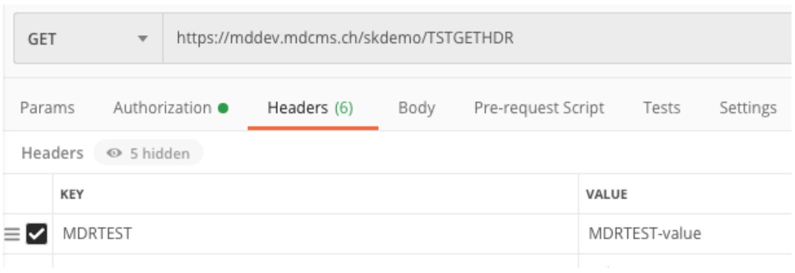
None

Example:

The code below reads the value from header called MDRTEST added by the client to the HTTP request, adds some text to it and writes it back to the client using the AddHdr function described above.

```
Begsr z_ProcGET;

// Logic from custom copybook

  // Write your database interaction logic here
  w_hdrname = 'MDRTEST';
  Gethdr(w_hdrname:w_hdrvalue);
  w_hdrvalue = %trim(w_hdrvalue) + '_response';
  AddHdr(w_hdrname:w_hdrvalue);
Endsr;
```

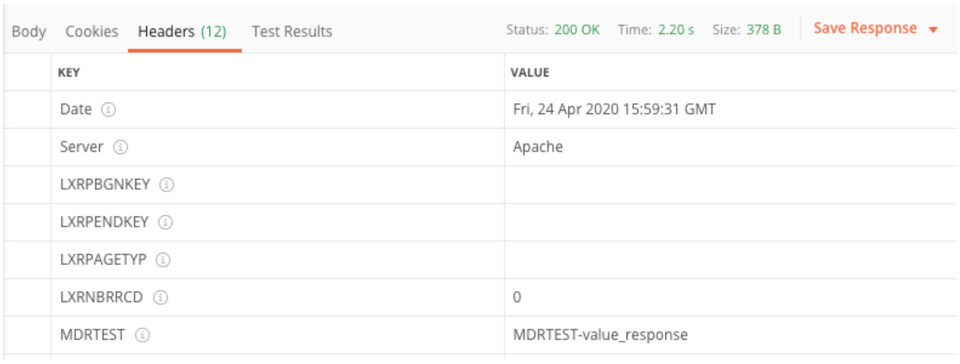
Using POSTMAN a header MDRTEST was added to a GET request to the MDREST4i API:



https://mddev.mdcms.ch/skdemo/TSTGETHDR

KEY	VALUE
MDRTEST	MDRTEST-value

Here is the response received in POSTMAN from the above MDREST4i API code:



Status: 200 OK Time: 2.20 s Size: 378 B

KEY	VALUE
Date	Fri, 24 Apr 2020 15:59:31 GMT
Server	Apache
LXRPBGNKEY	
LXRPENDKEY	
LXRPAGETYP	
LXRNBRRCD	0
MDRTEST	MDRTEST-value_response

18.4.7 GetPathParm

This function can be called from the REST service to extract the value of specific path parameter as expected in the REST service. This functions expects the fixed entry name as the first parameter and offset as the second parameter. The offset value could be negative or positive depending on the path parameter location from that fixed entry.

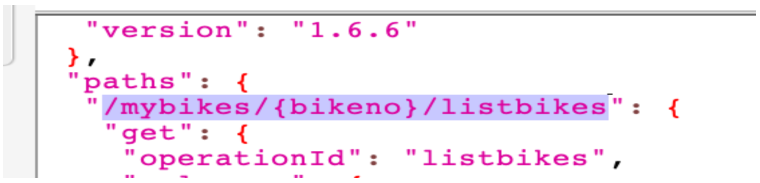
Arguments

Fixed Name	100a
Offset	5i,0

Returns

Parameter Value	2048a

Example



```
  "version": "1.6.6"
},
"paths": {
  "/mybikes/{bikeno}/listbikes": {
    "get": {
      "operationId": "listbikes",
```

In this case, the path parameter “bikeno” is one position right to the fixed entry “mybikes”. Therefore, if we are using “mybikes” as the first parameter, the second parameter will be 1 whereas if we use “listbikes” as the first parameter to this (i.e. GetPathParm function), the second parameter will have to be supplied as -1. As you can see, return value is assigned to w_str2. This value is then assigned to the variable expected to hold the parameter value and is used in later processing as needed.

```
// ====================================================================
// Set Path Parameters
// ====================================================================
Begsr z_GetPathParm;

  w_str2 = GetPathParm('mybikes':1);
  if w_str2 <> '*notFound';
    n_bikeno = *On;
    p_bikeno = %trim(%trim(w_str2));
  Endif;
Endsr;
```

18.4.8 GetReqBody

This function can be used to REST APIs for getting request body in a variable or write in the IFS file. This function have three parameters one is mandatory and other two parameters are optional. If we want to get JSON request in a variable, we need to use this with one parameter of 5 MB size (i.e. 5242880) and if we want to write the request body in the IFS file, we need to use this with three parameter where first is 5 MB string second in type where we need to supply it with the type ‘IFS’ and third last parameter we need to supply full path name with the file name where we want write the JSON of request body. Like below:

Ex. GetReqBody(w_inBuffer : 'IFS' : '/MDRest4i/test1.json');

Arguments

• ReqBody	5242880a	
• Type	3a	options(*nopass)
• IFSPath	100a	options(*nopass)

Returns

N/A

18.4.9 LoadReqHdrToDB

This function can be used to write the specific http header and its value to MDRJSONF file. This function would generally be used in z_Procxxx subroutines. The developer will have to specify which header is to be retrieved and written to DB file. This is because the REST service doesn’t receive all the headers in any environment variable or request body. The second parameter is the entry type under which you want to write the http headers in DB file. The third parameter can be used to supply the library name from where “MDRJSONF” file should be used. If not specified, DB file MDRJSONF in library list will be used instead. Below is the example:

```
Begsr z_ProcGET;

   // Get incoming headers
  LoadreqHdrToDB('X_MYHEADER':'OHD':'MDRDEMOD');

Endsr;
```

Arguments

• Http Header name	100a
• Entry Type	3a
• DBF Library	10a

Returns

N/A

18.4.10 LoadQryParmToDB

This function can be used to write all the query parameters (as received in REST service) to MDRJSONF file. The procedure has only one and optional parameter. You can supply the library name where MDRJSON file exists. If this parameter is not supplied, it will attempt to use the file in library list. Below is the example on how to use it.

```
Begsr z_ProcGET;

  // Get parameters in the MDRJSONF
  LoadqryparmtoDB('MDRDEMOD');

Endsr;
```

Arguments

• DBF Library	10a

Returns

N/A

18.4.11 LoadRspHdrFromDB

This function can be used to read the http headers and their values loaded in DB file and send those headers along with the response to REST service. The first parameter is the entry type which is supposed to be processed from DB file and the second parameter is optional. You can supply the library name which should be considered for using MDRJSONF file, otherwise, *LIBL will be used. Below is the complete example of using all the three functions.

```
Begsr z_ProcGET;

   // Get incoming headers
  LoadreqHdrToDB('X_MYHEADER':'OHD':'MDRDEMOD');

  // Get parameters in the MDRJSONF
  LoadqryparmtoDB('MDRDEMOD');

  // Add headers from MDRJSONF
  LoadrspHdrFromDB('IHD':'MDRDEMOD');


  Exsr z_PrcResponse;

Endsr;
```

Arguments

• Entry Type	3a
• DBF Library	10a

Returns

N/A

18.4.12 RspJsonFile ( )

This procedure can be used to send the response by reading an IFS file and then sending the content as the response.

Copy Book member:	LXRRESTP

Arguments

File Path	1024a

Returns

N/A

18.4.13 rtnHeader( )

This procedure can be used to retrieve the value of custom headers received in REST http request. This procedure returns the value of custom header supplied in the parameter.

Copy Book member:	LXRRESTP

Arguments

● Header Name	100

Returns

HeaderValue	VARCHAR(2048 A)

18.4.14 SetHttpStatus( )

This procedure sets the variables “w_status” and “w_reason” with the values received in the parameter. These values are used in LXRPush procedure for writing the same in custom header part.

Copy Book member:	LXRRESTP

Arguments

1. Status number	3S,0
2. Reason	50

Returns

None

Example:

```
SetHttpStatus(405: 'Record not found');
```

18.4.15 ValidateToken( )

This procedure retrieves the values of “CLIENTID”, “APPID” from http header and also retrieves the “Authorization” header that has been supplied as part of authentication. The “CLIENTID” and “APPID” are optional and the “Authorization” is expected to be a Bearer token.

It then tries to fetch the token details from the MDRest4i credential stores file to find the secret for validating the token to make sure it’s a valid token (i.e. existing in the database as well as its not expired) before proceeding ahead with the REST service. If the token is not valid, the error will be sent and REST service logic won’t continue further.

18.4.16 wlogSRV( )

This procedure is controlled by the conditional compiler directive LXR_LogService and is used for logging the requested message/severity in the LXRSRVLOG file under MDRSTxxx library. The logging happens only when the global indicator ng_customlog is *On.

Copy Book member:	LXRRESTP

Arguments

1. Log message	50
2. Message severity	10i,0 options(*noPASS)

Returns

N/A

18.4.17 WriteIFSFile( )

This procedure can be used to write the specific IFS Files received as part of request body at the specific IFS location. Supply the file name (e.g. “abc.pdf” as the first parameter and the IFS path (e.g. /MDRest4i/uploads/). Refer “Downloading the IFS files” section in this document for more details on how to use this function.

Copy Book member:	LXRRESTP

Arguments

● File name	200
● IFS Path	1024

Returns

None

18.4.18 wx( ) – Write XML tags and value

This procedure is used to write the xml response. If the request type is “SP”, it means this is special request (e.g. version information) and therefore write it directly. If the request type is “TB”, means the request is to write the beginning of the tag with the name of the tag received in “tag” parameter. If the request type is “TE”, it means the request is to write the end of the tag. If the request type is “DT”, it means write the begin of the tag followed by the value received in “value” field followed by the end of the tag. The actual writing happens via call to “Wrtxmldata” procedure.

Arguments

● tag	100
● value	1024
● request type	2

Returns

N/A

18.4.19 x( ) – Write XML tags and value

This procedure is used for writing the character string enclosed within the xml tags. The actual writing happens via call to “Wrtxmldata” procedure.

