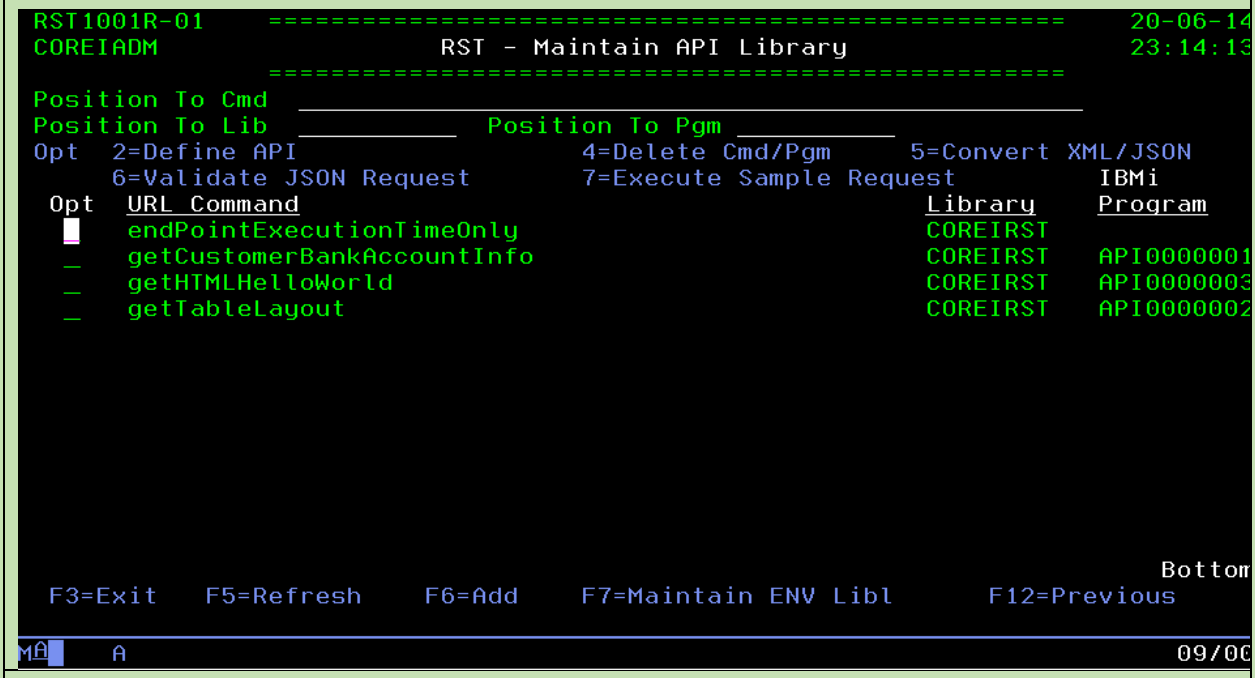
Core-i™ RST

REST/Webservice Framework

(middleware and API engine)

v 1.01 – Created 05/28/2020

OPTION 1. MAINTAIN API LIBRARY



Overview

This screen provides a security admin/developer with the ability to create, maintain, validate and execute the backend API program. If you choose to build your own API's to use with the library and framework and NOT use the auto-generate API feature, it is important to know that an API program can be any *PGM object. Core-i RST highly recommends creating those *PGM objects as type SQLRPGLE. The Core-i RST middleware component presumes the *PGM is of type SQLRPGLE, and if it is, then the API can also be called as an SQL stored procedure which returns an sql result set.

- **URL Command**
  - Every API will have an associated command. This command will appear in either 1 of 2 ways in which the API can be called.
    - REST method
      - getCustomerBankAccountInfo/1023
    - JSON request

