Coding Techniques for Mixed Language Applications under LE/VSE (involving Assembler)

Table of Contents

1 Mixed Language Applications under LE/VSE Involving Assembler

1.1 Introduction

When creating or maintaining mixed language applications in Language Environment for VSE/ESA (LE/VSE) various supported techniques are available.

This summary is intended to help selecting the right implementation. It primarily focuses on how to exploit …

- LE/VSE Assembler Macros (mainly: CEEENTRY, CEETERM, CEELOAD …)
- LE/VSE Preinitialization Service (CEEPIPI)
- LE/VSE C-run-time macros (EDCPRLG and EDCEPIL)

1.2 Key scenarios

- A LE/VSE-conforming Assembler (main or sub) routine calls a High Level Language (HLL) routine such as COBOL/VSE, PLI/VSE and C/VSE. This is a typical situation where the use of CEEENTRY/TERM macros is required.
- A non-LE/VSE-conforming Assembler driver calls HLL routines. Here CEEPIPI pre-initialization is the appropriate coding method.
- A C/VSE program calls an Assembler routine bridging to another C/VSE subroutine. This Assembler routine is a good candidate for EDPRLG/EPIL macros usage.

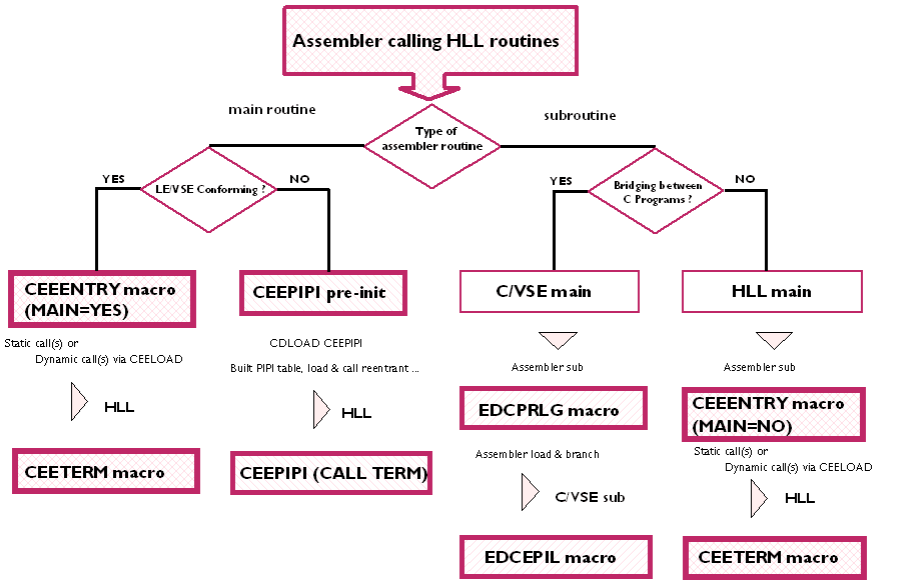

Fig. Assembler Calling High Level Language (HLL) Routines

1.3 General Usage Notes

- LE/VSE-conforming assembler macros (CEEENTRY/TERM etc.) should be used by assembler batch routines calling High Level Language (HLL) routines such as COBOL/VSE, PLI/VSE or C/VSE. These macros preserve the register and calling convent ions for the communication between LE/VSE and HLL applications. However this technique is not supported under CICS/VSE and CICS TS when MAIN=YES is used. Here EXEC CICS LINK and EXEC CICS XCTL commands should be used instead.
- CEEPIPI is a batch interface, which initializes and terminates an LE/VSE environment for specific HLL routines. Its basic use is to allow a non-LE/VSE-conforming assembler program to call HLL routines.
- C macros (EDCPRLG and EDCEPIL) are the right choice for assembler subroutine(s) called by a C/VSE main routine. EDC and CEE macros are mutually exclusive and must not be specified in the same assembler program.
- The CEELOAD macro is recommended to dynamically load LE/VSE-conforming subroutines; however there are restrictions when loading C modules. Info APAR II10024 for LE/VSE Programming Guide (SC33-6684)

will provide further information on this issue. An alternative to consider will be the CEEFETCH and CEERELES macros available since VSE/ESA 2.7.

1.4 Additional Documentation References

Further considerations about mixed language applications are made in LE/VSE Writing InterLanguage Communications Applications (SC33-6686). For additional assist in using the LE/VSE-conforming assembler macro CEEENTRY please refer to LE/VSE Programming Guide (SC33-6684) which e.g. describes capabilities such as the "RMODE=" and "BASE=' parameter settings to specify the residency mode and allow for using multiple base registers.

