


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

APIs by Example: Retrieve Subsystem Entries API

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 09/21/2006 (All day)

Some APIs can have very complex parameter lists and be an immense challenge to code. Other APIs, however, are pretty straightforward to tackle. Nevertheless there's also a first time when it comes to making the acquaintance of such "easy" APIs, so this is the topic for today's API by Example.

In fact, the List Subsystem Entries (QWDLSE) API has only four parameters, one of which is the common API error data structure, so calling the API correctly should hardly pose any insuperable problems. The API returns the subsystem entry information in a user space, and I therefore focus on the techniques involved in accessing and retrieving API output from user spaces. To wrap it all up in a practicable context, I've written the Work with Routing Entry (WRKRTGE) command.

Despite the simplicity, let's start off with the QWDLSE API parameter list. Here it is in its entirety:

1. Qualified user space name	Input	Char(20)
2. List format	Input	Char(8)
3. Qualified subsystem name	Input	Char(20)
4. Error code	I/O	Char(*)

The user space specified for the first parameter is typically created immediately before calling the API, using the Create User Space (QUSCRTUS) API. Defining a named constant for the qualified user space name requires you to specify the name in only one place and ensures that it's the same user space referenced on all subsequent API calls:

```
**-- Global constants:
D USRSPC_Q          C              'LSTRTGE  QTEMP'
```

The list format can be one of the following seven available formats, one for each different subsystem entry type that this API supports:

- SBSE0100 Routing entry list
- SBSE0200 Communications entry list
- SBSE0300 Remove locations entry list
- SBSE0400 Autostart job entry list
- SBSE0500 Prestart job entry list
- SBSE0600 Workstation name entry list
- SBSE0700 Workstation type entry list

Because this example requires the routing entry list, SBSE0100 should be specified for the second parameter.

The qualified subsystem name must be specified in the same format as the user space parameter, a 10-byte subsystem name followed by a 10-byte library name. Because the qualified subsystem name also happens to be the WRKRTGE command's only parameter, and CL commands pass library-qualified parameters in the same format, I simply pass the CPP's input parameter to the QWDLBSBSE API:

```
LstSbsEnt( USRSPC_Q: 'SBSE0100': PxSbsNam_q: ERRC0100 );
```

The final error code parameter has been discussed previously in great detail in this newsletter. Please refer to the following "Getting Started with APIs" article for all details:

<http://www.SystemiNetwork.com/article.cfm?id=18648>

After the call to the QWBLSBSE API, the first thing to do is check whether the call was successful. To check, we evaluate the Bytes Available subfield in the ERRC0100 data structure.

```
If  ERRC0100.BytAvl = *Zero;
  ExSr  PrcUsrSpc;
EndIf;
```

If zero bytes available is returned, it's safe to proceed. Otherwise, you would usually use the information in the ERRC0100 data structure to format and send an escape message, as in the following example:

```
If  ERRC0100.BytAvl > *Zero;

  If  ERRC0100.BytAvl
```

But in this case, the main candidate that could cause a failure, the qualified subsystem name, has already been validated in the WRKRTGE command's validity checking program (VCP), so I leave that out of the equation here.

If you decide to return an escape message to the caller, be sure to take into account that sending an escape message immediately terminates the current call level (i.e., the program sending the escape message). So you should make sure that necessary cleanup activities are handled, for example by registering an activation group exit program using the CEE4RAGE API. Links to API documentation are at the end of this article.)

The processing of the list data in the user space populated by the QWDLBSBSE API is based on the information in the generic header section of the user space. The following data structure describes the part of the layout of this information relevant for this example:

```

**-- User space generic header:

D  UsrSpcHdr          Ds                      Qualified  Based( pUsrSpc )

D  OfsInpSec          10i 0 Overlay( UsrSpcHdr: 109 )

D  SizInpSec          10i 0 Overlay( UsrSpcHdr: 113 )

D  OfsHdrSec          10i 0 Overlay( UsrSpcHdr: 117 )

D  SizHdrSec          10i 0 Overlay( UsrSpcHdr: 121 )

D  OfsLstEnt          10i 0 Overlay( UsrSpcHdr: 125 )

D  NumLstEnt          10i 0 Overlay( UsrSpcHdr: 133 )

D  SizLstEnt          10i 0 Overlay( UsrSpcHdr: 137 )

```

As you can see, the UsrSpcHdr data structure is defined as based on the pointer pUsrSpc. This means that the data structure maps to whatever location that this pointer is pointing to (i.e., the storage address defined by the pointer's current value).

