Language Environment

What is the Language Environment?

LE, or Language Enviroment, is a common run-time that provides services for Language Environment-conforming programs written in IBM C, C++, COBOL, Pl/I, Fortran, and assembler. Building mixed-language applications is easier in LE because there is a consistent environment for all languages. LE services include message handling, condition handling, and storage management. There are also common library of routines for math, time, date.

Assembler is the only language on z/OS that provides the option of being LE-conforming or not. Programs written in C, C++, COBOL, PL/i, and Fortran are automatically LE-conforming. For languages like COBOL with little or no condition handling function, LE provides a user-controlled method for robust error recovery. Additionally, different language routines can call each other without the need for initialization and termination of language specific run-time environments for each call. LE also provides a common dump for all conforming languages.

LE Terminology

There are three terms at the core of the LE program management model:

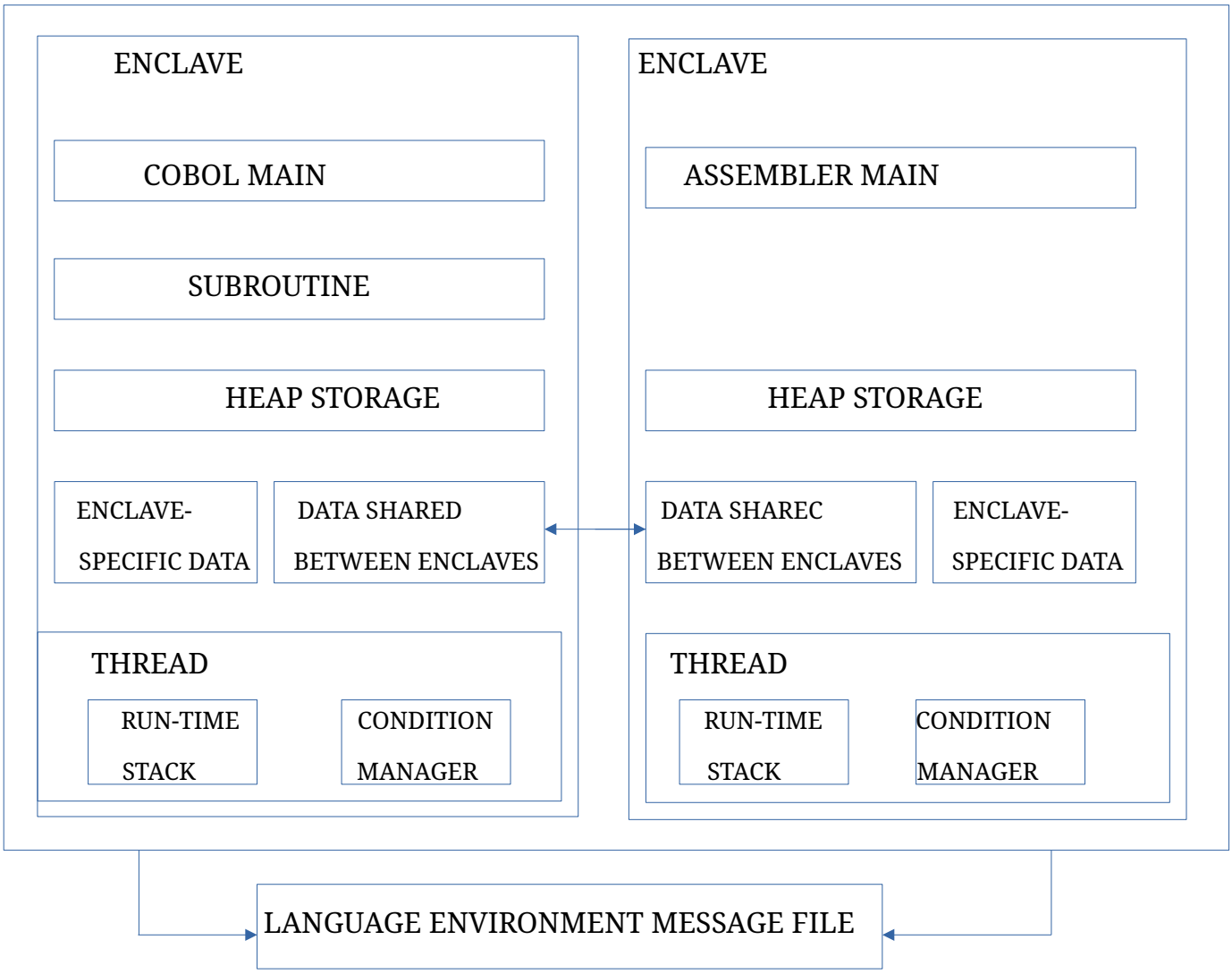
1) **Thread** – this is the basic run-time path within LE that is dispatched by the system with its own run-time stack, instruction, and registers.