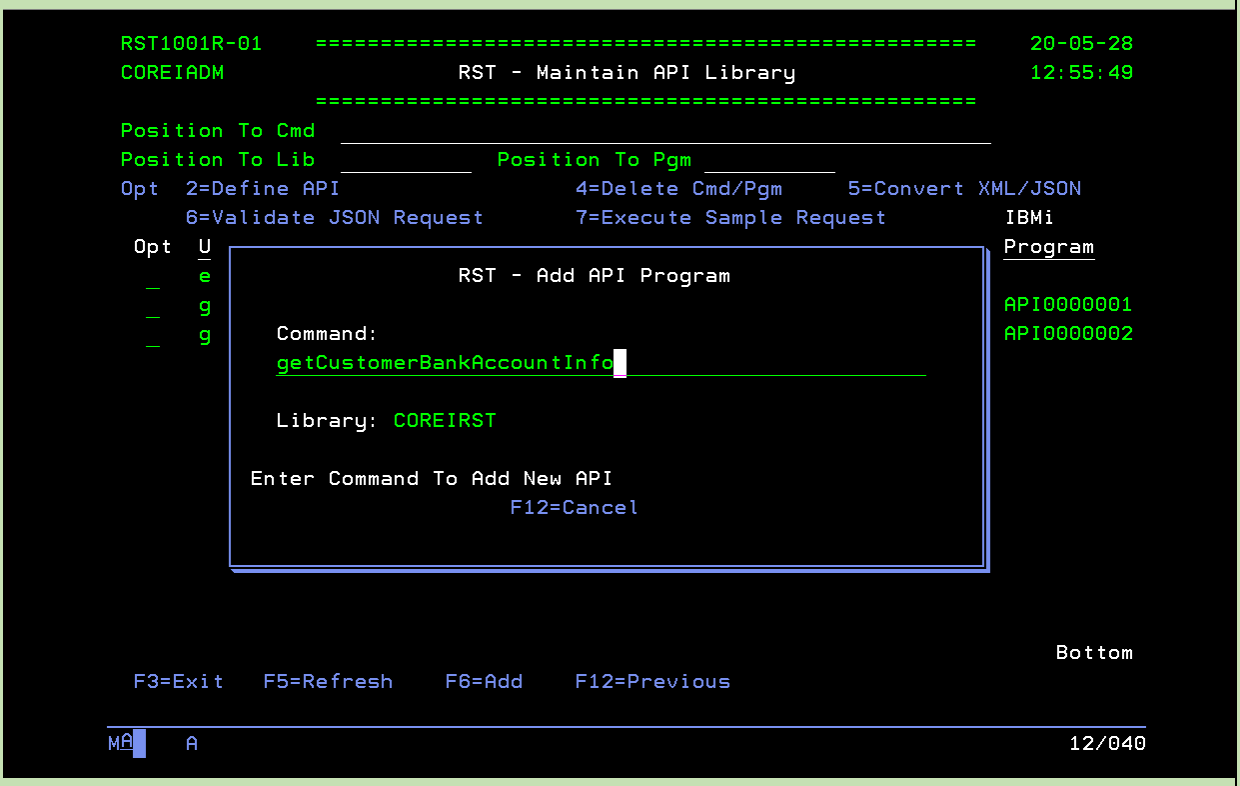
```
{
```

```
  "env":"xxx",
  "command":"getCustomerBankAccountInfo",
  "payload":[
    {
      "custNo":"11111"
    },
    {
      "custNo":"22222"
    }
  ]
}
```

- **Library**
  - This is the library where the API *PGM will live and be invoked from.
  - The auto-API feature only allows API's to be created in COREIRST.

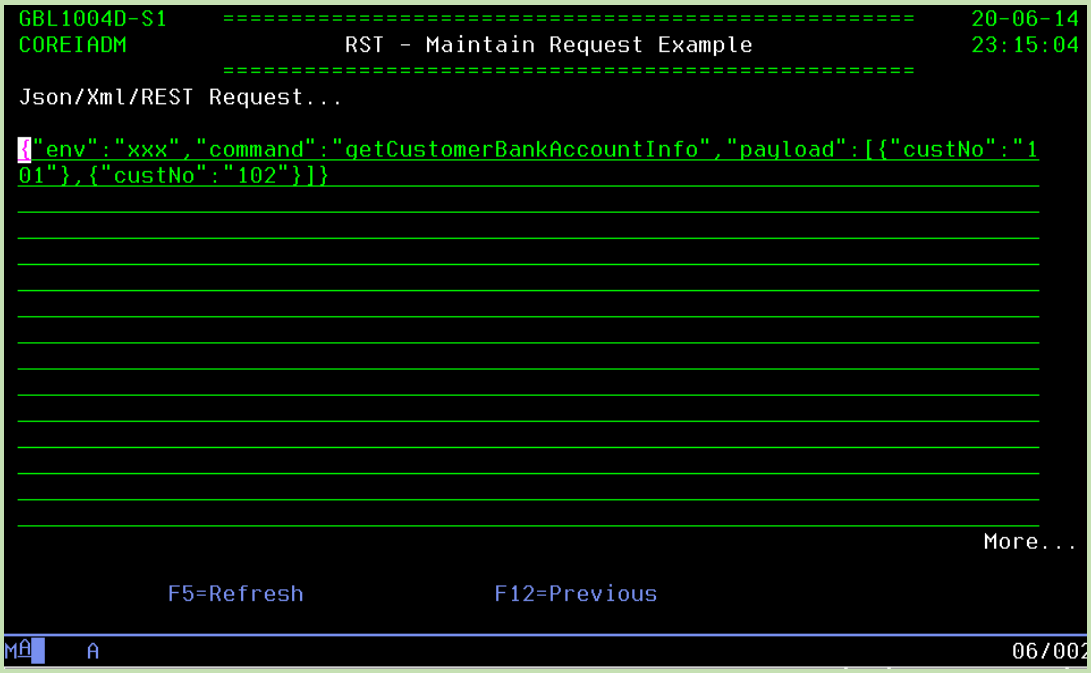
- **IBMi Program**
  - When F6=ADD is executed, Core-i RST will always create the next API program based on the APIxxxxxxx objects in COREIRST. So for example, when API0000005 is found in COREIRST, when the next auto-API is requested to be created, it will look in COREIRST library and find the API0000005 and create the next API program as COREIRST/API0000006.
  - When the auto-API creates a new API *PGM, the source is placed in COREIRST/COREIRST_S.

