SEE/Change

SEE/Change

PE Notes 4.6001

(Version 4.6001)

www.thenon.com

Many of the world leading companies use Thenon's products to change manage and test their software.

Thenon – designers of SEE/Change, the leading iSeries change management product.

1 Contents

2 Primary PE Enhancements

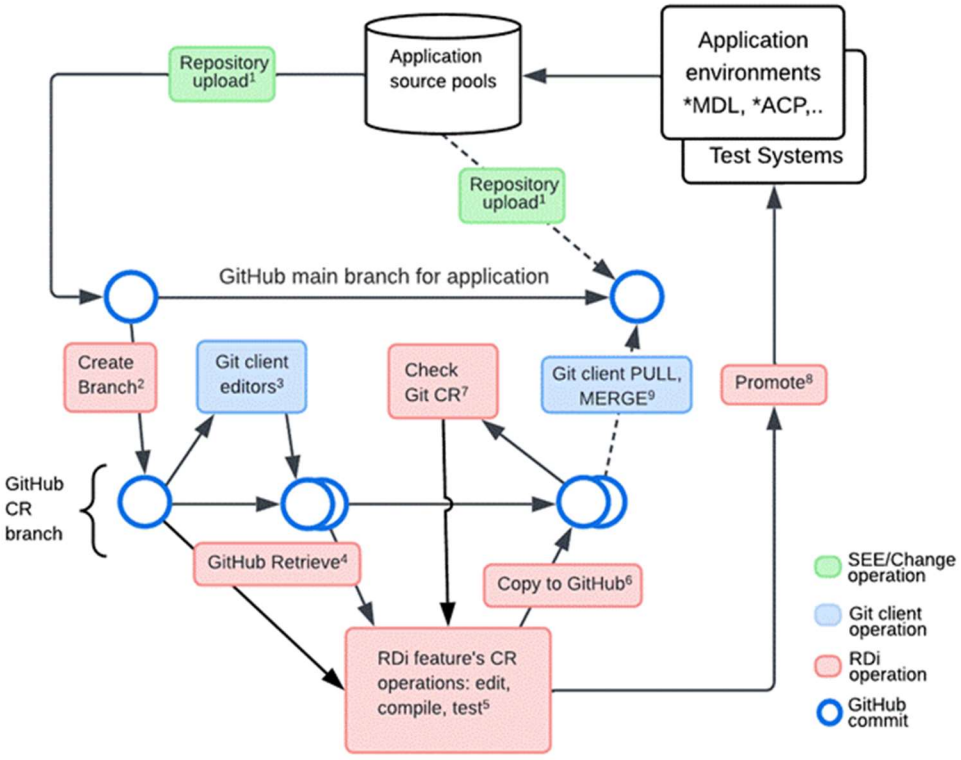
2.1 SEE/Change RDi Feature upgrade.

SEE/Change 4.5503 adds GitHub integration, allowing development in either GitHub or SEE/Change. Change Requests (CRs) while combining GitHub source control with SEE/Change change-management and deployment.

Version 4.6001 extends this to the RDI feature, enabling repository browsing, branch-change detection, and CRs that create or link to GitHub branches, including those created externally.

Source updates sync both ways: GitHub branch changes can be pulled into a CR for compile and test, and CR edits—via RDi LPEX or SEU—can be pushed back to the branch.

SEE/Change keeps the CR and its linked GitHub branch aligned, maintaining consistent source integrity across both environments.



GitHub/RDi Interface - CR workflow (for refs 1-9, please see text)

IMPORTANT NOTICE

PE 4.6001 adds GitHub support to SEE/Change through the RDi interface, requiring all RDi clients to upgrade to version 1.5 because versions 1.3 and 1.4 are no longer supported.

2.1.1 Prerequisites

- The SEE/Change RDI feature 1.5 runs on Rational Developer for i 9.8 or later, and requires an IBM-downloaded RDi installation configured to access your development System i.
- Documentation for downloading and upgrading the RDI feature is included in these PE notes, with additional RDI feature materials available in the Manuals section of the Thenon website at https://thenon.net.
- The RDI feature 1.5 requires SEE/Change version 4.6001 or above.
- The RDI feature works the same as version 1.4 when SEE/Change is not using the GitHub interface, with no additional prerequisites.
- RDI feature 1.5 using the GitHub interface requires the following to be configured in SEE/Change.
    - Each user must be configured for GitHub access using F18 in Work with User Enrolment. A classic GitHub personal access token (User → Settings → Developer settings → Personal access tokens → Tokens (classic)) must be assigned to the user, and the token is validated against GitHub.
    - Each Application that uses GitHub needs to be assigned to a GitHub repository using option 60=GitHub Link from Application Configuration. The user setting up this link must have their GitHub access token assigned and be authorised to the GitHub repository.
    - Application sources load into the GitHub repository through option 30 = Take On Jobs in Application Configuration. Running the “GITLOAD” job imports the entire local SEE/Change source pool into the repository and may take several minutes to complete.
    - Open CRs link to GitHub branches, using the CR description as the branch name. Branches can be created from SEE/Change or the RDI feature interface.

2.1.2 Upgrading RDi client

2.1.2.1 Uninstalling SEE/Change RDI 9.8 Feature 1.3 / 1.4

PC clients are using RDI 9.8 with the SEE/Change feature 1.3 / 1.4 applied, the existing RDI feature 1.3 / 1.4 will need to be removed before loading the 1.5 version.

To check your current RDI feature version

- Start RDI 9.8
- Help → About IBM Rational Developer for i → Confirm version shows 9.8.xxx

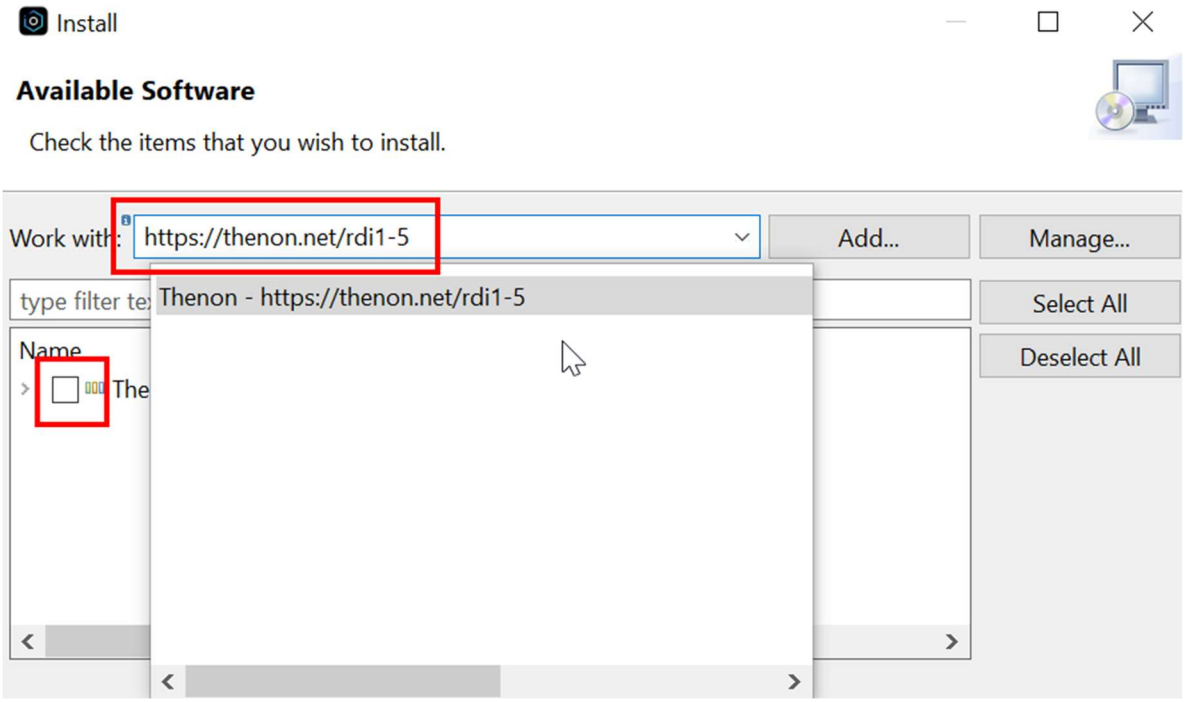
- Help → Installation Details → Scroll to find “SEE/Change CM feature for RDi” → Version shown in the Version column
- Installation Details → Select “SEE/Change CM feature for RDi” → Click Uninstall
- When prompted to restart, choose “Restart Now” to apply the updates.

This action removes versions 1.3 and 1.4 of the SEE/Change RDi feature.

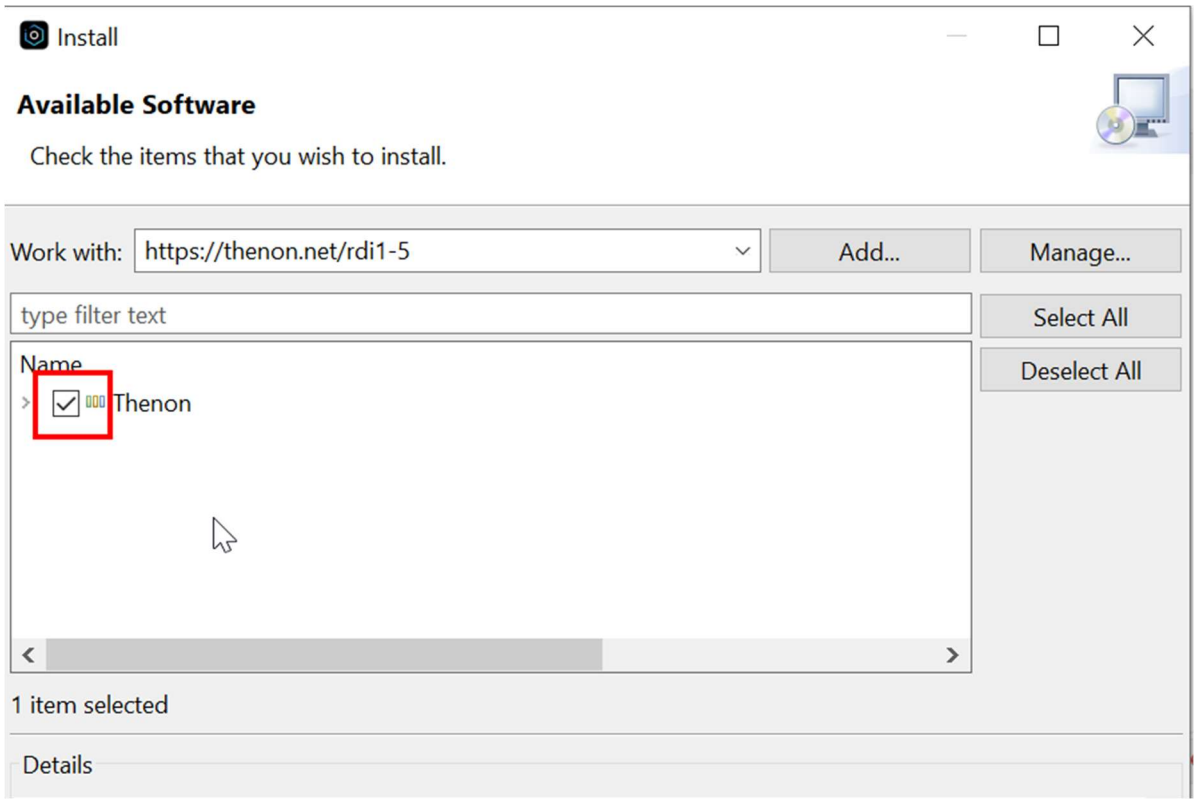
2.1.2.2 Installing SEE/Change RDI Feature 1.5

With RDi 9.8 restarted, you can now proceed to add the new software.

- Help → Install New Software
- Enter `https://thenon.net/rdi1-5` in the *Work With* field.



In the list, select “Thenon”.



