Generic Message Procedures

By now, most programmers that use API's have seen the long text (QUILNGTX) API that IBM has made available to Power Systems application developers. The API is a handy one to put in the tool box, though just using the API by itself may prove to be somewhat problematic. The does return an error code data structure, but a simple up-front test might be prudent.

First, it would seem wise to incorporate some test of the job attributes before issuing a display. It never was a good idea to open a display file in the batch mode. It doesn't seem much of a better idea to use the API to display a message in the batch mode.

```
H DEBUG(*YES)
H OPTION(*SRCSTMT : *NODEBUGIO) DFTACTGRP(*NO) ACTGRP('QILE')
D dspTxtMsg       pr             3a
D   msg                      65535a   varying const options(*varsize)
D GetJobAtr       Pr             1A
D thisAtr         S              1A
 /free
  IF GetJobAtr = 'I';
     IF dspTxtMsg('Enter to continue or F12 to cancel program')='F12';
        dsply 'F12 was pressed.';
     ENDIF;
  ENDIF;
  *INLR = *ON;
  RETURN;
 /end-free
```

Fortunately, IBM has also provided an API that makes it relatively simple to test whether the program is operating in a batch mode, or interactive mode. The procedure GetJobAtr returns a single-byte to describe if the program is interactive or a batch type job. A simple check of the job attributes can allow the developer to code to use the long-text API if the job is interactive, or to send a message to a specific user, or job queue, if the application is operating in the batch mode.

The QUSRJOBI API is a part of the function to get the job attribute. In this case, the data format used for the API is JOBI0100. JOBI0100 provides the job information that denotes whether the current job is interactive, batch, or a pre-start entry.

```
P GetJobAtr       B                   export
D GetJobAtr       PI             1A

DRtvJobA          Pr                  extpgm('QUSRJOBI')
D rtv_Data                      100a
D rtv_Length                     10i 0 const
D rtv_Format                      8a
D rtv_Job                        26a
D rtv_IntJob                     16a

D p_Rcvr          S              100
D p_Format        S                8    INZ('JOBI0100')
D p_ThisJob       S               26    INZ('*')
D p_IntJob        S               16
D JobType         S                1
 /free
    RtvJoba( p_Rcvr: %len(p_Rcvr): p_Format: p_ThisJob: p_IntJob);
    JobType = %SUBST(p_Rcvr : 61 :1);
    RETURN JobType;
 /end-free

P GetJobAtr       E
```

It is convenient to know the job type when the need arises to alert someone that there is an error, or a decision to be made. In the case of this simple test program, a test is made before using the long-text API. It is dangerous to assume the API would be coded for interactive work. But in the ILE model, reusable code modules can be purpose-built to be shared across a variety of applications—common modules may be shared between interactive and batch processes.

The QUSERJOBI information can help the long-text API, at least from the perspective of making sure it is only invoked in interactive mode. However, using the format, JOBI0100 does not provide a meaningful response to the program that invoked the API, which could leave the developer scratching their head, wonder what key the user pressed.

After the text message is presented to the user, it does make sense to interrogate the user action. The long-text API presentation includes F12 on the display, so it is safe to assume a certain number of users will react to the message with F12. QUSRJOBI can help with this.

```
P dspTxtMsg       B
D dspTxtMsg       PI             3a
D  i_msg                     65535a   varying const options(*varsize)
D  i_Title                      27    options(*nopass)

D QUILNGTX        PR                   ExtPgm('QUILNGTX')
D   text                     65535a   const options(*varsize)
D   length                      10i 0 const
D   msgid                        7a   const
D   qualmsgf                    20a   const
D   errorCode                32767a   options(*varsize)

D QUSRJOBI        PR                   ExtPgm('QUSRJOBI')
D   rcvVar                   65535a   options(*varsize)
D   rcvVarLen                   10i 0 const
D   format                       8a   const
D   qualJob                     26a   const
D   intJob                      16a   const
D   errorCode                32767a   options(*varsize:*nopass)

D JOBI0600        ds                   qualified
D   F12                          1n   overlay(JOBI0600:104)

d title           s              7a   inz(*blank)
d messageF        s             20a   inz(*blank)
d titleString     s             27a   varying inz(*blank)
d tx              s              3  0
d ctrString       s             27a   inz(*blank)
d thisErr         s             16a   inz(*blank)
 /free
    title = *blanks;
    messagef = *blanks;
    titleString = %trim(i_Title);
    IF titleString <> *blank and %len(%trim(titleString)) < 27;
       tx = %div(27-%len(titlestring):2) + 1;
       %subst(ctrString:tx) = %trim(titleString);
    ELSE;
      ctrString = titleString;
    ENDIF;
    title=%subst(ctrString:1:7);
    messageF=%subst(ctrString:8:20);
    QUILNGTX(i_msg: %len(i_msg): title: messageF: thisErr);
    QUSRJOBI(JOBI0600: %size(JOBI0600): 'JOBI0600': '*': *blanks);
    if JOBI0600.F12;
       return 'F12';
    else;
       return 'ENT';
    endif;
 /end-free
P                 E
```

By adding the QUSRJOBI API and the JOBI0600 data structure, you can determine if F12 was pressed (or not). This allows a developer the option of making a decision based on whether a user wanted to continue the program action, or abandon the program action based on whether F12 was pressed, or not. Otherwise the API is merely a note to the application user—which is not necessarily a bad thing, but not as versatile as actually managing the user response. It cannot remember where I first saw this technique used—I believe in may have been Scott Klement that had an example of this on a web forum.

There is an additional consideration when using the long-text message API. The job does not necessarily reset the cancel key. In other words, if the function does not exit the program but returns to running the job after the message display, the F12 (cancel) key retains the value of '1' after the first time the text box is displayed and interrogating the return will still produce the 'F12-was pressed' message, even if the user simply pressed the ENTER key.

```
D QWCCCJOB        PR                  ExtPgm('QWCCCJOB')
D   jobVar                    65535a   options(*varsize)
D   errorCode                    16a
```

In order to reset the value of the cancel key for the job, the system API, QWCCCJOB, can be invoked. IBM provided this API specifically to reset the job keys. The API has two parameters; the list of keys to reset (exit, cancel, etc.) and the standard API error structure.

```
d resetDS         ds
d   cancelK                      10i 0 inz(2)
d                                10i 0 inz(1)
d                                10i 0 inz(1)
d                                 1a   inz('0')
d                                10i 0 inz(2)
d                                10i 0 inz(1)
d                                 1a   inz('0')
```

The data structure shows the cancel key (key ID 2) being set to a value of '0'. In free-format code, the execution of the reset is just a single instruction line.

```
QWCCCJOB(resetDS:errorDS);
```

By adding this code to the long-text message procedure, the value of F12 is reset to '0' so the test of the user response to the message represents the last action for the job key.

Combining the IBM supplied API's QUILNGTX, QUSRJOBI, and QWCCCJOB into a service program, or adding the procedures to a common service program, could produce a generic, re-useable tool for messaging that could easily be incorporated into a CL program or any other ILE application.

Steve