Arguments

1. tag	200
2. value	2048

Returns

N/A

18.5 Downloading attachments in REST service

You can receive the attachments received in POST, PUT REST service. As of now, the attachments of type PDF, JAR, WAR, ZIP, text, png, jpeg and json files are supported. Other file types can also be managed if required.

18.5.1 Default IFS path setting

The data area “DFTUPLOADP” which is set to “/MDRest4i/uploads/” by default is used to identify where the IFS files received in the REST services should be saved. This data area is part of MDRST library. If you would like to change the default location of saving the attachments received via REST APIs, please change the value of this data area to the actual path (making sure slash in the beginning as well as at the end). If the data area is not found or any error occurs, the files will be saved in “/MDRest4i/uploads” folder.

18.5.2 Saving IFS files in non-default folders

By default, the IFS files are saved in default folder as explained in the above paragraph. In some specific API, you may like to save the received files in different folders (e.g. zip file in one folder, PDF in another folder etc.). In order to do this, you have to set the indicator “n_autowrite” to *off in z_CustomInit subroutine of your REST API. In this case, you have to write the necessary logic in “z_ProcPost” soubroutine (for POST service) and “z_ProcPut” (for PUT service) where you have to check the names of received files and write at the required paths. The array “r_filercvd” (which is defined in copybook MDRESTDFN) will have the file names loaded. You can check the name of the file and call the function “WriteIFSFile” specifying the file name (e.g. myPDFFile.pdf) in first parameter and the IFS path (e.g. /home/myfolder/) as the second parameter. The variable “i_filercvd” maintains the number of files received in r_filercvd array. If the file received are for example 5, the paramer will be 6 and if the value of this parameter is 1, means no file has been received. The prototype of this function can be found in MDRESTDFN copybook. e.g. you can use below statement to write the IFS files:

```
for w_idx1 = 1 to i_filercvd-1;
  WriteIFSFile(%trim(r_filercvd(w_idx1)):'/home/lxr/uploads/');
endfor;
```

18.5.3 Pre-requisites for receiving the IFS files in REST service.

The global "Content-Type" at request level should be received from the request body as "multipart/form-data" if you want the REST service to be capable of accepting the IFS files. If the requestbody contains JSON, it must be in the beginning of the request (i.e. before any pdf, zip, jar, war files). The second requirement is that, the "Content-Type" should be repeated for each file to specify the file type. The PDF files should have “Content-Type” set to "application/pdf", that for zip files should be "application/zip" and the jar/war files should have “Content-Type” as "application/octet-stream". These are http headers and therefore they must be followed by hex ‘0d25’ (i.e. \r\n). The file name including the file extension should also be available as http header like " filename=abc.pdf" after the presence of specific "Content-Type" in the request body. The end of the file is identified by the presence of hex '0d25' (i.e. \r\n followed by the string boundary string (i.e. the value for the header “boundary” in the response).

18.5.4 Loading JSON request body/sending response via DB file

When the REST service receives the request body in JSON format, you can use “JsonToDB” procedure to load the JSON content of the request body to the DB file. Likewise, if you have JSON content available in DB file, you can use “JsonFromDB” function to build the JSON response. Please refer the section “Automated parsing and writing of JSON using DB File” in this document for more details.

18.6 Useful Producer Variables

18.6.1 Important Compiler Directive Switches*:*

LOGERRORS – Write the compiler directive statement “/DEFINE LOGERRORS” just before the copy statement for MDRESTDFN to disable the default exception handling in the REST services generated using LXRRESTC, MDRESTDFN, LXRRESTP copybooks. The logs are written in LXERRLOG file.

REQBODYPARM – If you expect the POST, PUT, PATCH, DELETE service receives the query parameters as part of the request body, write the compiler directive statement “/DEFINE REQBODYPARM” just before the copy statement for MDRESTDFN to instruct the REST service pick the query parameters from the request body.

SAVEIFSSWITCH – If you want to save the incoming requests and outgoing responses on the IFS directory, write the compiler directive statement “/DEFINE SAVEIFSSWITH” just before the copy statement for MDRESTDFN. This will log the request/response of this particular API in dedicated IFS folder or the overridden path as per the settings made in z_CustomInit subroutine.

TRACE – Write the compiler directive statement “/DEFINE TRACE” just before the copy statement for MDRESTDFN to enable debugging of the generated MDRest4i service. This compiler directive causes the DSPLY statement executed and this makes the REST service job to go to message wait from where STRSRVJOB followed by STRDBG could be used to debug.

TRACKTIME – If you want to monitor the performance of the API, you can track the time taken in different parts of the REST API. In order to track the time, write the compiler directive statement “/DEFINE TRACKTIME” just before the copy statement for MDRESTDFN to instruct the REST service track time in file under MDRST library.

18.6.2 Important Pointer Variables*:*

t_readPointer – This pointer is used to read the large string data received as part of the POST http request through “CgiStandardRead1” procedure. This pointer points to the user space LXRSCHEMA in QTEMP library if the size is up to 5MB but otherwise, it gets assigned the memory allocated from heap.

t_sendPointer – This pointer points to the “LXRSENDP1” user space in QTEMP library if the size of requested response is limited to 5MB, otherwise it contains to the address of the allocated heap memory. Whenever we use “w” procedure to write the data to standard output, the data is stored at this pointer which is later converted to UTF8 and send to the requester/consumer.

18.6.3 Important Indicator Variables*:*

n_autowrite – This indicator controls the downloading of the large files if the indicator “n_largeFiles” is set to *On. When this indicator is set to *On (& this is the default), the files are downloaded in default directory, otherwise, you can set to *Off and use function “WriteIfsFile” and write at the required IFS location.

n_getrspsize/n_getreqsize – By default, MDRest4i reserves 5MB memory area for the outgoing response. However, this response could be much larger. You can use “w_maxrsplen”/”w_maxreqlen” variables to specify the size of the request/response. However, if you would like to know the exact response size sent by the REST API or the exact request size received in the API, set the indicators “n_getrspsize” (for the response size) and “n_getreqsize” (for the request body including headers). If these indicators are set *On, the API will respond back with the size of request or response (whatever has been requested by setting the indicators). When “n_getrspsize” indicator is set *On, the API by default reserves 500MB memory space in two pointers (i.e. 1GB per call) from the heap. If you however know that the size won’t be that large, you can set the maximum expected size in “w_debug_max_rsp_size” (e.g. 20971520 for 20 MB) and it will then only allocate two pointers with this size. Be very careful in doing this as large memory from heap is getting allocated to get this information.

n_largeFiles – If you want to enable the API receiving large IFS files in the request body (i.e. more than 5MB in total size), set the indicator “n_largeFiles” to *On in z_CustomInit subroutine. In this case, you have to manually add the statement “ alloc(*teraspace) “ in control specification of the program.

n_LogErrors – When this indicator is set to *On, it will log the exceptions on LXERRLOG file.

n_reqbdyprmfnd: This indicator is used to instruct the REST API to pick the query parameters from the request body and is applicable only for POST, PUT, PATCH and DELETE type requests.

n_saveIFSSwitch – The logging of the http request in an IFS file (as explained in w_saveIFSPath variable above) happens only when this variable is set *On. You can set this variable to *On before inclusion of the copybook LXRRESTC.

n_sendUtf8 – This indicator is set to *On by default and that means the response will be translated to UTF-8 before transmitting to the requesting consumer. If the response you are writing is already in UTF-8 form, you can disable the UTF-8 translation by switching this indicator to *Off in z_CustomInit subroutine of your REST service.

n_trace – This indicator is used to enable debugging. It is initialized to *On if the literal “trace” is defined under the conditional compiler directive (e.g. “/define trace”) before the inclusion of MDRESTDFN copybook. If this variable is initialized to *On, the job processing the http request will go to message wait and you can then use STRSRVJOB followed by STRDBG to debug it. You can also initialize this indicator in z_CustomInit subroutine if you don’t want to initialize via compiler directive “define” statement.

n_trackTime: This indicator (when set to *On) records the time tracking information in TRKTIME table under MDRST library. The tracking provides the information about the time taken in different sections of the REST API Processing.

ng_mdrJobHdr – This indicator is set to *off by default and you can switch it to *On if you want to send the job information to the consumer as part of HTTP header when the specific REST service is being called. The purpose could be you want to know which of the several CGI jobs is processing your request so that you debug it. The header name would be “x-mdrsrvjob” and the value would be in form “MYRESTPGM-MYPGMLIB-MYSERVERJOB-nnnnnn” (where nnnnnn is the job number).

ng_UTF8Conv – If this indicator is set to *ON in subroutine z_Custominit,   two work variables available for use:
w_utf8reqbody - contains POST request body in UTF8 format up to 5mb in size
w_utf8response - place response body in UTF8 format in this variable before end of z_ProcPost subroutine execution

ng_oldmethod – This variable is set *Off by default and that means the writing of the response is being done using standard beginObject, addChar etc functions. However, if you want to do it via old "w" function of MDRest4i, set this indicator to *On in "z_CustomInit" subroutine. In that case, the data must be written by "w" function and anything written via addChar, beginObject etc will be ignored.

18.6.4 Important arrays*:*

r_filercvd – When large files are being downloaded, this array will provide the list of files which have been received. This array can be used in z_ProcPost or z_ProcPut subroutines. The number of files received would be available in the variable "i_filercvd". If the files received are for example 4, the value of "i_filercvd" will be 5 and therefore use this variable by reducing 1 from the value.