1.5 Examples and Further Details

1.5.1 LE/VSE-Conforming Assembler (main) Calling HLL Routines

Attached coding example shows an assembler main routine calling other LE/VSE high level language routines by setting up itself via the **CEEENTRY/TERM macros**.

Preserving register conventions (by use of these LE/VSE provided assembler macros) will be essential for communication capabilities between LE/VSE and applications running in a common run-time environment.

Further details on coding LE/VSE-conforming assembler macros are provided in LE/VSE Programming Guide (SC33-6684), Chapter 26, "Assembler Considerations".

1.5.1.1 Sample Coding, LE/VSE Conforming Assembler Main

LE/VSE-Conforming Assembler Main Routine

```
* REGISTER EQUATES
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9              BASE REGISTER
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         PRINT ON,NOGEN
*
```



```
*  PROGRAM ENTRY
*
ASMLEPRG CEEENTRY PPA=MAINPPA,AUTO=WORKSIZE,MAIN=YES,EXECOPS=YES,
X
               PARMREG=1,BASE=9
*
*  NOTE: The assembler routine is setting up itself as 'MAIN=YES'
*
         USING WORKAREA,R13
*
         CALL  CEEMOUT,(BEGMSG,DEST,0)
*

*********************************************************************
*
*        OPTION 1: Setup Static Call to LE/VSE-Conforming Subroutine
*********************************************************************
*
         CALL  CEEMOUT,(STATMSG,DEST,0)
         XR    R1,R1
*        Static call to C/VSE subroutine
STATLOAD CALL  CVSESUB
         CALL  CEEMOUT,(STATEND,DEST,0)
*
*   NOTE: HLL routines (called statically) should be compiled using
*         an LE/VSE-conforming compiler, such as COBOL/VSE, PLI/VSE
*         or C/VSE. It's object(s) must be linked together with this
*         assembler MAIN=YES routine (single phase). In this case
*         a static call to a C/VSE subroutine is performed. Here the
*         required run-time is initialized prior to the actual call
*         for a static subroutine.
*

*********************************************************************
*        OPTION 2: Setup Dynamic Call to LE/VSE-Conforming Routine
*********************************************************************
         CALL  CEEMOUT,(DYNMSG,DEST,0)
         MVC   COBVNAM,=CL8'CBLDATM'    setup name of dynamic routine
         LA    R4,HLLNAME               address parm list for LE
macro
*        Dynamically call COBOL/VSE routine
DYNLOAD  CEELOAD NAMEADDR=(4),SCOPE=ENCLAVE,MF=I
*
*  NOTE: The purpose of the CEELOAD macro is to dynamically load an
*        LE/VSE-conforming subroutine. It ensures that LE/VSE will
*        initialize the corresponding run-time environment at load-
time
*        (dependent on the signature of the loaded routine). If the
*        above dynamic load fails, LE/VSE will signal the condition
*        and reason for the failure displaying it in the programs
*        output. The use of operating system macro (CDLOAD) isn't
*        appropriate in this context
*
*********************************************************************
*        Savings and Exit
*********************************************************************
         ST    R15,LOADADDR               save load addr
         AMODESW QRY                      get current amode setting
         ST    R1,MYAMODE                 save current amode
         L     15,LOADADDR                restore load address
         BASSM 14,15                      execute routine
```



```
         L     R4,MYAMODE                 restore our amode
         AMODESW SET,AMODE=(4)            reset if required
         CALL  CEEMOUT,(ENDDYN,DEST,0)
EXIT     EQU   *
         CALL  CEEMOUT,(ENDMSG,DEST,0)
         CEETERM RC=0,MODIFIER=0
*
DEST     DC    F'2'
*
BEGMSG   DC    Y(BEGLEN)
BEGTEXT  DC    C'ASMLEPRG - LE/VSE-conforming assembler begins'
BEGLEN   EQU   *-BEGTEXT
*
ENDMSG   DC    Y(ENDLEN)
ENDTEXT  DC    C'ASMLEPRG - LE/VSE-conforming assembler complete'
ENDLEN   EQU   *-ENDTEXT
*
STATMSG  DC    Y(STATML)
STATMSGT DC    C'ASMLEPRG - Static call to C/VSE sub begins'
STATML   EQU   *-STATMSG
*
STATEND  DC    Y(STATEL)
STATENDT DC    C'ASMLEPRG - Static call to C/VSE sub complete'
STATEL   EQU   *-STATEND
*
DYNMSG   DC    Y(DYNLEN)
DYNTEXT  DC    C'ASMLEPRG - Dynamic call to COBOL/VSE sub via
CEELOAD'
DYNLEN   EQU   *-DYNTEXT
*
ENDDYN   DC    Y(ENDDYNL)
ENDDYNT  DC    C'ASMLEPRG - Dynamic call of COBOL/VSE sub completed'
ENDDYNL  EQU   *-ENDDYN
*
MYAMODE  DS    F
HLLNAME  DS    0CL10
LEPREFIX DS    H
COBVNAM  DS    CL8
LOADADDR DS    F
         LTORG
MAINPPA  CEEPPA  ,
WORKAREA DSECT
         ORG   *+CEEDSASZ
CALLMOUT CALL  ,(,,),VL,MF=L
FBCODE   DS    3F
         DS    0D
WORKSIZE EQU   *-WORKAREA
         CEEDSA  ,
         CEECAA  ,
         END ASMLEPRG
```