After the user space has been created and the QWDLSE API has written data to it, I use the Retrieve Pointer to User Space (QUSPTRUS) API to retrieve the address of the first byte of the user space:

```
RtvPtrSpc( USRSPC_Q: pUsrSpc );
```

As the first parameter, I specify the user space's qualified name (using the named constant mentioned earlier), and if all works out, the QUSPTRUS returns the user space address in the second parameter. So by simply specifying the pointer that the user space generic header data structure is based on as the second parameter, I make the information in this data structure immediately available following the (successful) QUSPTRUS API call.

At the end of this article, I provide a link to the section in IBM's online API manual that describes in detail the layout of the various user space structures and their internal relationships. I recommend that you study this information to understand the objectives and thoughts behind this concept. Doing so will help you write the most flexible and robust code when dealing with APIs and user spaces.

Armed with the information in the user space generic header data structure, I initialize the pointers that the Input Parameter Section, the Header Section and the List Entry Structure are based on, to the address returned:

```

pInpInf = pUsrSpc + UsrSpcHdr.OfsInpSec;
pHdrInf = pUsrSpc + UsrSpcHdr.OfsHdrSec;
pLstEnt = pUsrSpc + UsrSpcHdr.OfsLstEnt;

```

At this point, I have access to the information in the Input Parameter Section and the Header Section. And if list entries are present, I also have access to the first list entry. However, I do not access the list entry data until I have verified that there actually is at least one entry in the list. The

Number of List Entries (NumLstEnt) subfield of the user space header holds that piece of information:

```

For  Idx = 1  to UsrSpcHdr.NumLstEnt;
    LstEnt.Option = *Zero;
    LstEnt.SeqNbr  = SBSE0100.SeqNbr;
    LstEnt.RtgPgm_q = SBSE0100.RtgPgm_q;
    LstEnt.RtgCls_q = SBSE0100.RtgCls_q;

    ...

    If  Idx

```

The preceding code snippet demonstrates how the user space list can be retrieved, one entry at a time. The For loop executes as many times as there are list entries available in the user space. For each entry, the retrieved routing entry information is processed, and finally the list entry pointer is advanced to the next entry.

The pointer is advanced by means of another header information subfield called "Size of List Entry." When the pointer is advanced, it's important to make sure you only do it as many times as there are list entries. Never point the pointer beyond the end of the list, or unpredictable results can occur.

Another technique that might be relevant to cover in this context relates to information being added to API return formats over time. Often an API data structure will have new fields that are added with a release upgrade. Other times, changes are introduced by means of PTFs. When the changes occur during a release upgrade, you can use compiler directives to distinguish between old and new formats as shown in the following example. This lets you use the same source member on different i5/OS releases.

```

/If Defined( *V5R3M0 )

    Format for V5R3 or later

/Else

    Format for V5R2 or earlier

/EndIf

```

Another method to safeguard your access to the returned information is to (again) use the information provided in the user space generic header data structure. For the QWDLSE API, release V5R3 introduced an addition to the SBSE0100 format. Three new resource affinity attributes were added. To ensure that these fields are referenced only if present, I added the following statement to the list entry processing part of my code:

```
If  UsrSpcHdr.SizLstEnt >= %Size( SBSE0100 );
  LstEnt.ThrRscAffGrp = SBSE0100.ThrRscAffGrp;
  LstEnt.ThrRscAffLvl = SBSE0100.ThrRscAffLvl;
  LstEnt.RscAffGrp    = SBSE0100.RscAffGrp;
EndIf;
```

Only if the list entry length returned by the QWDLSE API is equal to (or longer than) the SBSE0100 data structure do I access the newly added fields.