18.6.5 Important data Variables*:*

w_debug_max_rsp_size – When the indicator "n_getrspsize" is set *On to return the actual response size of the REST service, this variable can be used to specify the max debug response size for allocating the memory from heap storage and to find the response size. Refer the documentation of "n_getrspsize" in "Important Indicator Variables" section above.

w_maxrsplen/w_maxreqlen – The REST service by default reserves 5MB area for the outgoing response as well as the incoming request body. However, if your REST service is expected to receive or send larger data, you can use these variables to set the size. If the response size is expected to be larger than 5MB, set the value in "w_maxrsplen" in "z_CustomInit" subroutine of the program. Likewise, if you expect the POST/PUT/PATCH service may receive the request body larger than 5MB, set the variable "w_maxreqlen" in "z_CustomInit" subroutine. The value being set to either of these variables must be greater than 5MB (i.e. 5242880).

wg_Jsonlength – The length of the string to be translated from UTF-8 to EBCDIC should be stored in this variable whenever the string being translated is large and the JSON parsing is requested through the pointer tg_outptr. The call to JSONSAX procedure to parse the JSON uses the content at tg_outptr and length of the string from wg_jsonlength variable if the JSON string received in parameter is blank and tg_outptr is not null. Otherwise, it parses the JSON string received in the first parameter.

wg_prddalib – This variable can be assigned to the name of the library where LXERRH file exists. This is only required if you enable exception logging and you do not have MDRSTxx in library list of the http server. If the variable is not assigned, it will try to fetch the value from the MDRest4i Instance settings (i.e. call to "MRLOCINF" program).

w_reason – This is also related to the response of the http service. This is set to "OK" but if you are setting the above (i.e. "w_status" to some other status), you can set "w_reason" also to write the specific reason for the non-default response status.

w_reqbody – This variable is used to hold the request body received in the program. If there is multiform type content, it will only contain the JSON body which was received in first part of the request (i.e. before any file attachments). This variable will have the value available at the beginning of the z_Procxxx subroutines. This variable is 2MB in size.

w_response –This variable will hold the response which is being sent from the REST service. The subroutine "z_CustomExit" has been provided for manipulating the response. This is controlled by the compiler directive "LXR_CustomExit". If you want to use this variable, add the statement "/define LXR_CustomExit" just before the /copy statement on LXRRESTC. You will have to then define z_CustomExit subroutine in the program where you can use the variable "w_response" for whatever you want to check or act on the value. This variable is 2MB in size.

If you are setting the value of either variable greater than 16MB, you have to also add "alloc(*teraspace)" in H-spec (fixed form RPG) or CTL-OPT (in fully free form RPG) of your program to instruct the compile for using the teraspace.

w_saveIFSPath – This variable holds the IFS path where http request received in your REST service program will be logged. You can assign the path (e.g. /MDREST4i/temp/yourfile.json) in this variable in z_CustomInit subroutine of your program if you want to log the request at specific IFS path. If this variable is blank, the path to "log" folder is identified from the MDRest4i instance settings and the file name is named as "LXRLOGS" followed by timestamp.

w_status – This variable is set to '200' by default and is used to write the REST service response status. You can change the value of this variable in your REST API if you want to send some other status code (e.g. different success type or different failure type based on some data validations).

w_uploadpath – This variable name is used to hold the default upload path when the REST service receives PDF, ZIP, JAR or WAR files as part of POST, PUT request and you want to process them. If the variable is blank, it loads the path of "uploads" folder from the MDRest4i instance library settings.

w_utf8reqbody – contains POST request body in UTF8 format up to 5mb in size – *only available when ng_noUTFConv indicator is set to *on in z_Custominit*

w_utf8response – place response body in UTF8 format in this variable before end of z_ProcPost subroutine execution – *only available when ng_noUTFConv indicator is set to *on in z_Custominit*

19 JSON Handling

These functions are used in both the REST service and the consumer. They have the prototypes declared in MDRJSONYC and actual definition comes from MDRJSONY module available in LXRGLOBAL binding directory.

19.1 JSON Reading Functions

19.1.1 GetJsonStr

This function copies the full JSON string available at the JSON tree. This function is generally required in REST service when the JSON request body has been loaded to the tree. In consumer, this function can be used in order to receive the JSON part of REST service response. This copy is done at the pointer supplied as the first parameter. The second parameter is expected to have the value which is the memory size available at that pointer. The procedure returns the length of data copied at the supplied pointer.

Arguments

• Buffer Pointer	*
• Buffer Size	10i,0

Returns

Data Size	10i, 0

```
dcl-s w_maxreqlen int(10);
dcl-s w_readdta char(20480);
dcl-s t_readdta pointer Inz(%addr(w_readdta));
dcl-s w_rc int(10);
w_rc = GetJsonStr(t_readDta:w_maxreqlen);
```

19.1.2 GetRootNode

This function returns the address of root node for the JSON tree loaded via JSONSAX procedure call.

Arguments

None

Returns

Root Node Address	*

```
dcl-s t_rootNode pointer;
t_rootNode = GetRootNode();
```

19.1.3 JGetElementV

This function returns the value of simple array elements (e.g. “colours”: [“blue”,”green”,”black”]). In this case, if we call the function as JGetElementV(‘colours’:2), it will return the value ‘green’.

Arguments

• JSON Path	500A
• Array Index	10i,0

Returns

Element Value	500A

```
dcl-s w_path char(500);
dcl-s w_value char(500);
// {“customerId”:”C104780”,”orderDetails”:[{“orderID”:”P00578”,
“products”:[“P45678”,”P09849”]}],”creditLimit”:45000}
w_value = JGetElementV(‘orderDetails(1).products',2);
//w_value=’P09849’
```

19.1.4 JGetArrayIdx/ JGetArrayDim

This function returns the maximum array index of the last element supplied in JSON path. For example, the path is supplied as “country.city.schools”. In this case, suppose “country” and “schools” are the arrays and city is an object.

The function will return the maximum index of the “schools” array inside any element of the “country” array. However, if we supply “country(2).city.schools”, it will return the maximum index of schools array in the country array at index=2. This function therefore helps determine how data is loaded in the specific array.

Arguments

JSON Path	2048A

Returns

Array Index	10i, 0

```
dcl-s w_arrayDim int(10);
dcl-s w_path char(2048);
//In the example here, country is an array of objects, city is an object, area is again an array of objects and schools is an array
w_arrayDim = JPathN(‘country.city.area’);
//Above statement returns the maximum dimension of “area” array across all the “country” elements.

w_arrayDim = JPathN(‘country.city.area.schools’);
//Above statement returns the maximum array index of “schools” across all the “country”, “area” elements.

w_arrayDim = JPathN(‘country(2).city.area.schools’);
/Above statement returns the maximum array index of “schools” in second element of “country” and across all the “area” array elements.

w_arrayDim = JPathN(‘country(2).city.area(3).schools’);
/Above statement returns the maximum array index of “schools” in second element of “country” array and third element of “area” array.
```

19.1.5 JpathN

This procedure can be used to return the value of the requested JSON path in signed numeric form. If the requested path is not found or the value is non-numeric or the number of digits before decimal place is more than the value supplied in the second parameter, zero is returned. Otherwise, the numeric value is returned. If the value has negative sign, it is returned with sign.

Arguments

• Qualified JSON Path	2048A
• Length before decimal place	10i,0 options(*nopass)

Returns

1. JSON value for the specified qualified path	30P,9

```
dcl-s w_lenBefDec int(10);
dcl-s w_value packed(30:9);
// {“creditLimit”:64833.59 ,”balance”:5478432.25}
W_lenBefDec = 5
```

```
w_value = JPathN('creditLimit':w_lenbefDec);
//w_value=64833.590000000
w_value = JPathN('balance':w_lenbefDec);
//w_value=0.000000000
```

19.1.6 JpathU

This procedure is same as JPathV above, except that, it returns the value in upper case if the requested JSON path is found.

Arguments

• Qualified JSON Path	2048A
• IsNull	N Options(*nopass)

Returns

JSON value corresponding to the specified qualified path 10240A

```
dcl-s n_isNull ind;
dcl-s w_value char(2048);
// {"customerName":"Robert Wock" ,"creditLimit":null}
w_value = JPathV('customerName':n_isNull);
//w_value='ROBERT WOCK' and n_isNull = *off
```

19.1.7 JpathV

This procedure returns the value corresponding to the requested property (label in the qualified path). If the specified path not found, blank is returned. The second parameter is optional and if you expect the "null" values to be present in the supplied JSON and would like to track if any parameter was supplied with "null" value, supply second parameter. When the call to JPathV returns control back to program, check if the value in second parameter is *On which would mean the parameter is set as null in JSON.

Arguments

1. Qualified JSON Path	2048A
2. IsNull	N Options(*nopass)

Returns

JSON value corresponding to the specified qualified path 10240A

```
dcl-s n_isNull ind;
dcl-s w_value char(2048);
// {"customerId":"C104780","orderDetails":[{"orderID":"P00578",
"productDetails":{"packCode":"12","type":"box"}}],"creditLimit":null}
w_value = JPathV('orderDetails(1).orderID:n_isNull);
//w_value='P00578' and n_isNull = *off
w_value = JPathV('orderDetails(1).productDetails.packCode:n_isNull);
```

```
//w_value='12' and n_isNull = *off
w_value = JPathV('creditLimit':n_isNull);
//w_value = ' ' and n_isNull = *On
```

19.1.8 JpathVLong

This procedure returns the length of the JSON value and moves the value into the pointer specified in the second parameter.

Arguments

Qualified JSON Path for example "address.street"	2048A
A pointer name which should have set with the address of the variable where data is supposed to be returned. The JSON value is loaded into this pointer	*
The third parameter is supposed to contain the length of memory/variable being supplied at parameter# 2	10U 0
optional indicator that can be used to return the value was found to be null at specified path	n

Returns

Length of data found in JSON value	10u S

Example

```
d w_str5           s          81000
d t_str5           s               *
d wl_len           s             10u 0
d wl_len1          s             10u 0
………….
t_str5 = %addr(w_str5);
wl_len = 80000;
wl_len1 = JPathvLong(w_string:t_str5:wl_len);
```

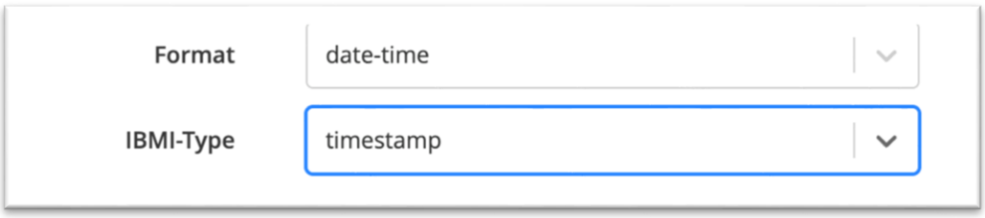
19.1.9 JpathZ

This is like JPathV 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.

Note: in the SDK UI, to have this function generated for a field in the parsing section of a provider or a consumer, the format must be set to **Date-Time**, and the x-ibmitype as **timestamp** and a maxlength of **26**



Format	date-time
IBMI-Type	timestamp

Arguments

@param (input) path **2048a** = character string representing a JSON path

@param (input/output) isNull **n** = Returns *ON if value is NULL, *OFF otherwise

Returns

@return value (in timestamp format) of the JSON data, or z'0001-01-01-00.00.00.000000' upon error.