User Option: F6=Add



- When F6=Add is invoked, the first thing that is required is a command name.
  - Enter the command in camel case to conform to a style standard
  - The library defaults to COREIRST and does not allow to be modified.
  - Press enter or F12 to cancel
  - At this point, the rest of the process will be the same functionality of 2=Define API

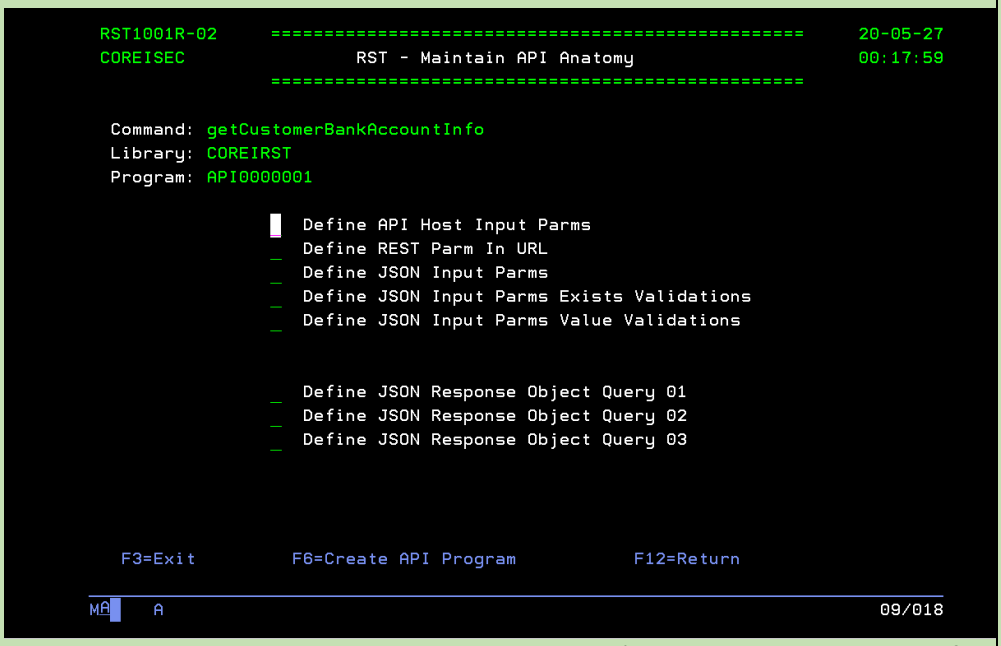
- **Maintain Request Example**



  - Core-i RST provides 32000 characters to allow you to specify your REST, JSON, or XML request, via the green screen maintenance. This is simply the ability to make quick simple tweaks to your request that can actually be up to 8MB in size.
  - About the structure
    - Env
      - Environment allows you to specify a 3 character environment code that you can associate a library list with. The env/library list maintenance is accessed via the F7=Maintain ENV Libl from the *RST – Maintain API Library* option. If an environment is specified that is not found to be setup, then the API job will continue with the same library list as defaulted for the user. This is a great feature for testing your API's within a DEV/QA/PRD environment layer.
    - Command
      - Every API must have an associated command name. It is required when the API is created with the F6=Add from the *RST – Maintain API Library* option. This is what determines whichh IBM I *PGM to be called.
    - Payload
      - The payload is the main parameter input portion of the JSON request structure. It contains nested objects, in which each nested object will result in a nested object response. In this example, two different customer numbers are requested. You may specify 1 or many nested objects.

