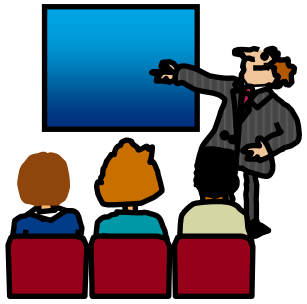
Slowed down by LE? Perhaps the CEEPIPI service can help!

Thomas Petrolino
IBM Poughkeepsie
tapetro@us.ibm.com

Session 9660



©Copyright IBM Corporation 2001, 2011
SHARE in Orlando, August 2011

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- CICS®
- DB2®
- Language Environment®
- OS/390®
- z/OS®

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.
UNIX is a registered trademark of The Open Group in the United States and other countries.
SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

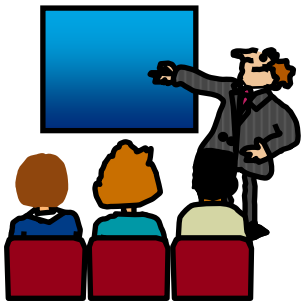
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Agenda

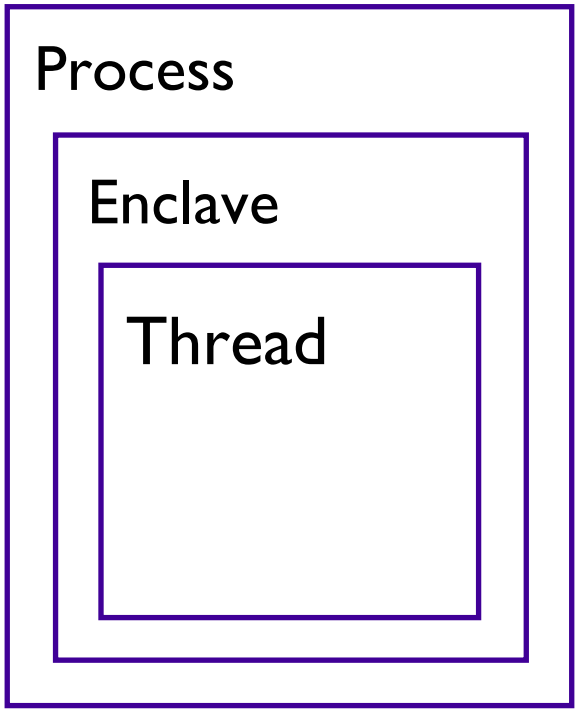
- Understanding The Basics of PreInitialization
- Writing a Preinit Application
- Other Preinit Topics
- A Preinit Example
- Sources of Additional Information

Understanding The Basics of PreInitialization



Background - LE Init/Term

- Process - Collection of Resources (LE message file, library code/data)
  - unaffected by HLL semantics, logically independent address space
- Enclave - Collection of Routines (Load modules, Heap, external data)
  - defines scope of HLL semantics, first routine is designated "main"
- Thread - "thread" of execution (Stack, raised conditions)
  - share the resources of the enclave