19.1.10 JSONSAX

This function loads the tree from the JSON supplied in different possible formats depending on the values supplied in the first parameter. The possible values are "*IFS" or "*STDIN" or blank.

If *IFS is supplied, the second parameter is expected to have fully qualified IFS path starting from root till the file name (e.g. /home/MDRest4i/JSONToParse.json) and in this case, the JSON from this file will be read and parsed. e.g.

```
dcl-s w_error varchar(500);
dcl-s t_bufptr pointer;
dcl-s w_buflen int(10);
dcl-s w_path char(1024) inz('/home/MDRest4i/JSONToParse.json');
JsonSax('*IFS':w_path:t_bufptr:w_buflen:w_error);
```

If the value in first parameter is "*STDIN", the procedure will read the JSON from the standard input (which in case of the REST service is the request body). In this case, second, third and fourth parameters are ignored. E.g.

```
dcl-s w_error varchar(500);
dcl-s t_bufptr pointer;
dcl-s w_buflen int(10);
dcl-s w_path char(1024);
JsonSax('*STDIN':w_path:t_bufptr:w_buflen:w_error);
```

If the first parameter is neither "*IFS" nor "*STDIN", it is expected that the third parameter is being supplied as the address of the JSON buffer and fourth parameter has the length of JSON data available at that pointer. e.g.

```
dcl-s w_error varchar(500);
dcl-s w_path char(1024);
dcl-s t_bufptr pointer inz(%addr(w_jsonbuf));
dcl-s w_jsonbuf char(20480);
```

```
dcl-s w_buflen int(10);
w_jsonbuf = '{"customerId":"C104780","creditLimit":200000}';
w_buflen = %len(%trim(w_jsonbuf));
JsonSax(*blanks:w_path:t_bufptr:w_buflen:w_error);
```

Arguments

1.	Parsing Method	10
2.	IFS Path	1024
3.	Buffer Pointer	*
4.	Buffer Length	10i, 0
5.	Error	500a

Returns

N/A

19.2 JSON Writing Functions

19.2.1 addBool( )

This procedure can be used to write the indicator value as "true" and "false" to the response. It receives the label name as the first parameter and the **indicator value** as the second parameter. If the indicator is set to *On, it sends the value as "true", otherwise, "false".

Arguments

•	Label name	200a
•	Value	n

Returns

N/A

19.2.2 addChar( )

This procedure is called from REST service or consumer program to write the json label/value for an element. It receives the label name as the first parameter and the value of the label as the second parameter. It writes the label and value in JSON format. Escaping is performed in json value (if the value contains double quote or back-slash). If the value of label is blank, it only writes the value and this type of call is used to write the values for normal (i.e. non-object) array in JSON.

Arguments

• Label name	200a
• Value	2048a

Returns

N/A

Example

```
addchar('custId' : d_s.lx_custid);
addDeci('price' : d_s.lx_price);
addIntr('cusno' : d_s.lx_cusno);
w_cnt1 += 1;
if w_cnt1 = w_count;
  endObject();
```

19.2.3 addCurr( )

This procedure can be used to write the numeric values (e.g. currency) such that if the first character is decimal, it will prefix zero to avoid JSON error. It writes the label and value in JSON format.

Arguments

• Label name	200a
• Value	15P,2

Returns

N/A

19.2.4 addDeci( )

This procedure can be used to write the decimal values to the response. It receives the label name as the first parameter and the decimal value of the label as the second parameter. It writes the label and value in JSON format. If the value received doesn't have any digit before the decimal, zero is added before decimal to make it valid JSON.

Arguments

• Label name	200a
• Value	15P,5

Returns

N/A

19.2.5 addIntr( )

This procedure can be used to write the integer values to the response. It receives the label name as the first parameter and the integer value of the label as the second parameter. It writes the label and value in JSON format.

Arguments

• Label name	200a
• Value	15P,0

Returns

N/A

19.2.6 addNumber( )

This procedure can be used to write number values which have more than 5 decimals to the response. It receives the label name as the first parameter and the number value of the label as the second parameter. It writes the label and value in JSON format.

Arguments

• Label name	200a
• Value	100a

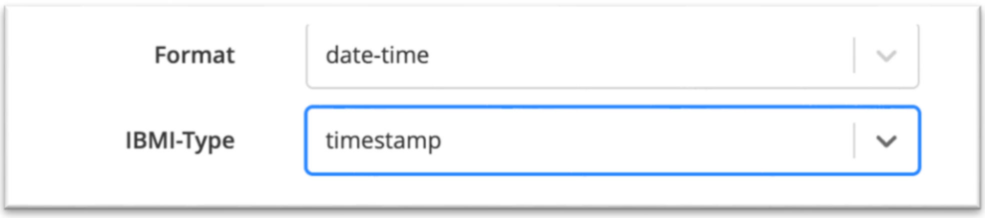
Returns

N/A

19.2.7 addTimestamp( )

This procedure can be used to write a timestamp that uses *ISO format(includes T and timezone suffix). It receives the label name as the first parameter and a valid RPG timestamp value as the second parameter. By default it uses UTC format, but this can be set with different input parameters, as described below.

Note: in the SDK UI, to have this function generated for a field in the parsing section of a provider or a consumer, the format must be set to **Date-Time**, and the x-ibmitype as **timestamp** and a maxlength of **26**



Format: date-time

IBMI-Type: timestamp

```
* MDR_addTimestamp
* Adds an ISO 8601 timestamp in:
*    YYYY-MM-DDTHH:MM:SS.sss
*    format to the JSON document.
*
*   @param (input)     name  = key name if within an object, or *omit
*   @param (input)     val  = standard RPG timestamp field
*   @param (input/opt) tz  = time zone to use, examples are:
*                  'Z' = UTC
*                  '+0500' = 5 hours east of UTC
*                  '-0600' = 6 hours west of UTC
```

```
 *                   '    ' = local time
 *                   '*UTC' (default) Convert to UTC
 *                   and report as a UTC timestamp
 *
 *  @param (input)     datesep = date separator (default: -)
 *  @param (input)     timesep = time separator (default: :)
 *  @param (input)     tzsep   = timezone sep (default: none)
 *
 *  @return returns yajl_gen_status_ok upon success,
 //        or generator status code upon failure.

d addTimestamp    pr                extproc('YAJL_ADDTIMESTAMP')
d  name                  65535    varying CONST OPTIONS(*VARSIZE:*OMIT)
d  val                      Z   CONST
d  tz                      6    CONST OPTIONS(*OMIT:*NOPASS)
d  datesep                   1    varying CONST OPTIONS(*OMIT:*NOPASS)
d  timesep                   1    varying CONST OPTIONS(*OMIT:*NOPASS)
d  tzsep                   1    varying CONST OPTIONS(*OMIT:*NOPASS)
```

Arguments

name	Varchar(65535) CONST OPTIONS(*VARSIZE:*OMIT);
val	Timestamp CONST;
tz	Char(6) CONST OPTIONS(*OMIT:*NOPASS);
datesep	Varchar(1) CONST OPTIONS(*OMIT:*NOPASS);
timesep	Varchar(1) CONST OPTIONS(*OMIT:*NOPASS);
tzsep	Varchar(1) CONST OPTIONS(*OMIT:*NOPASS);

Returns

yajl_gen_status_ok upon success, or generator status code upon failure

19.2.8 beginArray( )

This procedure is called from REST service or consumer program to open the object array or normal array. This procedure receives the label name as the parameter and if the name is blank, it writes open square bracket to the output pointer, otherwise it writes open square bracket with label.

Arguments

Label name	200a

Returns

N/A

19.2.9 beginObject( )

This procedure is called from REST service or consumer program to write labeled or non-labeled open curly bracket. The procedure receives the label name as the parameter and if the name is blank, it writes open curly bracket to the output pointer, otherwise it writes open curly bracket with label.

Arguments

• Label name	200a

Returns

N/A

19.2.10 endObject( )

This procedure is called from REST service or consumer program to write closing curly bracket.

Arguments

N/A

Returns

N/A

19.2.11 endArray( )

This procedure is called from REST service or consumer program to write the closing square bracket.

Arguments

N/A

Returns

N/A

19.2.12 wlong( )

This procedure can be used to write pre-formatted JSON. It is up to you to make sure the JSON is properly formatted and escaped. It is capable of much larger outputs Up to 5Mb.

Arguments

• pl_DataToSnd	1a	Data to be written to output (accepts data up to 5Mb)
• pl_datalen	10i 0	Length of data being sent in pl_DataToSnd

19.2.13 w( )

This procedure is deprecated but has been provided for backward compatibility of older version of MDRest4i. You would mostly use this function for the programs written using the older version of the product. In order to use this function, the variable “ng_oldmethod” must be set as *On. By default, this indicator is set to *Off and that means the writing of the JSON data is being done using standard beginObject, addChar etc functions. However, if you want to do it via “w” function, set this indicator to *On in “Initialize” procedure of consumer program or “z_CustomInit” subroutine of the REST service. When this indicator is *On, the data written by “w” function will be accepted and anything used via addChar, beginObject etc will be ignored.

Arguments

• http Data	20480a

Returns

N/A

19.3 JSON Utilities

19.3.1 CreateJSONF

While dealing with the JSON output to DB file or loading JSON data in DB file, this procedure can be called to create the empty object of “MDRJSONF” file from the MDRest4i product library (i.e. MDRSTxxx) to the library supplied as the first parameter. If the file already exists, no action is taken and “N” is returned back in the second parameter, otherwise, “Y” is returned back.

Arguments

• DB File Library	10A
• Success	1A

Returns

None.

19.3.2 ClearJSONF

This procedure can be called to delete the records of specified entry type from “MDRJSONF” file in the specified library. The first parameter is the library name where “MDRJSONF” file should be located, second parameter tells the entry type to be deleted. Third parameter returns “N” if error occurred while opening the file, otherwise, it returns “Y”. The last parameter returns the error message or the message indicating how many records have been deleted from the file.