- The 5250 subfile editor above is a means for reviewing and making small tweaks to your request structure. It is understood that this is a rigid method for making large scale edits to the structure, so in that case, it is advised to copy/paste the structure to a text editor that is 75 chars per row.
- Once the structure is copied back into the subfile (or inserted with SQL into table COREIRST.RST00001T.tjsonin, the option 6. Validate JSON Request can be executed (from Maintain API Library), to ensure the structure remains valid.
- Methods supported
  - REST
    - getCustomerBankAccountInfo/1023
  - JSON
    - {"env":"xxx","command":"getCustomerBankAccountInfo","payload":[{"custNo":"11111"}, {"custNo":"22222"}]}
  - XML
    - <payload><custNo>11111</custNo></payload><payload><custNo>22222</custNo></payload><env>xxx</env><command>getCustomerBankAccountInfo</command>
- JSON/XML elements
    - env
      - Not actually linked to any functionality at the moment, but plans to allow library lists to be set according to env values.
    - command
      - The command is what the middleware associates with the backend IBMi API program.
    - Payload
      - Contains the core input parameters of the request that the backend API library will utilize to build the response.
    - Core-i RST has plans to enhance the JSON/XML with many new elements that the Core-i RST will work with to enhance the functionality of the framework. For now, the following elements are required.

- **Maintain API Anatomy**



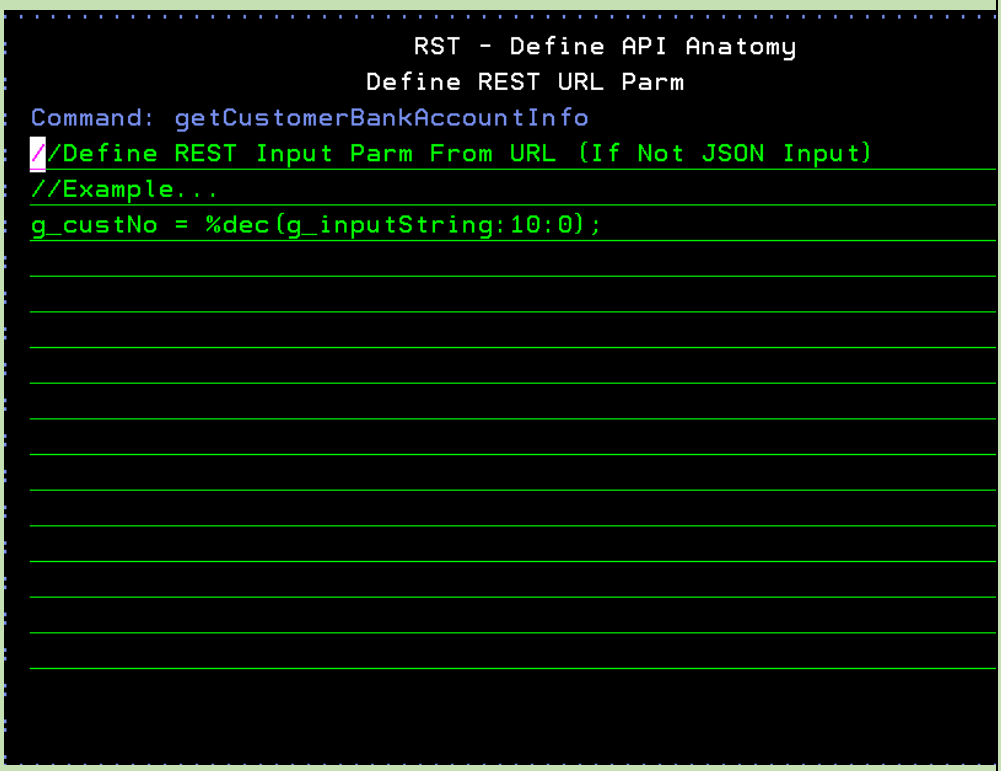
  - Any API program that was generated with auto-API (which we hope will be all of your API's), you have the ability to configure and maintain the crtical dynamic specifications of the query from the screen above.

