


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

## APIs by Example: Conversion of a Path Name

*System iNetwork Programming Tips Newsletter*

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 04/21/2005 (All day)

This week's APIs by Example will focus on the concept of path name format in general and the character data conversion required to resolve the path name in particular. As we all know, object and library names uniquely identify an object in the QSYS.LIB file system. But for APIs to identify objects through the IFS interface, a path name is instrumental. The path name format is a data structure that contains the path name as well as information about the character set of the path name.

Here's a brief description of the path name format structure:

Offset		Type	RPG/IV	Field
Dec	Hex			
(1) 0	0	BINARY(4)	10i 0	CCSID
4	4	CHAR(2)	2a	Country or region ID
6	6	CHAR(3)	3a	Language ID
9	9	CHAR(3)	3a	Reserved
(2) 12	C	BINARY(4)	10i 0	Path type indicator
16	10	BINARY(4)	10i 0	Length of path name
(3) 20	14	CHAR(2)	2a	Path name delimiter character
22	16	CHAR(10)	10a	Reserved
(4) 32	26	CHAR(*)	5000a	Path name

- (1) The Coded Character Set Identifier (CCSID) defines the CCSID of the 'Path name'.
- (2) Depending on the 'Path type indicator', the 'Path name' parameter could also be a space pointer instead of a character string.
- (3) The 'Path name delimiter character' is returned in the same CCSID as the 'Path name' itself.
- (4) The asterisk notation really means 'varying length', but a length of 5000 bytes should be sufficient in most situations.

All the details about the path name concept are documented in the Information Center under API concepts:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/concept.htm>

And the Path name format section here:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/pns.htm>

The path name format is used both as input and output parameter to APIs. For input use, the path name structure is not as complicated to set up as it might seem initially. Specifying the following values will tell the API that the path name uses the current job's settings:

Field:	Value:
CCSID	0
Country or region ID	x'0000'
Language ID	x'000000'

Please note that the reserved field must also be set to all x'00'.

The 'Path type indicator' will most often have the value zero, which specifies that the path name will be a character string and that the path name delimiter will be a single character. The 'Path name length', 'Path name delimiter', and 'Path name' fields are self explanatory.

Here are a few examples of APIs that expect a path name format structure as input parameter:

QSYRTVUA -- Retrieve Users Authorized to an Object API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qsyrtvua.htm>

QWCLOBJL -- List Object Locks API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qwclobjl.htm>

QPOLROR -- Retrieve Object References API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qpolror.htm>

When you call an API that outputs a path name format, you'll usually have to face the challenge of performing character data conversion. This is due to the fact that APIs usually return path name information in Unicode, because that ensures a correct data representation of the path name, regardless of which CCSID your job is currently using.

Performing a conversion from Unicode to your job's current CCSID is required for anybody to be able to read the path name. To give you an example of how this can be achieved, I've written a couple of commands, both based on the same Command Processing Program (CPP). The CPP retrieves a specified user profile's home directory attribute, using the QSYRUSRI (Retrieve User Information) API.

The QSYRUSRI returns both a user profile's 'Locale Path' and 'Home Directory' attributes using the path name format. In this example, I will focus on the latter, but the processing involved applies to all path name format structures.

The Display User Directory (DSPUSRDIR) and the Retrieve User Directory (RTVUSRDIR) commands display and retrieve, respectively, the specified user profile's home directory.

Here's what prompting the two commands will display:

```

= = = = =
                                Display User Directory (DSPUSRDIR)
Type choices, press Enter.
User profile . . . . . *CURRENT      Name, *CURRENT

= = = = =

                                Retrieve User Directory (RTVUSRDIR)
Type choices, press Enter.
User profile . . . . . *CURRENT      Name, *CURRENT

```

CL var for HOMDIR	(1024)	. .	Character value
= = = = =	= = = = =	= = = = =	= = = = =

The DSPUSRDIR command displays the home directory information in an informational message:

```
User profile JULIAN home directory is '/home/JULIAN'.
```

As usual, the commands' parameters are explained in detail in the included help panel groups.

In the CPP, once the Home Directory path name structure has been returned by the QSYRUSRI API, I use the QTQCVRT (Convert a Graphic Character String) API to perform the conversion from the Unicode CCSID to the CCSID of the current job.

The QTQCVRT API is capable of converting a character data string, based on a specified to- and from-CCSID. This API is a member of the CDRA (Character Data Representation Architecture) API family, which is described here:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/nls4.htm>

As the above documentation reveals, most of the CDRA APIs have a CDR prefix in their name, and these are located in library QSYS2. A functionally equivalent set of APIs are located in library QSYS, and instead of a QCD prefix, this set of APIs has names starting with QTQ. To avoid any possible library list issues, I normally recommend using the QTQ API set.

The CDRA APIs are a bit atypical in how they report back error conditions. Instead of the familiar ERRC0100 error data structure, a feedback array of three 32-bit binary values are returned. The first array element holds the status code in its first 16 bits, and the second array element holds the reason code in the first 16 bits.

In the event of an error condition, the status field and reason code will both be set to non-negative values, the first indicating the type of error and the second indicating the specific reason. If your primary intention is to ensure that the CDRA API completed successfully, you can simply specify the array as 4 byte integers and check that the status field (array element one) is equal to zero.

To fully exploit the CDRA APIs, IBM recommends that you consult the Character Data Representation Architecture Reference book, SC09-1390-01, which can be ordered in hardcopy from IBM Library Center:

<http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US>

If spending money on IBM manuals is not included in your budget, I have also found the following CDRA online documentation:

Character Data Representation Architecture Overview:

<http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/FOCOVRoo/CCONTENTS?DT=19951110145034>

Character Data Representation Architecture Reference and Registry:

<http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/FOCREFoo/CCONTENTS?DT=19951114134943>

Another approach to perform a CCSID based data conversion would be using the iconv() data conversion APIs that were demonstrated in the Programming Tips Newsletter on April 26, 2001:

<http://www2.systeminetwork.com/article.cfm?ID=10193>

For large data buffers or when you have a requirement to perform multiple data conversions, the `iconv()` approach works better due to its session-based conversion process.

The conversion initialization is performed as an individual step, returning a handle to the `iconv()` function, which can then be run over and over again until the conversion process is complete. At that point, the conversion session termination is performed in an individual, final step. This way the overhead from initialization and termination is only incurred one time, although the conversion process is performed multiple times.

The DSPUSRDIR and RTVUSRDIR commands include the following sources:

CBX134 -- Common command processing program.

CBX1341H -- DSPUSRDIR help text panel group.

CBX1341X -- DSPUSRDIR command definition source member.

CBX1342H -- RTVUSRDIR help text panel group.

CBX1342X -- RTVUSRDIR command definition source member.

Compilation instructions are found in the source headers.

This article demonstrates the following APIs:

Retrieve User Information (QSYRUSRI) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qsyrusri.htm>

Convert a Graphic Character String (QCDCVRT/QTQCVRT) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/CDRCVRT.htm>

Retrieve Job Information (QUSRJOB) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qusrjobi.htm>

Send Program Message (QMHSNDPM) API:

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

You can retrieve the source code for this API example from

[http://www.pentontech.com/IBMContent/Documents/article/50774\\_17\\_PathNameFmt.zip](http://www.pentontech.com/IBMContent/Documents/article/50774_17_PathNameFmt.zip).

The above article was written by Carsten Flensburg. If you have any questions, you can contact Carsten at [flensburg@novasol.dk](mailto:flensburg@novasol.dk).

**Source URL:** <http://iprodeveloper.com/rpg-programming/apis-example-conversion-path-name>