Arguments

• DB File Library	10A
• Entry Type	3A
• Success	1A
• Status Message	50A

Returns

None.

```
dcl-s w_success char(1);
dcl-s w_stsmsg char(50);
```

```
ClearJSON('MDRDEMOD':'REQ':w_success:w_stsmsg);
//w_success = 'Y' if the file MDRJSONF in MDRDEMOD library was cleared successfully
//w_stsmsg contains error if any
```

19.3.3 cleanTree

This function deallocates the dynamic memory allocated from heap for storing the JSON data. This procedure must be called from the REST service program and from the consumer program at the end of processing. The REST services or consumers generated via MDRGENPRD or the SDK already have the call to "cleantree" procedure.

Arguments

None

Returns

None

19.3.4 DeleteJSONF

This procedure can be called to delete the object of "MDRJSONF" file from the library supplied in the first parameter. If any error occurs, "N" is returned, else "Y" is returned via second parameter.

Arguments

• DB File Library	10A
• Success	1A

Returns

None.

19.3.5 GetJsonFromDBF

When the JSON is parsed and loaded in MDRJSONF table, this procedure returns the value of specific JSON label from this file. The first parameter is the JSON Path till the last label for which the value is to be determined.

Arguments

• Path	2048A
• DB File library	10A
• Entry type	3A

Returns

JSON label value	10240

You can use this function like below:

```
dcl-s w_value char(10240);
dcl-s w_stsmsg char(50);
w_value = GetJsonFromDBF('country(1).countrydetails.citydetails(1).language'
:'MDRDEMOD':'RSP');
//Above example assumes the JSON where "country" is an object array at root level,
"countrydetails" is an object within "country" array, "citydetails" is an object array
```

```
and language is an element within "citydetails" array.The second assumption is that the
MDRJSONF file exists in MDRDEMOD library where JSON is loaded (i.e. through the use of
"CREATEJSONF", "JSONTODB" functions. The function will find out the presence of
"language" element inside first array index of "citydetails" inside the "countrydetails"
object inside the first element of "country" array.
```

If the value is not found, it will return the value '*notFound'.

19.3.6 JsonFromDB

This procedure reads the JSON content available in DB file named "MDRJSONF" and converts to JSON format for sending as request body from the consumer or for sending as response from the REST service. The first parameter expects the library name where the "MDRJSONF" file exists and third parameter is to return back the success or failure status in form of Y or N. The second parameter tells what type of entries are to be processed (e.g. "REQ", "RSP" or any other 3 characters that you have written the records with) from the DB file.

Arguments

• DB File Library	10A
• Entry Type	3A
• Success	1A

Returns

None.

19.3.7 JsonToDB

This procedure parses the JSON content available in string form and writes to the DB file named "MDRJSONF" in the specified library. The first parameter expects the library name where the "MDRJSONF" file exists. The second parameter tells the value of entry type field (e.g. "REQ", "RSP" or any other 3 characters) while writing the parsed JSON entries to DB file. The third parameter is to return back the success or failure status in form of Y or N.

The fourth and fifth parameters are optional. You don't have to specify them when you are loading the JSON from the request body of REST service or response in consumer. However, if you are writing an standalone program that contains JSON string and you want to load that JSON from string to DB, you have to send the address of that string in fourth parameter and the length of the JSON content as fifth parameter.

Arguments

• DB File Library	10A
• Entry Type	3A
• Success	1A
• JSON Pointer	* options(*nopass)
• JSON Length	10i,0 options(*nopass)

Returns

None.

```
dcl-s w_jsonstr char(10240);

dcl-s w_success char(1);

w_jsonstr = '{"customerId":"C104780","orderDetails":[{"orderID":"P00578", +
"products":["P45678","P09849"]}],"creditLimit":45000}';

// Create or Clear MDRJSONF

CreateJSONF('MDRDEMOD':w_success);
```

```
// Write response in MDRJSONF
JsonToDB(‘MDRDEMOD’:'DBR’:w_success:%addr(w_jsonstr):%len(%trim(w_jsonstr)));
```

19.4 Automated parsing and writing of JSON using MDRJSONF File

MDRESt4i has two useful JSON functions:

JsonToDB – will automatically parse incoming JSON and write the name value data into a generic, predefined database file/table called MDRJSONF

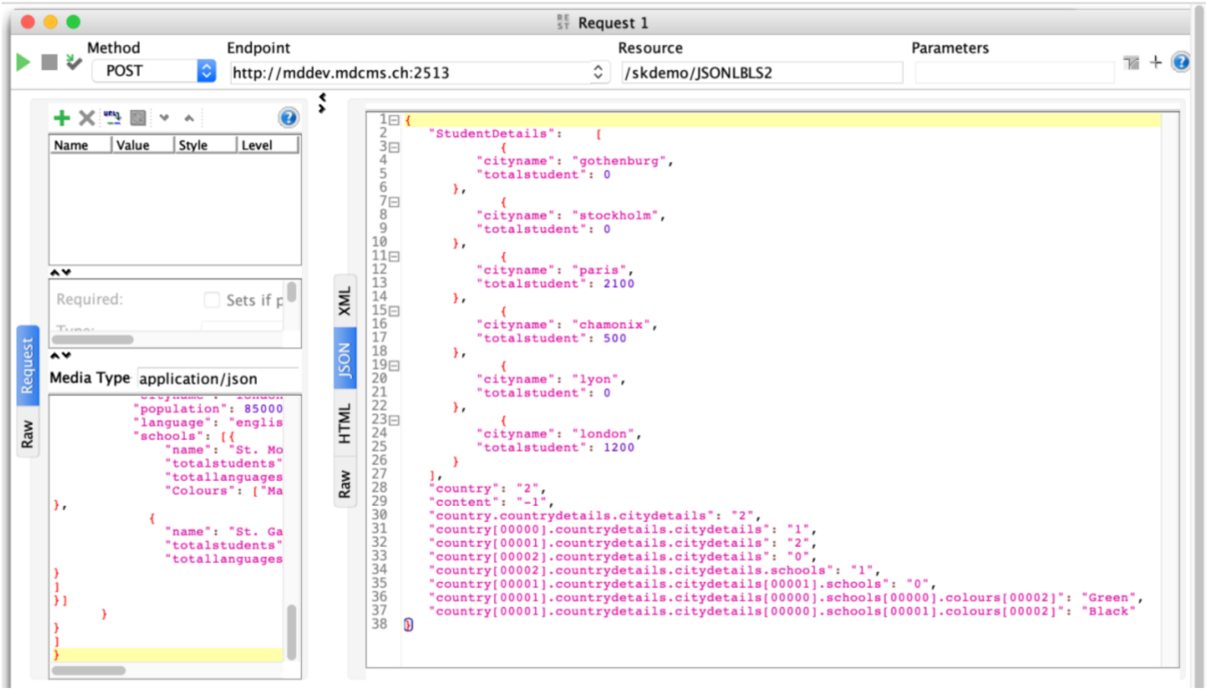
JsonFromDB – will build a structured JSON payload from a predefined database file/table called MDRJSONF.

The purpose of these is to avoid having to write any specific JSON parsing or JSON writing logic at all in the API or Consumer, and rather use SQL/native I/O to read or write the data being transported in JSON format.

For the full details of feature, please consult section ***“Automated parsing and writing of JSON using MDRJSONF File”*** in the ***MDRest4i_11_Tutorial_English*** document, and the JSON Utilities section above for the related functions such as CreateJSONF, ClearJSONF and DeleteJSONF

19.4.1 JSON Parsing Example

In order to understand what all functions you can use to extract the data from JSON arrays and objects, refer example JSONPARSE in QEXAMPLES source file under the product library (which has detailed documentation of each function used). The JSON content which is considered in this example is available in JSONPARSE source member in QEXJSON source file. In order to correlate the processing with the output, below is the response which is produced after calling it.



Method: POST
Endpoint: http://mddev.mdcms.ch:2513
Resource: /skdemo/JSONLBLS2
Parameters:
Media Type: application/json

```
            "population": 85000
            "language": "englis
            "schools": [{
                "name": "St. Mo
                "totalstudents"
                "totallanguages
                "Colours": ["Ma
},
            {
                "name": "St. Ga
                "totalstudents"
                "totallanguages
}
]
}]
        }
}
]
}
```

```
{
  "StudentDetails":    [
        {
      "cityname": "gothenburg",
      "totalstudent": 0
    },
        {
      "cityname": "stockholm",
      "totalstudent": 0
    },
        {
      "cityname": "paris",
      "totalstudent": 2100
    },
        {
      "cityname": "chamonix",
      "totalstudent": 500
    },
        {
      "cityname": "lyon",
      "totalstudent": 0
    },
        {
      "cityname": "london",
      "totalstudent": 1200
    }
  ],
  "country": "2",
  "content": "-1",
  "country.countrydetails.citydetails": "2",
  "country[00000].countrydetails.citydetails": "1",
  "country[00001].countrydetails.citydetails": "2",
  "country[00002].countrydetails.citydetails": "0",
  "country[00002].countrydetails.citydetails.schools": "1",
  "country[00001].countrydetails.citydetails[00001].schools": "0",
  "country[00001].countrydetails.citydetails[00000].schools[00000].colours[00002]": "Green",
  "country[00001].countrydetails.citydetails[00000].schools[00001].colours[00002]": "Black"
}
```

20 XML Handling

When you want to write the response to SOAP/REST services or you want to parse the XML content received as the request body, you can use various XML procedures for data writing or extraction depending on the requirement as explained below. These procedures are available in LXRXML module which can be bound by adding LXRXML binding directory. The prototype definitions of these functions are available in LXRXMLC copybook.

20.1 XML Writing Functions

20.1.1 wx( )

This procedure is used to write various types of XML entries to the output. If the requested type is ‘SP’, it means this is special request to just write the supplied content (e.g. XML version information) and therefore the value received in “tag value” is written to the output. If the request type is ‘TB’, the value supplied in “XML Tag” parameter is written as opening tag. If the request type is ‘TE’, the value supplied in “XML Tag” parameter is written as closing tag. If the request type is “DT”, this is same as “x” function above and therefore it writes the tag and value pair.

It also writes carriage return and new line characters (i.e. x’0d25’) at the end of the string.

Arguments

• XML Tag	200a
• Tag Value	2048a
• Entry Type	2a

Returns

N/A

Example:

```
wx(' ' :'<?xml version="1.0" encoding="UTF-8" ?>':'SP');
wx('root ' : ' ' :'TB');
wx('country':'UK':'DT');
wx('root ' : ' ' :'TE');
```

Above piece of code generates below XML content:

```
<?xml version="1.0" encoding="UTF-8" ?>

<root>

<country>UK</country>

</root>
```

20.1.2 x( )

This procedure is used to write tag and value pair in XML form. It also writes carriage return and new line characters (i.e. x’0d25’) after writing the requested tag.

