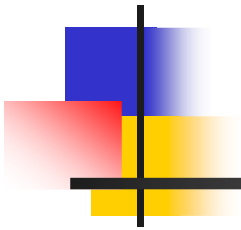
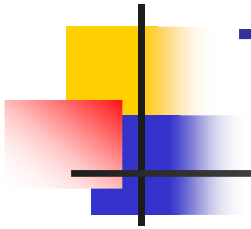


Exploit Condition Handling in Language Environment

Thomas Petrolino
IBM Poughkeepsie
tapetro@us.ibm.com



Trademarks

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

Language Environment

z/OS

CICS

* 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.

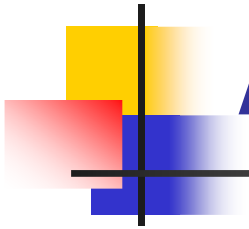
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations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

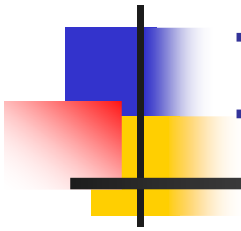
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products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

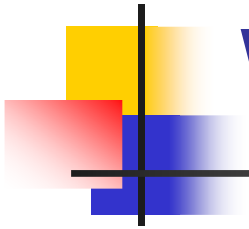


Agenda

- Introduction
- Condition Handling Terminology
- Language Environment Condition Handling Model
- Registering a Condition Handler
- Writing a Condition Handler
- Sources of Additional Information
- Appendix
  - Related Run-time Options
- Example COBOL Program (separate file)



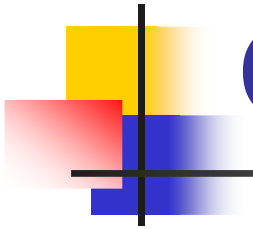
Introduction



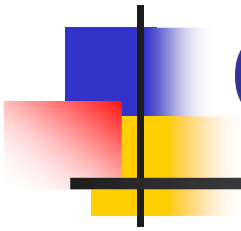
Why Use a Condition Handler?

- To detect and react to an issue encountered during the execution of an application
  - Recover from a failure that might have caused an application to terminate
  - Receive notification of a non-fatal situation
  - Capture diagnostic information to assist with trouble-shooting

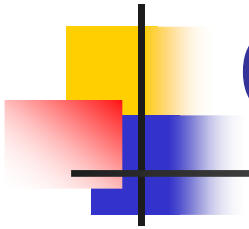
Language Environment Condition Handling



- A stack frame-based model
  - Application can handle conditions at the level at which they occur
- Can be written in language of choice
  - Assembler, C/C++, COBOL, PL/I
- Supports mixed-language environments

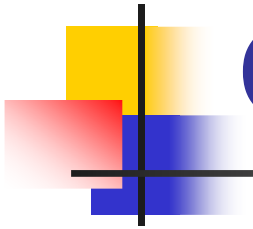


Condition Handling Terminology



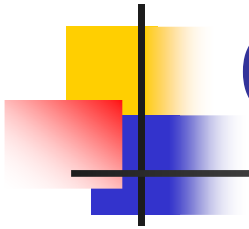
Condition

- Any change to the normal flow of a program
  - a.k.a. exception, interruption



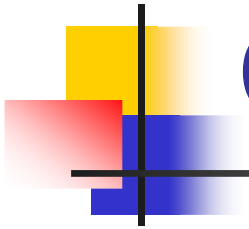
Condition...

- A Condition may occur because…
  - Hardware detects an interrupt
    - SOC7, SOC9, SOCB…
  - Operating system detects problem
    - Open error, some other file mismatch, out of memory, etc
  - Language detects some "situation"
    - COBOL "out of range" for table or reference modification and user has SSRANGE and CHECK(ON)
  - LE can generate condition via a callable service
    - Date "out of range" for CEEDAYS, for example
  - User routine "signals" a condition
    - Call to CEESGL from COBOL
    - raise() in C/C++
    - SIGNAL in PL/I



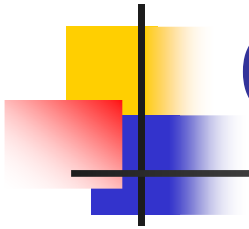
Condition Handler

- Routine invoked by Language Environment so that user programs can analyze and react to conditions
  - "Registered" to LE via:
    - call to CEEHDLR callable service
    - USRHDLR run-time option
    - Member language semantics, such as PL/I ON statements.
- Allows an application to resolve or at least react to problems that otherwise might have caused it to terminate



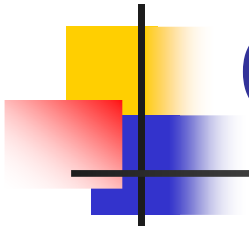
Condition Handler...

