

[print](#) | [close](#)

APIs by Example: Valid Target Release Levels for Save Commands

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 04/26/2007 (All day)

In a [recent thread](#) in the System iNetwork forums, the question came up whether it is possible to programmatically retrieve the supported target release values for the Save Object (SAVOBJ) and Save Library (SAVLIB) commands for any given release. There is actually a positive answer to that question, as long as you use APIs.

In this article, I present and demonstrate a couple of software product APIs that, when joining forces, can produce the information sought in the above thread. The Retrieve Product Information (QSZRTVPR) API can produce a wide range of information about a particular software product, including a list of valid operating system releases. The Check Target Release (QSZCHKTG) API has, among other things, the ability to verify if a specific release is a supported target release for the current release level's save commands.

Part of the API demonstration is handled by three new CL commands that make the target release information available to programmers:

- Display Target Release (DSPTGTRLS)
- Retrieve Target Release (RTVTGTRLS)
- Check Target Release (CHKTGTRLS)

Although these commands cover most immediate needs in relation to target release information in a programming context, they might also serve as a starting point for meeting your own, more specific requirements. Let me begin by showing you how the two software product APIs fit into the scheme.

You can pass a release name to the QSZCHKTG API to verify it's a valid target release for your system. This is useful for commands supporting a target release parameter, for example. You can specify either an array of valid target release values or the special value *SAV as input to the API. With *SAV, the API evaluates the specified target release against an internal register used for the same purpose by the SAVLIB and SAVOBJ commands. This function is what makes the QSZCHKTG API so useful for the target release commands.

The QSZCHKTG API further supports the target release special values *CURRENT and *PRV, and returns the actual release identifiers for these values in a separate output parameter. So, by calling the QSZCHKTG API specifying either of these values, you get back the VxRxMx formatted value that the special value resolves to for the current release level of the operating system. For example, for release level V5R3Mo, the *CURRENT special value resolves to V5R3Mo and the *PRV special value to V5R2Mo.

Now that I am able to verify whether a specific release level identifier is supported by the save commands of the current level of the operating system, I need a way to list all possible release levels of the operating system, to provide the necessary input to the QSZCHKTG API.

The QSZRTVPR API identifies the product load for which to return information based on an input parameter that defines a product information data structure. This structure can have one of two formats. Here's a brief description of the basic format used in this example, PRDIO100:

1. Product ID	Char 7	the product ID for which information is being requested
		You can use a special value for the product ID:
		*OPSYS The product ID for the operating system for the specified release level.
2. Release level	Char 6	the release level for which information is being requested;
		the release level must be a valid special value, or
		be in the format VxRxMy
		You can use the following special values for the release level:
		*CUR uses the release level of the currently installed operating system
		*ONLY uses the only release level for which a product load is found
		*PRV uses the previous release with modification level 0 of the operating system
3. Product option	Char 4	the option number for which information is being requested;
		use 0000 for the base option
4. Load ID	Char 10	the load ID for which information is being requested;
		for example, 2924 is the load ID for an English national language version (NLV)
		you can use a special value for the load ID:
		*CODE The Load ID of the code load for the given product ID, release level, and option.

If you run the Display Software Resources (DSPSFWRSC) command and press F11 once, you'll be able to recognize many of the above parameter values. For the specific use here, the special values allowed for the individual parameters will do the job. Product ID *OPSYS, Product option 0000, and Load ID *CODE ensure that the correct product load is identified for the specified release level.

The QSZRTVPR API supports many different return formats, all of which contain different types of information for specific product loads. This information includes:

- general information about a product load, including whether the product load is installed or not
- library, folder, object, and directory list of a product load
- current or previous release level of the operating system
- list of valid release levels of the operating system from a given release level through the currently installed release level
- primary language ID of a product

You can retrieve much more information using this API — a total of nine different return formats are supported for V5R3. Check out the manual for details (a link to the API documentation is at the end of the article).

As you can see from the above list, there's an option to retrieve a list of valid release levels from a given release through the currently installed release level. Therefore, if you specify a given release as input to the QSZRTVPR API (e.g., the first one available, V1R3Mo), the API will return a list of all valid releases of the operating system, up to and including the current one. The release list is returned in the PRDR0700 return format.

Now I have the resource to provide the list of valid release level identifiers for the QSZCHKTG API to verify. Because target releases normally include only the current and two prior levels of the operating system, it's probably (and hopefully) overkill in most cases to start the list at V1R3Mo. Feel free to change and adjust as you find appropriate.

The DSPTGTRLS command simply implements the routines described above. The command-processing program first lists all valid release levels using the QSZRTVPR API, then verifies each one with the QSZCHKTG API. All release identifiers that pass the test are returned in a separate completion message to the caller of the DSPTGTRLS command. Consequently, there are no input parameters to the DSPTGTRLS command.

The RTVTGTRLS command supports three different target release special values as input parameters to the command:

Retrieve Target Release (RTVTGTRLS)

Type choices, press Enter.

Release	*CURRENT	*CURRENT, *PRV,
*OLDEST		
CL var for RTNRLS	(6) . .	Character value

You can specify the special value *CURRENT to get the current target release level of the save commands, *PRV to get the previous one, and *OLDEST to get the earliest target release level. The resolved target release level is returned in a 6-byte character variable in the VxRxNx format, where x is a digit from 0–9. I have provided a test CL program to verify the RTVTGTRLS command, and compile and call CBX1722X to run the test. See the program source for more details.

The final command in today's example is CHKTGTRLS:

```

                                Check Target Release (CHKTGTRLS)
Type choices, press Enter.
Target release . . . . . VxRxMx

```

This command works similar to the CHKOBJ command in that it returns an exception in the form of an escape message if the check is not successful. You then have to monitor for this escape message in the CL program issuing the CHKTGTRLS command to be able to evaluate the outcome of the check. Look in the CHKTGTRLS command's help text to see the actual escape messages that you can monitor for this command. If successful, the CHKTGTRLS command returns a completion message indicating that the check was passed.

Here's an example of how to code the CHKTGTRLS command in a CL program:

```

Pgm
ChkTgtRls      TgtRls( V5R1M0 )
MonMsg        CPF0000      *NONE      Do
/*-- Insert code to handle missing support of V5R1M0 --*/
EndDo
EndPgm

```

As always, help text is provided for all three commands. Please check out the command help text for further information.

This APIs by Example includes the following sources:

```

CBX1721  -- RPGLE  -- Display Target Releases - CPP
CBX1721H -- PNLGRP -- Display Target Releases - Help
CBX1721X -- CMD    -- Display Target Releases

CBX1722  -- RPGLE  -- Retrieve Target Release - CPP
CBX1722H -- PNLGRP -- Retrieve Target Release - Help
CBX1722X -- CMD    -- Retrieve Target Release
CBX1722T -- CLP    -- Retrieve Target Release - Test

CBX1723  -- CLP    -- Check Target Release
CBX1723H -- PNLGRP -- Check Target Release
CBX1723X -- CMD    -- Check Target Release

CBX172M  -- CLP    -- Valid target releases - Build commands

```

To create these objects, compile and run CBX172M. Compilation instructions are in the source headers as usual.

This article demonstrates the following software product APIs:

Retrieve Product Information (QSZRTVPR) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qsztvpr.htm>

Check Target Release (QSZCHKTG) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qszechktg.htm>

You can retrieve the source code for this API example from:

http://www.pentontech.com/IBMContent/Documents/article/54563_197_TargetRls.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-valid-target-release-levels-save-commands>