

APIs by Example: Display Partition Information, and a DSPSYSCFG Command Update

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 09/23/2010 (All day)

In the preceding issue of APIs by Example ("[Tricky Retrieve APIs and How to Process the Receiver Variable](#)," August 26, 2010, article ID 65412), I left you with a promise to provide a Display Partition Information (DSPPTNINF) command. The foundation for a DSPPTNINF command is naturally the Retrieve Partition Information (dlpar_get_info) API, which was introduced with release 5.3. So in today's issue of APIs by Example, I discuss the dlpar_get_info API as well as introduce you to the DSPPTNINF command.

In the aforementioned article, the DSPPTNINF command was referenced on the display panel of the Display System Configuration DSPSYSCFG command presented in that article. Following the distribution of that issue of the *Programming Tips* newsletter, I received an email from Bryan Dietz conveying the following information: "Starting with release 5.3 IBM has started 'point' releases, i.e., V5R3Mo of OS and V5R3M5 of Licensed Internal Code (LIC). The same applies for release 5.4 and in release 6.1 there is a LIC version V6R1M1." I didn't want to miss out on that opportunity, so with this article, I also include an update of the DSPSYSCFG command supporting the relatively new LIC release information.

Bryan also noted that in release 5.4, IBM added a *Licensed Internal Code VRM* option to the Materialize Machine Attributes (MATMATR) built-in, and he kindly included the code for both this built-in as well as a release 5.3 workaround employing the Retrieve Product Information (QSZRTVPR) API to obtain the LIC VRM. I've added both code snippets to the DSPSYSCFG CPP and the LIC VRM information to the command's display panel and help text panel group.

While I was at it, I also changed the UIM Display Panel API to keep the current page when function key F5 is pressed in order to refresh the display panel. Some of the information specified on page 2 reflects the current usage of the system resources and is therefore subject to constant change. Keeping the page displayed when pressing function key F5 allows you to better compare the readings displayed and to note the actual differences. You'll find the updated DSPSYSCFG source members with this week's source code zip file. A big thank you to Bryan for both the information and the code provided.

The dlpar_get_info API and is different from the typical API as it is also indicated by the C-style documentation explaining it in the *General Configuration APIs* section of the online API manual. The dlpar_get_info API requires one output parameter and two input parameters, here presented in that order:

1. Receiver variable (data structure)
2. Receiver variable format
3. Receiver variable length

The receiver variable format defined by parameter 2 supports two return data formats, as documented in the following excerpt from the API manual:

Format of the data to return (*data_format*)

INPUT; BINARY(4)

This parameter determines what information is returned in the receiver

variable. The supported formats are:

Value	Description
1	Partition information that is unlikely to change without partition IPL
2	Partition information that may change at any time during partition Operation

Note: If some fields returned by this API are not supported by the installed

version of the hypervisor, these fields will be set to zero.

The information returned by the API is divided into two possible groups, one containing static partition information and another containing dynamic partition information. Since I include both in the DSPPTNINF display panel, I'll need to call the API twice, specifying format 1 on the first call and format 2 on the second call. Further, in the absence of a standard API error data structure to report any error condition detected by the `dlpar_get_info` API, the API returns an integer value to indicate the outcome of the API call. Again I resort to the manual's explanation of the `dlpar_get_info` API's return value:

Return Value

Positive value Returned Partition information was successfully retrieved.

value indicates number of bytes returned in the receiver variable.

Negative value return value API cannot return data because of error. The will be a negative number describing the error, as follows:

-1	Specified format is not supported by the API.
-2	Length of the receiver variable is negative.
-3	Address of the receiver variable is invalid.
-4	API encountered an exception during execution.

(See job log for the details about the exception)

-5 Required parameter omitted.

Consequently, in my DSPPTNINF CPP, I make sure to check that the return value following the API call contains a positive value before processing the data structure returned by the API. Depending on the context in which the `dlpar_get_info` API is employed, you could also include code to detect each of the above possible types of failure and return corresponding exception messages to the caller of your program. Since the types of errors, however, all indicate some sort of misconception on the API programmer's side, at the point where I have the API call working correctly, I simply verify that the API call was successful before continuing the execution path of the CPP.

