



Migrating to Language Environment-Conforming Assembler, If You Need To..



John Monti
IBM Poughkeepsie
jmonti@us.ibm.com





Agenda

- Definitions
- Language Environment-Enabled Assembler
- Language Environment-Conforming Assembler
- To Conform or Not to Conform
- Language Environment-Conforming Macros
- More About CEEENTRY Macro
- Language Environment-Conforming Conventions
- A Language Environment-Conforming Assembler Routine
- Invoking Callable Services
- Migrating to Language Environment-Conforming Assembler
- Service Considerations



Definitions

- **Language Environment-Enabled:**

- Routine that can run with Language Environment run-time, and may also run with previous run-times. Cannot make use of Language Environment callable services.

- **Language Environment-Conforming:**

- Routine that can run only with the Language Environment run-time library. Can make use of Language Environment callable services.



Language Environment-Enabled Assembler

- All routines run in Language Environment must at least be Language Environment-enabled.
 - Allows for Language Environment error recovery.
 - Routines must allocate their own register save area.
- Language Environment-enabled routines observe standard OS linkage, plus the following conventions:
 - R13 contains address of executing routine's register save area
 - Register save area back chain is set to valid 31-bit address
 - First two bytes of register save area must be hex zeros



Language Environment- Enabled Assembler

- Language Environment always handles register save area (R13) address as a 4 byte address whether it's a 24-bit or 31-bit address.
 - The first byte of a 24-bit address must be zeroed out!
 - For example, AMODE 24 Language Environment-enabled assembler routine:

```
    ...  
    BAL    13, JUMP          store RSA in R13  
SAVE     DS    18F          register save area  
JUMP     0H  
    LA    13, 0(, 13)      clear high order byte  
    ...
```



Language Environment- Conforming Assembler

- Must adhere to Language Environment common interface conventions, such as:
 - Language Environment-conforming entry points
 - Language Environment-conforming register usage
 - Language Environment-conforming linkage
- Language Environment provides macros that generate code following the required register conventions for Language Environment-conforming assembler routines.



To Conform or Not to Conform

- **For our sake, we can classify assembler applications into three categories:**
 1. Assembler routine that calls assembler routine(s)
 2. Assembler routine that calls Language Environment HLL routine(s)
 3. Language Environment HLL routine that calls assembler routine(s)



To Conform or Not to Conform

1. **Assembler routine that calls assembler routine(s)**
 - Language Environment-conforming assembler is required if a routine exploits Language Environment callable services.
 - Language Environment-conforming assembler is recommended if routine makes more than one call to Language Environment-conforming assembler routines.
 - Minimize Language Environment initialization / termination overhead



To Conform or Not to Conform

2. **Assembler routine calls Language Environment HLL routine(s)**
 - Language Environment-conforming assembler required if assembler exploits Language Environment callable services.
 - Language Environment-conforming assembler recommended if assembler makes more than one call to Language Environment HLL routines.
 - Minimize Language Environment initialization / termination overhead



To Conform or Not to Conform

3. **Language Environment HLL routine calls assembler routine(s)**
 - Language Environment-conforming assembler required if assembler routine exploits Language Environment callable services.
 - Number of calls to assembler routines doesn't affect Language Environment initialization / termination overhead.
 - In other words, assembler can be Language Environment-enabled without overhead considerations.
 - Let's say HLL routine calls assembler routine which calls another HLL routine. Still no overhead considerations if assembler is Language Environment-enabled!



To Conform or Not to Conform

Quick Reference – Do you need to migrate to Language Environment-Conforming assembler?

If your assembler “driver” calls...	then you...
zero Language Environment services,	don't need to conform.
one or more Language Environment services,	<u>must conform.</u>
zero or one Language Environment HLL or Language Environment-conforming ASM routine,	don't need to conform.
more than one Language Environment HLL or Language Environment-conforming ASM routines,	<u>should conform.</u>