1.5.1.2 Sample Coding LE/VSE-Conforming HLL Subroutines

C/VSE Respectively COBOL/VSE Being Called

OPTION 1: Statically Called C/VSE Subroutine

Note:

1. This sample program is similar to member EDCDATM.C (as shipped in LE/VSE product sublibrary PRD2.SCEEBASE and documented in LE/VSE Programming Reference Manual (SC33-6685).
2. However it's slightly changed representing a subroutine!

```
/*Module/File Name: EDCDATE   */

#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceeedcct.h>

int cvsesub(void) {

   _FEEDBACK fc;
   _INT4 lil_date = 139370;   /* May 14, 1964 */
   _VSTRING date_pic,date;
   _CHAR80 date_out;

   strcpy(date_pic.string,
     "The date is Wwwwwwwwwz, Mmmmmmmmmz ZD, YYYY");
   date_pic.length = strlen(date_pic.string);

   CEEDATE(&lil_date,&date_pic,date_out,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEDATE failed with message number %d\n",
              fc.tok_msgno);
      exit(2999);
   }
   printf("%.80s\n",date_out);
}
```

OPTION 2: Dynamically Called COBOL/VSE Subroutine

Note:

1. This sample program is shipped in LE/VSE product sublibrary PRD2.SCEEBASE (member: IGZTDATM.C) and documented in LE/VSE Programming Reference Manual (SC33-6685).

```
CBL LIB,APOST
      *Module/File Name: IGZTDATM
      ************************************************
      **                                            **
      ** Function: CEEDATM - convert seconds to     **
      **                     character timestamp    **
      **                                            **
      ** In this example, a call is made to CEEDATM **
      ** to convert a date represented in Lilian    **
      ** seconds (the number of seconds since       **
      ** 00:00:00 14 October 1582) to a character   **
      ** format (such as 06/02/88 10:23:45). The    **
      ** result is displayed.                       **
      **                                            **
      ************************************************
```



```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLDATM.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  DEST          PIC S9(9) BINARY VALUE 2.
 01  SECONDS                 COMP-2.
 01  IN-DATE.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                         of IN-DATE.
 01  PICSTR.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                        of PICSTR.
 01  TIMESTP                 PIC X(80).
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.
 PARA-CBLDATM.
************************************************
** Call CEESECS to convert timestamp of 6/2/88 **
**     at 10:23:45 AM to Lilian representation **
************************************************
     MOVE 20 TO Vstring-length of IN-DATE.
     MOVE '06/02/88 10:23:45 AM'
         TO Vstring-text of IN-DATE.
     MOVE 20 TO Vstring-length of PICSTR.
     MOVE 'MM/DD/YY HH:MI:SS AP'
         TO Vstring-text of PICSTR.
     CALL 'CEESECS' USING IN-DATE, PICSTR,
                          SECONDS, FC.

************************************************
** If CEESECS runs successfully, display result**
************************************************
     IF  CEE000 of FC  THEN
         DISPLAY Vstring-text of IN-DATE
             ' is Lilian second:  ' SECONDS
     ELSE
         DISPLAY 'CEESECS failed with msg '
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
```



```
*************************************************
** Specify desired format of the output.       **
*************************************************
     MOVE 35 TO Vstring-length OF PICSTR.
     MOVE 'ZD Mmmmmmmmmmmmmmz YYYY at HH:MI:SS'
             TO Vstring-text OF PICSTR.

*************************************************
** Call CEEDATM to convert Lilian seconds to   **
**     a character timestamp                   **
*************************************************
     CALL 'CEEDATM' USING SECONDS, PICSTR,
                          TIMESTP, FC.

*************************************************
** If CEEDATM runs successfully, display result**
*************************************************
     IF CEE000 of FC  THEN
         DISPLAY 'Input seconds of ' SECONDS
             ' corresponds to: ' TIMESTP
     ELSE
         DISPLAY 'CEEDATM failed with msg '
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

1.5.2 LE/VSE-Conforming Assembler (sub) Calling HLL Routines

Attached coding example shows an assembler subroutine itself setting up for calling an LE/VSE-conforming HLL routine. This assembler routine is statically called by a COBOL/VSE main and set up via **CEEENTRY/TERM (MAIN=NO)**. Reason for coding LE/VSE-conforming assembler macros in this subroutine context is that a subsequent call to another HLL is performed. So honoring appropriate register conventions is also important here. Furthermore LE/VSE will ensure proper stack segment allocation (so called 'Next Available Byte (NAB)' support), validating stack areas pointed to by R13. Again this will be essential for communication capabilities between LE/VSE and executing application(s) in a common run-time environment.

Further details on coding LE/VSE-conforming assembler macros are provided in LE/VSE Programming Guide (SC33-6684), Chapter 26, "Assembler Considerations".

1.5.2.1 Sample Coding, LE/VSE-Conforming COBOL/VSE Main

LE/VSE-Conforming COBOL/VSE Main Routine

Note:

1. This sample program is shipped in LE/VSE product sublibrary PRD2.SCEEBASE (member: IGZTSCEN.C) and documented in LE/VSE

Programming Reference Manual (SC33-6685). It's only modification made here is the actual call to the assembler subroutine 'ASMLESUB'.

2. It is assumed that this main routine is linked together with the statically called assembler subroutine ASMLESUB (forming a single phase unit).