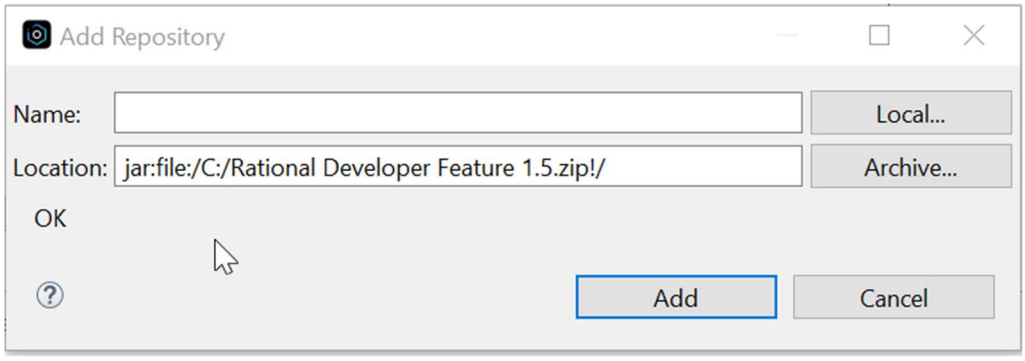
- At the bottom of the window, click Next. This will download the RDi 1.5 feature directly from the Thenon website.
- On the next window, click Finish (this takes a few seconds).

- RDi 9.8 will complete the installation of the SEE/Change 1.5 feature.
- Allow RDi to restart to complete the installation.

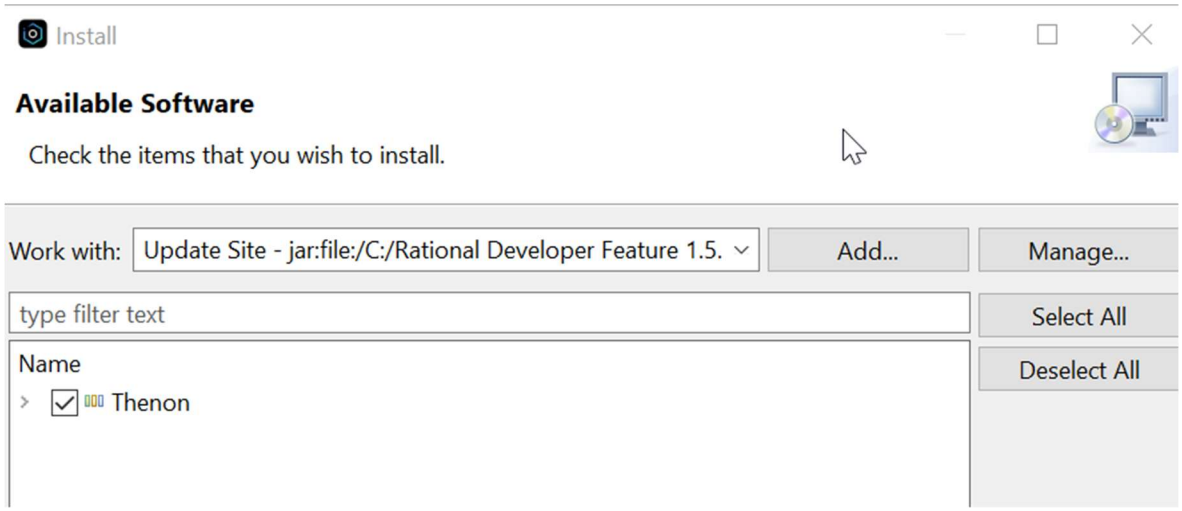
2.1.2.3 Installing RDI 9.8 SEE/Change feature 1.5 from download.

If needed, you can download SEE/Change RDi Feature 1.5 from the Thenon website (https://thenon.net). Under *Downloads*, select “Rational Developer Feature 1.5.zip”. To install from the downloaded ZIP file, follow the installation steps using the local archive instead of the online update site.

- In RDi, go to Help → Install New Software → Add → Archive.
- In the Repository Archive window, locate the downloaded ZIP file and click Open.



- Click Add, then in the Available Software window select “Thenon,” click Next, complete the install with Finish, and allow RDi to restart.



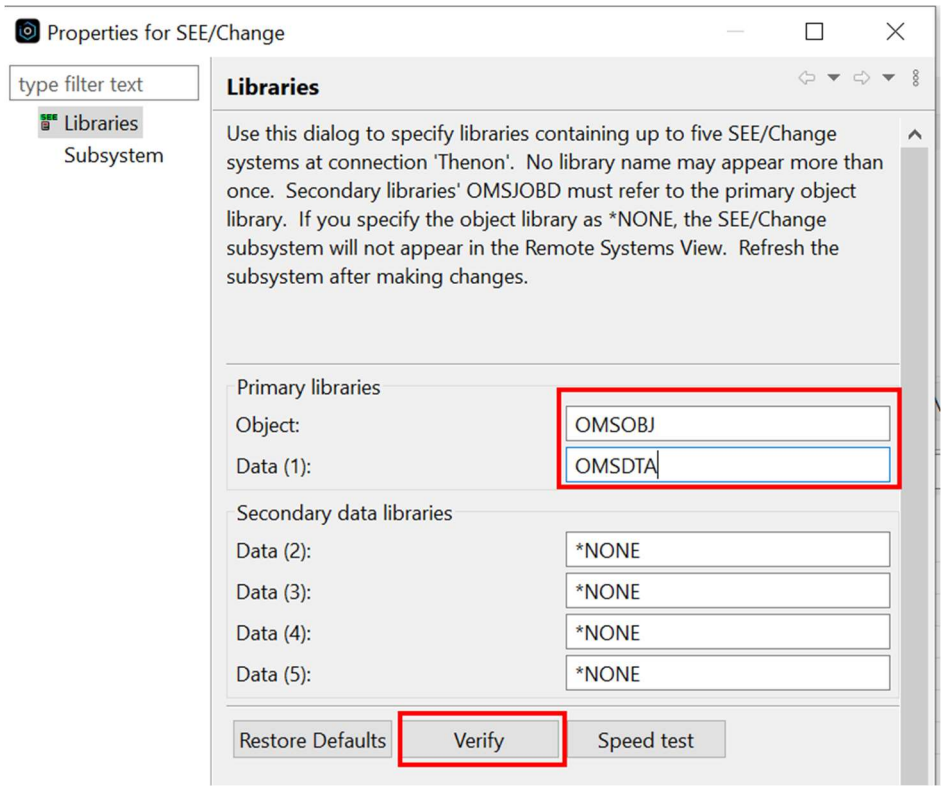
2.1.2.4 Checking RDI feature settings

- After the restart, SEE/Change should appear in the Remote Systems tree under your system.

Showing version 1.5. xxxx

If this is a new installation of the SEE/Change RDi feature, you may need to verify the SEE/Change data and object libraries using the steps below.

- Right-click SEE/Change in the Remote Systems tree, select Properties, adjust the data and object libraries if needed, then click Verify.



You will be prompted to enter your iSeries credentials to complete the verification.

- Click Apply and Close.

RDi 9.8 with the SEE/Change Feature 1.5 is now fully installed and ready to use.

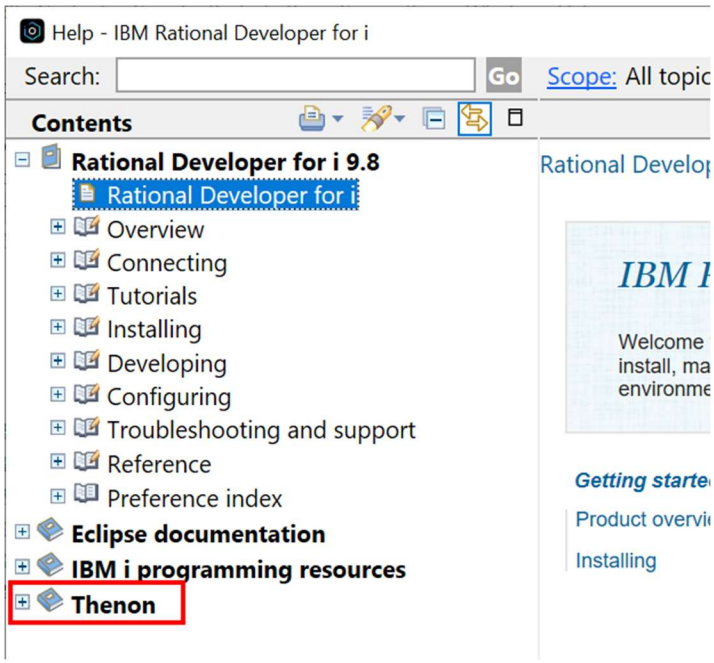
2.1.2.5 RDI feature 1.5 & GitHub.

This updated RDi Feature 1.5 now includes support for GitHub integration through SEE/Change. Be aware that the first time SEE/Change connects to GitHub, there may be a brief delay while the initial interaction completes.

2.1.3 Verification of RDI Feature install.

Once the SEE/Change RDI 1.5 feature has been installed and RDi has restarted, you should see SEE/Change in the Remote System Entry tree. To confirm that the RDi 1.5 feature is active, you can perform a few quick checks.

- Help → Help Contents. In the Help window, you should now see “Thenon” listed in the Help Contents. This section provides the usage documentation included with the RDi 1.5 feature release.



- For applications that are not GitHub-enabled (linked), the functionality of the RDi Feature 1.5 remains the same as in version 1.4.
- Applications. In the RSE tree, expand SEE/Change and select your development system. You'll see two items: *Change Requests* and *Applications*. When you expand *Applications*, you'll see the SEE/Change-configured applications. If an application is GitHub-enabled, an entry for the associated GitHub repository will appear. If it is not GitHub-enabled, this area behaves exactly as it did in version 1.4 of the RDi feature.
- With GitHub enabled for an application, expanding the Repository entry will display the GitHub branches. Expanding a branch gives you access to the sources stored in that branch. When you drill down to an individual source and right-click it, you can browse the source or retrieve it into a Change Request, provided the CR description matches the branch name.
- Within SEE/Change, for GitHub-enabled applications, the CRs include several additional functions that allow you to copy sources between the Change Request and the associated GitHub branch. You'll also find the "Check GIT CR" option, which verifies source synchronization between the branch and the CR.

2.2 OMSAPI REST API's

SEE/Change 4.6001 introduces support for RESTful API requests, enabling external applications to retrieve SEE/Change data and initiate actions programmatically. This allows any REST-capable system to access information or trigger operations within SEE/Change, such as querying activity data or performing workflow actions.

Data can be retrieved from SEE/Change using a GET request. Several predefined endpoints ("maps") are available, and you may also create custom GET mappings to extract any required SEE/Change data and produce tailored JSON output. This approach is not limited to the standard GET requests delivered with the product.

External applications can initiate SEE/Change actions using a POST request, which triggers a SEE/Change command. For example, a POST request can create a change request or advance a CR through its lifecycle. Several POST API command examples are provided, and additional POST endpoints (maps) can be configured to support other commands as required.

2.2.1 About the OMSAPI REST API

You can use OMSAPI to access SEE/Change from external software, enabling you to retrieve SEE/Change data, automate processes, and integrate with the SEE/Change application. For example, APIs can be used to retrieve activity data or perform actions within SEE/Change.

The REST API endpoints (OMSAPI maps) are highly configurable, allowing you to build comprehensive integrations tailored to your specific interface requirements. Several OMSAPI maps are included as part of the 4.6001 upgrade. The documentation below explains how these maps operate and provides a solid foundation for creating additional maps to meet your own API needs.

OMSAPI processing is handled through the Apache web server supplied by IBM. The SEE/Change OMSAPI configuration is compatible with existing web server setups and is described in the next section. This provides the following features:

- Utilises Apache I/O Control
- Authorisation is handled through Basic Authentication in the Apache web server, using System i user profiles for validation.
- Integrates with existing Apache webserver configurations.
- Optionally, you can run SEE/Change OMSAPI on a dedicated Apache web server instance if required.
- Optionally, SSL can be configured using the standard SSL facilities provided by Apache.
- Access auditing is handled through the Apache access logs (when configured).

2.2.2 Configuring OMSAPI into Apache Webserver

By default, IBM supplies an Apache web server with a standard configuration named “APACHEDFT.” However, some application packages that rely on the web server may disable this default configuration and replace it with their own application-specific setup. The SEE/Change OMSAPI configuration directives are very simple and can be added to any existing Apache instance. Configuration updates can be made either by editing the Apache configuration file directly or by using IBM Navigator for i. Both methods are described below.