2) **Enclave** – A collection of routines, one of which is named as the main routine. The enclave contains at least one thread.

3) **Process** – The highest level in the LE program managent model. It consists of resouces, program code, data, and at least one enclave.

A diagram on the next page illustrates the relationship of these terms within the environment.

PROCESS



Processes

1) A process consists of at least one enclave and is logically separate from othe processes.

2) Processes do not share storage.

3) Processes are not hierarchically related.

4) Message files are shared across enclaves.

5) Processes can create new processes and communicate with each other.

Enclaves

1) A collection of routines that make up an application.
2) A run unit in COBOL.
3) A main procedure and all of its subroutines in PL/I
4) A main C function and its subroutines.

Threads

1) The basic instance of a particular routine.
2) Has its own run-time stack.
3) Has an instruction counter, registers, and condition-handling mechanisms.
4) A thread can create a new enclave.

LE Storage Management Model

Common storage management services are provided all LE conforming languages. LE controls run-time stack and heap storage .

A run-time stack is automatically created when a thread is created, and it is freed when the thread terminates. Stacks can be tuned for better performance. Heap storage is allocated by a thread in no particular order. Heap storage is shared among all program units and all threads in an enclave.

LE Callable Services

Services are divided into the following groups:

- Communicating Conditions Services
- Condition Handling Services
- Date and Time Services
- Dynamic Storage Services
- General Callable Services
- Initialization/Termination Services
- Locale Callable Services
- Math Services
- Message Handling Services
- National Language Support Services

An Example of Invoking Callable Services in COBOL

```
01 Feedback.

COPY CEEIGZCT .

.
.
.
CALL "CEESERV" USING parm1 parm2 ... parmn fc
```

CEEIGZCT is a copybook of LE feedback codes. The feedback code, fc, can be omitted.

An Example of Invoking Callable Services in Assembler

Here is IBM's example of invoking a callable service:

```
         LA    R1,PLIST
         L     R15,=V(CEESERV)
         BASR  R14,R15
         CLC   FC(12),CEE000     CHECK IF FC IS ZERO
         BNE   ER1               ERROR IF NOT
...
PLIST    DS    0D                PARM LIST (VARIABLE SIZE)
         DC     A(PARM1)
         ...
         DC     A(FC+X'80000000')    FC LAST PARM, SO ADD A 1 IN HIGH ORDER BIT
PARM1    DC     F'5'              THE PARMS
         ...
FC       DS    12C
CEE000   DC
```

IBM’S Example of a simple main assembler program.

```
* ===================================================================
*
* Shows a simple main assembler routine that brings up the environment,
* returns with a return code of 0, modifier of 0, and prints a
* message in the main routine.
*
* ===================================================================
MAIN     AMODE    31
MAIN     RMODE    ANY
MAIN     CEEENTRY PPA=MAINPPA
*
*
         LA       1,PARMLIST
         L        15,=V(CEEMOUT)
         BASR     14,15
*
*   Terminate the Language Environment environment and return to the caller
*
         CEETERM  RC=0,MODIFIER=0
* ===================================================================
*              CONSTANTS AND WORKAREAS
* ===================================================================
PARMLIST DC    AL4(STRING)
         DC    AL4(DEST)
         DC    X’80000000’        Omitted feedback code
*
STRING   DC    AL2(STRLEN)
STRBEGIN DC    CL19’In the main routine’
STRLEN   EQU   *-STRBEGIN
DEST     DC    F’2’
         MAINPPA  CEEPPA          Constants describing the code block
         CEEDSA                   Mapping of the dynamic save area
         CEECAA                   Mapping of the common anchor area
         END   MAIN               Nominate MAIN as the entry point
```

Explanation

Assembler programs that run in LE must be reentrant. The CEENTRY and CEEPPA macros combine to generate a Program Prolog Area (PPA). CEENTRY generates reentrant code. The name of the PPA is MAINPPA.

- Register 2 cannot be used as the base register for the module.
- The caller’s registers (14 – 12) are saved in the DSA provided by the caller.
- The base register is set by default or can be specified by parameter.
- R12 is set with the address of the CEECAA (Common Anchor Area)
- R13 is set with the address of CEEDSA (Dynamic Save Area).
- PARMREG (Register 1 by default) is set based on PLIST

The program branches to CEEMOUT to print a message. The CEETERM macro is invoked to to terminate and return from an LE routine.