Arguments

10. XML Tag	200a
11. Tag Value	2048a

Returns

N/A

Example:

```
x('country':'UK');
```

Above code would write the XML tag/value as below:

```
<country>UK</country>
```

20.2 XML Reading Functions

We will use below XML for explaining the functions in this section:

```
<?xml version="1.0"?>
<catalog>
   <address>Johannesburg</address>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <author>Bradbury, Brad</author>
      <author>George, Ray</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
        <author>George, Ray</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
   </book>
</catalog>
```

20.2.1 XGetArrayIdx

This procedure returns the maximum index of the array element at the specified path. e.g. for “catalog/book”, it will return the maximum index of “book” (if its an array like the example shown above). If you supply “catalog/book(1)/author”, it will return the maximum index of the “author” inside the second array element of “book” if we supply “catalog/book/author”, it will return the maximum index of “author” across all the elements of the previous tags. If the last element in the supplied path is not an array, -1 is returned.

Arguments

• Path	500a

Returns

Index Value	10i,0

```
wl_idx2  = xGetArrayIdx('catalog/book');
wl_idx2  = xGetArrayIdx('catalog/book(1)/author');
wl_idx2  = xGetArrayIdx('catalog/book/author');
```

The first statement above will return the value 1 (which means there are two elements, first one with zero and second with 1 index). The second statement will return the value 1 (because the second index of “book” array has two elements of “author”. The third statement will return value 2 (as there are three elements of “author” inside the first element of “book” and there are two elements of “author” inside second element of “book”.

20.2.2 XGetAttr

This procedure returns the value of the specified XML attribute (e.g. in <book id="bk102">, if we supply the value “book” in the first parameter and “id” in second parameter, it will return the string ‘bk102’. If the value is not found ‘*notFound’ is returned. If there are duplicate values like in above example, it will return the attribute of the last element processed. If you want to get the value of any other elements of the array type entries, you should use this function in callback procedure when that specific element has been processed.

Arguments

• Label	100a
• Attribute	100a

Returns

Attribute Value	1024

Example:

```
w_string=xGetAttr('book':’id’);
```

Assuming the above statement is called after XML parsing is complete, the value ‘bk102’ will be returned in the w_string variable.

20.2.3 XGetPathV

This procedure returns the tag or element value corresponding to the specified XML path e.g. “catalog/address”. In order to retrieve the value of the tag inside an array, specify the index qualification e.g. “catalog/book(0)/id”. If the value is not found ‘*notFound’ is returned. This procedure can be used to retrieve any attribute or tag value across the entire XML as long as you supply the complete path with the required array qualification.

Arguments

• Path	500a

Returns

Value	1024

Example:

```
w_string = XGetPathV('catalog/address');
w_string = xGetPathV('catalog/book(0).id');
```

Above code will return the value ‘Johannesburg’ from the first line and ‘bk101’ from the second line.

20.2.4 XGetValue

This procedure returns the most recent value of the supplied tag. If the tag appears multiple times in XML, XGetValue can only be used to retrieve the values of last occurrence of the tag and not for the previous ones. You can either use “callback” or XGetPathV function to get any occurrence of the tag. If the value is not found ‘*notFound’ is returned.

Arguments

• Tag	100a

Returns

Tag Value	1024

Example:

```
   w_string = XGetValue('address');
   w_string = XGetValue('price');
```

Assuming the above statements are executed after XML parsing is complete, the first statement will set the value 'Johannesburg' in w_string variable and the second line will set the value 5.95 in the same variable.

20.2.5 XGetXMLValue

Sometimes, the XML content/document is embedded inside the main XML. This procedure returns the entire embedded XML content for the specified tag/label if that tag has embedded XML content.

Arguments

• Tag	100a

Returns

XML Content	20480a

Example: In below example, the tag "rawxmlLogString" contains embedded XML (highlighted in blue).

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <rawxmlLogString>&lt;?xml version="1.0" encoding="UTF-8"?&gt;&lt;XMLLog
            xmlns="http://www.nxml.org/IXLog/namespace/"
            xmlns:dtv="http://www.xmlcorp.com/xstore/"
         XMLLog.xsd"&gt;
            &lt;Transaction
                xmlns:dtv="http://www.datavantagecorp.com/xstore/"
dtv:TransactionType="MAIN_SALE"&gt;
            &lt;/Transaction&gt;
        &lt;/XMLLog&gt;
        </rawxmlLogString>
    </S:Body>
</S:Envelope>
```

Below is the relevant section of code in REST service. You have to declare the variable "wg_embeddedxml" as "import" type and then assign the tag name which will contain the embedded XML content ("rasxmlLogString" in above XML). After the XML parsing has been completed, you can use the function "xGetXMLValue" which accepts the tag name as the parameter for the embedded XML. In output, you will get the embedded XML content.

```
d w_string          s            20480
d wg_embeddedxml    s               50a    import
……
……
Begsr Z_CustomInit;
```

```
   wg_embeddedxml = 'rawxmlLogString';
Endsr;
……
……
Begsr z_ProcPost;
w_string = xGetXMLValue('rawxmlLogString');
Endsr;
```

After execution of the above code, “w_string” would contain the string highlighted in blue color in the XML given as example above for this function.

20.2.6 XMLSax

This procedure initiates SAX parsing. The XML string up to 5MB can be supplied in the first parameter. The second parameter should have the procedure address of “callback” procedure.

Arguments

• XML String	5242880a
• Callback Pointer	*

Returns

N/A

20.2.7 XMLSaxF

This procedure initiates SAX parsing for the XML content inside the IFS file. The first parameter is the IFS path of the XML file including the file name. The second parameter should have the procedure address of “callback” procedure.

Arguments

• File Path	1024a
• Callback Pointer	*

Returns

N/A

20.2.8 XMLSaxUS

This procedure initiates SAX parsing for the XML content inside the user space. If your XML is in a program variable rather than the user space, you can supply the address of that string to the pointer “tg_inPointer” (exported from LXRXML). Otherwise, the first parameter is the user space name (i.e. library name + user space name). The second parameter is the length of the data to be processed from the user space. The third parameter should have the procedure address of “callback” procedure and fourth parameter is optional and can be supplied if you want to start processing XML content at specific position instead of the beginning of the user space.

Arguments

• User Space	20a
• Data Length	10i, 0
• Callback Pointer	*

• Begin Offset	10i, 0

Returns

N/A

20.2.9 XMLSaxUSE

This procedure is same as XMLSAXUS but it gets executed from LXRXMLE copybook when you have embedded XML (i.e. XML document inside XML document and this internal XML is to be parsed separately while the main XML is already getting parsed). You can therefore use this procedure in such scenario by adding copybook LXRXMLE in callback procedure of the main program.

Arguments

• User Space	20a
• Data Length	10i, 0
• Callback Pointer	*
• Begin Offset	10i, 0

Returns

N/A

Example:

```
p Callback        b
d Callback        pi
d   pl_event                    10a   const
d   pl_label                          like(w_labelD) const
d   pl_value                          like(w_valueD) const
d   pl_stackLvl                 10i 0 const
d   pl_xmlPath                        like(w_jPathD) const
d wl_inspect      s          20480
d wl_embededxml   s          20480
d wl_embededlen   s             10i 0
d wl_soapenvfld1  s            200
d wl_soapenvfld2  s            200
 /free
   wl_inspect = XRemoveIndex(pl_xmlPath);

   select;
   when pl_event = 'strElement';
   when pl_event = 'endElement';

   // Below copybook contains the logic of Embedded XML extraction
   // and send it to data queue for processing it in a separate job
   // This program will again to back to receive CGI job entry
 /copy qrpglesrc,lxrxmle

   // if the logic inside LXRXMLE has instructed to proceed working
   // on embedded XML, send the data queue entry
   if n_PrcEmbedded = *On;
     n_prcEmbedded = *Off;
     w_dqData = %trim(wl_embededxml);
     w_dqlen = wl_embededlen;
     QSendDtaq(w_DQName: w_DQLib: w_dqLen: w_dqData);
   endif;

   when pl_event = 'strDoc';
   when pl_event = 'endDoc';
   endsl;
 /end-free
p Callback         e
```

20.2.10 XRemoveIndex

This procedure removes the index along with the open and closing square brackets from the supplied path/string.

Arguments

• Path	1024a

Returns

Return String	1024a

Example:

```
w_string = xRemoveIndex('root/transaction[0]/amount');
```

Above code will set the value 'root/transaction/amount' in w_string.

20.2.11 XRemovePath

This procedure extracts the last element from the qualified path from the XML document (e.g. for the string "catalog/book/author" it will return "author").

Arguments

• Path	1024a

Returns

Return String	1024a

Example:

```
w_string = xRemovePath('root/transaction[0]/amount[1]');
w_string = xRemovePath('root/transaction[0]/amount');
```

The first statement sets the value 'amount[1]' in w_string whereas the second line sets 'amount'.

20.3 XML Utilities

20.3.1 CleanXML

When the XML document is embedded inside main XML content, the extracted portion (i.e. embedded XML content) has the values &lt;, &gt;, &apos;, &quot; etc. This procedure can be used to replace those values with the actual characters (e.g. '<' for '&lt;').

Arguments

• Search String	1024a

Returns

Return String 1024a

Example: Referring the example for “xGetXMLValue” function as explained earlier, we can add one more statement to call “cleanXML”.