2.2.2.1 Apache webserver overview

The Apache “instances” are stored in the file *QATMHINSTC* in library *QUSRSYS*. This file contains one member for each Apache instance. In most environments you will see an entry for *APACHEDFT*, although this may have been replaced by an application-specific configuration. Each instance defines which configuration file it uses and whether the instance should automatically start when the system IPLs. For example:

```
                          Display Physical File Member
File . . . . . :   QATMHINSTC          Library  . . . . :   QUSRSYS
Member . . . . . :   APACHEDFT         Record . . . . . :   1
Control  . . . . .                     Column . . . . . :   1
Find . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
-apache -d /www/apachedft -f conf/httpd.conf -AutoStartN
                           ****** END OF DATA ******









F3=Exit    F12=Cancel    F19=Left    F20=Right    F24=More keys
```

In the example above, the APACHEDFT server is configured in the IFS at */www/apachedft*, with its configuration file located in the *conf* subdirectory. Your own web server configuration may differ depending on the applications installed on your system. Also note that, in the example shown, this server instance is set *not* to start automatically at system startup, as a different Apache instance is used on that machine.

2.2.2.2 Manual Apache Configuration changes for OMSAPI

In the example above, two sections need to be added to the configuration file. These updates can be made by manually editing the *httpd.conf* file located in the */conf/* directory, or by applying the changes through IBM Navigator for i. Your Apache web server configuration may differ depending on the applications running on your System i.

The following section should be inserted into the configuration after the “General setup directives.”

OMSAPI Security directive

```
<LocationMatch "/omsapi/*">
   Require valid-user
   Passwdfile %%SYSTEM%%
   AuthType Basic
   AuthName SEE/Change
</LocationMatch>
```

Directive to run OMSAPI software.

```
ScriptAliasMatch ^/omsapi/(.*) /QSYS.LIB/OMSOBJ.LIB/O#999.PGM
```

Note that the “omsapi” path can be changed if it conflicts with existing applications. (For the purposes of this documentation, “omsapi” will be used.)

If your SEE/Change installation does not use the standard *OMSOBJ* object library, you will need to update the configuration to reference the library where your SEE/Change objects reside.

After updating the web server configuration, you must restart the Apache instance. Use the *STRTCPSVR* command to start the web server.

2.2.2.3 Update Apache configuration via IBM Navigator for i

Your ADMIN web server must be running for this step. You can verify this by using the command `WRKACTJOB SBS(QHTTPSVR)`. In the subsystem, you should see the ADMIN jobs active. If the ADMIN web server instance is not running, you can start it with the command:

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

Note this can take several minutes to start.

Using a web browser, open Navigator for i. This is typically accessed at:

```
http://<your-system-ip>:2001
```

Supply User ID and password and *Log In*.

Select the *IBM i Tasks* page, then choose *IBM Web Administration for i (HTTP)*. This opens in a new browser tab and will prompt you to enter your credentials again.

Once loaded, the IBM Web Administration for i interface will appear.

Under *Manage All Servers*, select your web server instance (or *APACHEDFT* if that is the one you are using).

From the left-hand menu, choose *URL Mapping*. Ensure the *Server* area at the top is set to *Global configuration*.

Under the *Aliases* tab, click *Add*. You will be adding a Script Alias Match as follows.

The Script Alias Match URL path is `^/omsapi/(.*)` and the Host directory or file is `/QSYS.LIB/OMSOBJ.LIB/O#999.PGM`.

Click the *Apply* button.

This completes the configuration of the “processing” component for OMSAPI, directing requests to the O#999 program in the SEE/Change *OMSOBJ* library.

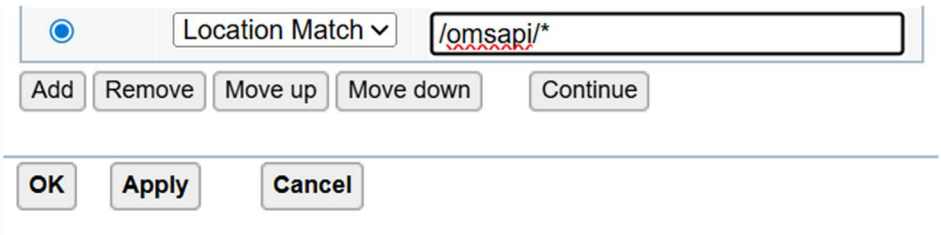
Next, we need to configure Basic Authentication for OMSAPI.

From the left-hand menu, under *Server Properties*, select *Container Management*.

Open the *Locations* tab and click *Add*. (You may need to scroll within the inner window to find the *Add* button.)

In the Add dialog, choose *Location Match* from the *Type* drop-down list, and enter `/omsapi/*` in the *URL path or expressions* field.

Click the *Apply* button.



That has created the “container.”

Next, we need to add security. (Security is applied at the container level, which is why the container had to be created first.)

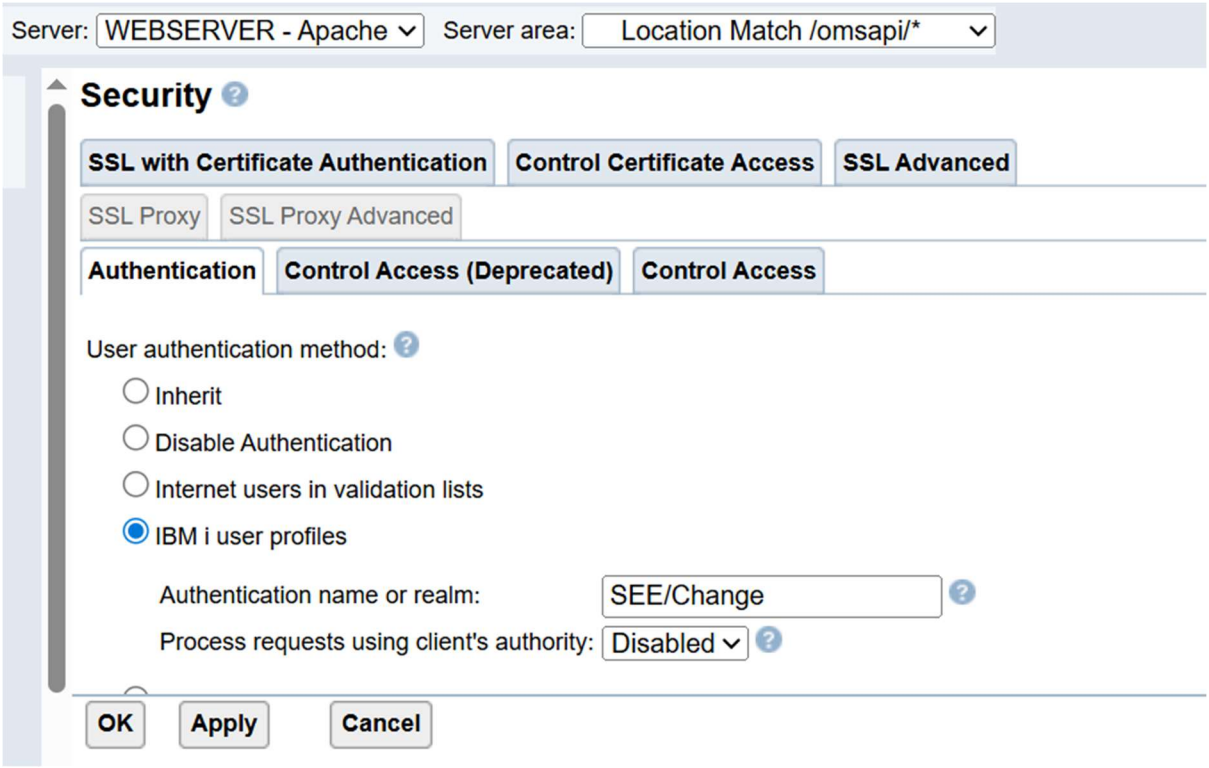
From the left-hand menu, select *Security*.

In the *Server Area* drop-down at the top, choose *Location Match* and enter `/omsapi/*`.

From the Security tabs, select *Authentication* and complete the fields as follows:

- User authentication method: *IBM i user profiles*
- Authentication name or realm: *SEE/Change*

Click *Apply*.



Note that “Process request using client’s authority” can be either enabled or disabled.

- **When this option is disabled:**
  The OMSAPI program *O#999* temporarily switches the job’s current user profile to the user supplied through Basic Authentication. It performs the API processing under that user profile, then restores the original job user when the request completes.
  This approach reduces job overhead and improves security by avoiding the need to spawn additional Apache jobs.
- **When this option is enabled:**
  Separate Apache jobs are created for each authenticated user. This can improve performance in some environments but will consume more system resources.

Checking the configuration in IBM Web administration for i.

Toward the bottom of the left-hand menu, under *Tools*, you can select *Display Configuration File*.

Review the configuration to confirm that it contains a section similar to the following.

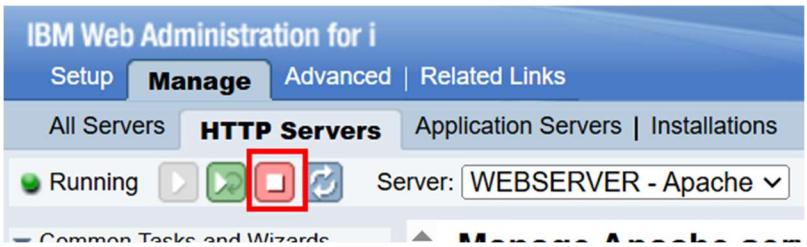
```
40 <LocationMatch "/omsapi/*">
41    Require valid-user
42    Passwdfile %%SYSTEM%%
43    AuthType Basic
44   AuthName SEE/Change
45 </LocationMatch>
```

And you should also see a one-line entry that looks like this.

```
56 ScriptAliasMatch ^/omsapi/(.*) /QSYS.LIB/OMSOBJ.LIB/O#999.PGM
```

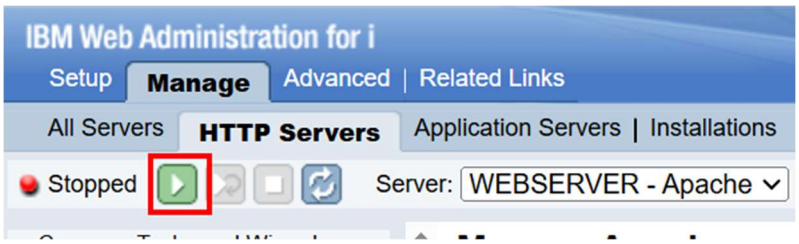
Once the configuration changes have been made, the web server must be stopped and then started again.

In the screenshot below, notice that *HTTP Server* is selected and the server name shown is *WEBSERVER – Apache*. Make sure this matches the web server instance you have been configuring before clicking the Stop icon.



Once the web server has stopped, use the Refresh button (next to the Stop icon) to confirm its status.

When it shows as fully stopped, click the Start icon to bring the server back online.



If the web server fails to start after a few seconds, check for diagnostic messages in *QSYSOPR* or the *QTMHHTTP* job logs. You can also review the Apache web server logs for details that may indicate the configuration error.

2.2.2.4 Testing the Apache Configuration for OMSAPI

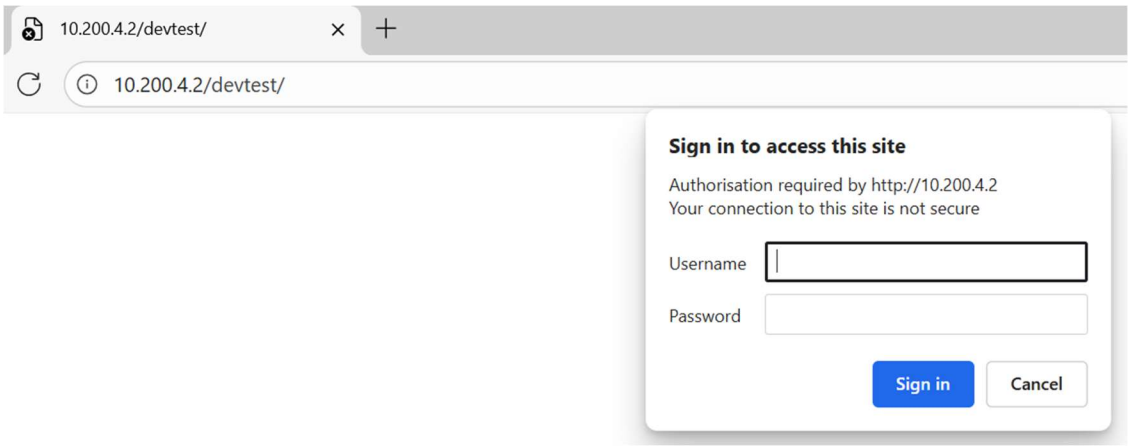
Once the configuration has been updated and the web server restarted, you should test the setup to confirm that the OMSAPI program is being called correctly.

