User Experience: LE Callable Services I Have Known

SHARE
August 2011
Session 09658

Craig Schneiderwent
cschneidpublic@yahoo.com
#include <std/disclaimer.h>

Any opinions expressed are my own and do not necessarily reflect those of my employer. You wouldn't believe the amount of red tape that one statement gets me out of.

Please don't interpret mention of a product with an endorsement thereof.

My background is in writing code, 13 languages on 5 different operating systems. I'm currently part of the IT Strategy and Architecture section at the Wisconsin DOT. Much of my day is spent consulting, answering questions that begin, "Can the mainframe do..."

What You Will See

- Code snippets from currently running applications
  - Mostly COBOL
  - CEEGMT, CEEGMTO, CEEDATM, CEECRHP, CEEGTST, CEEFRST, CEE3INF, CEESITST, CEE3DMP, CEEMSG
- Soapbox oration on error handling
- A development philosophy

We are z/OS 1.11, CICS TS 3.2, DB2 8, IMS v9. Our mainframe application portfolio consists of ~150M lines of mostly COBOL. We have exactly 0 C applications, approximately 70 program objects that contain code generated by the OS PL/I v1 and v2 compilers, and a small number of Assembler applications. Our motto is “Write COBOL until you can’t, then write Assembler until you don’t have to.” The unspoken corollary is “Don’t write PL/I, C, or Fortran.”

Some of that ~150M lines of COBOL was generated by an I-CASE tool originally known as IEF, currently answering to the name CA-Gen.

A Real-World Example

- CICS application needs a date/time formatted as per RFC 2616 (for more information, see the handouts for “User Experience: Writing a Web-enabled CICS/COBOL Program” which was presented earlier this week)
- 3 byte day of week, comma, space, day of month, space three by month abbreviation, space, four digit year, space, two digit hour, colon, two digit minute, colon, two digit second, space, time zone
- Mon, 11 Jan 2010 20:41:54 GMT
- Can I write code to do that? You bet...

...and then I thought, “Wait a minute, didn’t John Monti give a presentation on LE Callable Services at SHARE?”

A Short Philosophical Digression

- Don’t write something that’s already been written
- Don’t write something specific when you can write something generic
- Refactor and Reuse
- Temper this with YAGTNI

Something I learned from the Smalltalk development environment: always check to see if someone has already written what you are planning to write. In the context of this presentation, always check the LE Programming Reference to see if there is a callable service, or a set of callable services you can wire together, that does what you want.

If you’ve done object oriented programming you’re familiar with the idea of a class library. Many IT shops have a set of standard subroutines. It’s all the same concept, highly shareable code made available so the wheel need not be reinvented. The trick is to know where to look for reusable components and to recognize components from which you can build what you need.

While we are unlikely to be able to build our entire application from prefab parts, why not make as much use of preexisting code as we can?

YAGTNI is You Aren’t Going To Need It - or - don’t try to solve *every* problem or you’ll end up with an unmaintainable mess. Remember the Unix Way: "Write programs that do one thing and do it well."

As you can see, there is definitely a tug of war between creating a complete solution to a problem and creating a never-ending feature list that never gets implemented. If this were easy, we’d have automated it by now.

A Language Environment Solution

- CEEGMT
- CEEGMTO
- CEEDATM

I could write code to obtain the current date, break it up into its component pieces, look up the day of week in an internal table, look up the month name in an internal table, and format the whole thing according to the specification a few slides back. Or I could call a few LE Callable Services to do all that for me.

```
*    Obtain GMT time.
     CALL 'CEEGMT' USING
         GMT-LILIAN
         GMT-FLOT
         OMITTED
     END-CALL

     IF LCL-TM
*        If local time requested, obtain its offset from GMT
*        and compute local time.
         CALL 'CEEGMTO' USING
             OSET-HRS
             OSET-MIN
             OSET-SEC
             OMITTED
         END-CALL
         ADD OSET-SEC TO GMT-FLOT
     END-IF

*    Format the date according to the passed picture string.
     CALL 'CEEDATM' USING
         GMT-FLOT
         DT-PIC-STRN
         FMT-DT-TM
         OMITTED
     END-CALL
```


This is the Procedure Division of the COBOL program to solve the problem presented a couple of slides back. This program does not just solve that problem, it solves the problem of needing a date or time or both in pretty much any format you might need.

CEEGMT obtains the number of seconds from midnight 14-Oct-1582 as a 64-bit double floating point number. CEEGMTO obtains the local offset from GMT as a 64-bit double floating point number. CEEDATM takes a 64-bit double floating point number representing the date and time as the offset from midnight 14-Oct-1582 and formats it according to a picture string you provide. The LE Programming Reference documents all the different picture string options. For my purposes I used 'Www, DD Mmm YYYY HH:MI:SS GMT'.

Error Handling, Part 1

Note the OMITTED keyword in the previous slide. This instructs LE to perform its default error handling, which is to display the message corresponding to the error and then abend.