Understanding The Basics

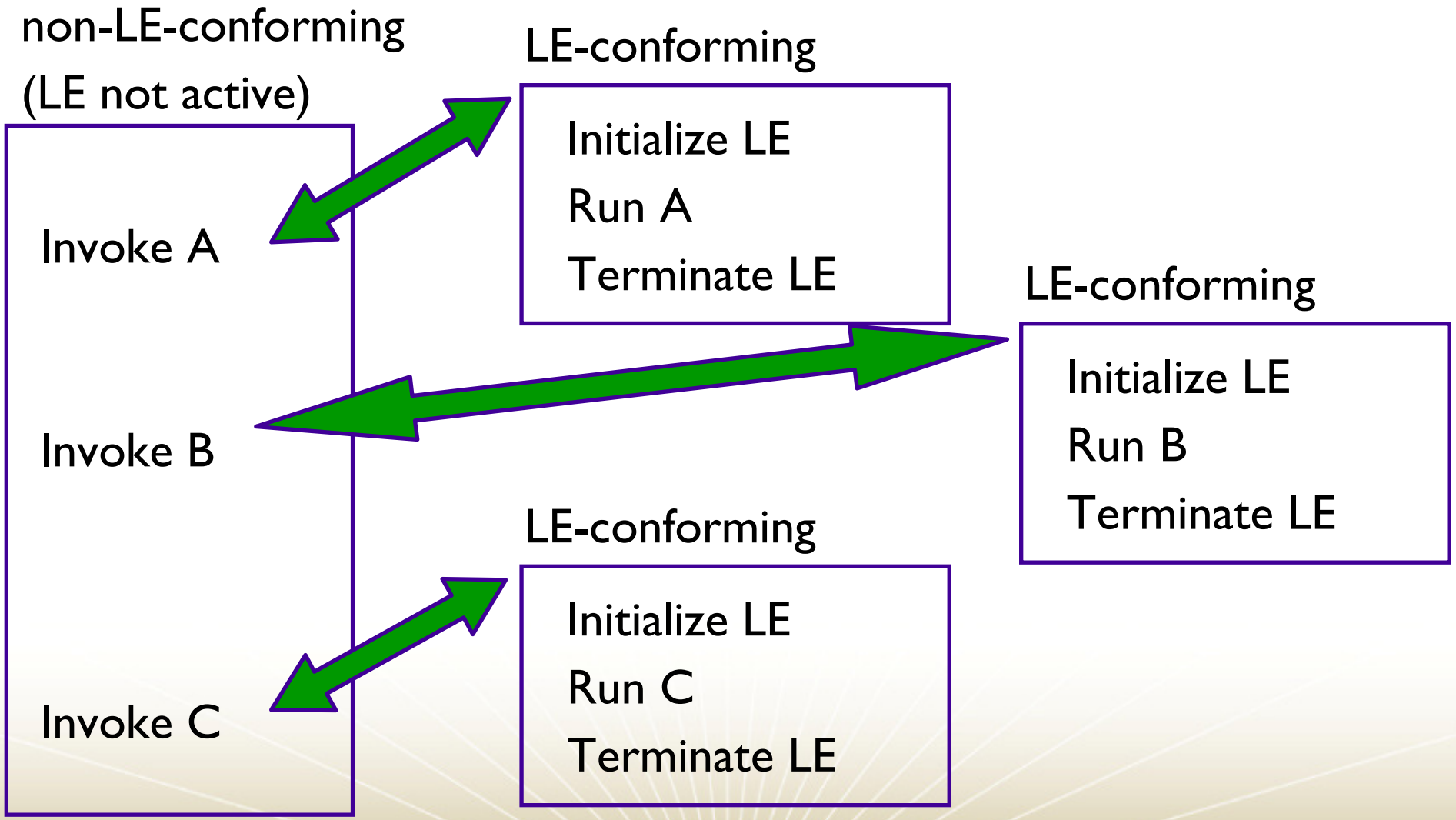
- Read Language Environment Programming Guide, Chapter 30 "Using preinitialization services" (SA22-7561)

- Read Language Environment Programming Guide for 64-bit Virtual Addressing Mode, Chapter 22 “Using preinitialization services with AMODE 64” (SA22-7569)