Another peculiarity facing you when working with the `dlpar_get_info` API output is the presence of bit fields in the return variable data structures. Bit fields are used by some APIs to return Boolean type of information—that is, information that can be expressed by an on or off value, as in yes or no. This allows you to store up to eight yes/no values in a single byte, as opposed to the eight bytes required to store this information in one byte each. Other languages, such as C, allow you to map each bit directly to a program variable; however, RPG does not. In case you're not already familiar with bit fields, I'll show you how to work with them in RPG in a moment. Both API return formats contain bit fields, and here's the relevant documentation for format 1:

Offset		Bit	Type	Field	Description
Dec	Hex				
44	2C		BINARY(4)	<code>lpar_flags</code>	
44	2C	0	Bit(30)	Reserved	Reserved
47	2F	6	Bit(1)	<code>lpar_smtbound</code>	Bound hardware threads indicator. If on, hardware threads are bound.
47	2F	7	Bit(1)	<code>lpar_dedicated</code>	Dedicated processors indicator. If on, partition uses dedicated processors.

Return format 1 defines a four-byte integer at decimal offset 44 as `lpar_flags`. As each byte contains eight bits, that makes a total of 32 bits. The documentation specifies the first 30 bits (from left to right) as being reserved for future use. That leaves the two rightmost bits, bit 31 and 32, to define the values of the two `lpar_flags`, `lpar_smtbound` and `lpar_dedicated`, respectively.

The current recommended approach of evaluating bit values involves the use of the RPG/IV Bitwise AND Operation `%BitAnd()` built-in function (BIF). Using `%BitAnd()`, the variable containing the bit(s) and a bit mask in the form of a hexadecimal value defining the bits you want to interrogate, you can determine whether the bit(s) in the bit mask are on or not. At the end of this article, I include a link to a Q&A article on the topic written by Scott Klement, explaining the `%BitAnd()` BIF and approach in more detail.

Alternatively, I sometimes for single bit tests use the Test Bit in String `tstbts()` MI built-in and C library function for the same purpose. The `tstbts()` function requires you to specify the address of the

variable containing the bit to test as the function's first argument, and the bit number to test as the second argument. The bits are numbered, left to right, from 0 (zero) and onward. Consequently, in a single byte the leftmost bit number is 0, and the rightmost bit number is 7. The `tstbts()` function returns an integer having the value 0 if the tested bit is off, and 1 if the bit is on.

In the above context, I need to find the value of bits 31 and 32, and given the zero offset mentioned, that implies to simply subtract 1 from each bit number to find the `tstbts()` function's second parameter, in this case adding up to bits 30 and 31, respectively:

```

**-- Partition information - format 1 (static):
D PtnInf01          Ds                      Qualified
D  FmtVer                      10u 0
D                               10u 0
D  MaxMem                    20u 0
D  MinMem                    20u 0
D  MemInc                    20u 0
D  DpcWRT                    20u 0
D  LparID                     10u 0
D  LparFlags                 10u 0
D  MaxPhyPrc                 10u 0
...

D LPAR_SMTBOUND    c                      30
D LPAR_DEDICATED   c                      31

/Free

    LparSmtBnd = tstbts( %Addr( PtnInf01.LparFlags ):
LPAR_SMTBOUND );
    LparDdcPrc = tstbts( %Addr( PtnInf01.LparFlags ):
LPAR_DEDICATED );

/End-Free

```

Another topic surfacing when dealing with the `dlpar_get_info` API output information relates to time granularity. Four of the APIs between-IPL CPU time fields are noted in nanoseconds. I must admit that I had to look up exactly how many nanoseconds fit into one second, and thought I'd spare you the Google challenge:

```

Nanosecond: One billionth of a second (0.000000001) or (10^-9) of a
second or
            1/1000 of a microsecond.

```

In other words, a nanosecond is a very, very short amount of time...

Anyway, now on to the `DSPPTNINF` command and display panels. When prompting the `DSPPTNINF` command, here's what you'll see:

```

Display Partition Information (DSPPTNINF)

```

Type choices, press Enter.

Output * , *PRINT

You specify as the only command parameter whether the command output should go to a display panel or a printed list. Here's what the command would look like if you decided to display the current partition's information on screen:

```
DSPPTNINF OUTPUT(*)
```

Following the execution of the above command, you'll be presented with a display panel having the following appearance:

```

                                Display Partition Information
WYNDHAMW                                                                18-09-10
16:59:15
Partition name . . . . . : 43-21CBA
Partition ID . . . . . : 1
Logical serial number . . . . . : 4321CDA1
Number of partitions . . . . . : 1
Firmware level . . . . . : 16

Partition group ID . . . . . : 32769
Shared processor pool ID . . . . . : 0

Minimum memory . . . . . : 320
Maximum memory . . . . . : 16384
Memory increment . . . . . : 64
Defined memory . . . . . : 15680