Schneiderwent's Rule: if all you're going to do with a bad LE feedback code is display a message and then abend, pass a null pointer for the feedback code parameter and let LE do its default error handling.

Caveat: check the possible LE feedback codes first.

I decided this how I wanted to handle bad feedback codes after a lot of mental running in circles. At one point, I wrote code to do exactly what LE probably does: call CEEMSG to display the message corresponding to the bad feedback code and then call CEE3ABD. Actually, at more than one point. I'm pretty sure every other example in this presentation shows why I adopted this rule.

Please note the caveat: in the case of the callable services in use in the previous slide, the errors for CEEGMT and CEEGMTO aren't something an application can deal with. CEEDATM could complain about the date format string, but pragmatically speaking that's probably hardcoded in the caller and they'll catch their error on the first call when they're testing.

Schneiderwent's rule isn't so much a rule as it is a guideline. Or maybe just good advice. Or just my opinion. It's one of those, you get to pick.

Error Handling, Part 2

If you're not going to abend in the case of a bad LE feedback code, you might want to provide as much information as possible regarding the context of the error.

- CEEMSG
- CEE3DMP

CEEMSG takes an LE feedback code and writes out the corresponding text. This saves you the trouble of translating all the possible bad feedback codes. CEE3DMP creates a core dump, just like what you'd get if you abended, but without abending. However, it isn't machine state at the time of the error, it's machine state after returning from the service that encountered the error.

A Real-World Example

- An application written with an I-CASE tool needs to do I/O to an arbitrary number of flat files, each of arbitrary LRECL and RECFM.
- Solution is LE conforming Assembler.
- A linked list of DCBs and DCBEs is maintained in an LE heap that exists below the 16M line.

If you're going to do LE conforming Assembler, be sure to look in the LE Programming Guide. There's a list of system services that **can** be used, **can but should not** be used, and **should not** be used.

The I-CASE tool in question is currently named CA-Gen. It was originally developed at Texas Instruments under the name IEF. There have been a number in intermediate names.

The solution was originally going to be 1 COBOL program per flat file. When the original developer heard that, he approached me and I proposed an Assembler solution. This was how I became familiar with writing LE conforming Assembler.

```
[...]
         IF    (CLC,HEAPID,EQ,=F'0') THEN
               CALL  CEECRHP,(HEAPID,ZERO,ZERO,                          X
               CEECRHP_OPT,LE_FC),MF=(E,CALL_LIST)
*              CREATE HEAP FOR CEEGTST
               IF    (CLC,LE_FC,NE,=3XL4'0') THEN
                     DSPL_ABND LE,=F'1'
               ENDIF
         ENDIF
         CALL  CEEGTST,(HEAPID,LISTMEMSZ,@MEM,LE_FC),MF=(E,CALL_LIST)
*              ALLOCATE STORAGE
         IF    (CLC,LE_FC,NE,=3XL4'0') THEN
               DSPL_ABND LE,=F'2'
         ENDIF
[...]
         CALL  CEEFRST,(@MEM,LE_FC),MF=(E,CALL_LIST)
*              FREE STORAGE
         IF    (CLC,LE_FC,NE,=3XL4'0') THEN
               DSPL_ABND LE,=F'3'
         ENDIF
```


We'll take a look at the DSPL_ABND macro on the next slide.

Of GETMAIN/FREEMAIN the LE Programming Guide says "Host services can, but should not, be used. Use of equivalent Language Environment storage management services is advised."

CEECRHP takes initial size to allocate, additional increment size and options as input and returns a heap ID to be used by CEEGTST. CEECRHP_OPT is initialized to F'76' which corresponds to HEAP(,,BELOW,). This is necessary because the DCB must be located below the 16M line. CEEGTST takes a heap ID and amount of storage to allocate as input and returns the address of the allocated storage. CEEFRST takes the address of storage to free and frees it.

```
          MACRO
*
*   INTERNAL MACRO TO FACTOR OUT THE DISPLAY-MESSAGE-ABEND CODE
*
          DSPL_ABND &MODE,&ABND_CD,&ACTN,&RC,&DDNAME
          AIF   ('&MODE' EQ 'LE').LE
          AIF   ('&MODE' EQ 'NOT_LE').NOT_LE
          MNOTE 12,'MODE MUST BE LE OR NOT_LE'
          MEXIT
[...]
.LE       ANOP
          CALL  CEEMSG,(LE_FC,CEEMSG_DEST_CD,CEEMSG_LE_FC),                X
                MF=(E,CALL_LIST)
          AIF   ('&ABND_CD' NE '0').ABND
          MEXIT
.ABND     ANOP
          MVC   CEE3ABD_CD,&ABND_CD           SAVE ABEND CODE
          CALL  CEE3ABD,(CEE3ABD_CD,CEE3ABD_CLEANUP),MF=(E,CALL_LIST)
          MEXIT
          MEND
```


Note that this is the moral equivalent of LE's default behavior in the event of an unexpected feedback code. The message corresponding to the feedback code is displayed and the application abends. Note that this isn't identical to LE's default behavior, as you're not getting machine state at time of error, but machine state shortly after time of error.