To do this, open a web browser and enter the following URL:

```
http://<your-system-ip>/omsapi/
```

(Note that the trailing slash is required.)

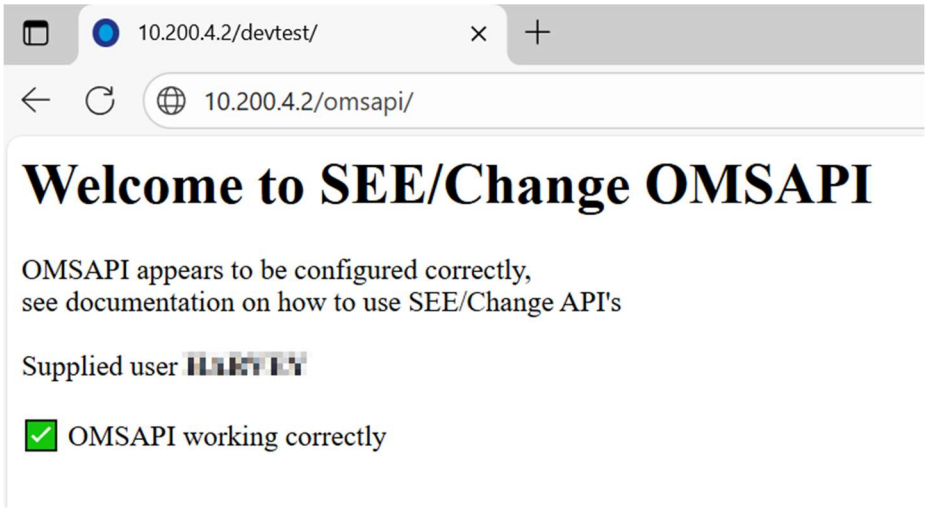
If the configuration is correct, you should see a screen like the example shown below, depending on your browser.



You will be prompted to enter your IBM i (iSeries) username and password, then click *Sign In*.

This is the Basic Authentication step, which OMSAPI uses to identify the user making the API request.

With valid credentials, you should see a screen like the example shown below.



2.2.2.5 Basic Authentication in API calls

The HTTP header for OMSAPI calls must include the Basic `Authorization` directive.

Most external tools or integration platforms that support API calls allow you to specify this in their scripting or configuration settings.

The resulting HTTP header typically looks like this:

```
<header name="Authorization" value="Basic uuuu:pppp" />
```

Where `uuuu:pppp` represents the IBM i username and password separated by a colon, the entire string is then Base64-encoded.

So the process is:

1. Combine the username and password into a single string:

   `uuuu:pppp`

2. Encode that string using Base64.
3. Place the encoded value after the word **Basic** in the Authorization header.

This is what allows OMSAPI to identify and authenticate the caller using Basic Authentication.

2.2.3 OMSAPI GET requests.

A GET API request is used to return data from SEE/Change in JSON format.

The data returned depends on the GET “map” name, which determines how OMSAPI processes the request.

In its simplest form, a GET request can be invoked directly from a web browser, although in most real-world scenarios it will be issued by an external application.

SEE/Change includes a number of GET “maps” supplied as part of release 4.6001, and we will cover these in more detail shortly.

For example, a Jira form might be scripted to make an API call to retrieve the current number of Change Requests in development using the map count.

Eg

`http://<your-system-ip>/omsapi/count?stat=01`

Count is the map name (end point) to be used.

`?stat=01` is a parameter to pass into the map.

In this instance, we are asking for the map *count* where the parameter *stat* is set to “*01”*.

The returned JSON data provides the number of Change Requests (CRs) currently in development status.

Different map names return different sets of data, depending on the purpose of the map and how it has been defined within SEE/Change.

2.2.3.1 Maintaining GET OMSAPI maps.

Unlike many API interfaces, SEE/Change is not restricted to a predefined set of fixed GET requests with rigid, unchangeable structures.

Instead, SEE/Change allows you to configure your own GET maps.

This design gives you complete flexibility:

- You decide **what data** you want to extract
- You decide **how that data is structured**
- You decide **which parameters** the map accepts
- You can tailor each map to the exact needs of your integrating application

This means you can retrieve any information from SEE/Change in whatever format or structure best fits your external system—whether that's Jira, ServiceNow, custom dashboards, automation scripts, or other enterprise tools.

We have supplied several GET maps as working examples. These can be copied and modified, or you may create entirely new GET maps as needed.

To maintain or create GET maps, you (or a group you belong to) must have authority to this function through the Work with User Enrolment option WRKAPICFG.

API configuration maintenance is located in:

8 – Configuration Manager

60 – More Configuration Manager Options

19 – Work with API Configuration

This screen lists all existing GET data maps.

An example screen is shown below.

```
                      SEE/Change Development Environment.
                          Maintain API GET Data Maps
 2=Change  3=Copy  4=Delete  5=Display

Opt  Map          Description
 _   COUNT        Count of CR at a status
 _   DSPIRCR      Display IR/CR Details
 _   LSTCROBJ     List CR objects
 _   LSTIRCR      List IR/CR combination
 _   LSTIRCRJ     List IR/CR combination Filter on text
 _   LSTPARMTYP   List parameters types
 _   LSTPARMVAL   List parameter values
 _   LSTRLS       List releases

                                                                    Bottom

 F1=Help  F3=Exit  F5=Refresh  F6=Create  F9=Cmd  F11=POST  F12=Previous
```

A GET API map consists of four key elements: a map name, a title, an SQL statement to retrieve the data, and the JSON template used to format the returned output.

Using the example **DSPIRCR** shown above, this map retrieves details about an IR/CR combination.

By selecting **2 = Change** or **5 = Display**, you can view the components that make up this map—its SQL, its JSON formatting rules, and any parameters it accepts.

```
                      SEE/Change Development Environment.
                              Change GET Data Map
 Map. .: DSPIRCR    Display IR/CR Details
 Sql. .: Select irirno, IRORGD , CRSEQN, IR#USR,   irtext, crappl, crcrtp, crrln
o, crstat, crtext, PRPARM FROM xir join  xcr on irirno=crirno
join xpr on PRPRMC='CRST' and PRPRMV = ('        ' || crstat)
WHERE irirno = &IRNO  and crseqn = &SEQN









 Default Sort. . .:


 F1=Help  F3=Exit  F9=Cmd  F10=Parms  F11=JSON  F12=Previous  F18=Run SQL
 F16=DSP Substitute fields
```

In the above SQL SELECT statement, we return fields from three files joined together (XIR, XCR, and XPR).

Notice in the **WHERE** clause that two variables are prefixed with an ampersand (`&`).

These are **substitution variables**, and their values are supplied directly from the query string in the URL.

For example:

`http://<your-system-ip>/omsapi/dspircr?irno=000134&seqn=01`

In this request:

- **dspircr** is the GET map being called
- **irno=000134** provides the value for `&irno`
- **seqn=01** provides the value for `&seqn`

OMSAPI substitutes these values into the SQL before executing it, allowing the map to return the specific IR/CR details requested.

2.2.3.2 Substitute variables.

Variables that can be used as substitution variables can be viewed by pressing F16 = Dsp Substitute fields.

This option displays all the fields that the GET map is allowed to reference as &variables in the SQL statement. It's a useful way to confirm which parameters your map can accept and how they should be named before you build or modify the SQL and JSON sections.

```
                      SEE/Change Development Environment.
                                Change GET Data Map
 Map. .: DSPIRCR    Display IR/CR Details
 Sql. .: Select irirno, IRORGD , CRSEQN, IR#USR,   irtext, crappl, crcrtp, crrln
o, crstat, crtext, PRPARM FROM xir join  xcr on irirno=crirno
join xpr on PRPRMC='CRST' and PRPRMV = ('        ' || crstat)
WHERE irirno = &IRNO  and crseqn = &SEQN




.............................................................................
:                      O#PRM API Parameter fields                           :
: & Field    Type     Size   Description                                    :
: APIUSER    CHAR     10     User for stored parms                          :
: PAGENO     NUMERIC  10.0   Page number                                    :
: REL        CHAR     5      Relative to page prev,next,last,first          :
: PERPAGE    NUMERIC  10.0   Entries per page                               :
: SORT       CHAR     200    Sorting Field and order                        :
: SYSM       CHAR     3      Dev Centre System                              :
: APPL       CHAR     3      Appl Code                                      :
:                                                                  More... :
:............................................................................:
 F16=DSP Substitute fields
```

There are quite a few substitution fields available, and these are defined in file *O#PRM*.

This file contains the fields as parameter names that a GET map is allowed to reference using the `&variable` syntax in the SQL statement. When you press **F16 = Dsp Substitute fields**, the system displays the fields in O#PRM, showing all valid substitution variables that can be used within your GET map.

2.2.3.3 Setting Substitute variables for testing.

You will notice on the previous screen that option *F18 = Run SQL* is available. This allows you to test your SQL statement directly from within the GET map definition.

However, because the SQL contains substitution variables (the `&variables` that normally come from the URL), the system needs values for those variables before the SQL can be executed.

To supply these values during testing, use F10 = *Parms*.

F10 lets you enter test values for each substitution variable, exactly the same values that would normally be passed in the query string of the API call. Once the parameters are set, you can press F18 to run the SQL and verify that your map returns the expected data.

```
                      SEE/Change Development Environment.
                      API Testing Parameters maintenance
 Parameters for testing by user THISUSER
 Parameter values
 SYSM. : DEV         APPL. : AP1               IRNO. : 000134
 SEQN. : 01          TEXT. :
 OREF. :                                       SITE. :
 CATG. : CRST        DATE. :       0           CRTY. :
 PRTY. : A           RLNO. :                   STAT. : 01
 USER. :             JOBN. :                   JOB#. :
 RQSN. :             RQSS. :                   OBJN. :
 OLIB. :             SRCF. :                   SLIB. :
 MBRN. :             OBJT. :                   OBJA. :
 DATE2 :       0     DATE3 :       0
 NUM1. :           .00000                      NUM2. :           .00000
 CHAR10:
 CHAR50:
 TEXT80A . . .:

 TEXT80B . . .:
```

There are two pages of potential substitution variables, and all of them are available for use when building or testing a GET map.

These variables are defined in file *O#PRM*, which give every parameter that a map is allowed to reference using the `&variable` syntax.

For testing purposes, the values you enter are **specific to your user profile**. This means:

- The test values you set are **only for your own use**
- They are **remembered across screens**, so you don't need to re-enter them each time
- They allow you to run and validate SQL using F18 without needing an actual API call

On the example screen, you can see that some variables already have values assigned.

In this case:

- **IRNO** has been set to `000134`
- **SEQN** has been set to `01`

These correspond exactly to what would be passed in the URL:

`http://<your-system-ip>/omsapi/dspircr?irno=000134&seqn=01`

2.2.3.4 Testing the SQL Select

Using F18 = *Run SQL* from the GET Data Map screen executes the SQL SELECT statement with all substitution variables replaced by the test values you entered.
This is an essential step because it confirms two things:

- Your SQL syntax is valid
- The SQL returns the data you expect before you expose it through an API map

If the SQL is invalid, **F18 will fail** and an error message will appear on the message line at the bottom of the screen.

To diagnose the issue:

1. Place the cursor on the error message
2. Press **F1** for more information
3. Then press **F10** to open the job log

The job log will show the exact SQL error, making it much easier to correct the statement.

For example, if the SQL in the map is deliberately altered so that it becomes invalid, pressing F18 will fail immediately and display the error message. From there, F1 → F10 will reveal the detailed SQL diagnostic information.

.

`SQL failed with SQL code -206. See Job log for more details`

Using F1 on the message and F10 we can see the SQL error.

```
2>> /*      */
    /*      */
    OMSTEMP in QTEMP type *FILE not found.
    Column or global variable IRIRNOX not found.
    SQL failed with SQL code -206. See Job log for more details
```

This helps you correct the SQL. Once the SQL has been fixed, pressing F18 again will run the statement successfully and display the data returned using the SELECT statement together with the substitution values you set via F10 = Parms.