Language Environment- Conforming Macros

- The following assembler macros generate Language Environment-conforming code:
 - **CEEENTRY** – generates Language Environment-conforming prolog
 - **CEETERM** – generates Language Environment-conforming epilog
 - **CEECAA** – generates CAA mapping
 - **CEEDSA** – generates DSA mapping
 - **CEEPPA** – generates a PPA
 - **CEELOAD** – dynamic load of Language Environment-conforming routine
 - **CEEFETCH** – dynamic load of Language Environment-conforming routine
 - **CEERELES** – dynamic delete of Language Environment-conforming routine
 - **CEEPCALL** – call Language Environment-conforming routine
 - **CEEPDDA** – define a data item in WSA
 - **CEEPLDA** – return the address of data item defined by CEEPDDA



More About CEEENTRY Macro

- **CEEENTRY Macro:**
 - provides a Language Environment-conforming prolog
 - generates code in cooperation with CEEPPA macro
 - generates reentrant code
 - must be used in conjunction with CEETERM, CEECAA, CEEDSA, and CEEPPA
 - assumes that registers adhere to certain conventions (discussed in a few more slides)



More About CEEENTRY Macro

○ CEEENTRY Parameters:

- **name** entry name (and CSECT if this is first call to CEEENTRY)
- **PPA=** label of corresponding PPA generated using CEEPPA macro. if unspecified uses “PPA”
- **AUTO=** total number of bytes (used by prolog) for DSA and auto data (default is CEEDSASZ – no auto data)
- **MAIN=YES** indicates Language Environment should be brought up. designates routine as main routine in enclave. YES is default.
- **MAIN=NO** indicates this is subroutine in enclave. should be specified when Language Environment already active (only prolog code needed).
- **BASE=** establishes base register(s) for this module. register 11 assumed if none specified.



More About CEEENTRY Macro

- **CEEENTRY Parameters (continued):**
 - **EXECOPS=YES** indicates that main routine honors run-time options on inbound parameter string. applicable only when MAIN=YES in effect. YES is default.
 - **EXECOPS=NO** indicates no run-time options honored on inbound parameter string. Language Environment will not attempt to process run-time options from inbound parameter string.
 - **PARMREG=** specifies register to hold inbound parameters. register 1 is default.
 - **PLIST=** indicates that main routines are to honor PLIST format on inbound parameter string. applicable only when MAIN=YES in effect. ignored if MAIN=NO specified. HOST format assumed if value unspecified.



More About CEEENTRY Macro

- **CEEENTRY Parameters (continued):**
 - **NAB=YES** indicates previous DSA has NAB. YES is default. use if always called by Language Environment-conforming assembler.
 - **NAB=NO** indicates previous DSA may not contain NAB. code generated to find NAB. use if called by non-Language Environment-conforming assembler.
 - **EXPORT=** indicates whether this entry point is exported.
 - **NO** entry point can only be called from other routines link-edited into same program object.
 - **YES** entry point marked as exported DLL function.
 - **ENCLAVE=YES** indicates Language Environment should create nested enclave. requires MAIN=YES.
 - **ENCLAVE=NO** indicates new enclave not needed. NO is default.



More About CEEENTRY Macro

- CEEENTRY automatically sets module to AMODE ANY and RMODE ANY.
 - When migrating, if macros were coded using AMODE 24, they should be changed to AMODE 31. If not possible, module must be set to RMODE 24 for link-edit.
- When CEEENTRY macro is invoked more than once in an assembly, the programmer is responsible for coding DROP statements for base registers set up by previous invocation.



Language Environment-Conforming Assembler Conventions

- **On entry...**

- to Language Environment-conforming assembler main routine, or subroutine with NAB=NO, registers must contain:

R0	Reserved
R1	Address of parameter list, or zero
R13	Caller's register save area
R14	Return address
R15	Entry point address

- These are passed without change to the CEEENTRY macro.
- NAB=NO on CEEENTRY macro means routine is called by non-Language Environment-conforming routine.



Language Environment-Conforming Assembler Conventions

- **On entry...**

- to Language Environment-conforming assembler subroutine with NAB=YES, registers must contain:

R0	Reserved
R1	Address of parameter list, or zero
R12	Common anchor area (CAA) address
R13	Caller's DSA
R14	Return address
R15	Entry point address
All Others	Undefined

- NAB=YES on CEEENTRY macro means routine is called by Language Environment-conforming routine.