Also note that the ABEND macro is another system service listed as "can, but should not, be used" in the LE Programming Guide.

CEEMSG takes an LE feedback code and destination code as input and writes the message corresponding to the feedback code to the destination corresponding to the destination code. The only destination code currently supported is 2, which is the DDNAME specified in the MSGFILE LE runtime option. This is usually SYSOUT. CEE3ABD takes an abend code and a cleanup indicator. The latter can take a number of different documented values indicating what sort of dump you want, we always use 1 meaning give us a CEEDUMP. There is also a CEE3AB2 callable service that allows specification of a reason code to go with your abend code.

A Real-World Example

- Application needs to know if it is running in CICS or not
- Solution, until z/OS 1.9, was an Assembler program walking through z/OS control blocks until it found the name of the first program executed and comparing the first 3 bytes of the name to ‘DFH’
- LE now provides a better alternative

I am not making this up.

While the application only needs to know whether it’s running in CICS, in the spirit of solving not just the current problem but the next one I can see on the horizon, I did a bit more.

```
[...]
       *    Get information about our environment
            CALL 'CEE3INF' USING
                WS-SYS
                WS-ENV
                WS-LANG
                WS-LE-VERS
                CEE3INF-LEFB-CD
            END-CALL

            MOVE FST-DBL-WORD OF CEE3INF-LEFB-CD TO LE-FEEDBACK-CD
            IF CEE000
                PERFORM 0100-INIT
                PERFORM 1000-PRCS
            ELSE
                MOVE CEE3INF-LEFB-CD TO LEFB-CD
                PERFORM 9998-LE-ERR
                MOVE 8 TO RETURN-CODE
            END-IF
[...]
            PERFORM VARYING BIT-TO-TEST FROM 0 BY 1
            UNTIL BIT-TO-TEST > 31 OR WS-RC NOT = 0
              PERFORM 8010-BIT-TEST
          *   Bits are numbered from the right, subscripts from the left
              COMPUTE BIT-SUB = 32 - BIT-TO-TEST
              EVALUATE TRUE
                WHEN BIT-IS-ON
                     MOVE '1' TO SYS-BYTES(BIT-SUB)
                WHEN BIT-IS-OFF
                     MOVE '0' TO SYS-BYTES(BIT-SUB)
              END-EVALUATE
            END-PERFORM
[...]
```


CEE3INF returns 4 fullwords. The first 3 are all bit switches indicating many different things about the current environment. The last is the LE version number.

What I'm building here is a table containing x'F1' for each bit that is on, and x'F0' for each bit that is off. My reasoning is that I can't think of a caller that would be unable to interpret the result.

This piece of code with the PERFORM VARYING is illustrative but not comprehensive. The program does something similar with each of the 3 bit switch fullwords.

The CEE000 88-level comes from a copybook provided by IBM in hlq.CEE.Samplib(CEEIGZCT).

```
 8010-BIT-TEST.
*    Call LE service to test bits
     CALL 'CEESITST' USING
         FULL-WORD-TO-TEST
         BIT-TO-TEST
         CEESITST-LEFB-CD
         BIT-TEST-RSLT-SW
     END-CALL

     MOVE FST-DBL-WORD OF CEESITST-LEFB-CD TO LE-FEEDBACK-CD
     IF CEE000
         CONTINUE
     ELSE
         MOVE CEESITST-LEFB-CD TO LEFB-CD
         PERFORM 9998-LE-ERR
         MOVE 8 TO WS-RC
         STRING MYNAME
           ' unrecognized return value from CEESITST'
           DELIMITED SIZE
           INTO CEE3DMP-TITL-SPFC
         END-STRING
         PERFORM 9997-DUMP-CORE
    END-IF
    .
```


There are a number of different ways to do bit testing in COBOL, I chose to call CEESITST, the LE callable service to test bits.

CEESITST takes a fullword you want to test a bit in, a fullword indicating which bit you want to test, and returns the result in a fullword as either 1 or zero indicating the specified bit is either on or off, respectively. If you want to test bits in a single byte, you'll have to embed that byte in a fullword if you want to use CEESITST.

```
 9997-DUMP-CORE.
*    Create a core dump to assist in debugging problems
     CALL 'CEE3DMP' USING
         CEE3DMP-TITL
         CEE3DMP-OPTIONS
         CEE3DMP-LEFB-CD
     END-CALL
     .

 9998-LE-ERR.
*    Display LE message associated with feedback code
     CALL 'CEEMSG' USING
         LEFB-CD
         CEEMSG-DEST
         CEEMSG-LEFB-CD
     END-CALL
     .
```


As you can see, error handling for LE callable services usually involves more LE callable services. Once you start, you just can’t stop.

Questions?