```

```

Online memory . . . . . : 15680

Maximum physical processors . . . . . : 2

Physical processors in the system . . . : 2

Physical processors in shared pool . . . : 0

More...
F3=Exit    F5=Refresh    F12=Cancel    F19=Display system
configuration
F20=Work with processor resources    F24=More keys

```

In addition to the partition information displayed, there's a function key shortcut to [the DSPSYSCFG command presented last time](#) as well as a function key to show the full partition name, in case it exceeds 32 bytes. Function keys F20 and F21 provide access to the Work with Hardware Resources (WRKHDWRSC) command for processor resources and storage resources, respectively. To display the second page of partition information, press the Page Down button:

```

                                Display Partition Information

WYNDHAMW
                                18-09-10
16:59:15
Minimum/maximum virtual processors . . . : 1      2

Defined virtual processors . . . . . : 2

Online virtual processors . . . . . : 2

Minimum/maximum processing capacity . . : 100      200

Processing capacity increment . . . . . : 100

Defined processing capacity . . . . . : 200

Minimum required processing capacity . . : 100

Maximum licensed capacity . . . . . : 200

Processing capacity . . . . . : 200

Unallocated processing capacity . . . . : 0

```

```

Minimum/maximum interactive capacity . . :    0          10000

Defined interactive capacity . . . . . :    10000

Interactive capacity . . . . . :    10000

Interactive threshold . . . . . :    10000

Unallocated interactive capacity . . . . :    0

More...
F3=Exit   F5=Refresh   F12=Cancel   F19=Display system
configuration
F20=Work with processor resources   F24=More keys

```

To refresh the information displayed, press function key F5. To show the third and final page of partition information, press the Page Down button:

```

                                Display Partition Information

WYNDHAMW                                                                18-09-10
16:59:15
Total CPU time . . . . . :    131896875000000

Interactive CPU time . . . . . :    4812306000000

Interactive CPU time above threshold . . :    0

Unused CPU time in shared pool . . . . . :    0

Variable capacity weight . . . . . :    0

Defined variable capacity weight . . . . :    0

Unallocated variable capacity weight . . :    0

Dispatch wheel rotation time . . . . . :    10006868

Dispatch latency . . . . . :    0

Bound hardware threads . . . . . :    *YES

```

```

Dedicated processors . . . . . : *YES

Capped partition . . . . . : *YES

Shared pool data . . . . . : *NO

Hardware multi-threading . . . . . : *YES

    SMT threads per processor . . . . . : 2

    Bottom
    F21=Work with storage resources    F22=Display entire name
    F24=More keys

```

The display panel and all fields shown are, as always, explained in the cursor-sensitive help text associated with the display. Just point the cursor to the area or field of interest and press function key F1 to access the help text provided.

This APIs by Example includes the following sources:

```

CBX219  -- RPGLE  -- Display Partition Information - CPP
CBX219E -- RPGLE  -- Display Partition Information - UIM General Exit
CBX219H -- PNLGRP -- Display Partition Information - Help
CBX219P -- PNLGRP -- Display Partition Information - Panel Group
CBX219X -- CMD    -- Display Partition Information

CBX219M -- CLP    -- Display Partition Information - Build command

```

To create all these objects, compile and run the CBX219M program, following the instructions in the source header. You'll also find compilation instructions in the respective source headers.

The following sources have been updated in order to provide LIC release support for the DSPSYSCFG command:

```

CBX218  -- RPGLE  -- Display System Configuration - CPP
CBX218E -- RPGLE  -- Display System Configuration - UIM General Exit
CBX218H -- PNLGRP -- Display System Configuration - Help
CBX218P -- PNLGRP -- Display System Configuration - Panel Group

```

These sources are included in the DSPPTNINF command source zip file provided at the end of this article.

Related articles:

[Testing Bits in Free-Form RPG](#)

[APIs by Example: Tricky Retrieve APIs and How to Process the Receiver Variable](#)

[Wiki on Bit Fields](#)

This article demonstrates the following APIs and MI Built-ins:

[Retrieve Partition Information \(dlpar_get_info\) API](#)

[Materialize Machine Attributes \(MATMATR\) MI Built-in](#)

[Test Bit in String \(TSTBTS\) MI Built-in](#)

[i5/OS Machine Interface](#)

[Retrieve the source code for this API example.](#)

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-display-partition-information-and-dspsyscfg-command-update>