```
CBL LIB,APOST
      *Module/File Name: IGZTSCEN
      ************************************************
      **                                            **
      ** CBLSCEN - Call CEESCEN to set the LE       **
      **           century window                   **
      **                                            **
      ** In this example, CEESCEN is called to change **
      ** the start of the century window to 30 years  **
      ** before the system date. CEEQCEN is then      **
      ** called to query that the change made.  A     **
      ** message that this has been done is then      **
      ** displayed.                                   **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLSCEN.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  STARTCW                 PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLSCEN.
      ************************************************
      ** Specify 30 as century start, and two-digit
      **     years will be assumed to lie in the
      **     100-year window starting 30 years before
      **     the system date.
      ************************************************
           MOVE 30 TO STARTCW.

      ************************************************
      ** Call CEESCEN to change the start of the century
      **     window.
      ************************************************
           CALL 'CEESCEN' USING STARTCW, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY 'CEESCEN failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
```



```
 PARA-CBLQCEN.
*************************************************
** Call CEEQCEN to return the start of the century
**     window
*************************************************
     CALL 'CEEQCEN' USING STARTCW, FC.

*************************************************
** CEEQCEN has no non-zero feedback codes to
**     check, so just display result.
*************************************************
         DISPLAY 'The start of the century '
             'window is: ' STARTCW

*************************************************
** Call the LE/VSE-Conforming Assembler Subroutine
*************************************************
         CALL 'ASMLESUB'

     GOBACK.
```

1.5.2.2 Sample Coding, LE/VSE-Conforming Assembler Subroutine

LE/VSE-Conforming Assembler Subroutine (MAIN=NO)

```
* REGISTER EQUATES
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9              BASE REGISTER
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         PRINT ON,NOGEN
*
*  PROGRAM ENTRY
*
ASMLESUB CEEENTRY PPA=MAINPPA,AUTO=WORKSIZE,MAIN=NO,EXECOPS=YES,
X
               PARMREG=1,BASE=9
*
*  NOTE: This assembler subroutine is setting up itself via 'MAIN=NO'
*
         USING WORKAREA,R13
*
         CALL  CEEMOUT,(BEGMSG,DEST,0)
*
*
```