- **Define API Host Input Parms**

```
                          RST - Define API Anatomy
                     Define Host Input Parameters
Command: getCustomerBankAccountInfo
//Define Host Vars Which Correspond To JSON Input Parms
//Example...
g_custNo    zoned(10:0) inz;
```

- This code snippet will be a free-format RPG syntax
- The API program global host variables which reference the request will be defined here.
- Essentially we will be defining variables in a ds_inputParms.
- When the API *pgm is generated, this source snipped will be constructed as…
  - dcl-ds ds_inputParms;
    //?JSONHOST?
    //Define Host Vars Which Correspond To JSON Input Parms
    //Example...
    g_custNo   zoned(10:0) inz;
    end-ds;

- **Define REST URL Parm**



- This code snippet will be free-format RPG syntax.
- It is possible with Core-i RST API's to utilize both a REST and/or JSON/XML request. Yes, both can be executed in a single API program. Before we begin to cover the details of extracting the REST parms, let's look at the auto-generated API code for this code snippet…

    - ```
      //extract only the REST parm...
      g_inputString = %subst(g_inputString
                             :%scan('/'
                             :g_inputString) + 1);
      //?RESTPARM?
      //Define REST Input Parm From URL (If Not JSON Input)
      //Example...
      g_custNo = %dec(g_inputString:10:0);
      ```

- You will see in the code snippet above that Core-i RST will auto generate the API program and place the entire REST command/parm inside a g_inputString host variable. Then that command is stripped from the g_inpuString along with the "/", leaving you only with the parm value(s) in g_inputString… so in this case g_inputString would contain '1023'
      ```

- **REST methods with a single parameter**
  - getCustomerBankAccountInfo/1023
  - simply define the API host variable g_custNo as the decimal version of the g_inputString.
- **REST methods with multiple parameters** –
  - getCustomerBankAccountInfo/1023/2030405
  - Simply parse the g_inputString and extract values between the "/" character.
  - An example can be found in the getTableLayout API that came as a demo with Core-i RST. getTableLayout/coreirst.rst00001t

  - ```
    g_schema = %subst(g_inputString
                      :1
                      :%scan('.'
                             :g_inputString) -1);
    g_table  = %subst(g_inputString
                      :%scan('.'
                             :g_inputString) + 1);
    ```
    ```

- **Define JSON Input Parms**

```
:                          RST - Define API Anatomy
:                      Define JSON Input Parms
: Command: getCustomerBankAccountInfo
: //Define Json Input Parms From JSON Input
: //Example...
: "custNo" decimal(10,0)
:
:
:
:
:
:
:
:
:
:
:
:
:
```

- This will be rpg embedded sql syntax.
- Like the rpg global host variables, we are defining the JSON request input parameters that we will extract values into our rpg host variables.
- So essentially, any global rpg host variable that was defined prior under (Define API Host Input Parms), we will define them here as well but in embedded sql syntax)
- This particular API only has one incoming request parameter, but if it had two we would just continue with the syntax as…
- "custNo" decimal(10,0)
    - ,"inputParm2" char(5)
    - ,"etc" numeric(7,2)

- **Define JSON Input Parms Exists Validations**

```
                          RST - Define API Anatomy
                  Define JSON Input Exists Validations
Command: getCustomerBankAccountInfo
//Define JSON Input Parm Exists Validation
//Example...
p_errMsg = %trim(psds.program) + ' - custNo not found ' +
           'in JSON input';
```

- This code snippet will be free-format RPG syntax.
- This is essentially a continuation from the prior definition where we defined the sql json column names.
- To ensure requests are providing the input parameters we are expecting and require, we will be sure they are in the JSON/XML request. If they are not, we will assign the error here.
- Again, if there were multiple input parameters within the request structure, we would simply duplicate this code statement for every column we have defined in *Define JSON Input Parms*

- **Define JSON Input Parms Value Validations**