This gives you a clear, reliable way to validate your GET map:

- You confirm the SQL syntax is correct
- You verify that the substitution variables are being applied properly
- You see the exact dataset that OMSAPI will return when the map is called through a browser or external application

With the corrected SQL in place, F18 shows the live results using the parameter values you entered, making it easy to confirm that the map behaves exactly as intended before exposing it as an API endpoint.

```
                                    Display Report
                                                                  Report width . . . . . :     210
Position to line  . . . . .                                    Shift to column  . . . . . .
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10....+...11....+...12....
       IR       Origintd   Seq  Entered by  Text                                  Appl  CR    Release  CR      Text
       Nbr      Date       Nbr  User                                              Code  Type  Nbr      Status
                CYYMMDD
000001 000134  1,250,721   01   USERID      REF-1234 Change to order entry status AP1   *MOD   00380     02    REF-12
****** ********  End of report  ********
                                                                                                                Bottom
```

In the next section, we look at how the SQL-returned data is formatted into JSON.

Up to this point, the focus has been on building and validating the SQL portion of the GET map—ensuring the query runs correctly, the substitution variables work as expected, and the returned dataset is accurate.

The next step is where OMSAPI becomes truly flexible:
the JSON formatting section takes the SQL result set and transforms it into the JSON structure that will be returned to the calling application.

This is where you define:

- Which SQL fields appear in the JSON
- How they are named
- How the JSON is structured (single object, array, nested objects, etc.)

- Any literal text or formatting you want included

This JSON template is what external systems such as Jira, ServiceNow, dashboards, or automation scripts will consume when they call the GET map.

Next we walk through the JSON formatting screen and break down how each part works.

2.2.3.5 JSON formatting

From the GET Data Map screen, pressing F11 toggles between the SQL SELECT statement and the JSON formatting statement.

The JSON section always begins with:

select json_object

This is because **json_object is an SQL scalar function** used to construct JSON output directly from the SQL result set.
IBM's SQL reference manual provides full details on how json_object works, including supported syntax, nesting, arrays, and formatting rules.

In the GET map:

- The **SQL page** defines *what data* is retrieved
- The **JSON page** defines *how that data is presented* to the API caller

This separation is what makes OMSAPI maps flexible—you can change the JSON structure without altering the SQL, or vice-versa.

We'll walk through a real JSON formatting example next and break down how each part maps to the SQL fields.

IBM Manual contains more information on "json_object".

https://www.ibm.com/docs/en/db2/11.5.x?topic=functions-json-object

Using the DSPIRCR map, pressing F11 switches to the JSON view, where you can see exactly how the SQL-returned data is transformed into the JSON output.

This JSON section is the second half of the GET map definition, and it controls the structure of the API response. When you toggle to this view:

- You will see the statement beginning with
- select json_object
- Each JSON key is mapped to a field returned by your SQL SELECT
- Literal text, nested objects, and arrays can all be defined here
- The final JSON produced by the API call is built entirely from this template

This makes the JSON page the "presentation layer" of the GET map—separate from the SQL logic but tightly linked to the fields it returns.

Below is the DSPIRCR JSON and explain how each element maps to the SQL result set.

```
                      SEE/Change Development Environment.
                                 Change GET Data Map
 Map. .: DSPIRCR    Display IR/CR Details
 JSON .: select json_object(
'IRNO' value cast(irirno as varchar(6) ccsid 37),
'IRText' value cast(trim(irtext) as VARCHAR(50) CCSID 37),
'IR_Date' value IRORGD+19000000,
'Created_By' value cast(trim(ir#usr) as varchar(10) ccsid 37),
'SEQN' value cast(crseqn as varchar(2) ccsid 37),
'APPL' value cast(trim(crappl) as varchar(3) ccsid 37),
'TYPE' value cast(trim(crcrtp) as varchar(4) ccsid 37),
'CRSTS' value cast(trim(crstat) as varchar(2) ccsid 37),
'CRSTSD' value cast(substr(prparm,1,4) as varchar(4) ccsid 37),
'CRText' value cast(trim(crtext) as varchar(50) ccsid 37),
'RLNO' value cast(trim(crrlno) as varchar(5) ccsid 37)
)




 F1=Help  F3=Exit  F9=Cmd  F10=Parms  F11=SQL  F12=Previous  F18=Run SQL/JSON
 F16=DSP Substitute fields
```

When building a JSON object from SEE/Change data:

1. **Character fields require CCSID conversion**
   Most fields are stored in EBCDIC, so cast them to UTF-8 to ensure correct JSON output.
2. **Fixed-length CHAR fields must be converted to VARCHAR**
   SEE/Change pads CHAR fields with spaces; converting to VARCHAR (and trimming) prevents trailing blanks in JSON.
3. **Packed date fields (7,0) use CYYMMDD format**
   SEE/Change dates store the century as C (0 = 1900s, 1 = 2000s). Adding 19000000 converts them to a standard CCYYMMDD integer that's easier for API consumers.

2.2.3.6 Testing the JSON formatting

- F18 = Run SQL/JSON
- On the JSON screen, F18 runs both parts of the GET map:
  - The SQL statement with substituted variables
  - The JSON_OBJECT definition
- Internally, F18 builds a single SQL statement:

  SELECT JSON_OBJECT(…)

  FROM (SELECT …substituted SQL…)
- F18 then executes this combined SQL and writes the resulting JSON to a temporary IFS file so you can view it.
- Error Handling with F18
- If the SQL fails, you see:by example

  SQL prepare failed with SQL code -104
- To diagnose the issue:

- Put the cursor on the message
- Press **F1**, then **F10**
- This shows the detailed SQL error text

- Example: Detecting a Typo

  By example example, the JSON contained a deliberate typo: VARXHAR instead of VARCHAR.

  F18 reported:

  Column or global variable IRNO not found

  Derived operands not valid for operator JSON_OBJECT

  Character conversion between CCSID 65535 and CCSID 1208 not valid

  Token VARXHAR was not valid. Valid tokens: … VARCHAR …

  This makes the problem obvious: **VARXHAR** is invalid, so the JSON formatting fails.

- **After Fixing the Typo**

  Once corrected, pressing **F18** again:

- Rebuilds the SQL
- Executes it
- Displays the generated JSON based on:
  - The SQL with substituted variables
  - The JSON_OBJECT formatting

```
 Browse : /tmp/omstmp574270.json
 Record :       1   of       2 by  18                    Column :    1    132 by 131
 Control :

....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+....0....+....1....+....2....+....3.
 ************Beginning of data**************
{"IRNO":"000134","IRText":"REF-1234 Change to order entry status","IR_Date":20250721,"Created_By":" USER01","SEQN":"01","APPL":"AP1",
"TYPE":"*MOD","CRSTS":"02","CRSTSD":"*TST","CRText":"REF-1234 Change to order entry status","RLNO":"00380"}
 ************End of Data********************
```

The 5250 screen shows the API's JSON exactly as returned as one unformatted, continuous string. The terminal doesn't pretty-print or modify it; it simply displays the raw payload verbatim.

If the DSPIRCR API is invoked from a web browser using the same IRNO and SEQN parameters, some browsers may offer a built-in "Pretty-print" option that formats the JSON for easier reading.


```
{
  "IRNO": "000134",
  "IRText": "REF-1234 Change to order entry status",
  "IR_Date": 20250721,
  "Created_By": [redacted],
  "SEQN": "01",
  "APPL": "AP1",
  "TYPE": "*MOD",
  "CRSTS": "02",
  "CRSTSD": "*TST",
  "CRText": "REF-1234 Change to order entry status",
  "RLNO": "00380"
}
```


Where a list of data is required, the SQL will return one or more records. The JSON output uses **JSON_ARRAYAGG** to format these as a JSON array. IBM provides full details in its documentation:

https://www.ibm.com/docs/en/ias?topic=functions-json-arrayagg

As an example, consider the API GET map **LSTIRCR**.
Its SELECT statement is shown below.

```
                      SEE/Change Development Environment.
                                 Change GET Data Map
 Map. .: LSTIRCR    List IR/CR combination
 Sql. .: SELECT irirno, IRORGD , CRSEQN, IR#USR,   irtext, crappl, crcrtp, crrln
o, crstat, crtext, PRPARM FROM xir join  xcr on irirno=crirno
join xpr on PRPRMC='CRST' and PRPRMV = ('        ' || crstat)
WHERE irsysm = &SYSM









 Default Sort. . .: irirno desc


 F1=Help  F3=Exit  F9=Cmd  F10=Parms  F11=JSON  F12=Previous  F18=Run SQL
 F16=DSP Substitute fields
```

There is one substitution variable, **&SYSM**, which we set to **"DEV"** using **F10 = Parms**.
Running the SQL with **F18** returns the list of IR/CR records shown in the screenshot.
They appear in **descending order**, which matches the default sort we specified on **IRIRNO DESC**.

```
                                        Display Report
                                                                                Report width . . . . . :     210
Position to line  . . . . .                                                Shift to column  . . . . . .
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10....+...11....+...12....
       IR       Origintd   Seq  Entered by  Text                                  Appl  CR    Release  CR      Text
       Nbr      Date       Nbr  User                                             Code  Type  Nbr      Status
                CYYMMDD
000001 000135  1,251,111   01   USER1       Client demo 11/11/25 10:30            AP1   *MOD            01    Client
000002 000134  1,250,721   01   USER1       REF-1234 Change to order entry status AP1   *MOD   00380    02    REF-12
000003 000134  1,250,721   02   USER1       REF-1234 Change to order entry status AP1   *MOD   00374    04    Test S
000004 000134  1,250,721   03   USER1       REF-1234 Change to order entry status V43   *MOD   00380    07    Nigel
000005 000134  1,250,721   04   USER1       REF-1234 Change to order entry status V43   *MOD            01    new te
000006 000133  1,250,709   01   USER1       RV RDi testing 2 (GH Disabled)        AP1   *MOD            01    RV RDi
000007 000133  1,250,709   02   USER1       RV RDi testing 2 (GH Disabled)        AP1   *MOD            04    RV RDi
000008 000132  1,250,708   01   USER1       RV RDi testing 1                      AP1   *MOD   00373    09    RV RDi
000009 000132  1,250,708   02   USER1       RV RDi testing 1                      AP1   *MOD            01    RV RDi
000010 000131  1,250,526   02   User2       RS RDi test 1                         ACM   *NEW            01    RS RDi
000011 000131  1,250,526   03   USER2       RS RDi test 1                         ACM   *EMG            01    RS RDi
000012 000131  1,250,526   04   USER2       RS RDi test 1                         ACM   *NEW            01    RS RDi
000013 000131  1,250,526   05   USER2       RS RDi test 1                         ACM   *NEW            01    RS RDi
000014 000131  1,250,526   06   USER2       RS RDi test 1                         ACM   *EMG            01    RS RDi
000015 000129  1,250,428   01   USER1       Test new obj type                     AP1   *NEW   00371    09    Test n
000016 000129  1,250,428   02   USER1       Test new obj type                     AP1   *NEW   00370    09    Test n
000017 000128  1,250,423   01   USER1       Test SQLPROC and SQLFUNC              AP1   *MOD   00368    09    RV Tes
                                                                                                            More...
F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split      F22=Width 80
```

Under the F11=JSON we need to specify that we are constructing a JSON array of entries to represent the list of records the SQL selects.

```
                        SEE/Change Development Environment.
                                Change GET Data Map
 Map. .: LSTIRCR    List IR/CR combination
 JSON .: select json_object('CR_details' value(JSON_ARRAYAGG(JSON_OBJECT(
'IRNO' value cast(irirno as varchar(6) ccsid 37),
'IRTEXT' value cast(trim(irtext) as VARCHAR(50) CCSID 37),
'IRDATE' value IRORGD+19000000,
'CREATED' value cast(trim(ir#usr) as varchar(10) ccsid 37),
'SEQN' value cast(crseqn as varchar(2) ccsid 37),
'APPL' value cast(trim(crappl) as varchar(3) ccsid 37),
'CRTYPE' value cast(trim(crcrtp) as varchar(4) ccsid 37),
'CRSTS' value cast(trim(crstat) as varchar(2) ccsid 37),
'CRSTSD' value cast(substr(prparm,1,4) as varchar(4) ccsid 37),
'CRTEXT' value cast(trim(crtext) as varchar(50) ccsid 37),
'RLSN' value cast(trim(crrlno) as varchar(5) ccsid 37)
   ))))




 F1=Help  F3=Exit  F9=Cmd  F10=Parms  F11=SQL  F12=Previous  F18=Run SQL/JSON
 F16=DSP Substitute fields
```