```
*********************************************************************
*        Setup Dynamic Call to LE/VSE-Conforming Routine
*********************************************************************
         CALL  CEEMOUT,(DYNMSG,DEST,0)
         MVC   PLINAME,=CL8'IBMDATE'    setup name of dynamic routine
         LA    R4,HLLNAME               address parm list for LE
macro
*        Dynamically call PLI/VSE routine
DYNLOAD  CEELOAD NAMEADDR=(4),SCOPE=ENCLAVE,MF=I
*
*  NOTE: The purpose of the CEELOAD macro is to dynamically load an
*        LE/VSE-conforming subroutine. It ensures that LE/VSE will
*        initialize the corresponding run-time environment at load-
time
*        (dependent on the signature of the loaded routine). If the
*        above dynamic load fails, LE/VSE will signal the condition
*        and reason for the failure displaying it in the programs
*        output. The use of operating system macro (CDLOAD) isn't
*        appropriate in this context
*
*********************************************************************
*        Savings and Exit
*********************************************************************
         ST    R15,LOADADDR               save load addr
         AMODESW QRY                      get current amode setting
         ST    R1,MYAMODE                 save current amode
         L     15,LOADADDR                restore load address
         BASSM 14,15                      execute routine
         L     R4,MYAMODE                 restore our amode
         AMODESW SET,AMODE=(4)            reset if required
         CALL  CEEMOUT,(ENDDYN,DEST,0)
EXIT     EQU   *
         CALL  CEEMOUT,(ENDMSG,DEST,0)
         CEETERM RC=0,MODIFIER=0
*
DEST     DC    F'2'
*
BEGMSG   DC    Y(BEGLEN)
BEGTEXT  DC    C'ASMLESUB - LE/VSE-conforming assembler sub begins'
BEGLEN   EQU   *-BEGTEXT
*
ENDMSG   DC    Y(ENDLEN)
ENDTEXT  DC    C'ASMLESUB - LE/VSE-conforming assembler sub complete'
ENDLEN   EQU   *-ENDTEXT
*
DYNMSG   DC    Y(DYNLEN)
DYNTEXT  DC    C'ASMLESUB - LE/VSE CEELOAD to load PLI/VSE program'
DYNLEN   EQU   *-DYNTEXT
*
ENDDYN   DC    Y(ENDDYNL)
ENDDYNT  DC    C'ASMLESUB - LE/VSE CEELOAD routine complete'
ENDDYNL  EQU   *-ENDDYN
*
MYAMODE  DS    F
HLLNAME  DS    0CL10
LEPREFIX DS    H
PLINAME  DS    CL8
LOADADDR DS    F
         LTORG
MAINPPA  CEEPPA  ,
WORKAREA DSECT
```



```
         ORG   *+CEEDSASZ
CALLMOUT CALL  ,(,,),VL,MF=L
FBCODE   DS    3F
         DS    0D
WORKSIZE EQU   *-WORKAREA
         CEEDSA  ,
         CEECAA  ,
         END ASMLESUB
```

1.5.2.3 Sample Coding, LE/VSE-Conforming PLI/VSE Routine

LE/VSE-Conforming PLI/VSE Routine Dynamically Called from Assembler

Note:

1. This sample program is shipped in LE/VSE product sublibrary PRD2.SCEEBASE (member: IBMDATE.P) and documented in LE/VSE Programming Reference Manual (SC33-6685). It's only modification made here is the actual setup via the "PROC OPTIONS(FETCHABLE)" statement to allow for being called via CEELOAD (staying in a subroutine context).
2. This routine needs to be generated as a separate phase (compile/link unit).

```
*PROCESS MACRO;
   /*Module/File name: IBMDATE                    */
   /*********************************************/
   /**                                           */
   /** Function: CEEDATE - convert Lilian date to */
   /**                     character format       */
   /**                                           */
   /** In this example, a call is made to CEEDATE */
   /** to convert a date in the Lilian format     */
   /** (the number of days since 14 October 1582) */
   /** to a date in character format. This date   */
   /** is then printed out.                       */
   /**                                           */
   /******************************************** */
   PLIDATE: PROC OPTIONS(
FETCHABLE
);

      %INCLUDE  CEEIBMAW;
      %INCLUDE  CEEIBMCT;

      DCL LILIAN  INT4 ;
      DCL PICSTR  VSTRING;
      DCL CHRDATE CHAR80 ;
      DCL 01 FC   FEEDBACK ;

      LILIAN = 152385;  /* input date in Lilian format */
      /* picture string that describes how converted   */
      /* date is to be formatted                       */
      PICSTR = 'ZD Mmmmmmmmmmmmmmz YYYY';

      /* Call CEEDATE to convert input Lilian date to  */
      /* a date in the character format specified in   */
```

```
    /* PICSTR                                              */
    CALL CEEDATE ( LILIAN , PICSTR , CHRDATE , FC );