```
d w_string         s           20480
d wg_embeddedxml   s              50a    import
……
……
Begsr Z_CustomInit;

   wg_embeddedxml = 'rawxmlLogString';

Endsr;
……
……
Begsr z_ProcPost;

w_string = xGetXMLValue('rawxmlLogString');
w_string = CleanXML(w_string);

Endsr;
```

The content in “w_string” before calling “CleanXML” procedure is below:

```
&lt;?xml version="1.0" encoding="UTF-8"?&gt;&lt;XMLLog

            xmlns="http://www.nxml.org/IXLog/namespace/"

            xmlns:dtv="http://www.xmlcorp.com/xstore/"

         XMLLog.xsd"&gt;

            &lt;Transaction

                xmlns:dtv="http://www.datavantagecorp.com/xstore/"
dtv:TransactionType="MAIN_SALE"&gt;

            &lt;/Transaction&gt;

        &lt;/XMLLog&gt;
```

After the execution of “CleanXML”, it would return below XML content:

```
<?xml version="1.0" encoding="UTF-8"?><XMLLog

            xmlns="http://www.nxml.org/IXLog/namespace/"

            xmlns:dtv="http://www.xmlcorp.com/xstore/"

         XMLLog.xsd">

            <Transaction

                xmlns:dtv="http://www.datavantagecorp.com/xstore/"
dtv:TransactionType="MAIN_SALE">

            </Transaction>

        </XMLLog>
```

20.3.2 XSearchNameSpace

This procedure returns the namespace prefix for the supplied namespace value.

Arguments

- Search String 1024a

Returns

Return String	1024a

Example:

```
w_string = xsearchNameSpace('http://schemas.xmlsoap.org/soap/envelope/');
```

It will return the value “S” considering below XML.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <rawxmlLogString>value</rawxmlLogString>
    </S:Body>
</S:Envelope>
```

20.4 Useful XML Variables

tg_outXML: This pointer variable contains the address of the next available memory bytes where data will be written by the “x” and “wx” procedures. The variable is set in LXRRESTC copybook for REST services. The pointer is incremented to point the next available space after writing the requested string.

tg_inPointer: When you are using XMLSAXUS procedure for parsing the XML content, you can set the address of the XML string in this pointer variable from the called procedure before calling XMLSAXUS procedure. This variable has to be declared with “import” keyword in the calling module which for REST services is already done in MDRESTDFN copybook. When the value is set in this pointer variable, the first parameter to XMLSAXUS is ignored, otherwise, it fetches the pointer to the user space name and library as supplied in first parameter and uses for SAX parsing of the content at that address.

wg_outXMLlen: When the data is written in XML form using “x” and “wx” functions as mentioned in “tg_outXML” pointer explained above, the length of the data is added in this variable and therefore it contains the complete length of XML content available at the output pointer. For REST services, this variable is declared in MDRESTDFN and used in LXRRESTC copybooks.

21 IFS Handling

These functions are used to perform the IFS file open, close, read, write operations. The prototypes are declared in LXRIFSPRO and the actual function definition exists in LXRIFS module. You can use the binding directory LXRIFS in control specification of the program where IFS operations are needed.

21.1 File Handling Functions

21.1.1 getErrorNo

Return the error number for the last error encountered.

w_errno = getErrorNo();

Returns

● Error Number	10i 0

21.1.2 iClose

Closes the file.

Arguments

• File Handle	10i 0

Returns

- Indicator
- On for file closed successfully
- Off for error

21.1.3 iFileHandle

Return the file handle for a file name if that file has been opened already using iOpenxx function.

Arguments

• File Name	1kB

Returns

• File Handle	10i 0

21.1.4 iFileName

Return the file name for a file handle if the supplied handle is valid and belongs to an open file.

Arguments

• File Handle	10i 0

Returns

• File Name	1kB

21.1.5 iFilePath

Return the IFS path for the supplied file handle if the handle is valid and belongs to an open file.

Arguments

• File Handle	10i 0

Returns

• File Path	1kB

21.1.6 iOpenA

Open file for append operation.

Arguments

• File Path	1kB

Returns

• File Handle	10i 0

21.1.7 iOpenN

Open a new file. If the file already exists, it will be deleted and recreated.

Arguments

• File Path	1kB

Returns

• File Handle	10i 0

21.1.8 iOpenR

Open file for read operation.

Arguments

• File Path	1kB

Returns

• File Handle	10i 0

21.1.9 strError

Call this procedure to get the error description for the last error encountered. The error number obtained via getErrorNo procedure should be supplied as the parameter. The return value should be accepted in a pointer and the text description of the error can be obtained either via based pointer or %str function.

Arguments

• Error Number	10i 0

Returns

- Pointer to the string containing error description

21.2 IFS Data Writing Functions

21.2.1 appendIFS

This function can be used when you want to append the data to an existing IFS file only once (i.e. no subsequent write/append operations in the same file). This function accepts two mandatory parameters (the file name where data would be appended and the variable holding the data) and one optional parameter. If the variable being supplied in second parameter has the data more than 20480 bytes, supply the length of the data as third parameter. It can be used similar to “writeIFS” example above.

Arguments

• File Name	200a
• Data to append	20480a
• Data Length	10i,0 options(*nopass)

Returns

- N/A

21.2.2 iW

Write a stream to a file. File must be open for write or append. The file must be open for write or append *(see iOpenA, iOpenN, iOpenR)*

Arguments

● File Handle	10i 0
● Write Stream	10kB

Returns

- Indicator
  - Off for error
  - On if write was successful

Example

```
d myFile        s                     10i 0
...
 /free
      myFile = iOpenN('/tmp/newFile.txt');
      iW(myFile, 'Hello I am a file');
..
      iClose(myFile);
```

21.2.3 iWs

Write a short string to an IFS file. The file must be open for write or append *(see iOpenA, iOpenN, iOpenR)*

Arguments

● File Handle	10i 0
● Write Stream	100A

Returns

- Indicator
- Off for error
- On if write was successful

Example

```
d myFile        s                     10i 0
...
 /free
      myFile = iOpenN('/tmp/newFile.txt');
      iWs(myFile, 'Hello I am a file');
..
      iClose(myFile);
```

21.2.4 iWnb

Write a number of characters to a file. This function can be used for streams that are too large for the 10kB restriction on iW(). The file must be open for write or append *(see iOpenA, iOpenN, iOpenR)*

Arguments

• File Handle	10i 0
12. Start Character	pointer or string of any size
13. Number of Bytes	10i 0

Returns

- Indicator
- Off for error
- On if write was successful

Example

```
h bnddir('LXRIFS':'LXRGLOBAL') dftactgrp(*no)
d/copy lxrifspro
d w_FileName      s              100a
d w_FileId        s               10i 0
d w_data          s            65535
d w_dtaLen        s               10i 0
 /Free
  w_filename = '/home/mdrest4i/logs/NewFile.txt';
  w_fileId = iOpenn(w_filename);
  w_data = '{"FirstName":"Stuart","LastName":"Milligan"}';
  w_dtalen = %len(%trim(w_data));

  iWnb(w_FileId: w_data: w_dtaLen);

   *Inlr = *On;
 /End-free
```

21.2.5 iX

Write a single XML tag and value. File must be open for write or append. *(see iOpenA, iOpenN, iOpenR)*

Arguments

• File Handle	10i 0
• XML Tag	1kB
• XML Value	10kB

Returns

- Indicator
- Off for error
- On if the write was successful

21.2.6 writeIFS

This function can be used when you want to write the complete data to an IFS file only once (i.e. no subsequent write operations in the same file). The function accepts two mandatory parameters (the file name where data would be written and the variable holding the data) and one optional parameter. If the variable being supplied in second parameter has the data more than 20480 bytes, supply the length of the data as third parameter.

Arguments

• File Name	200a
• Data to write	20480a

- Data Length 10i,0 options(*nopass)

Example

```
HBnddir('LXRGLOBAL') dftactgrp(*no)

 /copy lxrifspro
D  w_valuea        s        5242880a
D  w_filename      s           1024a
D w_datalen        s             10i 0
 /Free
  w_datalen = %size(w_valuea);
  w_filename = '/home/MDRest4i/BankingSwagger.json';
  readIFS(w_filename:w_valuea:w_datalen);
  w_datalen = %len(%trim(w_valuea));
  writeIFS('/MDRest4i/temp/test1.json':w_valuea:w_datalen);

  *Inlr = *On;
 /End-free
```

Returns

- N/A

21.3 IFS Data Functions

21.3.1 iReadLn

Read one line up to line feed. The file must be open for read.

Arguments

- File Handle 10i 0

Returns

- Buffer 10kB

21.3.2 iReadNb

Read a number of bytes from a file. The file must be open for read

Arguments

- File Handle 10i 0
- Number of Bytes 10i 0

Returns

- Buffer 64kB

21.3.3 readIFS

This function can be used when you want to read the complete data from an IFS file. This function has two mandatory parameters (i.e. the file name where data is to be read and the variable to return the data) and one optional parameter (i.e. data length). If your data is expected to be larger than 65535 bytes, supply the variable with enough size as the second parameter and supply the size of that variable as the third parameter.

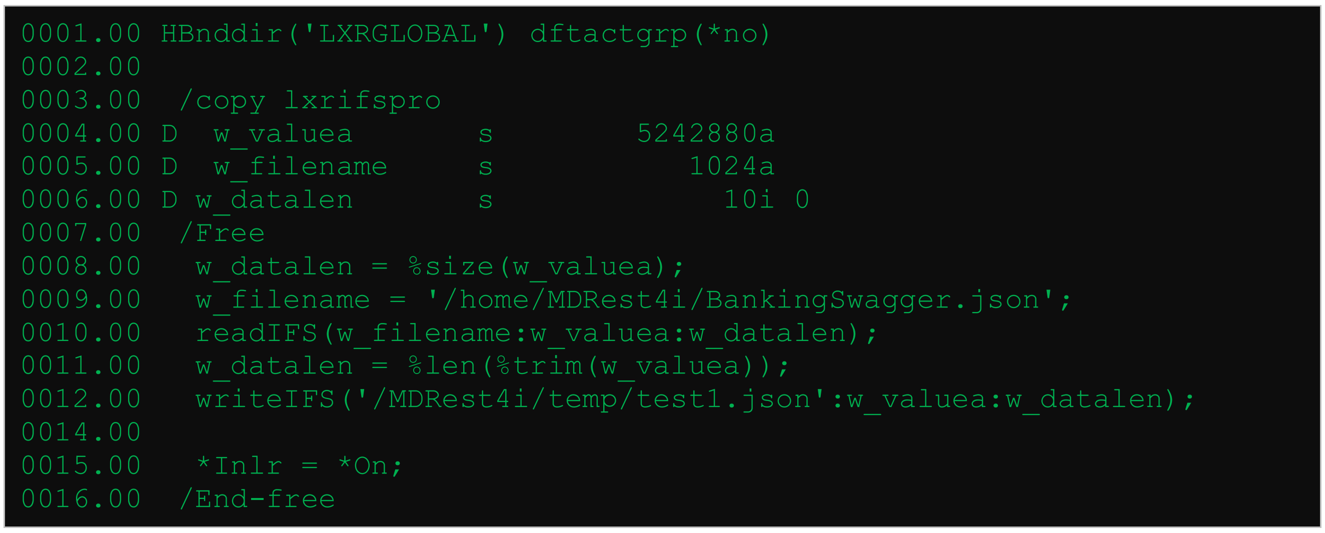
Arguments

● File Name	200a
● File Data	65535a
● Data Length	10i,0 options(*nopass)

Returns

- N/A

Example


```
HBnddir('LXRGLOBAL') dftactgrp(*no)

 /copy lxrifspro
D  w_valuea        s         5242880a
D  w_filename      s            1024a
D w_datalen        s              10i 0
 /Free
  w_datalen = %size(w_valuea);
  w_filename = '/home/MDRest4i/BankingSwagger.json';
  readIFS(w_filename:w_valuea:w_datalen);
  w_datalen = %len(%trim(w_valuea));
  writeIFS('/MDRest4i/temp/test1.json':w_valuea:w_datalen);

  *Inlr = *On;
 /End-free
```

21.4 Read/Write Pointer movement Functions

21.4.1 iShiftA

Shift the read/write pointer to absolute position relative to the first byte of the file.