Language Environment-Conforming Assembler Conventions

- **On entry...**

- to a Language Environment-conforming assembler routine, CEEENTRY:
 - Loads caller's registers (R14 through R12) into DSA (or register save area) provided by caller
 - Kicks off initialization if Language Environment if MAIN=YES
 - Allocates DSA (which sets NAB field in new DSA) and sets first halfword of DSA to hex zero and sets backchain



Language Environment-Conforming Assembler Conventions

- **At all times..**

- while Language Environment-conforming assembler routine is running, R13 must point to routine's DSA

- **At call points..**

- R12 must contain address of CAA, except when:
 - calling a COBOL program
 - calling an assembler routine that's not Language Environment-conforming
 - calling a Language Environment-conforming assembler routine that specifies NAB=NO on CEEENTRY macro



Language Environment-Conforming Assembler Conventions

- **On exit...**

- from Language Environment-conforming assembler routine, registers contain:

R0	Undefined
R1	Undefined
R15	Undefined
All Others	The contents they had upon entry



A Language Environment- Conforming Assembler Routine

- Using CEEENTRY, CEETERM, CEEPPA, CEEDSA, and CEECAA:

```
MAIN      CEEENTRY PPA=MAINPPA
          ... Your code ...
          CEETERM  RC=0,MODIFIER=0
          ... constants, definitions, etc. ...

MAINPPA   CEEPPA
          CEEDSA
          CEECAA
          END      MAIN
```



Invoking Callable Services

- Calling fictitious CEESERV callable service using a feedback code:

```
...  
LA    R1,PLIST           set up parm list  
L     R15,=V(CEESERV)   set up entry point  
BALR  R14,R15           branch to service  
CLC   FC(12),CEE000     optional fc check  
BNE   ER1               branch to error path  
...
```




Invoking Callable Services

- Parameter definitions for CEESERV. Services can be called with feedback code (usually as last parameter):
 - Can use CEEBALCT (SCEESAMP) to define feedback codes

```
...
PLIST    DS    0D
         DC    A(PARM1)                parm 1
         ...                               parms 2 through n
         DC    A(FC+X'80000000')      fc is last parm
PARM1    DC    F'5'                    parm 1
         ...                               parms 2 through n
FC       DS    12C                      fc is last parm
CEE000   DC    12X'00'                 good fc
...
```



Invoking Callable Services

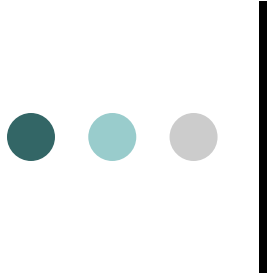
- Services can also be called without feedback code (signaling condition on failure, potential ABENDING if unhandled):

```
      ...
PLIST  DS    0D
       DC    A(PARM1)           parm 1
       ...                     parms 2 through n
       DC    A(X'80000000')     omitting fc parm
PARM1  DC    F'5'              parm 1
       ...                     parms 2 through n
```



Migrating to Language Environment- Conforming Assembler

- In the accompanying sample we will migrate a Language Environment-enabled assembler routine that repeatedly calls a COBOL routine.
 - Notice Language Environment-enabled assembler routine's performance hit
 - Language Environment initialization / termination overhead
 - We'll also call a Language Environment callable service from the migrated routine.



Migrating to Language Environment- Conforming Assembler

```
ASMENBL  CSECT
ASMENBL  AMODE 31
ASMENBL  RMODE ANY
*
*      GENERATE LANGUAGE ENVIRONMENT-ENABLED PROLOG
*
          STM      14,12,12(13)  LOAD CALLER'S REGS IN CALLER'S RSA
          LR       11,15        MOVE BASE REG TO R11
          USING   ASMENBL,11    SET R11 AS BASE REG
          LR       10,13        BACKUP CALLER'S RSA
          LA      13,SAVEAREA   SET R13 TO OUR RSA
          ST      10,4(13)      SET BACKCHAIN IN OUR RSA
*          ST      13,8(10)     SET CALLER'S FWD CHAIN TO OUR RSA
```



Migrating to Language Environment-Conforming Assembler

```
*
*      ISSUE MESSAGE
*
*           WTO          ' IN LE-ENABLED ASM ROUTINE '
*
*      LOOP AND CALL COBOL ROUTINE SEVERAL TIMES
*
*           L           2, LOOPNUM
*           SR          1, 1
LOOP      L           15, =V(CBLTST)
*           BALR        14, 15
*           BCT         2, LOOP
*
*
```