    /* Print results if call to CEEDATE succeeds     */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
        PUT SKIP LIST( 'Lilian day ' || LILIAN
           || ' is equivalent to ' || CHRDATE );
       END;
    ELSE  DO;
       DISPLAY( 'CEEDATE failed with msg '
          || FC.MsgNo );
       STOP;
       END;

END PLIDATE;
```

1.5.3 Non-LE/VSE-Conforming Assembler Calling HLL routines

This is a useful technique for non-LE/VSE-conforming driver programs (batch environment) communicating with LE/VSE-conforming routines. CEEPIPI preinitialization does not set the driving assembler program as a 'main'. However it initializes the environment for called HLL applications. There exist different preinitialization services that can support multiple executions of main and subroutines (init_main, init_sub, init_sub_dp).

On the contrary LE/VSE-conforming assembler via use of CEEENTRY/TERM macros does not allow to start "non-LE/VSE enabled" nor will support multiple executions of HLL routines in a persistent environment (until explicitly terminated).

Further details on coding CEEPIPI are provided in LE/VSE Programming Guide (SC33-6684), Chapter 27, "Using Preinitialization Services". Sample shown below is documented there, too.

Note:

It is assumed that all routines shown here are compiled/linked into separate phase units and that the called LE/VSE-conforming subroutine (option 1, 2 or 3) is named HLLPIPI.PHASE, regardless which language was chosen for its implementation. In summary this example illustrates how to call HLLPIPI via the CEEPIPI interface.

1.5.3.1 Sample Coding, Assembler Driver for Preinitializion

Assembler Driver Preparing for a Preinitialized Environment

In the following example, the assembler program ASMPIPI ASSEMBLE invokes CEEPIPI to:

- Initialize a subroutine environment under LE/VSE
- Load and call a reentrant HLL subroutine
- Terminate the LE/VSE environment

This assembler setup is used to subsequently call HLL versions of HLLPIPI (alternatives show below are written in C, COBOL, and PL/I).

```
*COMPILATION UNIT:  LEASMPIP
*********************************************************************
**
*
*
*   Function :  CEEPIPI - Initialize the PIPI environment,
*
*                         call a PIPI HLL program, and terminate
*
*                         the environment.
*
*
*
* 1.Call CEEPIPI to initialize a subroutine environment under LE.
*
* 2.Call CEEPIPI to load and call a reentrant HLL subroutine.
*
* 3.Call CEEPIPI to terminate the LE PIPI environment.
*
*
*
* Note:  ASMPIPI is not reentrant.
*
*
*
*********************************************************************
**
*
*
=====================================================================
* Standard program entry conventions.
*
=====================================================================
ASMPIPI  CSECT
         STM   R14,R12,12(R13)    Save caller's registers
         LR    R12,R15            Get base address
         USING ASMPIPI,R12        Identify base register
         ST    R13,SAVE+4         Back-chain the save area
         LA    R15,SAVE           Get addr of this routine's save
area
         ST    R15,8(R13)         Forward-chain in caller's save area
         LR    R13,R15            R13 -> save area of this routine
*
* Load LE CEEPIPI service routine into main storage.
*
         CDLOAD CEEPIPI           Load CEEPIPI routine dynamically
         ST    R0,PPRTNPTR        Save the addr of CEEPIPI routine
*
* Initialize an LE PIPI subroutine environment.
*
INIT_ENV EQU   *
         LA    R5,PPTBL           Get address of PIPI Table
         ST    R5,@CEXPTBL        Ceexptbl-addr -> PIPI Table
         L     R15,PPRTNPTR       Get address of CEEPIPI routine
*                                 Invoke CEEPIPI routine
```