Arguments

● File Handle	10i 0
● Absolute Position	10i 0

Returns

- Indicator
- On for error
- Off if shift was successful

21.4.2 iShiftL

Shift file offset (i.e. read/write pointer) to the left by the specified number of bytes.

Arguments

● File Handle	10i 0
● Number of Characters	10i 0

Returns

- Indicator
- On for error
- Off if shift was successful

21.4.3 iShiftR

Shift file offset (i.e. read/write pointer) to the right by the specified number of bytes.

Arguments

• File Handle	10i 0
• Number of Characters	10i 0

Returns

- Indicator
- On for error
- Off if shift was successful

22 Language Translation Functions

These functions are used in both the REST service and the consumer. In REST service, the final response is first converted to UTF-8 and then sent to the requester. When the service has received request body, it is first translated from UTF-8 to host code and then utilized for JSON or XML parsing. The prototype definitions are available in LXRLANGC and actual procedure definition exists in LXRLANG module. You can declare BNDDIR('LXRLANG') in control specification of your program where language translation functions are required.

22.1 Convert

Converts a string from one language to other language. The returned string is sent in wg_output_value but that variable has the limitation of 32700 characters. If the string to be translated is expected to be longer, it's better to allocate the memory in TG_LANGPTR variable which is exported from LXRLANG module and can be imported in the program using it. The logic in this procedure checks if the pointer is not null, the conversion string is stored at TG_LANGPTR pointer and the caller can use basing pointer or %str built-in function to get the translated string. The translated string length is returned as the output. This length might be different than the one in received format.

Arguments

1. Input_Pointer	*
2. Input_Pointer_Length	10i,0

Returns

• Length of Translated String	10i,0

22.2 SetConvert

Allocates the language CCSID Elements and opens the conversion descriptor:

Arguments

1. InputCCSID	10i,0
2. OutputCCSID	10i,0

Returns

Return Code	10i,0

0 for success and -1 for failure

22.3 EndConvert

De-Allocate the language CCSID Elements.

Arguments

1.	Conversion descriptor	10i,0

Returns

Return Code	10i,0

0 for success and negative for failure

23 General Utility Functions

There are some other functions in different modules that can be used for some specific purpose. Below sections describe different types of functions.

23.1 Global Functions

These general purpose functions have the declaration in LXRGLOBALC copybook and the actual definition in LXRGLOBAL module. The binding directory LXRGLOBAL is used to access these functions.

23.1.1 Cap

This procedure translates the received string first to lower case and then the first character is translated to upper case. The translated string is returned from the procedure.

Arguments

• Input String	1024

Returns

• Output String	1024

```
myString = 'AbCdEF';
Capitalized = Cap(myString);
// Capitalized = 'Abcdef';
```

23.1.2 CountParms

This procedure counts the number of arguments/parameters in URL query string.

Arguments

• Query String	1024

Returns

• Number of parameters	5P,0

23.1.3 DecodeURL

This procedure is to decode the encoded URL. Supply the URL encoded string and it will return the actual URL string.

Arguments

• Input Number	4096A

Returns

• String	4096A

23.1.4 DetermineParm

This procedure returns the value of a given parameter in a URL query string. If the value is not found, the function returns *notFound.

Arguments

• Parameter Name	20A
• Query String	1024A

Returns

• String	200A

23.1.5 EditN

This procedure edits a number. It removes the leading and trailing zeroes as well as the decimal point for integer values.

Arguments

• Input Number	15P,5

Returns

• String	20A

23.1.6 EncodeURL

This procedure is for the url encoding in REST consumers. Supply the URL string and it will encode the eligible characters in URL string and return the converted string.

Arguments

• Input Number	4096A

Returns

• String	4096A

23.1.7 PCode

This procedure converts all percentage encoded strings to corresponding EBCDIC characters.

Arguments

• Input String	32767

Returns

• Output String	32767

23.1.8 SQLMsg

This Procedure can be used in REST APIs and Consumer programs to return SQL error message in a parameter.

Arguments

• SQL Message	200

23.1.9 Upcase

This procedure translates the received string to upper case and returns the translated string.

Arguments

• Input String	1024

Returns

• Output String	1024

23.2 CGI Functions

The sending/receiving of the data over the REST/SOAP requires CGI APIs/functions to read the data from the incoming connection channel and write the response. The source member LXRCGIPRO contains the prototype declarations for various CGI operations with the target procedure provided in C language library. These functions are accessed through the binding directory names “CGIBNDDIR”.

23.2.1 CgiGetEnvironmentVariable

Retrieve a CGI environment Variable value.

Arguments

1. Buffer	pointer to string
2. Buffer Length	10i 0
3. Response Length	10i 0
4. Variable Name	64
5. Variable Length	10i 0
6. Error	16A

This is a CGI standard API.

23.2.2 CgiStandardRead

Read the input buffer in a CGI program.

Arguments

1. Buffer	pointer
2. Buffer Length	10i 0
3. Response Length	10i 0
4. Error	16A

23.2.3 CgiStandardRead1

Read the input buffer in a CGI program.

Arguments

1.	Buffer	pointer to string for storing the data
2.	Buffer Length	10i 0
3.	Response Length	10i 0
4.	Error	32767A

This is a CGI standard API.

23.2.4 CgiStandardWrite

Write to the CGI output stream.

Arguments

1.	Buffer	pointer to string
2.	Buffer Length	10i 0
3.	Error	16A

23.2.5 CgiStandardWrite1

Write to the CGI output stream.

Arguments

1.	Buffer	pointer
2.	Buffer Length	10i 0
3.	Error	16A

23.2.6 CgiConvertToDbFile

CGI convert to database file and the function maps to the CGI API QtmhCvtDb.

Arguments:

1.	Qualified File Name	20A
2.	Buffer	pointer
3.	Buffer Length	10i 0
4.	Structure	60A
5.	Structure Length	10i 0
6.	Actual Length	10i 0
7.	Response Code	10i 0
8.	Error	16A

23.2.7 Getenv

Retrieves the environment variable.

Example:

Arguments

1.	Environment Variable	pointer (String variable name)

Returns

1.	Environment Variable Value	pointer (String result)

Example

Below statement fetches the content length available on POST http request.

```
w_dtalen = %int(%str(getenv('CONTENT_LENGTH')));
```

24 HTML Handling

In order to provide html capabilities for sending the data in html form, MDRest4i provides the module LXRHTML and the binding directory of the same name. The prototypes are defined in LXRHTMLC and LXRHTMLPRO copybooks.

24.1 HTML Writing Functions

24.1.1 HSetValue

Changes the value of a replacement variable.

Arguments

1.	Field Name	50A
2.	New Value	2kB
3.	Force Long	indicator

Returns

- Indicator
- On if the value was set
- Off if the value was not set

24.1.2 HWrite

Writes a free format string to the HTTP server.

Arguments

• Buffer	64kB

Returns

- Indicator
  - On for successful write
  - Off for error

24.1.3 HWriteLn

Writes a free format line to the HTTP server.

A line feed character is appended to the end of the string.

Arguments

• Buffer	32000 bytes

Returns

- Indicator
- On for successful write
- Off for error

24.1.4 HWriteSection

Writes a section of HTML to the HTTP Server. The first occurrence of the section is written i.e. if the same section appears on two files, the section from the first file will be written. The replacement values are handled at this stage. The sections are written immediately and are not buffered for consolidated write operation.

Arguments

• Section	50

Returns

- Indicator
- On for successful write
- Off for error

24.2 HTML Reading Functions

24.2.1 HGetEnv

Get the Value for a CGI Environment Variable.

Arguments

• Environment Variable Name	64A

Returns

• Environment Variable Value	64kB

24.2.2 HGetQuery

Retrieve the Query portion of the URL for HTTP Get method.

Returns

• Query String	64kB

24.2.3 HLoadHTML

Load an HTML file into a user space.

Arguments

• File Path	1kB

Returns

- File Load Indicator
- On if the file was loaded successfully
- Off if the file failed to load

24.2.4 HReceive

Receive posted data into user space LXRHTMLIN in library QTEMP.

Returns

- Indicator
- *On for success
- *Off in failure

24.2.5 HReset

Initialise HTML work area.

24.2.6 HUnpack

Gets a value from a form that was sent using the POST method

This function requires a call to HReceive().

This function is depreciated because Web2.0 browsers post form data differently.

Arguments

• Field Name	50A

Returns

• Field Value	2kB

25 CSV Functions

These functions are used to perform CSV file operations. The copybook LXRCSVP contains the prototype definitions and the actual definitions exist in LXRCSV module which is attached to binding directory LXRCSV.

25.1 CSV

Parse a CSV string and return an Array of 100 element of 1000a.

Arguments

1. CSV String	100000A

Returns

- Array of 100 elements.

- Each element is 1000A.
- Each element corresponds to a column
- The index for the array is the column number

Example

```
myFile = iOpenR('/path/file.txt');
inData = iReadLn(myFile);
fields = CSV(inData);
itemNo = %int(fields(1));
description = fields(2);
price = %dec(fields(2): 11: 2);
```

Further Information & Support

For information about support, training, education and customization services, please contact us at:

support@midrangedynamics.com