



APIs by Example: Using the ERRC0200 Data Structure

[System iNetwork](#)

System iNEWS Staff

Thu, 06/24/2004 (All day)

In this week's APIs by Example, Carsten Flensburg has written a sample program that demonstrates the ERRC0200 data structure.

The downloadable code for this article implements a DSPRTGDTA command that, when run, will display the routing data of your job in the message line of your screen.

The Retrieve Job Information (QUSRJOBI) API is used to retrieve the routing data. If you wanted to, you could change this utility to display a different job attribute such as the job's internal ID, run priority, or time slice, as this API returns all of this information.

Here's a screen shot of the DSPRTGDTA command

```
-----
                        Display Job Routing Data (DSPRTGDTA)
Type choices, press Enter.
Job name . . . . . *                Name, *
  User . . . . .                Name
  Number . . . . .                000000-999999
                                   Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use
F24=More keys
-----
```

The message that it displays on the screen looks like this: Job 604420/KLEMSCOT/QPADEV0001 has routing data "QCMDI".

HOW DOES IT WORK?

In the article "Getting Started with APIs, part 2" from the May 20, 2004 issue of this newsletter, it was explained that there is a standard error code data structure that gets passed to an API. When an error occurs, the API will populate that data structure with error information and pass it back to your program. This standard error code data structure is called "ERRC0100."

The "ERRC0200" data structure is an alternative to the original error code structure that contains an important addition: The CCSID of the message.

WHAT'S A CCSID?

If you're not familiar with the term "CCSID" (pronounced "see-sid") it stands for "Coded Character Set Identifier" and it tells the system the character set and codepage of text that you want to show to a user, store in a file, etc. It's necessary because the different languages, conventions and cultures around the world require different characters to be available.

Since the characters in each country aren't the same, simply copying the bytes from one place to another can result in the wrong characters being displayed. For example, in the United States we typically use CCSID 37, in the United Kingdom they use 285, and in Denmark (where Carsten lives) they use 277. The hex character x'5B' displayed in the U.S. appears as a dollar sign. If it's displayed in the United Kingdom, the same hex value displays as the symbol for Pounds Sterling. In Denmark, it displays as the letter A with a small circle above it.

When you know the CCSID that text is written in, you can convert it using a translation table. If you take the same example of a dollar sign written in CCSID 37 and convert it to CCSID 285, it will still be displayed as a

dollar sign. If you take it again and convert it to CCSID 277, it will still be displayed as a dollar sign.

HOW DO I USE ERRCo200?

The "error code" parameter of an API can accept either the original error code structure (as I demonstrated in the May 20 article) or an ERRCo200 data structure.

The Information Center describes the ERRCo200 data structure as having the following format:

Offset		Use	Type	Field
Dec	Hex			
0	0	INPUT	BINARY(4)	Key
4	4	INPUT	BINARY(4)	Bytes provided
8	8	OUTPUT	BINARY(4)	Bytes available
12	C	OUTPUT	CHAR(7)	Exception ID
19	13	OUTPUT	CHAR(1)	Reserved
20	14	OUTPUT	BINARY(4)	CCSID of the CCHAR data
24	18	OUTPUT	BINARY(4)	Offset to the exception data
28	1C	OUTPUT	BINARY(4)	Length of the exception data

The first field, called "Key", should always contain a value of -1. This is important because it's how the API knows that you want to use ERRCo200! Since the first field of ERRCo100 is always zero or higher, when the API receives a -1, it knows that ERRCo200 is desired.

The bytes provided field tells the API how much space is available for it to use when returning error information.

The bytes available field is set by the API and contains the amount of bytes of error data that the API returned to your program. This field is set to zero if the API encounters no errors. All of the other fields in the data structure are unchanged if no error occurs.

The exception ID field contains the message ID of the error message, such as "CPF3C58" to indicate that the job name was specified incorrectly.

The CCSID of CCHAR data field contains the CCSID as described above. The term CCHAR is short for "Convertible Character" and refers to data that can be converted from one CCSID to another.

The offset of exception data and length of exception data fields are used to locate the values that are used to fill in variables in the message description as described in the May 20, 2004 issue of this newsletter.

The following is an example of coding this data structure in an ILE RPG program:

D	ERRCo200	Ds	Qualified
D	Key	10i 0	Inz(-1)
D	BytPro	10i 0	Inz(%Size(ERRCo200))
D	BytAvl	10i 0	
D	MsgId	7a	
D		1a	
D	CcsId	10i 0	
D	MsgDtaOfs	10i 0	
D	MsgDtaLen	10i 0	
D	ExtraSpace	1024a	

The BytAvl field will be set to zero if the API succeeds, or set to the length of the error information if it fails. It is used in the program to determine if anything went wrong, as illustrated in the following code snippet:

C	CallP	RtvJobInf(JOBI0400
C		: %Size(JOBI0400)
C		: 'JOBI0400'
C		: PxJobId
C		: *Blank
C		: ERRCo200
C)
**		
C	If	ERRCo200.BytAvl > *Zero
**		

```
... an error has occurred ...
```

When an error has occurred, the message data is extracted from the ERRC0200 data structure. The data itself will be somewhere in the field that I called "ExtraSpace" above. The position that it's located in is found by checking the value of the MsgDtaOfs field. You can extract the message data with the %subst() BIF as follows:

```
C          Eval      MsgDta = %Subst( ERRC0200
C                                     : ERRC0200.MsgDtaOfs + 1
C                                     : ERRC0200.MsgDtaLen
C                                     )
```

In the DSPRTGDTA utility, the Send Program Message (QMHSNDPM) API is used to send an escape message when an error has occurred. The message that was extracted is passed to this API, as well as the MsgId and CCSID returned in the ERRC0200 data structure.

The Send Program Message API will ensure that the characters are properly translated to the job's CCSID so that characters will always be displayed as they were intended to be.

The following code runs the Send Program Message API:

```
C          CallP      SndPgmMsg( ERRC0200.MsgId
C                                     : 'QCPFMSG *LIBL'
C                                     : MsgDta
C                                     : %Len( MsgDta )
C                                     : '*ESCAPE'
C                                     : '*PGMBDY'
C                                     : 1
C                                     : MsgKey
C                                     : ERRC0200
C                                     : 10
C                                     : '*NONE *NONE'
C                                     : *Zero
C                                     : '*CHAR'
C                                     : ERRC0200.CcsId
C                                     )
```

The "Display Routing Data" utility demonstrates the following APIs and API concepts:

Retrieve Job Information (QUSRJOBI)

<http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/apis/qusrjobi.htm>

Send Program Message (QMHSNDPM)

<http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/apis/QMHSNDPM.htm>

Information about the ERRC0100 and ERRC0200 error code data structures

<http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/apis/error.htm>

Information about CCSIDs

<http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/nls/rbagsccsidref.htm>

CCSIDs as they relate to messages

<http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/nls/rbagsccsidmsgsup2.htm>

You can download the sample code for this article from

<http://www2.systeminetwork.com/noderesources/code/clubtechcode/DisplayRoutingData.zip>.

The above source code was written by Carsten Flensburg. You can contact Carsten at

<mailto:flensburg@novasol.dk>.

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-using-errc0200-data-structure>