```
         CALL  (15),(INITSUB,@CEXPTBL,@SRVRTNS,RUNTMOPT,TOKEN)
*                                Check return code:
         LTR   R2,R15               Is R15 = zero?
         BZ    CSUB                 Yes (success).. go to next
section
*                                   No (failure).. issue message
         WTO   'ASMPIPI : call to CEEPIPI(INIT_SUB) failed',ROUTCDE=2
         C     R2,=F'8'             Check for partial initialization
         BE    TSUB                   Yes.. go do PIPI termination
*                                   No.. issue message & quit
         WTO   'ASMPIPI : INIT_SUB failure RC is not 8.',ROUTCDE=2
         DUMP  RC=(2)               Cancel with bad RC and dump memory
*
* Call the subroutine, which is loaded by LE
*
CSUB     EQU   *
         L     R15,PPRTNPTR         Get address of CEEPIPI routine
         CALL  (15),(CALLSUB,PTBINDEX,TOKEN,PARMPTR,
X
               SUBRETC,SUBRSNC,SUBFBC)   Invoke CEEPIPI routine
*                                Check return code:
         LTR   R2,R15               Is R15 = zero?
         BZ    TSUB                 Yes (success).. go to next
section
*                                   No (failure).. issue message &
quit
         WTO   'ASMPIPI : call to CEEPIPI(CALL_SUB) failed',ROUTCDE=2
         DUMP  RC=(2)               Cancel with bad RC and dump memory
*
* Terminate the environment.
*
TSUB     EQU   *
         L     R15,PPRTNPTR         Get address of CEEPIPI routine
         CALL  (15),(TERM,TOKEN,ENV_RC)  Invoke CEEPIPI routine
*                                Check return code:
         LTR   R2,R15               Is R15 = zero ?
         BZ    DONE                 Yes (success).. go to next
section
*                                   No (failure).. issue message &
quit
         WTO   'ASMPIPI : call to CEEPIPI(TERM) failed',ROUTCDE=2
         DUMP  RC=(2)               Cancel with bad RC and dump memory
*
* Standard exit code.
*
DONE     EQU   *
         LA    R15,0                Passed return code for system
         L     R13,SAVE+4           Get address of caller's save area
         L     R14,12(R13)          Reload caller's register 14
         LM    R0,R12,20(R13)       Reload caller's registers 0-12
         BR    R14                  Branch back to caller
*
*
*********************************************************************
* CONSTANTS and SAVE AREA.
*
*********************************************************************
SAVE     DC    18F'0'
PPRTNPTR DS    A                    Save the address of CEEPIPI routine
*
* Parameters passed to a CEEPIPI(INIT_SUB) call.
```



```
*
INITSUB  DC    F'3'              Function code to initialize for
subr
@CEXPTBL DC    A(PPTBL)          Address of PIPI Table
@SRVRTNS DC    A(0)              Addr of service-rtns vector, 0 =
none
RUNTMOPT DC    CL255' '          Fixed length string of runtime
optns
TOKEN    DS    F                 Unique value returned (output)
*
* Parameters passed to a CEEPIPI(CALL_SUB) call.
*
CALLSUB  DC    F'4'              Function code to call subroutine
PTBINDEX DC    F'0'              The row number of PIPI Table entry
PARMPTR  DC    A(0)              Pointer to @PARMLIST or zero if
none
SUBRETC  DS    F                 Subroutine return code (output)
SUBRSNC  DS    F                 Subroutine reason code (output)
SUBFBC   DS    3F                Subroutine feedback token (output)
*
* Parameters passed to a CEEPIPI(TERM) call.
*
TERM     DC    F'5'              Function code to terminate
ENV_RC   DS    F                 Environment return code (output)
*
*
*********************************************************************
* PIPI Table.
*
*********************************************************************
PPTBL    CEEXPIT  ,              PIPI Table with index
         CEEXPITY HLLPIPI,0      0 = dynamically loaded routine with
*                                    re-entrant option
         CEEXPITS ,              End of PIPI table
*
*
         LTORG
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END   ASMPIPI
```

1.5.3.2 Sample Coding, Subroutines Called from Assembler Driver

C Subroutine Called by ASMPIPI

OPTION 1

```
/*Module/File Name:  EDCPIPI  */
 /************************************************************/
 /*                                                          */
 /* HLLPIPI is called by an assembler program, ASMPIPI.      */
 /* ASMPIPI uses the LE preinitialized program               */
 /* subroutine call interface. HLLPIPI can be written        */
 /* in COBOL, C, or PL/I.                                    */
 /*                                                          */
 /************************************************************/
#include <stdio.h>
#include <string.h>
#include <time.h>
HLLPIPI ()
{
 printf ( "C subroutine beginning\n" );
 printf ( "Called using LE PIPI call\n" );
 printf ( "Subroutine interface.\n" );
 printf ( "C subroutine returns to caller\n" );
}
```

COBOL Subroutine Called by ASMPIPI

OPTION 2

```
CBL LIB,APOST
      *Module/File Name: IGZTPIPI
      ************************************************************
      *                                                          *
      * HLLPIPI is called by an assembler program, ASMPIPI.      *
      * ASMPIPI uses the LE preinitialized program               *
      * subroutine call interface. HLLPIPI can be written        *
      * in COBOL, C, or PL/I.                                    *
      *                                                          *
      ************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. HLLPIPI.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       PROCEDURE DIVISION.
           DISPLAY 'COBOL subroutine beginning'.
           DISPLAY 'Called using LE PIPI '.
           DISPLAY 'Call subroutine interface.'.
           DISPLAY 'COBOL program returns to caller.'.

           GOBACK.
```

PLI/VSE Subroutine Called by ASMPIPI

OPTION 3

```
/*Module/File Name:  IBMPIPI                                */
/************************************************************/
```



```
/*                                                          */
/* HLLPIPI is called by an assembler program, ASMPIPI.      */
/* ASMPIPI uses the LE preinitialized program               */
/* subroutine call interface. HLLPIPI can be written        */
/* in COBOL, C, or PL/I.                                    */
/*                                                          */
/************************************************************/
 HLLPIPI: PROC OPTIONS(FETCHABLE);
          DCL RESULT FIXED BIN(31,0) INIT(0);
          PUT SKIP LIST
              ('HLLPIPI  : PLI subroutine beginning.');
          PUT SKIP LIST
              ('HLLPIPI  : Called LE PIPI Call ');
          PUT SKIP LIST
              ('HLLPIPI  : Subroutine interface.   ');
          PUT SKIP LIST
              ('HLLPIPI  : PLI program returns to caller.');
          RETURN;
 END HLLPIPI;
```

1.5.4 C/VSE (main) Calling Assembler Bridge & C/VSE Subroutine

Attached coding example shows a C/VSE main routine calling an assembler routine itself setting up via C macros EDCPRLG/EDCEPIL for subsequent call of a C library function. It follows the OS linkage conventions.

Further details on coding C Macros are provided in LE/VSE C Run-Time Programming Guide (SC33-6688), Chapter 15, "Combining C and Assembler". Sample coding shown below is documented there, too.

Note: It is assumed that all routines shown in this chapter are linked together in the sequence being involved (forming a single phase unit).

1.5.4.1 Sample Coding, C/VSE Main Routine

Main C/VSE Routine Calling Assembler

```
/* EDCXGCA4 (Main C/VSE Routine)                                */
 /* The example demonstrates C/Assembler ILC.                    */
 /* Part 1 of 3 (other files involved are: EDCXGCA2, EDCXGCA5) */

#pragma linkage(callprtf, OS)

int main(void) {

callprtf();


   return(0);
}
```

1.5.4.2 Sample Coding, Bridging Assembler Subroutine

Called Assembler Subroutine (Coding EDCPRLG and EDCEPIL Macros)


```
* EDCXGCA2
* This example demonstrates C/Assembler ILC.
* Part 2 of 3 (other files involved are EDCXGCA4, EDCXGCA5)
CALLPRTF CSECT
CALLPRTF AMODE 31
CALLPRTF RMODE ANY

         EDCPRLG
         LA    1,ADDR_BLK              parameter address block in r1
         L     15,=V(@PRINTF4)         address of routine
         BALR  14,15                   call it
         EDCEPIL

ADDR_BLK DC   A(FMTSTR)                parameter address block with..
         DC   A(X'80000000'+INTVAL)    ..high bit on the last address
FMTSTR   DC   C'Sample formatting string'
         DC   C' which includes an int -- %d --'
         DC   AL1(NEWLINE,NEWLINE)
         DC   C'and two newline characters'
         DC   AL1(NULL)
*
INTVAL   DC   F'222'            The integer value displayed
*
NULL     EQU  X'00'             C NULL character
NEWLINE  EQU  X'15'             C \n character
         END
```

1.5.4.3 Sample Coding, C/VSE Subroutine Called by Previous …

C/VSE Subroutine Called by Assembler Bridge

```
 /* EDCXGCA5 (C/VSE Routine called by previous)               */
 /* This example demonstrates C/Assembler ILC.               */
 /* Part 3 of 3 (other files involved are EDCXGCA2, EDCXGCA4) */
 /*********************************************************
 * This routine is an interface between assembler code       *
 * and the LE/VSE C Run-Time library function printf().  OS  *
 * linkage will not tolerate variable length parameter       *
 * lists, so this routine is specific to a formatting string *
 * and a single 4-byte substitution parameter.  It is        *
 * specified as an int here.                                 *
 *********************************************************/

#pragma linkage(_printf4,OS) /*function will be called from
assembler*/

#include <stdio.h>

int _printf4(char *str,int i) {

   return printf(str,i);  /* call LE/VSE C Run-Time library function
*/

}
```


2 Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

CICS, IBM, Language Environment, VSE/ESA, z/VSE

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names, may be the trademarks or service marks of others.

3 Comments and Questions

Comments or questions on this documentation are welcome. Please send your comments to:

vseesa@de.ibm.com