Migrating to Language Environment- Conforming Assembler

```
*      GENERATE LANGUAGE ENVIRONMENT-ENABLED EPILOG
*
      LR      13,10      REESTABLISH CALLER'S RSA REG AND
      LM      14,12,12(13)      CALLER'S OTHER REGS
      LA      15,0      ZERO OUT R15
      BR      14      BRANCH BACK TO CALLER
*
*
=====
*      CONSTANTS AND WORKAREAS
*
=====
*
LOOPNUM  DC      F'99'
SAVEAREA DS      18F'0'      DYNAMIC SAVE AREA
                END      ASMENBL      NOMINATE ASMENBL1 AS ENTRY POINT
```



Migrating to Language Environment- Conforming Assembler

- CPU time spent when Language Environment-enabled assembler calls COBOL routine 99 times..

```
IEF403I RUNASM - STARTED - TIME=11.31.44
```

```
-----  
-                               REGION          --- STEP TIMINGS ---  
- STEPNAME PROCSTEP PGMNAME      CC      USED      CPU TIME  ELAPSED TIME  
- STEP1              ASMENBL      00      92K      00:00:00.41  00:00:08.89
```

```
IEF404I RUNASM - ENDED - TIME=11.31.53
```

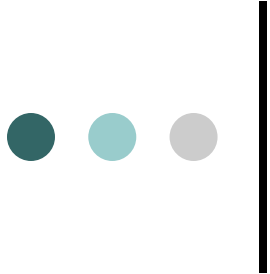
```
-----  
- NAME-BRYNTCO              TOTALS: CPU TIME= 00:00:00.41  
-----
```

```
$HASP395 RUNASM ENDED
```



Migrating to Language Environment- Conforming Assembler

```
*      INITIALIZE LANGUAGE ENVIRONMENT AND GENERATE LANGUAGE  
*      ENVIRONMENT-CONFORMING PROLOG  
*  
ASMCONFM CEEENTRY PPA=MAINPPA  
*
```

Migrating to Language Environment- Conforming Assembler

```
*      INVOKE CEEMOUT CALLABLE SERVICE TO ISSUE MESSAGE
*
          LA          1,PARMLIST
          L           15,=V(CEEMOUT)
          BALR       14,15
*      CLC          FC(12),CEE000      OPTION TO CHECK FEEDBACK CODE
*      BNE          ER1                BRANCH TO ERROR PATH IF FC !0
*
*      LOOP AND CALL COBOL ROUTINE SEVERAL TIMES
*
          L           2,LOOPNUM
          SR          1,1
LOOP     L           15,=V(CBLTST)
          BALR       14,15
          BCT        2,LOOP
*

```



Migrating to Language Environment- Conforming Assembler

```
*      GENERATE LANGUAGE ENVIRONMENT-CONFORMING EPILOG AND  
*      TERMINATE LANGUAGE ENVIRONMENT  
*  
          CEETERM  RC=0,MODIFIER=0  
ER1      CEETERM  RC=12,MODIFIER=0  
*
```



Migrating to Language Environment- Conforming Assembler

```
* =====  
*                CONSTANTS AND WORKAREAS  
* =====  
*  
PARMLIST DC    AL4 (STRING)  
          DC    AL4 (DEST)  
          DC    X'80000000'          OMITTED FEEDBACK CODE  
*          DC    A (FC+X'80000000')  OPTION TO USE FEEDBACK CODE PARM  
*  
LOOPNUM  DC    F'99'  
STRING   DC    AL2 (STRLEN)  
STRBEGIN DC    CL28 'IN LE-CONFORMING ASM ROUTINE'  
STRLEN   EQU   *-STRBEGIN  
DEST     DC    F'2'  
CEE000   DC    12X'00'              GOOD FEEDBACK CODE  
FC        DS    12C                 OPTION TO USE FC FOR CALLABLE SERVICE  
MAINPPA  CEEPPA                      CONSTANTS DESCRIBING CODE BLOCK  
          CEEDSA                      MAPPING OF THE DYNAMIC SAVE AREA  
          CEECAA                      MAPPING OF THE COMMON ANCHOR AREA  
          END    ASMCONFM            NOMINATE ASMCNMN1 AS ENTRY POINT
```