In this example, JSON_ARRAYAGG builds a CR_details JSON object that contains an array of JSON objects for the extracted rows. When F10=Parms has SYSM set to DEV, running F18=Run SQL/JSON returns the JSON array.

```
 Browse : /tmp/omstmp574270.json
 Record :       1   of      49 by  18                    Column :    1    132 by 131
 Control :

....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+....0....+....1....+....2....+....3.
 ************Beginning of data**************
{"CR_details":[{"IRNO":"000135","IRTEXT":"--------------- 11/11/25 10:30","IRDATE":20251111,"CREATED":"USER01","SEQN":"01","APPL":"A
P1","CRTYPE":"*MOD","CRSTS":"01","CRSTSD":"*DEV","CRTEXT":"--------------- 11/11/25 10:30","RLSN":""},{"IRNO":"000134","IRTEXT":"REF
-1234 Change to order entry status","IRDATE":20250721,"CREATED":"USER01","SEQN":"01","APPL":"AP1","CRTYPE":"*MOD","CRSTS":"02","CRST
SD":"*TST","CRTEXT":"REF-1234 Change to order entry status","RLSN":"00380"},{"IRNO":"000134","IRTEXT":"REF-1234 Change to order entr
y status","IRDATE":20250721,"CREATED":"USER01","SEQN":"02","APPL":"AP1","CRTYPE":"*MOD","CRSTS":"04","CRSTSD":"*MDL","CRTEXT":"Test
SQLDML movement to redevelopment","RLSN":"00374"},{"IRNO":"000134","IRTEXT":"REF-1234 Change to order entry status","IRDATE":2025072
1,"CREATED":"USER01","SEQN":"03","APPL":"V43","CRTYPE":"*MOD","CRSTS":"07","CRSTSD":"*RDY","CRTEXT":"Nigel Demo diff app","RLSN":"00
380"},{"IRNO":"000134","IRTEXT":"REF-1234 Change to order entry status","IRDATE":20250721,"CREATED":" USER01","SEQN":"04","APPL":"V43
","CRTYPE":"*MOD","CRSTS":"01","CRSTSD":"*DEV","CRTEXT":"new text via api CRTCR","RLSN":""},{"IRNO":"000133","IRTEXT":"RV RDi testin
g 2 (GH Disabled)","IRDATE":20250709,"CREATED":" USER01","SEQN":"01","APPL":"AP1","CRTYPE":"*MOD","CRSTS":"01","CRSTSD":"*DEV","CRTEX
T":"RV RDi testing 4 demo","RLSN":""},{"IRNO":"000133","IRTEXT":"RV RDi testing 2 (GH Disabled)","IRDATE":20250709,"CREATED":" USER01
","SEQN":"02","APPL":"AP1","CRTYPE":"*MOD","CRSTS":"04","CRSTSD":"*MDL","CRTEXT":"RV RDi testing 4 demo 2a","RLSN":""},{"IRNO":"0001
32","IRTEXT":"RV RDi testing 1","IRDATE":20250708,"CREATED":" USER01","SEQN":"01","APPL":"AP1","CRTYPE":"*MOD","CRSTS":"09","CRSTSD":
"*LIV","CRTEXT":"RV RDi testing 1","RLSN":"00373"},{"IRNO":"000132","IRTEXT":"RV RDi testing 1","IRDATE":20250708,"CREATED":" USER01"
,"SEQN":"02","APPL":"AP1","CRTYPE":"*MOD","CRSTS":"01","CRSTSD":"*DEV","CRTEXT":"RV RDi testing 2 spare","RLSN":""},{"IRNO":"000131"
,"IRTEXT":"RS RDi test 1","IRDATE":20250526,"CREATED":"RICHARD","SEQN":"02","APPL":"ACM","CRTYPE":"*NEW","CRSTS":"01","CRSTSD":"*DEV
","CRTEXT":"RS RDi test 1","RLSN":""},{"IRNO":"000131","IRTEXT":"RS RDi test 1","IRDATE":20250526,"CREATED":"RICHARD","SEQN":"03","A
PPL":"ACM","CRTYPE":"*EMG","CRSTS":"01","CRSTSD":"*DEV","CRTEXT":"RS RDi test 1 test CR creation","RLSN":""},{"IRNO":"000131","IRTEX

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

As noted earlier, the JSON output isn't formatted; this is the raw JSON returned to the API requester.

If the API call is made from a browser, you can usually view the response in a pretty-printed JSON format.

Ps in the above ---------- is used to obscure client and user data.

IMPORTANT: - When the SQL selection returns no records, JSON data is still created with array contents of null. By example

```
{"CR_details":null}
```

Because data is being returned, the http status would be 200 (OK), not 404 (Not Found).

Note the resultant JSON is longer than shown in the above.

For more information on JSON aggregated arrays (JSON_ARRAYAGG) see IBM manual. https://www.ibm.com/docs/en/ias?topic=functions-json-arrayagg

2.2.3.7 JSON formatting a simple overview.

A single JSON object (one returned record) is simply a list of field–value pairs, separated by commas, inside a json_object structure.

```
Select json_object(
'json_field1' value SQLfield1,
'json_field2' value cast(SQLfield2 as varchar(10) ccsid 37),
'json_field3' value 'constant value'
)
```

An array/list is a JSON object like the one above, but wrapped inside a JSON array. The array name becomes the JSON field, and its value is a JSON_ARRAYAGG containing repeated json_object entries—one for each retrieved row.

```
Select json_object(
'array_name' value(JSON_ARRAYAGG(JSON_OBJECT(
'json_field1' value SQLfield1,
'json_field2' value cast(SQLfield2 as varchar(10) ccsid 37),
'json_field3' value 'constant value'
)
)
```

In the above, the "json_object" contains an "array_name" with a value of an embedded "json_object" (in this case with json_field1, json_field2 and json_field3).

.

2.2.3.8 List / Array sort.

By default, SQL returns data in an unpredictable order, which can create problems for downstream systems and for pagination. In the SQL screen, you'll see a **Default Sort** option. Unless the URL specifies a sort order, this setting controls the sequence of returned results.

For example, in **LSTIRCR**, we can sort by **IRIRNO** (descending) and **CRSEQN** (ascending), matching the order shown in **WRKCRDEV** (Work with Parts using SCDM). To do this, set the **Default Sort** in the SQL definition accordingly.

```
irirno desc, crseqn asc
```

Although this is the default sort sequence for **LSTIRCR**, it can be overridden in the URL using the **sort** parameter. If a different sort order is required, simply include it in the request URL, for example:

```
http://<your-system-ip>/omsapi/lstircr?sysm=DEV&sort=irirno desc,crseqn desc
```

This would return the list in IRIRNO descending and CRSEQN descending.

We could request the list in ascending order.

```
http://<your-system-ip>/omsapi/lstircr?sysm=DEV&sort=irirno,crseqn
```

Notice the ascending is not specified. The default is ascending. So in the above URL would behave the same as.

```
http://<your-system-ip>/omsapi/lstircr?sysm=DEV&sort=irirno asc,crseqn asc
```

2.2.3.9 List entries per page and starting page.

By default, OMSAPI returns 30 records starting at page 1. You can control the number of records and the starting page using the URL parameters **perpage** and **pageno**. If these parameters are omitted, **perpage = 30** and **pageno = 1** are used automatically. For example, the following URL will return only two records.

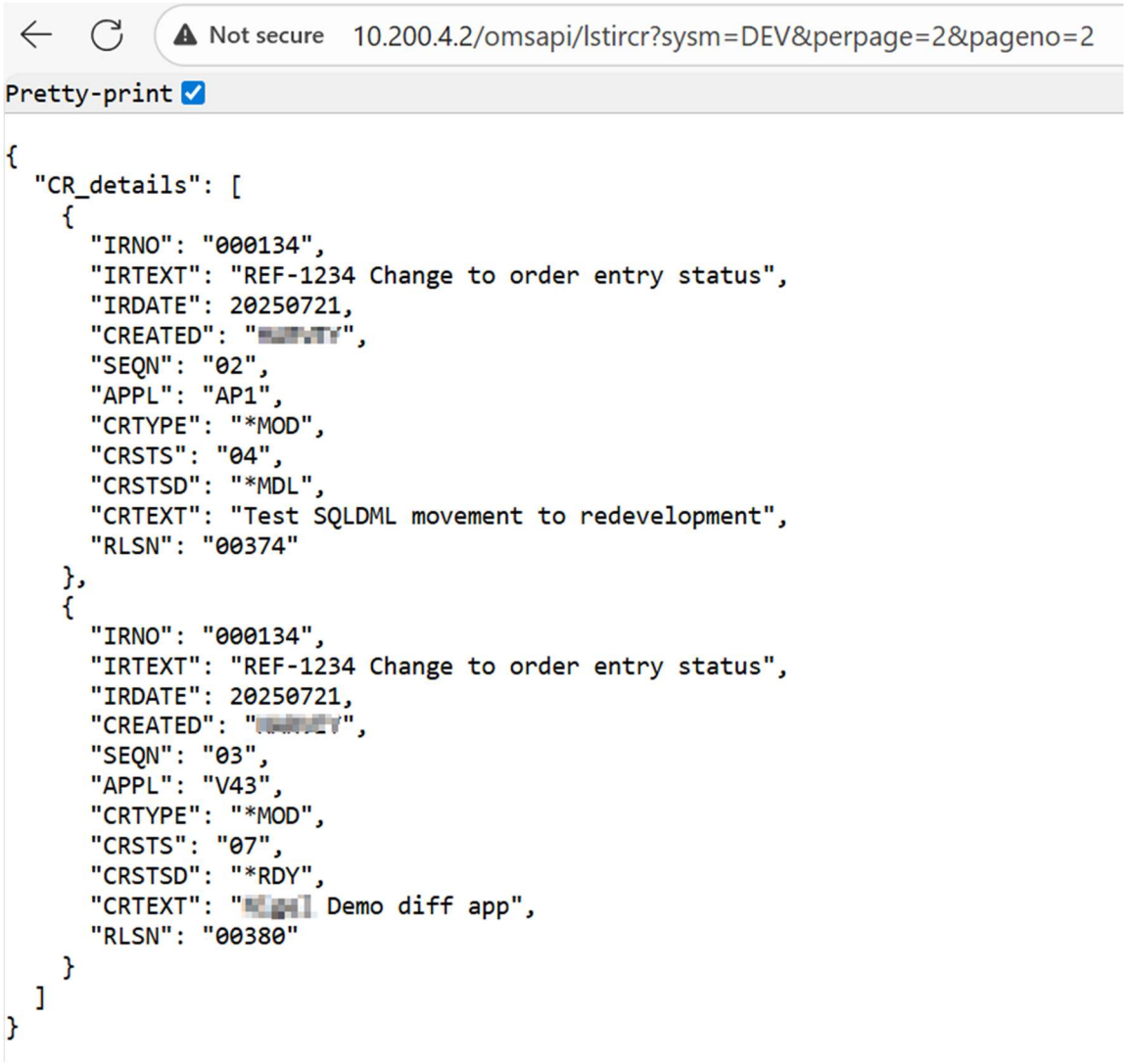
`http://<your-system-ip>/omsapi/lstircr?sysm=DEV&perpage=2`

Because no “sort” has been specified, the list will be sorted by the default sort specified on the LSTIRCR map.