```
                          RST - Define API Anatomy
                  Define JSON Input Values Validations
Command: getCustomerBankAccountInfo
/Define JSON Input Parm Values Validation
//Example...
when g_custNo = 0;
  p_errMsg = %trim(psds.program) + ' - Input Parm Value Requ
             'For Parm custNo';
```

- This code snippet will be free-format RPG syntax.
- Just like the prior edit where we check to be sure the expected request input parameter is present, we will not be sure that it contains a value we are expecting.
- So for this particular API we are just ensuring the customer number was sent to us with at least some numeric value in it.
- Again, if there were multiple input parameters within the request structure, we would simply duplicate this code statement for every column we have defined in *Define JSON Input Parms*

- **Define JSON Response Object Query 01**

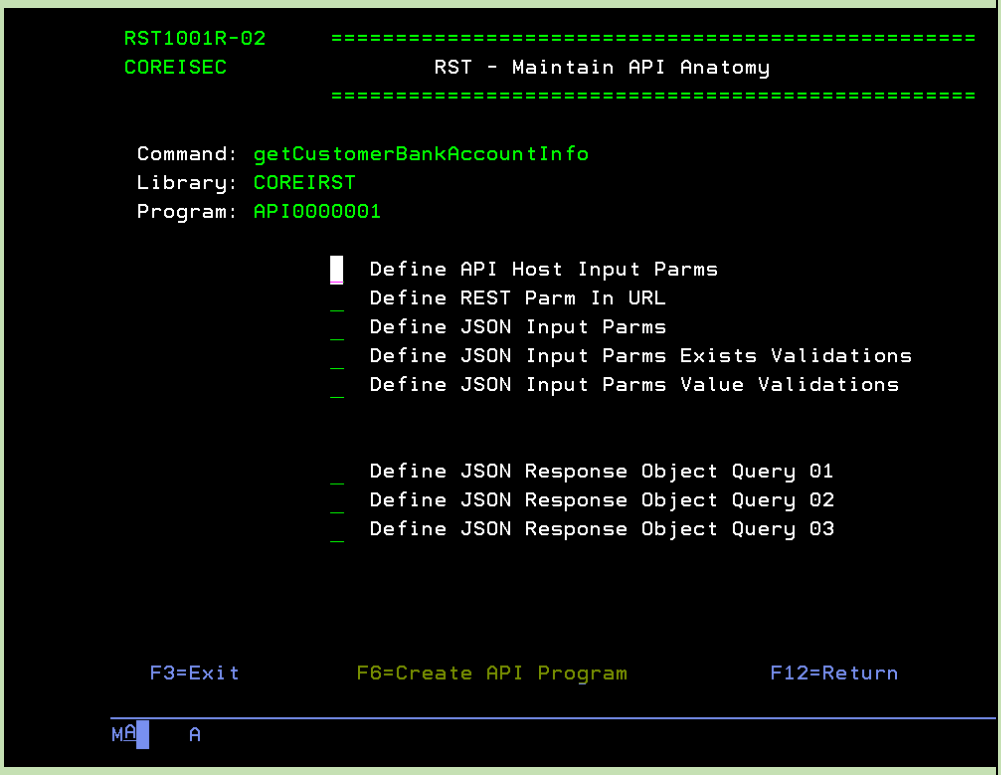
```
                          RST - Define API Anatomy
                  Define JSON Response Object Query 01
Command: getCustomerBankAccountInfo
//Define Actual SQL/JSON Statement To Return JSON Object
//Example...
exec sql
values
varchar(json_object(
                     key 'custNo'
                     value trim(:g_custNo)
                    ,key 'custInfo'
                     value (select json_object(
                                    'firstName'   :trim(cfname)
                                   ,'lastName'    :trim(clname)
                                   ,'address1'    :trim(cadd1)
                                   ,'address2'    :trim(cadd2)
                                   ,'city'        :trim(ccity)
                                   ,'state'       :trim(cstate)

                                   ,'zip'         :trim(czip)
                                   ,'routing'     :trim(arout)
                                   ,'accountNo'   :trim(aacct)
                                                  )
                                from coreirst.testcust
                                    ,coreirst.testacct
                               where ccustNo = trim(:g_custNo)
                                 and acustNo = ccustno)
                    format json )
       ,32000)
into :l_jsonOutString;
```

- This will be rpg embedded sql syntax.
- This definition, along with *Define JSON Response Object Query 02* and *Define JSON Response Object Query 03* will be the sole code snippets for producing the JSON response!

- For large responses you may use all 3 query definition areas. It is never good to build a single SQL statement so large and complex it becomes hard to maintain. Core-i RST will append each individual query output to each other producing a fine large JSON response, not to surpass 8MB.
- You will observe in this query, that we are utilizing the g_custNo host variable we coded for to locate the corresponding customer number rows in the tables we are querying. We define our selection criteria and leave the rest up to SQL to generate the JSON object.

- **F6=Create API Program**



- After all definitions are defined, they are stored as control records.
- Pressing function key F6 will now generate the code from the Core-i RST auto-API compiler utilizing your definitions.
- The source will be generated to COREIRST/COREIRST_S.APIxxxxxxx
- If there are any errors, it is advised that you re-compile from PDM and review the compile listing for errors. Revisit your definitions and retry the F6=Create API Program.

End Of Document