When I'm through processing the list, the program continues doing what it's further supposed to do. There's one final responsibility for the programmer in relation to the user space initially created. When the program is about to end, it's good practice to perform the necessary cleanup duties and ensure that the user space is deleted:

```
DltUsrSpc( USRSPC_Q: ERRC0100 );
```

As for the purpose of all these efforts, let me give you a brief introduction to the WRKRTGE command. Here's the command prompt:

```
Work with Routing Entries (WRKRTGE)
Type choices, press Enter.

Subsystem . . . . . Name
Library . . . . . *LIBL Name, *LIBL, *CURLIB
```

You simply specify the name of the subsystem whose routing entries you want to work with, as in the following example:

```
WRKRTGE SBS(QINTER)
```

Running the above command leads to the display of a list panel similar to the one below:

```
Work with Routing Entries WYNDHAMW
16-09-06 21:05:58
Subsystem . . . : QINTER Subsystem status : *ACTIVE
Library . . . : QSYS
Type options, press Enter.
2=Change 3=Copy 4=Remove
Seq
Opt Nbr Program Library Compare value Start Pos.
10 QCMD QSYS QCMDI 1
15 QCMD QSYS QIGC 1
20 QCMD QSYS QS36MRT 1
40 QARDRIVE QSYS 525XTEST 1
```

700	QCL	QSYS	QCMD38	1
9999	QCMD	QSYS	*ANY	0
				Bottom
Parameters or command				
===>				
F3=Exit	F4=Prompt	F5=Refresh	F6=Add routing entry	
F11=View 2	F12=Cancel	F21=Print list	F24=More keys	

The list panel offers three alternate views, all together displaying all available routing entry attributes. The list options let you run the three routing entry CL commands: Change Routing Entry (CHGRTGE), Add Routing Entry (ADDRTGE) (based on an existing routing entry, thus providing the copy option), and finally Remove Routing Entry (RMVRTGE).

Among other facilities, the function keys offer access to a print list function, the ADDRTGE command, and the WRKSBSD (Work with Subsystem Description) command. Cursor-sensitive help text is also provided for both list panel and command.

If you want to learn more about routing entries and the role they play in how jobs get processed in a subsystem, follow the link at the end of this article for "How works get processed," as well as the other links for related aspects of work management.

This APIs by Example includes the following sources:

```
CBX162  -- Work with Routing Entries - CCP
CBX162E -- Work with Routing Entries - UIM Exit Program
CBX162H -- Work with Routing Entries - Help
CBX162P -- Work with Routing Entries - Panel Group
CBX162V -- Work with Routing Entries - VCP
CBX162X -- Work with Routing Entries

CBX162M -- Work with Routing Entries - Build command
```

To create all these objects, compile and run CBX162M. Compilation instructions are in the source headers, as usual.

"How work gets processed":

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/rzaks/rzakshowwrkgetsproc.htm>

The CEE4RAGE API documentation:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/apis/CEE4RAGE.htm>

User space structure documentation:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/apis/usf.htm>

This article demonstrates the following APIs:

List Subsystem Entries (QWDLSE) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qwdlse.htm>

Retrieve Subsystem Information (QWDRSBSD) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/apis/qwdrbsd.htm>

Create User Space (QUSCRTUS) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/quscrtus.htm>

Delete User Space (QUSDLTUS) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qusdltus.htm>

Retrieve Pointer to User Space (QUSPTRUS) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qusptrus.htm>

Open Display Application (QUIOPNDA) API:

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

Close Application (QUICLOA) API:

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

Display Panel (QUIDSPP) API:

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

Put Dialog Variable (QUIPUTV) API:

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

Get Dialog Variable (QUIGETV) API:

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

Add List Entry (QUIADDLE) API:

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

Get List Entry (QUIGETLE) API:

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

Update List Entry (QUIUPDLE) API:

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

Remove List Entry (QUIRMVLE) API:

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

Retrieve List Attributes (QUIRTVLA) API:

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

Set List Attributes (QUISETLA) API:

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

Delete List (QUIDLTL) API:

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

Print Panel (QUIPRTP) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/quiprtp.htm>

Add Print Application (QUIADDPA) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/quiaddpa.htm>

Remove Print Application (QUIRMVPA) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/quirmvpa.htm>

Retrieve Message (QMHRTVM) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QMHRTVM.htm>

Send Program Message (QMHSNDPM) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QMHSNDPM.htm>

Retrieve Object Description (QUSROBJD) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qusrobjd.htm>

You can retrieve the source code for this API example from

http://www.pentontech.com/IBMContent/Documents/article/53255_117_RtvSbsEnt.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-retrieve-subsystem-entries-api>