```
{
  "CR_details": [
    {
      "IRNO": "000135",
      "IRTEXT": "[illegible] 11/11/25 10:30",
      "IRDATE": 20251111,
      "CREATED": "[illegible]",
      "SEQN": "01",
      "APPL": "AP1",
      "CRTYPE": "*MOD",
      "CRSTS": "01",
      "CRSTSD": "*DEV",
      "CRTEXT": "[illegible] 11/11/25 10:30",
      "RLSN": ""
    },
    {
      "IRNO": "000134",
      "IRTEXT": "REF-1234 Change to order entry status",
      "IRDATE": 20250721,
      "CREATED": "[illegible]",
      "SEQN": "01",
      "APPL": "AP1",
      "CRTYPE": "*MOD",
      "CRSTS": "02",
      "CRSTSD": "*TST",
      "CRTEXT": "REF-1234 Change to order entry status",
      "RLSN": "00380"
    }
  ]
}
```


To retrieve the next two entries, set **pageno** to the next page number and **perpage** to 2 in the URL.

`http://<your-system-ip>/omsapi/lstircr?sysm=DEV&perpage=2&pageno=2`




```
{
  "CR_details": [
    {
      "IRNO": "000134",
      "IRTEXT": "REF-1234 Change to order entry status",
      "IRDATE": 20250721,
      "CREATED": "[redacted]",
      "SEQN": "02",
      "APPL": "AP1",
      "CRTYPE": "*MOD",
      "CRSTS": "04",
      "CRSTSD": "*MDL",
      "CRTEXT": "Test SQLDML movement to redevelopment",
      "RLSN": "00374"
    },
    {
      "IRNO": "000134",
      "IRTEXT": "REF-1234 Change to order entry status",
      "IRDATE": 20250721,
      "CREATED": "[redacted]",
      "SEQN": "03",
      "APPL": "V43",
      "CRTYPE": "*MOD",
      "CRSTS": "07",
      "CRSTSD": "*RDY",
      "CRTEXT": "[redacted] Demo diff app",
      "RLSN": "00380"
    }
  ]
}
```


Limiting the number of JSON entries per page and choosing a starting page improves response times and reduces the amount of data each API call must process, resulting in a smoother user experience.

2.2.3.10 Searching with a GET API request.

The **GET MAP** uses SQL to retrieve data, typically with a **WHERE** clause to select specific records or filter results. This makes it useful for searching and filtering. Because it relies on SQL, the **LIKE** operator can be used—for example, by copying *LSTIRCR* and modifying its WHERE clause. When this map runs, the caller can specify characters that must appear in the CR's text.

Since **LIKE** is used, the parameter (e.g., &TEXT) must include SQL wildcards (_ and %). This allows the API caller to use single-character or multi-character wildcards, as well as standard SQL pattern matching. The **MATCHES** operator can also be used.

```
                    SEE/Change Development Environment.
                              Change GET Data Map
 Map. .: LSTIRCRJ   List IR/CR combination Filter on text
 Sql. .: SELECT irirno, IRORGD , CRSEQN, IR#USR,   irtext, crappl, crcrtp, crrln
o, crstat, crtext, PRPARM FROM xir join  xcr on irirno=crirno
join xpr on PRPRMC='CRST' and PRPRMV = ('        ' || crstat)
WHERE upper(crtext) like upper('%&TEXT%')









 Default Sort. . .: irirno desc, crseqn asc


 F1=Help  F3=Exit  F9=Cmd  F10=Parms  F11=JSON  F12=Previous  F18=Run SQL
 F16=DSP Substitute fields
```

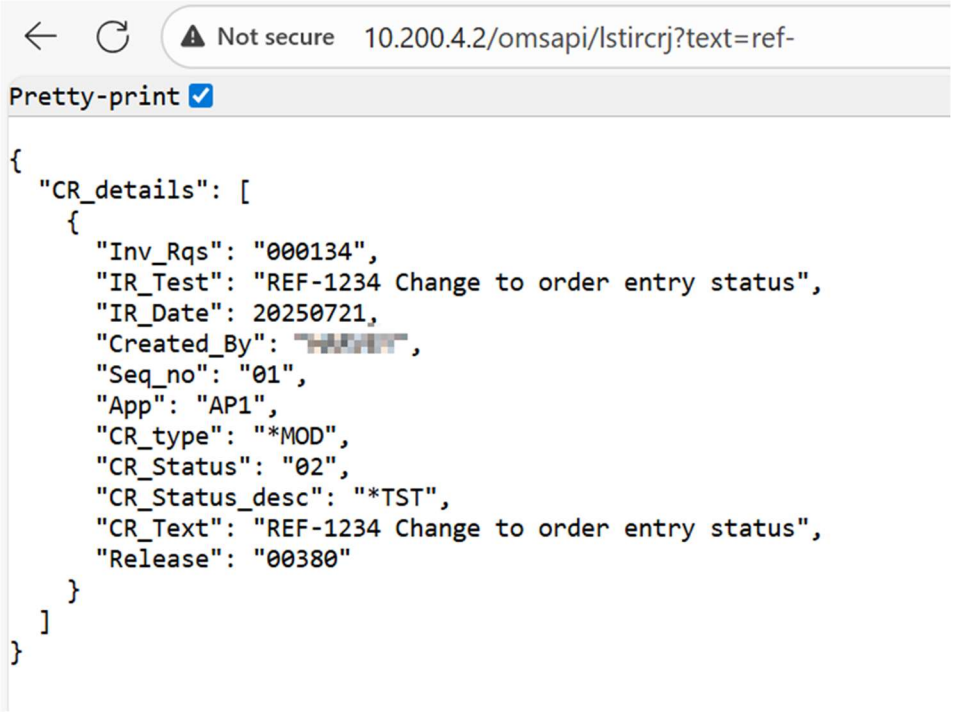
If we want to find CRs where the text begins with **REF-**, we use the SQL **LIKE** operator. This can be tested in maintenance by pressing **F10=Parms**, setting **TEXT** to REF-, and running the SQL with **F18**.

Because **LIKE** is used, the substitution variable (e.g., &TEXT) must include the required single quotes and the % wildcard. Unlike standard comparison operators (=, <>, *NE, etc.), character-field quotes are not optional here.

The URL to invoke this would look like the following (note the different map name):

`http://<your-system-ip>/omsapi/lstircrj?text=ref-`

In this example we only have one CR with text starting REF-




```
{
  "CR_details": [
    {
      "Inv_Rqs": "000134",
      "IR_Test": "REF-1234 Change to order entry status",
      "IR_Date": 20250721,
      "Created_By": "[redacted]",
      "Seq_no": "01",
      "App": "AP1",
      "CR_type": "*MOD",
      "CR_Status": "02",
      "CR_Status_desc": "*TST",
      "CR_Text": "REF-1234 Change to order entry status",
      "Release": "00380"
    }
  ]
}
```


You'll notice the JSON response uses different field names. This is because the **LSTCRIRJ** map defines a different JSON structure than **LSTIRCR**, so the returned fields do not match.

2.2.3.11 Parameter fields

As described above, OMSAPI supports several parameter fields for both GET and POST requests. These fields are defined in file **O#PRM** and can be viewed in map maintenance using **F16 = DSP Substitute fields**. Below is the current layout of the available substitute fields. Any combination of them may be used in GET or POST calls.

In the map, substitute fields begin with an ampersand followed by the field name.

Field	Type	Length	Scale	Description
APIUSER	NUMERIC	10	-	User for stored parms
PAGENO	CHAR	10	0	Page number
REL	NUMERIC	5	-	** For future use
PERPAGE	CHAR	10	0	Entries per page
SORT	CHAR	200	-	Sorting Field and order
SYSM	CHAR	3	-	Dev Centre System
APPL	CHAR	3	-	Appl Code
IRNO	CHAR	6	-	IR Nbr
SEQN	CHAR	2	-	Seq Nbr
TEXT	CHAR	50	-	Text
OREF	CHAR	30	-	User Reference
SITE	CHAR	3	-	Site/ Location
CATG	DECIMAL	10	-	Category
DATE	CHAR	7	0	Date CYYMMDD
PRTY	CHAR	10	-	Priority
CRTP	CHAR	4	-	Type
RLNO	CHAR	5	-	Release No.
STAT	CHAR	4	-	Status
USER	CHAR	10	-	Assigned User Profile
JOBN	CHAR	10	-	Job Name
JOB#	CHAR	6	-	Job Number
RQSN	CHAR	5	-	Request Name
RQSS	CHAR	5	-	Request sequence
OBJN	CHAR	10	-	Object Name
OLIB	CHAR	10	-	Object Library
SRCF	CHAR	10	-	Source File
SLIB	CHAR	10	-	Source Library
MBRN	CHAR	10	-	Member name
OBJT	CHAR	10	-	Object Type
OBJA	DECIMAL	10	-	Object Attribute
DATE2	DECIMAL	7	0	Date2 CYYMMDD
DATE3	NUMERIC	7	0	Date3 CYYMMDD
NUM1	NUMERIC	15	5	Generic Number1
NUM2	CHAR	15	5	Generic Number2
CHAR10	CHAR	10	-	Character field 10a
CHAR50	CHAR	50	-	Character field 50a
TEXT80A	CHAR	80	-	Text field 80a A
TEXT80B	CHAR	80	-	Text field 80a B

2.2.3.12 SEE/Change files.

A list of SEE/Change files and data areas can be found in support document SEE_Change_Files_and_Data_Areas-4_6001 on the Thenon web site Manuals and PE Notes page. https://thenon.net/manuals.html

2.2.4 OMSAPI POST requests

POST API requests in SEE/Change execute actions using predefined commands. Responses use standard HTTP status codes, and both success and failure return JSON to support diagnostics.

In release 4.6001, POST requests do **not** process POST data; any body content is ignored. All required variables are passed through the POST URL, and the POST *map name* determines how OMSAPI handles the request. In its simplest form, a POST triggers the command associated with that map.

SEE/Change 4.6001 includes several built-in POST maps, which are described later. A typical use case is a Jira form invoking a SEE/Change action. by example, running **CHKCR** on a specific CR.

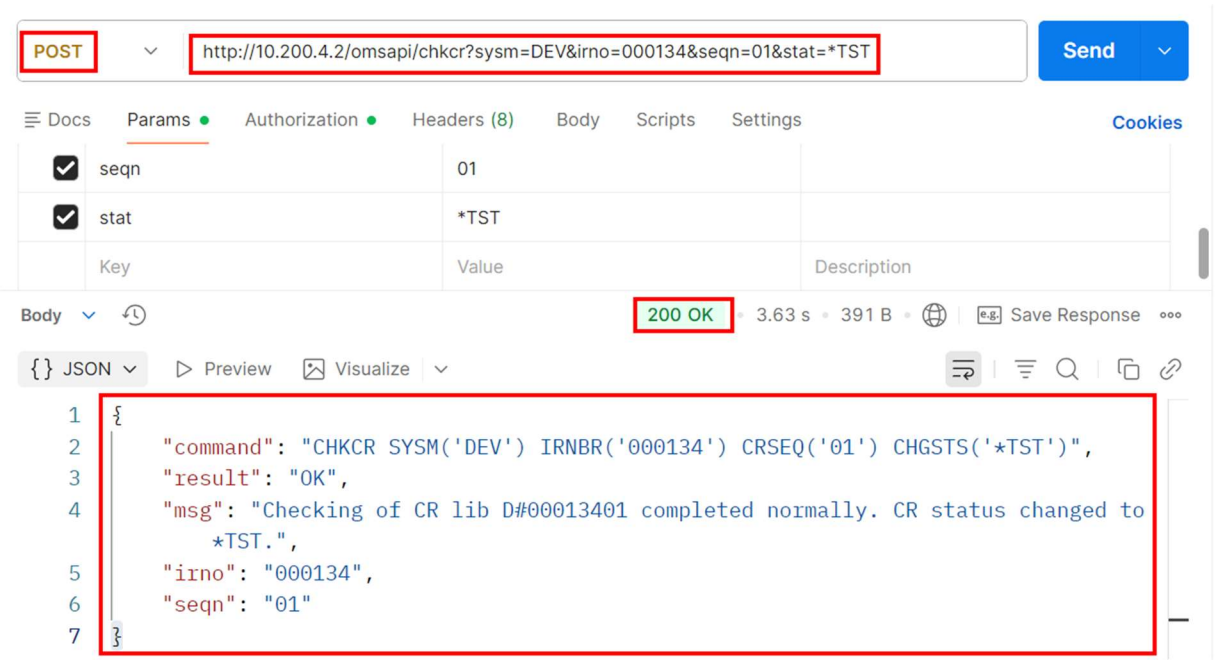
All POST examples in this document use Postman to issue the request, including Basic Auth credentials.

Eg (POST examples)

```
http://<your-system-ip>/
omsapi/chkcr?sysm=DEV&irno=000134&seqn=01&stat=*TST
```

/omsapi/ indicates that an OMSAPI call should be executed. chkcr is the POST map (endpoint). The ?.... portion contains the URL parameters passed into the map.

In this example, the POST map **chkcr** runs the **CHKCR** command. Postman is used to issue the request.



The returned JSON shows the command OMSAPI executed for the **chkcr** map, including substituted variables. The command completed with status **OK**, reflected in the result field. msg contains any message returned by the command. irno and seqn are internal SEE/Change fields set during **CHKCR** processing. If the command cannot run, see the “Post Map command failures” section.

2.2.4.1 Maintaining POST OMSAPI maps

Unlike fixed-function APIs, SEE/Change lets you define your own POST maps, allowing any SEE/Change command to be run. Several sample POST maps are provided and can be copied, modified, or replaced with new ones.

Access to this feature requires the appropriate authority for **WRKAPICFG** in Work with User Enrolment.

API configuration is located under: **8** Configuration Manager **60** More Configuration Manager Options **19** Work with API Configuration

The default view shows GET maps; press **F11** to switch to POST maps.

```
                      SEE/Change Development Environment.
                        Maintain API POST Actions Maps
 2=Change  3=Copy  4=Delete  5=Display

Opt  Map          Command
     CHKCR        CHKCR SYSM(&SYSM) IRNBR(&IRNO) CRSEQ(&SEQN) CHGSTS(&STAT)
     CRTCR        CRTCR IRNO(&IRNO) APPL(&APPL) TEXT(&TEXT) CRTP(&CRTP) PRTY(&...
     CRTIRCR      CRTIRCR TEXT(&TEXT) APPA(&APPL) LOCN(&SYSM) CATG(*SOFT) PRTY...
     MOVCROBJ     MOVCROBJ SYSM(&SYSM) IRNBR(&IRNO) CRSEQ(&SEQN) MOVTYP(&STAT)

                                                                        Bottom
 F1=Help  F3=Exit  F5=Refresh  F6=Create  F9=Cmd  F11=GET   F12=Previous
```

In this example, four POST maps are configured. The POST map name is independent of the command it runs. However in this case, the **CHKCR** map runs the **CHKCR** command with substituted parameters.

A POST map simply defines a map name and the command it executes. Using **2=Change** or **5=Display**, you can view the details of any map.

```
                      SEE/Change Development Environment.
                           Change POST Action Map
 Map. .: CHKCR
 Cmd. .: CHKCR SYSM(&SYSM) IRNBR(&IRNO) CRSEQ(&SEQN) CHGSTS(&STAT)

 F1=Help  F3=Exit  F4=Prompt  F9=Cmd  F12=Previous  F16=DSP Substitute fields
```

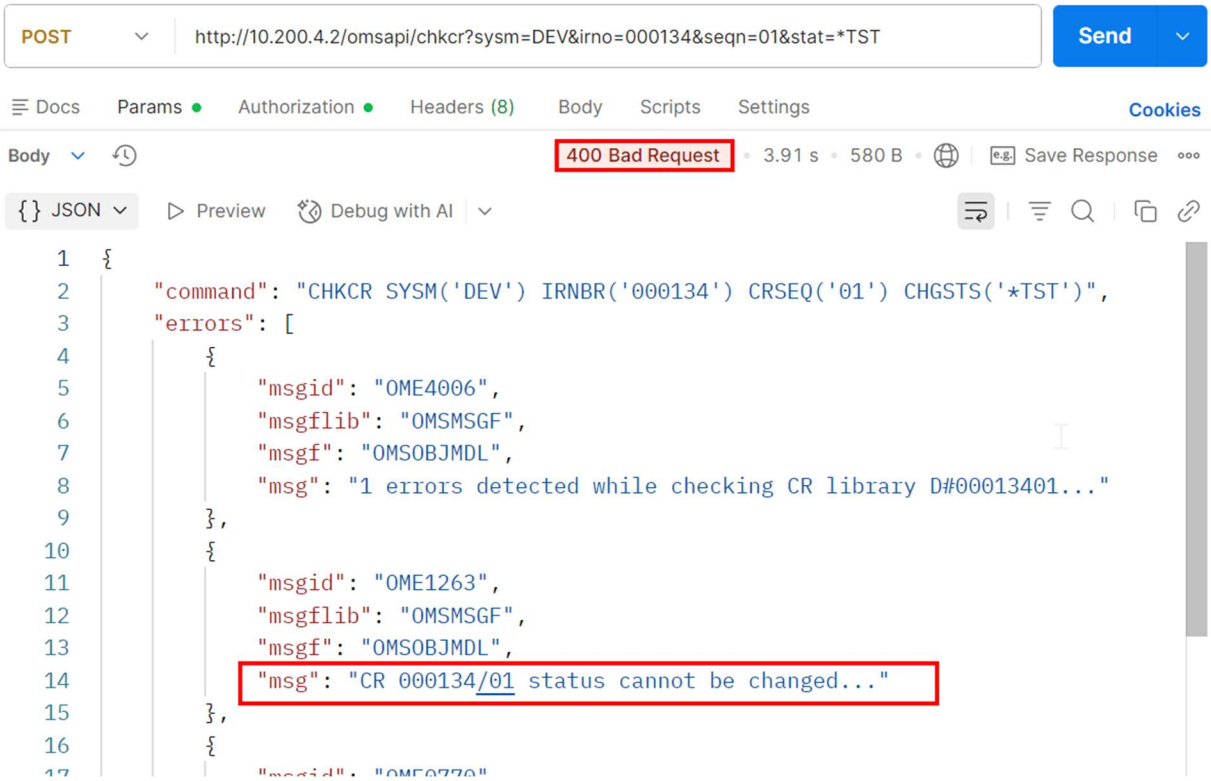
F4 prompts the command, and pressing Enter validates it and updates the map. The prompt supports substitution variables from **O#PRM**, supplied through the URL.

Prompting and validation ensure the command itself is valid, but not necessarily valid with the provided substitution values. You can use **F9** to open a command line and test the command manually.

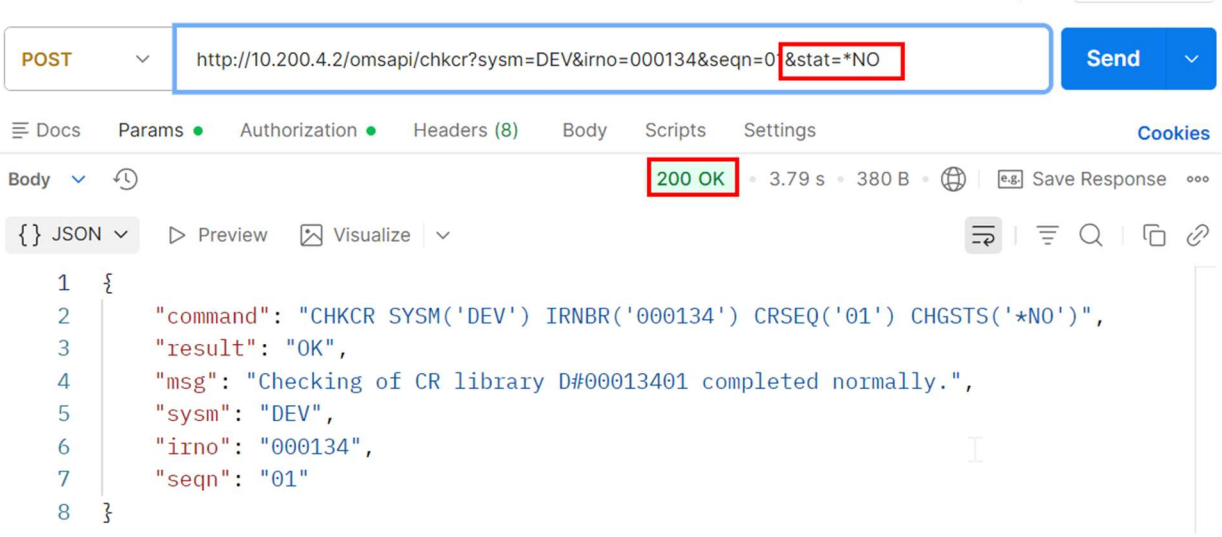
2.2.4.2 POST map Command failures.

In the earlier example, Postman sent a POST request to **CHKCR**, which returned **HTTP 200 OK** and a JSON result of **OK**, moving CR **000134/01** to *TST status.

A second POST fails because the CHKCR command requires moving the CR *to* testing, which is impossible once it's already there. The POST then returns **HTTP 400 Bad Request** with an errors array explaining the failure.



If the request is run with *NO for the status change. The CHKCR runs correctly even when the CR is in *TST status.



The response to an OMSAPI POST map depends on how well-behaved the underlying command is.

2.2.4.3 POST map Command performance.

Many SEE/Change commands run in batch. Although the earlier examples show CHKCR running directly in the API request, it could just as easily submit itself to batch using **SBMJOB**. In that case, the API response reflects the **SBMJOB** submission—not the CHKCR command running in the batch job.

When CHKCR is submitted to batch, your application may need to issue a **GET** on map **DSPIRCR** afterward to check the CR's status once the submitted job has had time to complete.

2.2.5 OMSAPI with multiple SEE/Change databases

OMSAPI normally uses the library list from *OMSOBJ/OMSJOBD* job description, so it can access only one SEE/Change database library (typically *OMSDTA*). To work with alternate SEE/Change database libraries, specify the desired library directly in the OMSAPI URL.

2.2.5.1 Normal use

`http://<your-system-ip>/omsapi/map?querysting`

Uses the library list defined in OMSJOBD job description within the SEE/Change object library..

2.2.5.2 Using an alternate SEE/Change database

Include the alternate database library in the URL so OMSAPI uses the OMSJOBD from that library to set the library list.

`http://<your-system-ip>/omsapi/`**`altdblib`**`/map?querysting`

3 Software Performance Reports

The following table lists software performance reports that have been resolved in this PE.

SPR Log Number	Description
5950	WRKRLS in version 4.6000 an issue prevented select of two or more CR's for different applications. Using F7=Previous application / F8=Next application would deselect the selection on the other screens preventing a release from being created for more than a single application.
5951	WRKAPPTOJ GITLOAD authority check issue when user is part of an authorised group.
5953	VERSION incorrectly reporting Authorisation Code status.

4 Installation

4.1 Warnings

Use the SEE/Change upgrade guide to ensure a smooth upgrade; it's available from the thenon.com Download Area or via support. Be sure to preserve any custom code, especially when reinstalling product libraries and running a conversion.

4.2 Special Instructions

Before upgrading from version 4.6000, end work notification processing using **ENDWRKNTFY** if notifications are in use (or if unsure). This step is required for a successful upgrade to 4.6001. After the upgrade, restart notifications with **STRWRKNTFY**.

4.3 Dependencies

- Must be applied after 4.6000.
- SEE/Change 4.6001 will run on IBM i 7.4 or higher.
- Must be applied to all remote sites? **See compatibility chart.**

5 RDi Plug-in Compatibility Chart

SEE/Change Rational Developer Feature Compatibility

SEE/Change Server Version	IDE Version	
	RDP 8.x.x.x	RDP 9.x.x.x
4.5401	1.4.2	1.4.2
4.5402	1.4.2	1.4.2
4.5403	N/A	1.4.3
4.5500	N/A	1.4.3
4.5501	N/A	1.4.3
4.5502	N/A	1.4.3
4.5503	N/A	1.4.3
4.6000	N/A	1.4.3 **
4.6001	N/A	**1.5.0**

** Not compatible with GitHub enabled applications

6 SEE/Change Compatibility Chart

SEE/Change Release Compatibility Chart

Production \ Development	4.5200	4.5201	4.5300	4.5400	4.5401	4.5402	4.5403	4.5500	4.5501	4.5502	4.5503	4.6000	4.6001
4.5200	Yes	1	1,2	1,2	1,2	1,2,3	1,2,3	1,2,3	1,2,3	1,2,3	1,2,3	1,2,3	1,2,3
4.5201	No	Yes	2	2	2	2,3	2,3	2,3	2,3	2,3	2,3	2,3	2,3
4.5300	No	No	Yes	Yes	Yes	3	3	3	3	3	3	3	3
4.5400	No	No	No	Yes	Yes	3	3	3	3	3	3	3	3
4.5401	No	No	No	No	Yes	3	3	3	3	3	3	3	3
4.5402	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	4	4
4.5403	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	4	4
4.5500	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	4	4
4.5501	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	4	4
4.5502	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	4	4
4.5503	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	4	4
4.6000	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
4.6001	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes

Notes

Note 1: Yes, however the Alternative ALTER TABLE support will be ignored on production machine

Note 2: Yes, however changes to the default object delivery sequence will be ignored on production machine

Note 3: Yes, however SSH distribution is not supported.

Note 4: Yes, however Extended SQL source files are not supported for distribution

Minimum i5/OS Levels

4.5200	V5R3
4.5201	V5R4
4.5300	V5R4
4.5400	V6R1
4.5401	V6R1
4.5402	V6R1
4.5403	V6R1
4.5500	V7R1
4.5501	V7R3
4.5502	V7R3
4.5503	V7R4
4.6000	V7R4
4.6001	V7R4