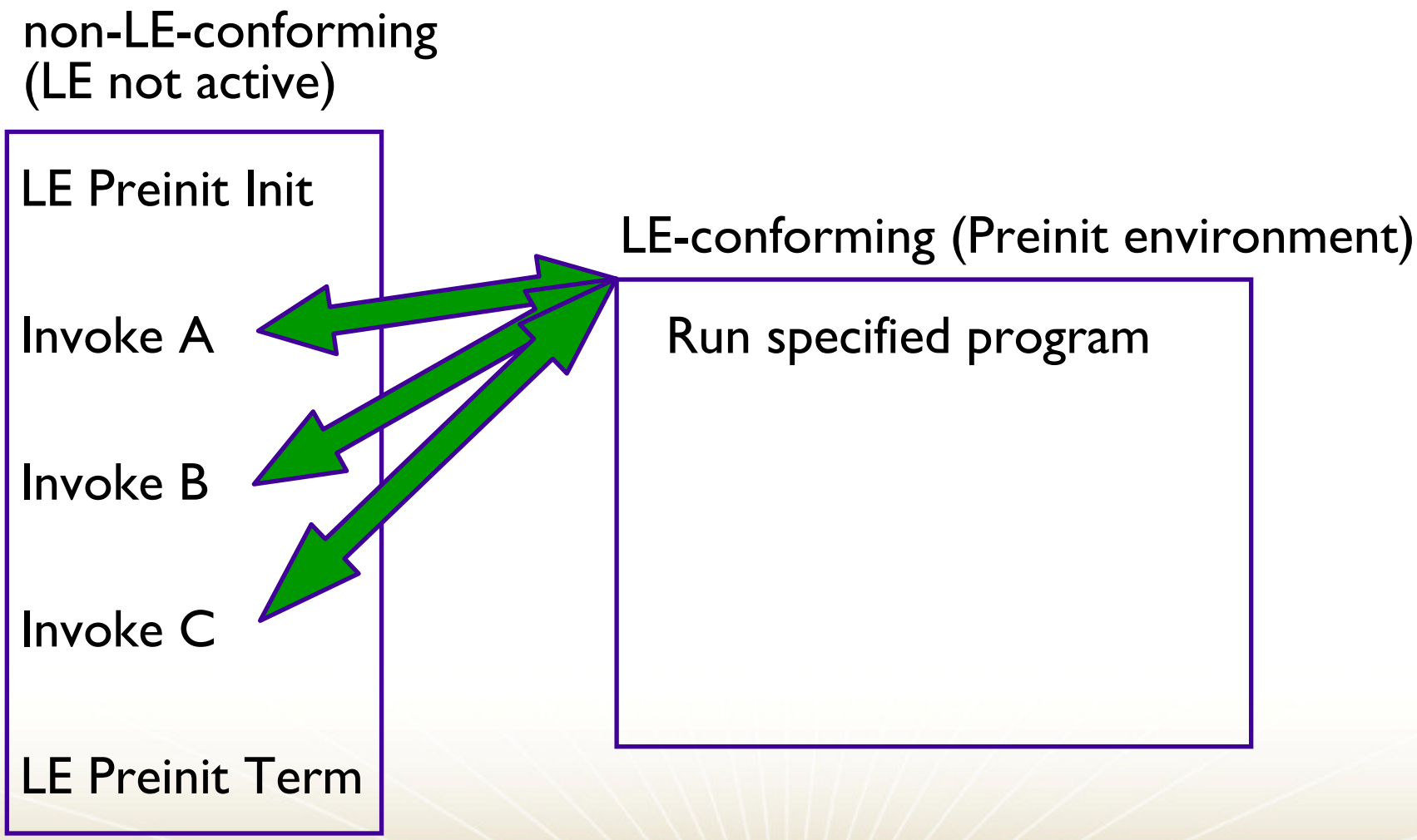
Understanding The Basics…

- You can use preinitialization to enhance the performance of certain applications
- Preinitialization lets a non-LE-conforming application (eg. Assembler) initialize an LE environment once, perform multiple executions of LE-conforming programs using that environment, and then explicitly terminate the LE environment
- Because the environment is initialized only once (even if you perform multiple executions), you free up system resources and allow for faster responses to your requests.

A non-Preinit scenario



Same application using Preinit



Older forms of preinitialization

- The following is a list of pre-LE language-specific forms of preinitialization. These environments are supported by LE but will not be enhanced.
  - C and PL/I -- supports prior form of C and PL/I preinitialization (PICI) through use of Extended Parameter List
  - C++ -- no prior form of preinitialization
  - COBOL -- supports the prior form of COBOL preinitialization through use of RTEREUS run-time option and ILBOSTP0 and IGZERRE functions
  - Fortran -- no prior form of preinitialization
- LE Library Routine Retention (LRR) is also supported but is not the "preferred" method

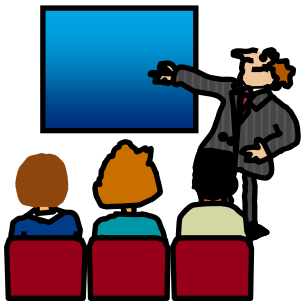
Restrictions on pre-LE preinitialization

- POSIX(ON)
- XPLINK
- AMODE 64

Users of preinitialization

- Numerous IBM products currently utilize preinitialization
  - Program Management Binder – for C++ demangler
  - DB2 – for stored procedures
  - CICS – TS V3.1 for XPLINK support
  - . . .
- Many IBM customers…

Writing a Preinit Application



The Preinit Application

- A Preinit application consists of:
  - One or more HLL routines
  - A Preinit Table
  - A Preinit Assembler Driver

HLL Routines

An example subroutine: Notice anything unusual?

```
CBL LIB,QUOTE
   IDENTIFICATION DIVISION.
   PROGRAM-ID. HLLPIPI.
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   PROCEDURE DIVISION.
         DISPLAY "COBOL subprogram beginning".
         DISPLAY "Called using LE Preinitialization".
         DISPLAY "Call subroutine interface.".
         DISPLAY "COBOL subprogram returns to caller.".
         GOBACK.
```

Your answer should be “Nope!”