- Handler can choose to:
  - Resume – after corrective action taken, control returns to a 'resume cursor'
    - Either back to point of failure, or to a new resume point set by the condition handler
    - Example: Data Exception/S0C7 – application knows how to handle this, wants to continue processing
  - Percolate - decline to handle the condition, LE calls next condition handler
    - Example: Operation Exception/S0C1 – usually unexpected, application may want to just give up



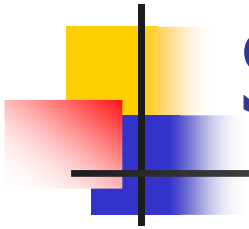
Condition Handler...

- Handler can choose to:
  - Promote - change condition meaning and percolate
    - Example: Warning-level condition CEE3QR / CEE3931W from CEEDLYM received, but application needs to treat it as critical
  - Fix-up and resume – resume with new input value to service or new output value from service



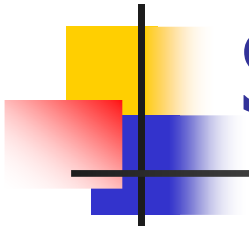
Condition Token

- a.k.a. Feedback Code, Message
  - Identifies a specific detected condition
  - 12 bytes of information, including:
    - Facility ID (CEE, AFH, IBM, IGZ, EDC…)
    - Message Number
    - Severity
    - Instance-specific information



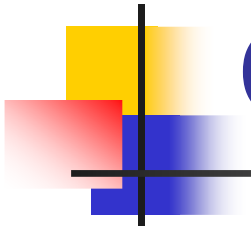
Stack Frame / DSA

- Physical representation of the activation of a routine
  - An area of storage allocated on a LIFO stack
- Contains:
  - Register Save Area
  - Automatic / local variables



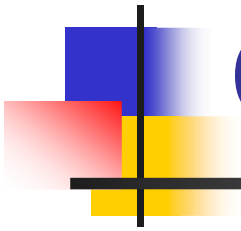
Stack Frame / DSA...

- A Stack Frame is created for:
  - Function call in C/C++
  - Entry into a compile unit in COBOL (not nested)
  - Entry into procedure or begin block in PL/I
  - Entry into ON-Unit in PL/I
  - Entry into a main or subprogram in Fortran
- A Stack Frame is destroyed when the driven program unit ends
- LE allocates "Stack Frame 0" to mark the start of the stack

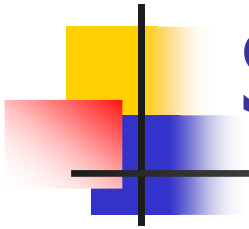


Cursors

- Resume cursor
  - Points to the initial "where to resume" location
  - Always on the move as the application executes, tracking the NSI (next sequential instruction)
  - When condition or signal occurs, resume cursor is positioned after the *machine* instruction that caused it
  - Can be "moved" to change the resume location
- Handle cursor
  - Points to the current condition handler being processed during Condition Handling



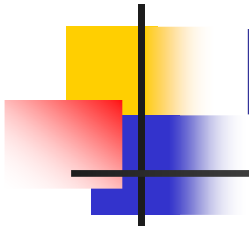
Language Environment Condition Handling Model



Stack Frame-based Model

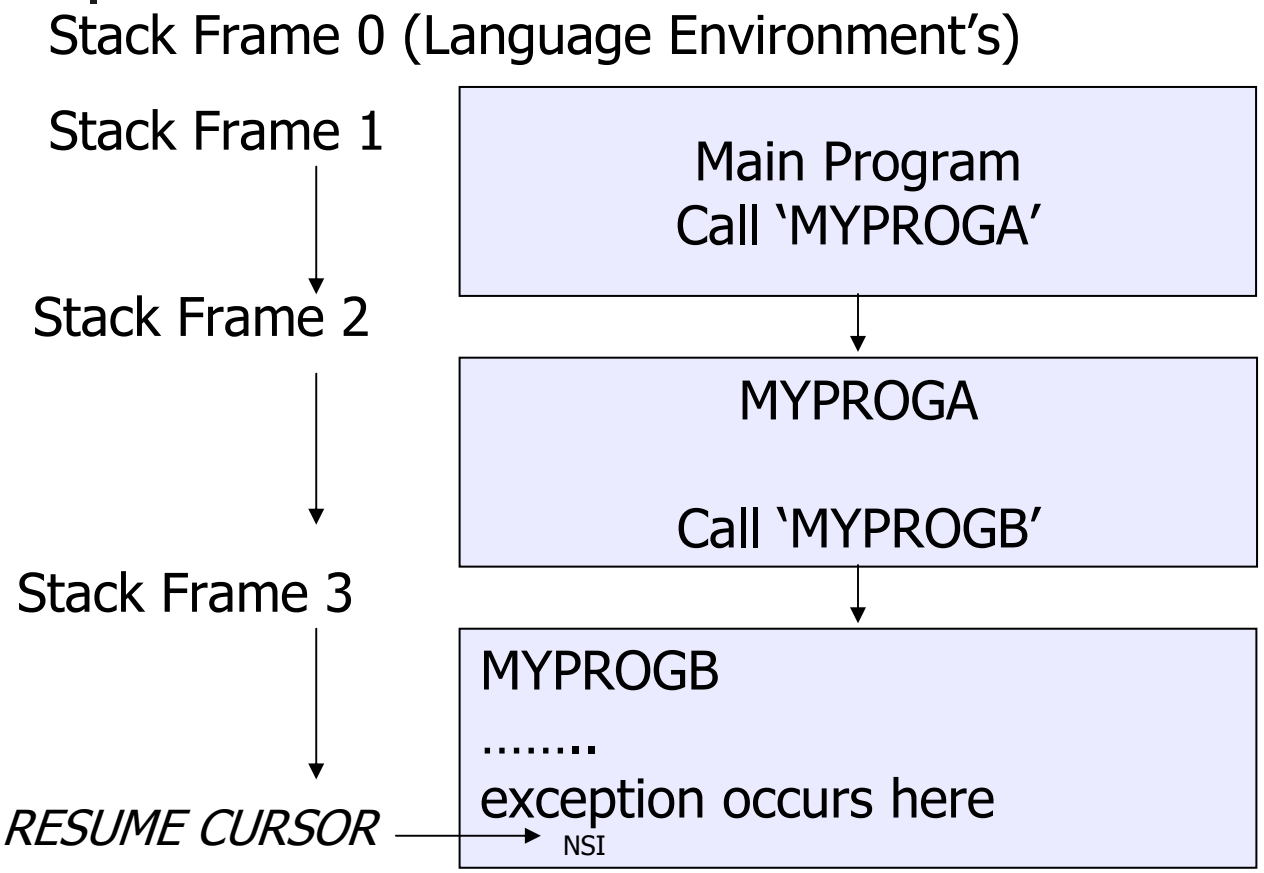
- User-registered condition handler is associated with the stack frame that registered it
- USRHDLR run-time option can associate a user-written condition handler with:
  - Stack frame 0; and/or
  - Stack frame that incurred the condition
- Member language handlers may be called at each stack frame to perform language-specific processing

Basic Condition Handling Flow



- Starts with the most recently activated stack frame
  - The stack frame that incurred the condition
- For each stack frame:
  - Looks for and calls user-written handlers for this stack frame
  - Looks for and calls member language-specific handlers
- If a handler is called, and it percolates or promotes the condition, LE continues to look for another handler to call
- Condition handling is complete if a handler requests "resume"
  - Processing resumes at the location defined in the Resume Cursor
- If all handlers have been called and no resume request occurred, then normal LE and/or language rules take over to finish

An Example

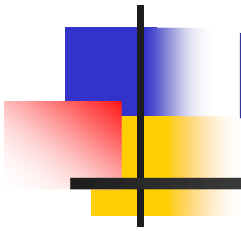


And if none found, then Language Environment or language rules apply

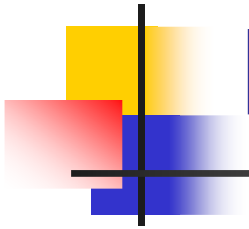
Still looking for someone to "handle" the condition..