A More Fully-Developed Program

What follows are LE COBOL and Assembler programs. The assembler program invokes LE Callable services. It also calls the COBOL program. An explanation of the program follows this listing.

```
//KC02486A JOB (KC024861),'WOOLBRIGHT',REGION=0M,CLASS=A,MSGCLASS=H,
// NOTIFY=KC02486,MSGLEVEL=(1,1),TIME=(0,1)
//STEP0     EXEC IGYWCL,REGION=250M,
//  PARM.COBOL='TEST,NORENT,APOST,OBJECT,NODYNAM,LIST,NOTHREAD'
//COBOL.STEPLIB DD DSN=IGY630.SIGYCOMP,DISP=SHR
//COBOL.SYSLMOD   DD DSN=KC02486.ASM.LOAD(COBSUB),DISP=SHR
//COBOL.SYSPRINT  DD SYSOUT=*
//COBOL.SYSIN     DD *
       IDENTIFICATION DIVISION.
       PROGRAM-ID. COBSUB.
      *
       ENVIRONMENT DIVISION.
       DATA DIVISION.
        WORKING-STORAGE SECTION.
        01 Z    PIC S9(5) COMP-3 VALUE 0.
     *
      LINKAGE SECTION.
        01 X    PIC S9(5) COMP-3.
        01 Y    PIC S9(5) COMP-3.
        01 ZOUT PIC +99,999.
     *
      PROCEDURE DIVISION USING BY REFERENCE X, Y, ZOUT.
          DISPLAY "HI I MADE IT TO THE SUBROUTINE"
          DISPLAY X
          DISPLAY Y
          COMPUTE Z = X + Y
          MOVE Z TO ZOUT
          GOBACK
          .
/*
//LKED.SYSLMOD DD DSN=KC02486.ASM.LOAD(COBSUB),DISP=SHR
//STEP1    EXEC PROC=HLASMC
//SYSIN   DD *
         YREGS
         PRINT ON,NODATA,NOGEN
* ENTER THE PROGRAM, SET UP THE PROLOGUE:
MAIN1    CEEENTRY PPA=MAINPPA,BASE=(11),AUTO=WORKSIZE
         USING WORKAREA,R13
         PRINT ON,NODATA,NOGEN
*
* CALL FOR THE INCOMING PARM FROM THE OS
*
```

```
         CALL  CEE3PRM,(MSG,FBCODE),VL,MF=(E,CALLCOB)
*
* CALL TO PRINT A MESSAGE
*
         MVC   MSGL,=H'80'
         CALL  CEEMOUT,(MSGAREA,DEST,FBCODE),VL,MF=(E,CALLMOUT)
         MVI   ZREC,C' '               CLEAR THE BUFFER FOR PRINTING
         MVC   ZREC+1(L'ZREC-1),ZREC
         MVC   ZLAB,=CL6'SUM = '
         MVC   ZLEN,=H'80'             SET THE STRING LENGTH
*
* CALL THE COBOL SUBPROGRAM
*
* NOT USING PLIST, SO ZERO R1
         SR    R1,R1
         CALL  COBSUB,(XPK,YPK,ZOUT),VL,MF=(E,CALLCOB)
*
* CALL TO PRINT THE RESULTS IN ZREC
*
         CALL  CEEMOUT,(ZREC,DEST,FBCODE),VL,MF=(E,CALLMOUT)
*
* EXIT THE PROGRAM
*
         CEETERM  RC=0,MODIFIER=0
* ================================
* CONSTANTS
* ================================
DEST     DC    F'2'                DESTINATION CODE FOR MESSAGES
PGMNAME  DC    CL8'COBSUB'         PROGRAM NAME FOR CALL
XPK      DC    PL3'10'             LOCAL DATA TO PASS
YPK      DC    PL3'7'              LOCAL DATA TO PASS
         LTORG
*
* SET UP THE PPA
*
MAINPPA  CEEPPA  ,                 PROGRAM PROLOG AREA
* ================================
* WORK AREA AND DSA
* ================================
WORKAREA DSECT
         ORG   *+CEEDSASZ        DSA HAS A FIXED PART UP FRONT
CALLCOB  CALL  ,(,),VL,MF=L      2-ARG PARM LIST
CALLMOUT CALL  ,(,,),VL,MF=L     3-ARG PARM LIST
FBCODE   DS   3F                 1-BYTE FEEDBACK CODE IN LE

*******************
* OUTPUT PRINT BUFFER
*******************
               DS   0H
ZREC     DS   0CL82
ZLEN     DS    H
ZLAB     DS    CL6
ZOUT     DS    CL7                DATA THAT IS RETURNED
               DS    CL67
*******************
*TOKEN   DS    CL4
         DS   0H
```

```
MSGAREA  DS   0CL82
MSGL     DS    H                  MESSAGE LENGTH
MSG      DS    CL80               THE MESSAGE AREA
         DS   0H
WORKSIZE EQU   *-WORKAREA
         CEEDSA  ,                MAPPING OF THE DYNAMIC SAVE AREA
         CEECAA  ,                MAPPING OF THE COMMON ANCHOR AREA
         END   MAIN1              NOMINATE MAIN AS THE ENTRY POINT
/*
//C.SYSLIB  DD
//          DD DSN=CEE.SCEEMAC,DISP=SHR
//LKED     EXEC  PGM=HEWL,PARM='MAP,XREF,LIST,CALL'
//SYSPRINT DD    SYSOUT=A
//SYSLMOD  DD    DSNAME=KC02486.ASM.LOAD,DISP=SHR
//SYSLIB   DD    DSNAME=CEE.SCEELKED,DISP=SHR
//SYSLIN   DD  DSN=&&OBJ,DISP=(OLD,DELETE)
//         DD    DSNAME=KC02486.ASM.LOAD(COBSUB),DISP=SHR
//         DD    DDNAME=SYSIN
//SYSIN    DD    *
     NAME  MAIN1(R)
/*
//RUNIT    EXEC PGM=MAIN1,PARM='THIS IS A PASSED MESSAGE'
//STEPLIB  DD  DSN=KC02486.ASM.LOAD,DISP=SHR
//SYSUDUMP DD  SYSOUT=*,OUTLIM=5000
//SYSABOUT DD  SYSOUT=*,OUTLIM=5000
//SYSDBOUT DD  SYSOUT=*,OUTLIM=5000
//SYSOUT   DD  SYSOUT=*
//FILEIN   DD  *
THIS IS RECORD 1
THIS IS RECORD 2
THIS IS RECORD 3
THIS IS RECORD 4
/*
FILEOUT DD SYSOUT=*
//
```

The above example illustrates a more fully-developed assembler program that calls a COBOL subroutine. The COBOL routine is defined first. That routine expects, three parameters, X, Y and Z. X and Y are two packed-decimal fields, and Z is a numeric edited field that will contain the result, X + Y. This is a standard-looking COBOL program. Nothing special is coded in order for it to run in LE.

The assembler program which follows it begins by invoking CCEENTRY:

```
MAIN1    CEEENTRY PPA=MAINPPA,BASE=(11),AUTO=WORKSIZE
```

The entry macro names the PPA, establishes R11 as the base register, and indicates the number of bytes needed for any automatic variables (non-static variables) plus the DSA needed for the prolog.

WORKAREA is the DSECT that defines the variables that will be dynamically allocated for each user. This includes space up front for the DSA, list versions of two CALL

macros, a twelve-byte feedback code, an output buffer, and a message area. Static variables are defined below the executable code and above the WORKAREA DSECT.

The program then invokes the callable service CEE3PRM to read the JCL parameter that was passed to the program by this line:

```
//RUNIT    EXEC PGM=MAIN1,PARM='THIS IS A PASSED MESSAGE'
```

The parm is read with this call:

```
         CALL  CEE3PRM,(MSG,FBCODE),VL,MF=(E,CALLCOB)
```

The program prints a message:

```
         CALL  CEEMOUT,(MSGAREA,DEST,FBCODE),VL,MF=(E,CALLMOUT)
```

Calls the COBOL program:

```
         CALL  COBSUB,(XPK,YPK,ZOUT),VL,MF=(E,CALLCOB)
```

Prints the results:

```
         CALL  CEEMOUT,(ZREC,DEST,FBCODE),VL,MF=(E,CALLMOUT)
```

And terminates:

```
         CEETERM  RC=0,MODIFIER=0
```

Since the program is reentrant, the CALLs are qualified with references to the executable versions of these macros. The list versions of the macros are defined in dynamic storage.

To learn more about LE Callable Services consult the **z/OS Language Environment Programming Reference**. Other helpful manuals include,

z/OS Language Environment Concepts Guide
z/OS Language Environment Debugging Guide
z/OS Language Enviroment Programming Guide for 64-bit Virtual Addressing Mode
z/OS Language Enviroment Runtime Messages