HLL Routines

- Written in
  - C
  - C++
  - PL/I
  - COBOL
- May be main or subroutine
  - If using an XPLINK or AMODE 64 subroutine, it must be declared “fetchable”

The Preinit table

- The Preinit table identifies routines to be executed (and optionally loaded) in a Preinit environment
  - It contains routine names and/or entry point addresses
  - It is possible to have an "empty" Preinit table with empty rows
    - routines can be added later using the Preinit *add_entry* interface
- In the Preinit table, entry point addresses are maintained with the High Order Bit set to indicate AMODE of routine
  - HOB on, routine is AMODE31 and invoked in 31 bit mode
  - HOB off, routine is AMODE24 and invoked in 24 bit mode
- CEEBXITA (Asm User Exit), CEEBINT (HLL User Exit), CEEUOPT are obtained from *first entry in Preinit table*

Generate the Preinit table

- LE provides the following assembler macros to generate the Preinit table
  - **CEEXPIT** generates a header for the Preinit table
  - **CEEXPITY** generates an entry within the Preinit table
    - specify entry *name* and/or *entry_point* address of the routine
    - each invocation generates a row in the Preinit table
    - if *name* is blank and *entry_point* is zero, then an empty row is added to the Preinit table
  - **CEEXPITS** identifies the end of the Preinit table
  - **CELQPIT**, **CELQPITY**, **CELQPITS** for AMODE 64
- The size of the Preinit table cannot be increased dynamically

The Preinit Table

Declared in the data section of the Preinit Assembler Driver:

```
          :
          :
* ================================================================
* Preinitialization Table.
* ================================================================
*
PPTBL     CEEXPIT ,                    Preinitialization Table with index
          CEEXPITY HLLPIPI,0           dynamically loaded routine
          CEEXPITY ,HLLEXTRN           statically-bound routine
          CEEXPITY ,                   empty Table slot
          CEEXPITS ,                   Endof PreInit table
*
          EXTRN    HLLEXTRN
*
          :
          :
```

The Preinit Assembler Driver

- The Preinit Assembler Driver is responsible for:
  - Loading the Preinit Interface module
  - Initializing / Terminating the Preinit environment
  - Calling HLL routines using the Preinit environment

The Preinit Interface Module

- The main Preinit interface is the loadable module "CEEPIPI“
  - The AMODE 64 Preinit interface is the loadable module “CELQPIPI”
- CEEPIPI handles the requests and provides services for:
  - LE Environment Initialization
  - Application Invocation
  - LE Environment Termination
- All requests for services by CEEPIPI must be made from a non-Language Environment environment
- The parameter list for CEEPIPI is an OS standard linkage parameter list
  - First parameter on each call to CEEPIPI is a Preinit function code

Loading CEEPIPI

```
          :
          :
*
* Load LE CEEPIPI service routine into main storage.
*
          LOAD  EP=CEEPIPI            Load CEEPIPI routine dynamically
          ST    R0,PPRTNPTR           Save the addr of CEEPIPI routine
*
          :
          :
```

Preinit Initialization

- LE supports three forms of preinitialized environments
- They are distinguished by the level of initialization
  - **init_main** - supports the execution of main routine
    - initializes LE environment through process-level
    - each **call_main** invocation initializes enclave- and thread-level
  - **init_sub** - supports the execution of subroutines
    - initializes LE environment through process-, enclave-, and thread-level
    - each **call_sub** invocation has minimal overhead
  - **init_sub_dp** - a special form of the init_sub that allows multiple preinitialized environments, for executing subroutines, to be created under the same task (TCB). For AMODE 64 **init_sub** is comparable.
    - Only one POSIX(ON) environment per TCB

Preinit Initialization...

- ***main*** Environment
  - Advantages
    - A new, pristine environment is created
    - Run-Time options can be specified for each application
  - Disadvantages
    - Poorer performance
- ***sub*** Environment
  - Advantages
    - Best performance
  - Disadvantages
    - The environment is left in what ever state the previous application left it (including WSA, working storage, etc)
    - Run-Time options cannot be changed

Initializing a Preinit Environment

```
*
* Initialize an LE Preinitialization main environment.
*
INIT_ENV EQU   *
         LA    R5,PPTBL                  Get address of Preinit Table
         ST    R5,@CEXPTBL               Ceexptbl_addr ->Preinit Table
         L     R15,PPRTNPTR              Get address of CEEPIPI routine
* Invoke CEEPIPI routine
         CALL  (15),(INITMAIN,@CEXPTBL,@SRVRTNS,TOKEN)
* Check return code:
         LTR   R2,R15                    Is R15 = zero?
         BZ    CMAIN                     Yes (success)..go to next section
* No (failure)..issue message
         WTO   'ASMPIPI: call to (INIT_MAIN) failed',ROUTCDE=11
         C     R2,=F'8'                  Check for partial initialization
         BE    TMAIN                     Yes..go do Preinit termination
* No..issue message & quit
         WTO   'ASMPIPI: INIT_MAIN failure RC is not 8.',ROUTCDE=11
         ABEND (R2),DUMP                 Abend with bad RC and dump memory
```

Initializing a Preinit Environment

```
*
* Initialize an LE Preinitialization subroutine environment.
*
INIT_ENV EQU   *
         LA    R5,PPTBL                  Get address of Preinit Table
         ST    R5,@CEXPTBL               Ceexptbl_addr ->Preinit 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 (INIT_SUB) failed',ROUTCDE=11
         C     R2,=F'8'                  Check for partial initialization
         BE    TSUB                      Yes..go do Preinit termination
* No..issue message & quit
         WTO   'ASMPIPI: INIT_SUB failure RC is not 8.',ROUTCDE=11
         ABEND (R2),DUMP                 Abend with bad RC and dump memory
```

Calling the HLL Routine

- Language Environment provides services to invoke either a main routine or subroutine.
  - When invoking **main** routines, the environment must have been initialized with **init_main**
  - When invoking **subroutines**, the environment must have been initialized with **init_sub** or **init_sub_dp**
- The Preinit environment identified by ***token*** is activated before the specified routine is called
- After the called routine returns, the environment becomes "dormant“
- The parameter list is passed to the application as-is
  - XPLink & 64-bit convert from OS format to XPLink

Calling the HLL Routine…

- It is important to provide the parameter list in the exact format that the compiled routine is expecting
  - C Example: 'TESTPGM 10 5' when interactively invoked
  - C function prototype: main(int argc, char **argv)
  - Assembler parameter list layout:

```
PARMPTR  DC A(PARMLIST)           Pointer to PARMLIST
*
PARMLIST DS 0A                    Parameter List
ARGC     DC F'3'                  Number of arguments
ARGVPTR  DC A(ARGV)               Pointer to Argument Array
*
ARGV     DS    0A                 Argument Array
ARCV0    DC    A(ARGV0S)          Pointer to Argument 1
ARGV1    DC    A(ARGV1S)          Pointer to Argument 2
ARGV2    DC    A(ARGV2S)          Pointer to Argument 3
*
ARGV0S   DC    C'TESTPGM',X'00'   Argument 1
ARGV1S   DC    C'10',X'00'        Argument 2
ARGV2S   DC    C'5',X'00'         Argument 3
```

Calling a HLL Main

```
          :
          :
*
* Call the main, which is loaded by LE
*
CMAIN    EQU   *
         L     R15,PPRTNPTR            Get address of CEEPIPI routine
         CALL  (15),(CALLMAIN,PTBINDEX,TOKEN,RUNTMOPT,PARMPTR,          X
               ENCRETC,ENCRSNC,APPLFBC)
* Check return code:
         LTR   R2,R15                  Is R15 = zero?
         BZ    TMAIN                   Yes (success)..go to next section
* No (failure)..issue message & quit
         WTO   'ASMPIPI: call to (CALL_MAIN) failed',ROUTCDE=11
         ABEND (R2),DUMP               Abend with bad RC and dump memory
         :
         :
```

Calling a HLL Subroutine

```
          :
          :
*
* 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)
* 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 (CALL_SUB) failed',ROUTCDE=11
          ABEND (R2),DUMP               Abend with bad RC and dump memory
          :
          :
```

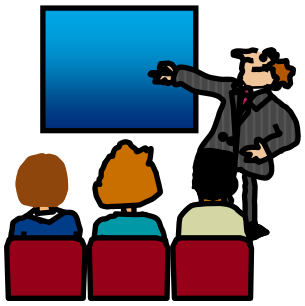
Preinit Termination

- The Preinit application terminates the Preinit environment once it is no longer needed
- Termination performs cleanup of the resources associated with the environment
- A single Termination service handles all types of Preinit environments

Terminating the Preinit Environment

```
         :
         :
*
* Terminate the environment
*
TSUB     EQU   *
         L     R15,PPRTNPTR          Get address of CEEPIPI routine
         CALL  (15),(TERM,TOKEN,ENV_RC)
* 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 (TERM) failed',ROUTCDE=11
         ABEND (R2),DUMP             Abend with bad RC and dump memory
*
         :
         :
```

Other Preinit Topics



Reentrancy Considerations

- You can make multiple calls to **main** routines or **subroutines**
- In general, you should specify only reentrant routines for multiple invocations:
  - Multiple calls to a reentrant **main** routine are not influenced by a previous execution of the same routine
  - For example, external variables are reinitialized for every call to a reentrant **main**

☞ If you have a nonreentrant COBOL program, condition IGZ0044S is signalled when the routine is invoked again

☞ If you have a nonreentrant C main() program that uses external variables, then when your routine is invoked again, the variables will be in last-use state

☞ Multiple calls to reentrant **subroutines** reuse the same working storage, it is only initialized once during (*call_sub*)

Stop Semantics in Preinit subs

- When one of the following occurs within a preinitialized environment *for subroutines*, the logical enclave is terminated:
  - C exit(), abort(), or signal handling function specifying a normal or abnormal termination
  - COBOL STOP RUN statement
  - PL/I STOP or EXIT
  - an unhandled condition causing termination of the (only) thread
- The process level of the environment is retained
- Modules in Preinit table are not deleted
- The next call to a subrtn in this environment will initialize a new enclave (possibly with different user exits)

Additional Preinit Services

- Calling a Subroutine By Address
  - call_sub_addr: Invoke a subroutine by address within an already initialized environment

- Improving Performance of a Sequence of Calls
  - start_seq: Start a sequence of uninterruptible calls to a number of subroutines
  - end_seq: Terminate a sequence of uninterruptible calls to a number of subroutines

Additional Preinit Services…

- Managing the Preinit Table
  - add_entry: Dynamically add a routine to an environment
  - delete_entry: Delete an entry from the Preinit table, making it available to a later add_entry
- Extracting Information from an Environment
  - identify_environment: Determine characteristics of a Preinit environment
  - identify_entry: Identify the language of an entry in the Preinit table
  - identify_attributes: Identify the attributes of an entry in the Preinit table

User Exit Invocation

	init_sub, init_sub_dp	call_main	call_sub or call_sub_addr ended with STOP semantics	term for "clean" init_sub or init_sub_dp environment	term
CEEBXITA (enclave init)	X	X	X(next call)		
CEEBINT (HLL exit)	X	X	X(next call)		
C atexit() functions		X	X	X	
CEEBXITA (enclave term)		X	X	X	
CEEBXITA (process term)				X	X

- Main environments: CEEBXITA and CEEBINT application-specific user exits are taken from the main routine being called.
- Sub environments: CEEBXITA and CEEBINT application-specific user exits are taken from the first entry in Preinit table.
- All other occurrences are ignored!

XPLINK Preinit

- Preinit applications can run XPLINK-compiled programs in a Preinit environment.
- LE initializes *either* an XPLINK environment or a "regular" (non-XPLINK) environment
  - Main: XPLINK environment if routine in first Preinit Table entry is XPLINK
  - Subroutine: XPLINK environment if routine in first Preinit Table entry is XPLINK, or if XPLINK(ON) run-time option is specified

XPLINK Preinit...

- call_main may cause an environment switch
  - If running a non-XPLINK environment, *and* either the program was compiled XPLINK or XPLINK(ON) was specified, the environment will be rebuilt XPLINK, *and remain that way.*
- Sub environments do not switch
  - A call to an XPLINK subroutine in a non-XPLINK environment will result in a "mismatch" error.
- Recommendation: Do not use non-XPLINK routines in an XPLINK Preinit environment.

Service routines

- Under Preinit, you can specify several service routines for use with running a main routine or subroutine in the preinitialized environment
- To use the routines, specify a list of addresses of the routines in a service routine vector
  - Pass the address of this list on the *init_main, init_sub, or init_sub_dp* interfaces
  - The *service_rtns* parameter that you specify contains the address of the vector itself
  - If this pointer is specified as zero (0), LE routines are used instead of the service routines
- Why?
  - Execution environment has its own storage or program management services
- Now supported in AMODE 64 Language Environment
  - z/OS V1.9: @Load and @Delete service routines
  - z/OS V1.11: @Getstore, @Freestore, and @Msgrtn service routines

Service routines...

- Count
  - the number of fullwords that follow
- User Word
  - passed to the service routines
  - provides a means for your routine to communicate to the service routines
- @Workarea
  - address of a work area of at least 256 bytes that is doubled word aligned. First word contains the length of area provided. Required if service routines present in vector
- @Load
  - loads named routines for application management
- @Delete
  - deletes routines for application management

Service routines...

- @Getstore
  - allocates storage on behalf of the storage manager. This routine relies on the caller to provide a save area, which can be the @Workarea
- @Freestore
  - frees storage on behalf of storage manager
- @Exceprtn
  - traps program interrupts and abends for condition management
- @Msgrtn
  - allows error messages to be processed by caller of the application

Preinit Diagnostics

- Preinit Trace Table
- IPCS Support to format Preinit control blocks and trace table

Preinit Diagnostics…

- Preinit Trace Table Characteristics
  - Tracing is always active
    - Begins when the Preinit environment is initialized and ends when the environment is terminated
  - Trace is kept in an in-storage trace table
    - Fixed size (4096 bytes)
    - Wraps when the end has been reached

Preinit Diagnostics…

- New keyword for the LEDATA IPCS Verbexit:
  - **PTBL**(value) - Formats Preinit control block and trace table based on value:
    - "**CURRENT**“ - Preinit data associated with the current or specified TCB is displayed.
    - <**address**> - Preinit data at that address is displayed.
    - “*“ – Data for all active and dormant Preinit environments within the current address space are displayed; *** *This option is time-consuming* ***.
    - "**ACTIVE**“ – Display Preinit data associated with each TCB in the address space.

Preinit Diagnostics…

LEDATA PTBL Output – Preinit Control Block

```
=== > VERBEXIT LEDATA 'PTBL(CURRENT)'

PreInitialization Programming Interface Trace Data
 CEEPIPI Environment Table Entry and Trace Entry :
  Active CEEPIPI Environment (  Address 25805CB0 )
  Eyecatcher  : CEEXIPTB
  TCB address : 008D1B08

 CEEPIPI Environment :
  Non-XPLINK Environment
  Environment Type : MAIN
  Sequence of Calls not active
  Exits not established
  Signal Interrupt Routines not registered
  Service Routines are not active

  CEEPIPI Environment Enclave Initialized
  Number of CEEPIPI Table Entries = 2
```

Preinit Diagnostics…

LEDATA PTBL Output – Preinit Control Block…

```
CEEPIPI Table Entry Information :
     CEEPIPI Table Index 0   ( Entry 1    )
      Routine Name   = HLLCRTN
      Routine Type   = C/C++
      Routine Entry Point   = A5810B38
      Routine Function Pointer   = A5810CC0
      Routine Entry is Non-XPLINK
      Routine was loaded by Language Environment
      Routine Address was resolved
      Routine Function Descriptor was valid
      Routine Return Code     = 0
      Routine Reason Code     = 0
```

Preinit Diagnostics…

LEDATA PTBL Output – Preinit Control Block…

```
Entry of routine in CEEPIPI Table for Index 0 ( 25805DB8 )

     +000000 25805DB8  A5810CC0 25811B30 80000000 00000000
                       00000000 00000000 00000000 00000000
                       |va...a.........................|
     +000020 25805DD8  00000000 00000000 00000000 A5810B38
                       00000003 258117C8 00000003 25810B38
                       |............va.......a.H.....a..|
     +000040 25805DF8  A5810B38 000014C8 C8D3D3C3 D9E3D540
                       00000000 00000000 00000000 00000000
                       |va.....HHLLCRTN ................|

    CEEPIPI Table Index 1   ( Entry 2    )  not in use.
```

Preinit Diagnostics…

LEDATA PTBL Output – Preinit Trace Table

```
CEEPIPI Trace Table Entries :
     Call Type =  INIT_MAIN
      PIPI Driver Address  = A5800A82
      Load Service Return Code   = 0
      Load Service Reason Code   = 0
      Most Recent Return Code    = 0
      Most Recent Reason Code    = 0
      An ABEND will be issued if storage can not be obtained
      PreInit Environment will not allow EXEC CICS commands
      Service RC = 0 :A new environment was initialized
```

Preinit Diagnostics…

LEDATA PTBL Output – Preinit Trace Table…

```
Call Type =  ADD_ENTRY
 Routine Table Index         = 1
 Routine Name  = HLLPIPI
 Routine Address = A5812E20
 Load Service Return Code    = 0
 Load Service Reason Code    = 3
 Service RC = 0 :The routine was added to the PreInit table.

Call Type =  CALL_MAIN
 Routine Table Index      = 1
 Enclave Return Code      = 0
 Enclave Reason Code      = 0
 Routine Feedback Code    = 0000000000000000
 Service RC = 0 :The environment was activated and the
    routine called.
```

Preinit Diagnostics…

LEDATA PTBL Output – Preinit Trace Table…

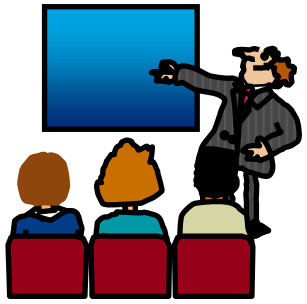
```
Call Type =  DELETE_ENTRY
 Routine Table Index     = 1
 Routine Name  = HLLCOBOL
 Routine Address = A5812E20
 Service RC = 0 :The routine was deleted from the
    PreInit table.

Call Type =  CALL_MAIN
 Routine Table Index     = 0
 Enclave Return Code     = 0
 Enclave Reason Code     = 0
 Routine Feedback Code   = 0000000000000000
 Service RC = 0 :The environment was activated and
    the routine called.
```

A Preinit Example

The following example provides an illustration of an assembler program ASMPIPI ASSEMBLE invoking CEEPIPI to:

- **Initialize a LE Preinit subroutine environment**
- **Load and call a reentrant C/COBOL/PLI subroutine**
- **Terminate the LE Preinit environment**



Example

- Following the assembler program are interchangeable examples of the program HLLPIPI written in:
  - C, COBOL, and PL/I
- HLLPIPI is called by an assembler program, ASMPIPI.
- ASMPIPI uses the Language Environment preinitialized program subroutine call interface
- You can use the assembler program to call the HLL versions of HLLPIPI.

Example...

```
*
*COMPILATION UNIT: LEASMPIP
*************************************************************************
*                                                                       *
* Function: CEEPIPI - Initialize the Preinitialization                  *
*                     environment,call a Preinitialization              *
*                     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 Preinitialization environment.    *
*                                                                       *
* Note: ASMPIPI is not reentrant.                                       *
*                                                                       *
*************************************************************************
```

Example...

```
* ======================================
* 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.
*
         LOAD  EP=CEEPIPI               Load CEEPIPI routine dynamically
         ST    R0,PPRTNPTR              Save the addr of CEEPIPI routine
```

Example...

```
*
* Initialize an LE Preinitialization subroutine environment.
*
INIT_ENV EQU   *
         LA    R5,PPTBL                  Get address of Preinit Table
         ST    R5,@CEXPTBL               Ceexptbl_addr ->Preinit 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 (INIT_SUB) failed',ROUTCDE=11
         C     R2,=F'8'                  Check for partial initialization
         BE    TSUB                      Yes..go do Preinit termination
* No..issue message & quit
         WTO   'ASMPIPI: INIT_SUB failure RC is not 8.',ROUTCDE=11
         ABEND (R2),DUMP                 Abend with bad RC and dump memory
```

Example...

```
*
* 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)
* 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 (CALL_SUB) failed',ROUTCDE=11
         ABEND (R2),DUMP                 Abend with bad RC and dump memory
```

Example...

```
*
* Terminate the environment
*
TSUB     EQU   *
         L     R15,PPRTNPTR              Get address of CEEPIPI routine
         CALL  (15),(TERM,TOKEN,ENV_RC)
* 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 (TERM) failed',ROUTCDE=11
         ABEND (R2),DUMP                 Abend 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
```

Example...

```
* ================================================================
* CONSTANTS and SAVE AREA.
* ================================================================
SAVE             DC    18F'0'
PPRTNPTR         DS    A                  Save the address of CEEPIPI routine
*
* Parameters passed to an (INIT_SUB) call.
INITSUB          DC    F'3'               Function code to initialize for subr
@CEXPTBL         DC    A(PPTBL)           Address of Preinitialization 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 (CALL_SUB) call.
CALLSUB          DC    F'4'               Function code to call subroutine
PTBINDEX         DC    F'0'               The row number of Preinit 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)
```

Example...

```
*
* Parameters passed to a (TERM) call.
TERM     DC    F'5'                          Function code to terminate
ENV_RC   DS    F                             Environment return code (output)
* ===============================================================
* Preinitialization Table.
* ===============================================================
*
PPTBL    CEEXPIT ,                           Preinitialization Table with index
         CEEXPITY HLLPIPI,0                  0=dynamically loaded routine
         CEEXPITS ,                          Endof PreInit table
*
         LTORG
R0       EQU   0
R1       EQU   1
...
R14      EQU   14
R15      EQU   15
         END   ASMPIPI
```

Example...

C Subroutine Called by ASMPIPI

```
#include <stdio.h>

HLLPIPI ()
{
     printf("C subroutine beginning \n");
     printf("Called using LE PreInit call \n");
     printf("Subroutine interface.\n");
     printf("C subroutine returns to caller \n");
}
```

Example...

COBOL Program Called by ASMPIPI

```
CBL LIB,QUOTE
      *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 subprogram beginning".
           DISPLAY "Called using LE Preinitialization".
           DISPLAY "Call subroutine interface.".
           DISPLAY "COBOL subprogram returns to caller.".
           GOBACK.
```

Example...

PL/I Routine Called by ASMPIPI

```
/*Module/File Name: IBMPIPI                                        */
/*******************************************************************/
/*                                                                 */
/* HLLPIPI is called by an assembler program, ASMPIPI.             */
/* ASMPIPI uses the LE preinitializedprogram                       */
/* 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: CalledLE Preinit Call ');
          PUT SKIP LIST
              ('HLLPIPI: Subroutine interface.');
          PUT SKIP LIST
              ('HLLPIPI: PLI program returns to caller.');
          RETURN;
END HLLPIPI;
```

Sources of Additional Information

- LE Debug Guide and Runtime Messages
- LE Programming Reference
- LE Programming Guide (64-bit too!)
- LE Customization
- LE Migration Guide
- LE Writing ILC Applications
- Web site
  - http://www.ibm.com/servers/eserver/zseries/zos/le/