And keeps walking back up the STACK frames…looking…..

*Language Environment **starts looking here** for a "condition handler" at this frame*

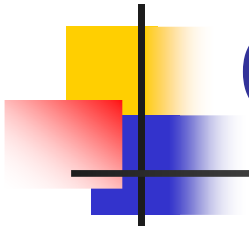


Registering a Condition Handler



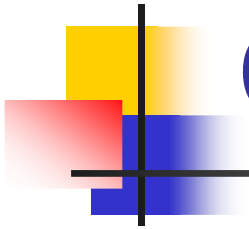
Mechanisms for Registering

- Via code: CEEHDLR / CEEHDLU
- Via run-time option: USRHDLR
- (Also language-specific semantics)



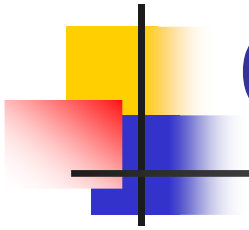
CEEHDLR

- Register User-written Condition Handler
- CEEHDLR(routine, token, fc)
  - routine – entry point of the handler
  - token – fullword integer of information you want passed to the handler
  - fc – optional feedback code indicating result of the call to CEEHDLR



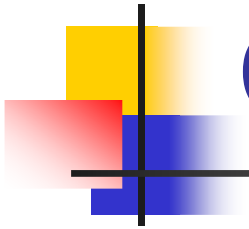
CEEHDLR...

- Condition handler is registered with the current stack frame
- Can register multiple condition handlers from multiple locations
  - Can have specific handlers for specific conditions
  - Handlers are called in LIFO order
- Handlers are automatically unregistered when the owning stack frame is destroyed



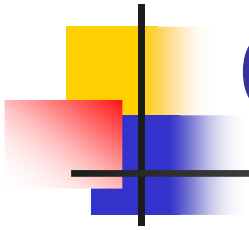
CEEHDLR...

```
*************************************************************
* REGISTER PROGRAM ECH911 AS A CONDITION HANDLER.
*************************************************************
SET PGMPTR TO ENTRY 'ECH911'
     MOVE 0000 TO MY-ABEND-TOKEN
     CALL 'CEEHDLR' USING PGMPTR, MY-ABEND-TOKEN, FEEDBACK
     IF FB-SEV = ZEROS
       DISPLAY 'ECHMAIN  - ECH911 REGISTERED'
     ELSE
       DISPLAY 'ECHMAIN  - ECH911 REGISTRATION FAILED'
       DISPLAY 'FB-MSG = ' FB-FAC, FB-MSG
     END-IF.
```



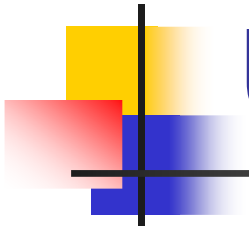
CEEHDLU

- Unregister User-written Condition Handler
- CEEHDLU(routine, fc)
  - routine – entry point of the handler
  - fc – optional feedback code indicating result of the call to CEEHDLR



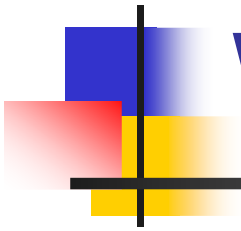
CEEHDLU...

```
*************************************************************
* UNREGISTER CONDITION HANDLER ECH911
*************************************************************
     SET PGMPTR TO ENTRY 'ECH911'
     CALL 'CEEHDLU' USING PGMPTR, FEEDBACK
     IF FB-SEV = ZEROS
       DISPLAY 'ECHMAIN  - ECH911 UNREGISTERED'
     ELSE
       DISPLAY 'ECHMAIN  - ECH911 UNREGISTRATION FAILED'
       DISPLAY 'FB-MSG = ' FB-FAC, FB-MSG
     END-IF.
```

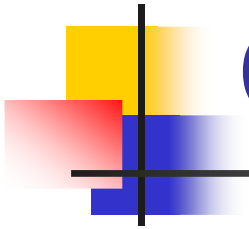


USRHDLR Run-time Option

- Register User-written Condition Handler without code modifications
- USRHDLR(lmname1,lmname2)
  - lmname1 – name of load module containing condition handler to be registered at stack frame 0
  - lmname2 – name of load module containing condition handler to get control before any other user condition handlers (“Super”)



Writing the Condition Handler

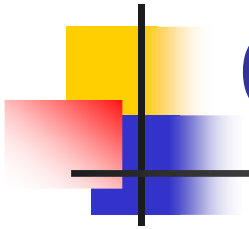


Condition Handler Interface

- condition_handler(c_ctok, token, result_code, new_condition)
  - c_ctok – Condition token that identifies the current condition being processed
  - token – 4-byte integer value provided when handler was registered with CEEHDLR
  - result_code – Instructs LE condition handling on how the handler wants to respond (resume, percolate, promote, fix-up and resume)
    - If unset, LE will assume "percolate"
  - new_condition – The promoted condition, or the fix-up action (new input / new output)

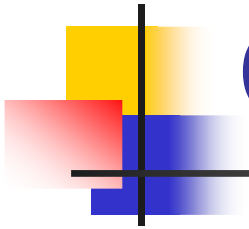
Condition Handler Interface…

- Condition token values are generally referred to by their symbolic feedback code
  - CEE34B = CEE3211S = X'00030C8B59C3C5C5'
  - LE provides definitions in SCEESAMP for feedback codes in supported languages
    - Assembler – CEEBALCT, AFHBALCT, EDCBALCT, IBMBALCT
    - C/C++ - CEEEDCCT, AFHEDCCT, EDCEDCCT, IBMEDCCT, IGZEDCCT
    - FORTRAN – CEEFORCT, AFHFORCT, IBMFORCT
    - PL/I – CEEIBMCT, AFHIBMCT, EDCIBMCT, IBMIBMCT, IGZIBMCT
    - COBOL – CEEIGZCT, AFHIGZCT, EDCIGZCT, IGZIGZCT, IBMIGZCT



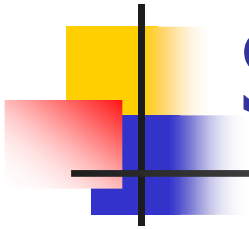
Condition Handler Processing

- Handler processing depends on application needs
  - May handle one condition or many
  - May capture diagnostic information
  - May correct problem
  - May drive clean-up activities
  - Decides whether to resume / percolate / promote / fix-up with resume



Condition Handler Processing...

- Resume point can also help
  - Bad record may be written to error file or report
  - Bad record may be marked as error
  - Resume point may pass control to a place that can read next record and continue processing
- Coordination between application program and condition handler is usually a good idea
  - Communication Area, footprints are helpful



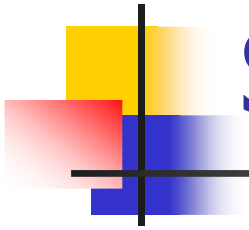
Simple Resume

- Condition Handler sets the result_code to indicate that the handler wants to resume at the instruction following the one that incurred the condition

Simple Resume...

Code from main routine ECHMAIN:

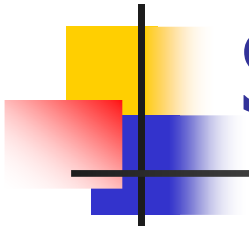
```
*************************************************
* FORCE A DECIMAL DIVIDE EXCEPTION BY DIVIDING BY ZERO.
* WHEN ECH911 DETECTS THIS EXCEPTION, IT WILL REQUEST A
* RESUME TO THE NEXT INSTRUCTION FOLLOWING THE DIVIDE.
*************************************************
  MOVE 'N' TO ERROR-INDICATOR
  DISPLAY "ECHMAIN  - ATTEMPTING DIVIDE"
  COMPUTE Z = 1 / DIVISOR.
  IF ERROR-INDICATOR = 'Y'
    DISPLAY "ECHMAIN - RESUMED AFTER DECIMAL DIVIDE EXCP"
  ELSE
    DISPLAY "ECHMAIN - DID NOT RESUME AFTER DEC DIVIDE"
  END-IF.
```



Simple Resume...

Code in condition handler ECH911 recognizes Decimal Divide Exception (symbolic feedback code CEE34B), sets the result_code to indicate 'resume':

```
WHEN CEE34B
  DISPLAY 'ECH911 - ENTERED FOR DECIMAL DIVIDE EXCP'
  MOVE 'Y' TO ERROR-INDICATOR
  SET RESUME TO TRUE
  DISPLAY 'ECH911 - RESUMING AFTER DECIMAL DIVIDE EXCP'
```



Simple Resume...

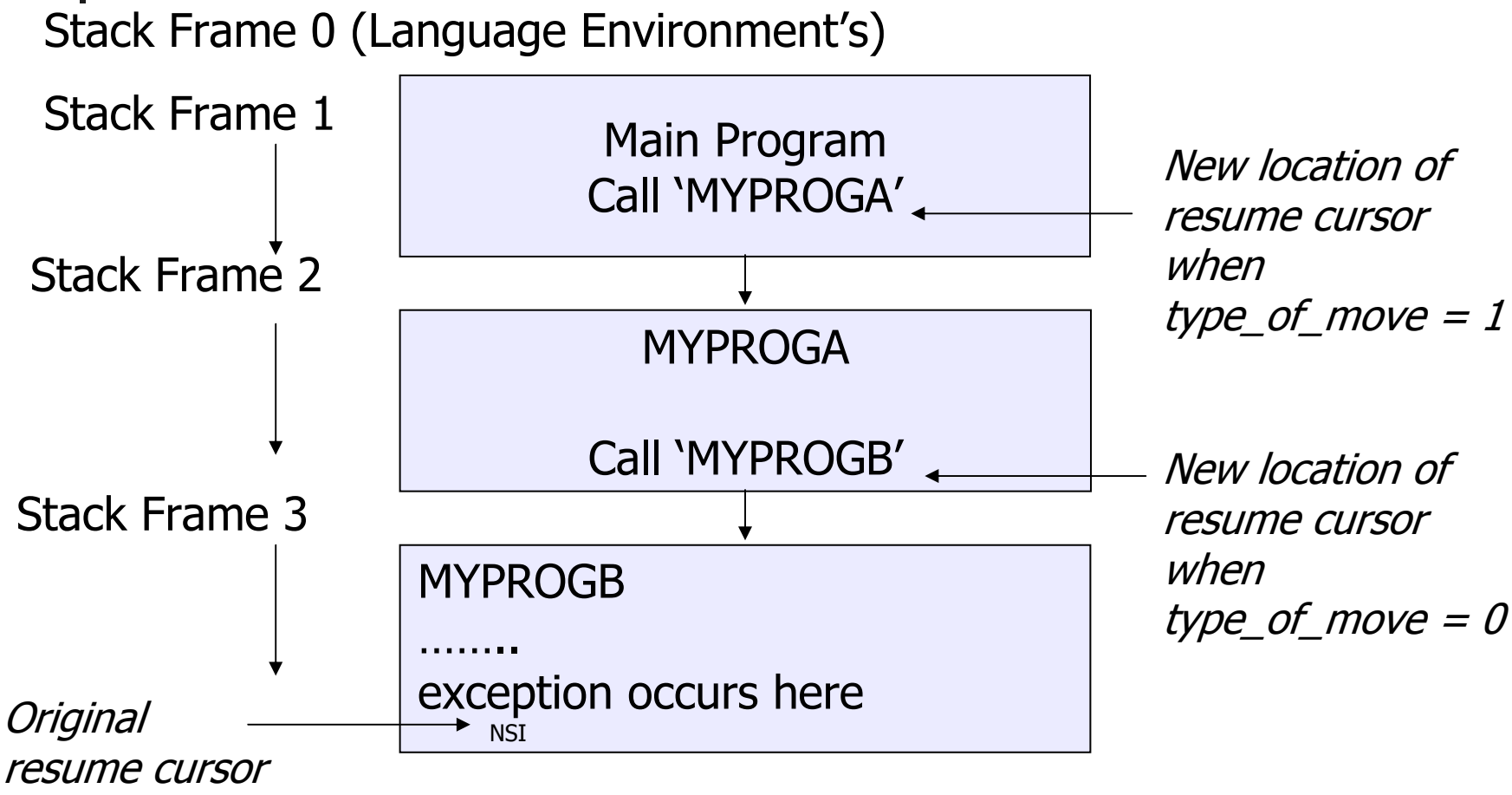
Program Output:

```
ECHMAIN - ATTEMPTING DIVIDE
ECH911  - ENTERED
ECH911  - ENTERED FOR DECIMAL DIVIDE EXCP
ECH911  - RESUMING AFTER DECIMAL DIVIDE EXCP
ECHMAIN - RESUMED AFTER DECIMAL DIVIDE EXCP
```