Migrating to Language Environment- Conforming Assembler

- CPU time spent when Language Environment-conforming assembler calls COBOL routine 99 times!

```
IEF403I RUNASM - STARTED - TIME=11.43.44
```

```
-----  
-                               REGION                --- STEP TIMINGS ---  
- STEPNAME  PROCSTEP  PGMNAME      CC      USED      CPU TIME  ELAPSED TIME  
- STEP1                ASMCONFM    00      92K      00:00:00.01  00:00:00.21
```

```
IEF404I RUNASM - ENDED - TIME=11.43.44
```

```
-----  
- NAME-BRYNTCO                TOTALS: CPU TIME= 00:00:00.01  
-----
```

```
$HASP395 RUNASM  ENDED
```



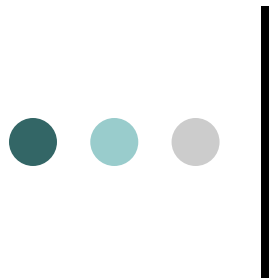
Service Considerations

Host Service	Language Environment Equivalent	Usability
ABEND	Call CEESGL with a severity 4 condition, call CEE3ABD, or have the assembler user exit request an abend at termination.	Host services can, but should not, be used. Use of equivalent Language Environment services is advised. ABEND can be used as a last resort.
ATTACH/ DETACH/ CHAP	No equivalent Language Environment function.	These services can be used.
ENQ/DEQ	No equivalent Language Environment function.	These services can be used.
(E)STAE/ (E)SPIE/ SETRP/ STAX	Use Language Environment's condition management callable services: CEEHDLR, CEEHDLU, and CEESGL.	Host services should not be used; instances should be changed to use Language Environment condition management callable services. Otherwise, unpredictable results may occur.



Service Considerations

Host Service	Language Environment Equivalent	Usability
EXEC CICS LOAD/DELETE	Use the Language Environment CEEFETCH assembler macro.	Host services can be used, but you must manage the loaded routines.
EXEC CICS XCTL/LINK	No equivalent Language Environment function.	These services can be used.
GETMAIN/ FREEMAIN EXEC CICS GETMAIN/ EXEC CICS FREEMAIN	For automatic storage (block- related), use Language Environment's stack storage. For non-block-related storage (that is, the storage persists beyond the current activation), use Language Environment heap storage.	Host services can, but should not, be used. Use of equivalent Language Environment storage management services is advised. Any heap storage allocated by Language Environment will automatically be freed at termination.



Service Considerations

Host Service	Language Environment Equivalent	Usability
LOAD/DELETE CVSRTL	Use the Language Environment CEEFETCH assembler macro.	<p>If you are introducing a new language into the environment, host services must not be used. The new language is not properly initialized.</p> <p>If you are not introducing a new language into the environment, host services can be used. However, you must manage the loaded routines.</p>
OPEN/CLOSE GET/PUT READ/WRITE	No equivalent Language Environment function.	Host services can be used.
PC (Program Call instruction)	High level language call statements, such as assembler BALR/BASSM.	Not supported by Language Environment.
SNAP	Call CEE3DMP.	This service can be used.



Service Considerations

Host Service	Language Environment Equivalent	Usability
STIMER	No equivalent Language Environment function.	This service can be used.
TIME	Call Language Environment date and time services.	This service can be used.
SVC LINK	No equivalent Language Environment function.	This service can be used. For compatibility, Language Environment supports the LINK boundary crossing and treats it as a new enclave.
WAIT/POST/EVENTS	No equivalent Language Environment function.	Host services can be used.
WTO	Call CEEMOUT. This writes to the error log or the terminal.	Host services can be used.
XCTL	No equivalent Language Environment function.	Host services can, but should not, be used.



The End...

Thank You!

Session 8220



SHARE

Technology · Connections · Results



Appendix - Reference

- z/OS Language Environment Programming Guide
- z/OS Language Environment Vendor Interfaces
- z/OS Language Environment Writing ILC Applications
- z/OS Language Environment Concepts Guide
- z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode