

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

APIs by Example: User Index APIs, Part Two

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 10/14/2004 (All day)

In this follow-up to User Index APIs, Part One, I will show how the User Index APIs can be put to work to create a Convert Display File Constants (CVTDSPFCNS) command.

As discussed in part one, user indexes offer sort and search functionality, very much like a keyed file does, but the process is significantly faster. The CVTDSPFCNS utility presented in this article employs the Add User Index Entry and Retrieve User Index Entry APIs, the Create User Index command that I demonstrated in the previous installment, and the native Delete User Index command.

The utility itself was inspired by a facility offered by DDS, whereby display file constants can be externalized into message descriptions, giving you the option to change the constants without recompiling the display file.

This especially comes in handy when you need to enable applications to support more than one language. Simply copy the message file holding the message descriptions to another library, for example APPLIBDEU (Application Library - German), and translate the copied message file's message descriptions to German. If you now add the APPLIBDEU library before the original application library in your job's library list, the display file will start "speaking" German.

The system automatically retrieves the message text of the message IDs referred by the MSGID keyword in the display file record format every time an output operation is performed against the record format.

Using this technique, you will only need to maintain one display file no matter how many languages you have to support. Adding a new language only requires you to translate the messages again. No programming is needed! Optionally, you can also choose to externalize all display file error messages defined by the ERRMSG keyword.

The task solved by the user index in the CVTDSPFCNS command is to keep track of the message IDs generated during the conversion process. The message ID prefix is either passed as an explicit value by the command or generated from each record format name. An incrementing integer is then appended to this prefix. To avoid possible duplicate message IDs, it is necessary to store each message ID prefix's current integer value so that when you want to add to the message file, you have a way of knowing what the next available message ID will be.

This requirement meant that I had to make the utility perform an initial pass through the display file source to retrieve any already existing MSGID and ERRMSGID referenced message IDs. That facility also simplifies the task of adding new constants to a display file that already has had its constants externalized; simply add the new constants and rerun the CVTDSPFCNS command.

Following the conversion and recompilation of a display file, you will also need to recompile all programs referencing the converted display file. This is necessary in order for the program to pick up the new display file record format level IDs. Otherwise, a record format level check exception will occur the next time the program tries to open the display file.

Here's what you will see when you prompt the CVTDSPFCNS command:

```

= = = = =
                                Convert Display File Constants (CVTDSPFCNS)

Type choices, press Enter.

From file . . . . . Name
  Library . . . . . *LIBL Name, *LIBL, *CURLIB
From member . . . . . Name
To file . . . . . *FROMFILE Name, *FROMFILE
  Library . . . . . *LIBL Name, *LIBL, *CURLIB
To member . . . . . *FROMMBR Name, *FROMMBR
Allow replace of to member . *NO *NO, *YES
Message file . . . . . *DSPF Name, *DSPF
  Library . . . . . *DSPFLIB Name, *DSPFLIB, *CURLIB
Message identifier prefix . *CALC Name, *CALC
Convert blank constants . . *YES *YES, *NO
Convert error messages . . . *YES *YES, *NO

= = = = =

```

The help text offers further explanation of the command and each of its parameters.

To help you build the utility, I have included the CBX125 CL program. Please note the instructions in the CBX125 source header to ensure a correct compilation and assembly of the CVTDSPFCNS command.

In the event that this article has inspired you to put user indexes to work in your own applications and utilities, I have included a sample program containing all the user index API prototypes as well as code to guide you through, and demonstrate the process of, calling each individual API. You will find the sample program in the source called CBX125T.

This utility demonstrates the Add User Index Entries API (QUSADDUI):

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

The Retrieve User Index Entries API (QUSRTVUI):

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

(QCAPCMD) <http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/apis/qcapcmd.htm> Send Program Message (QMHSNDPM)

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

The sample program CBX125T further demonstrates the following User Index APIs:

The Create User Index API (QUSCRTUI):

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

The Delete User Index API (QUSDLTUI):

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

The Remove User Index Entries API (QUSRMVUI):

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

The Retrieve User Index Attributes API (QUSRTVUA):

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

You can retrieve the source code for this utility from

<http://www2.systeminetwork.com/noderesources/code/clubtechcode/UserIndex2.zip> .

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

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-user-index-apis-part-two>