Resume with Move Cursor Relative

- Condition Handler uses CEEMRCR to move the resume cursor to a location relative to the location of the condition
- CEEMRCR(type_of_move, fc)
  - type_of_move – the target of the resume cursor movement
    - 0 – Move the resume cursor to the call return point of the stack frame associated with the handle cursor.
    - 1 - Move the resume cursor to the call return point of the stack frame prior to the stack frame associated with the handle cursor. (Cannot be used if nested COBOL routine!)

Resume with Move Cursor Relative...



Resume with Move Cursor Relative...

Code from main routine ECHMAIN:

```
*************************************************
* CALL PROGRAM ECHOUTBD, WHICH WILL FORCE A TABLE
* REFERENCE OUT OF BOUNDS ERROR. WHEN ECH911 DETECTS
* THIS EXCEPTION, IT WILL USE CEEMRCR TO MOVE THE RESUME
* CURSOR TO THE RETURN POINT FOLLOWING THE ECHOUTBD CALL.
*************************************************
     MOVE 'N' TO ERROR-INDICATOR
     DISPLAY "ECHMAIN  - CALLING ECHOUTBD"
     CALL "ECHOUTBD".
     IF ERROR-INDICATOR = 'Y'
       DISPLAY "ECHMAIN  - RESUMED AFTER OOB ERROR"
     ELSE
       DISPLAY "ECHMAIN  - DID NOT RESUME AFTER OOB
  ERROR"
     END-IF.
```

Resume with Move Cursor Relative...

Code from main routine ECHOUTBD:

```
*********************************************************
* FORCE A TABLE REFERENCE OUT OF BOUNDS ERROR.
* WHEN ECH911 DETECTS THIS EXCEPTION, IT WILL USE
* CEEMRCR TO MOVE THE RESUME CURSOR TO THE RETURN POINT
* FOLLOWING THE ORIGINAL CALL TO THIS ROUTINE.
*********************************************************
     DISPLAY 'ECHOUTBD - ATTEMPTING AN OUT OF BOUNDS REFERENCE'
     MOVE 9 TO J.
     MOVE 1 TO SLOT (J).

* EXECUTION SHOULD NOT REACH HERE WHEN COMPILED WITH SSRANGE.
     IF ERROR-INDICATOR = 'Y'
       DISPLAY "ECHOUTBD - NOT COMPILED WITH SSRANGE"
     ELSE
       DISPLAY "ECHOUTBD - NOT COMPILED WITH SSRANGE"
     END-IF.
     DISPLAY 'ECHOUTBD - ENDING'
     GOBACK.
```

Resume with Move Cursor Relative...

Code in condition handler ECH911 recognizes Table Reference Out of Bounds Exception (symbolic feedback code IGZ006), uses CEEMRCR to move the resume cursor to the return point of the caller, and sets the result_code to indicate 'resume':

```
WHEN IGZ006
        DISPLAY 'ECH911   - ENTERED AFTER TABLE REF OOB'
        MOVE 'Y' TO ERROR-INDICATOR
        CALL 'CEEMRCR' USING MOVE-TYPE, FEEDBACK
        IF FB-SEV = ZEROS
          DISPLAY 'ECH911   - CEEMRCR SUCCESSFUL'
        ELSE
          DISPLAY 'ECH911   - CEEMRCR FAILED'
          DISPLAY 'FB-MSG = ' FB-FAC, FB-MSG
        END-IF
        SET RESUME TO TRUE
        DISPLAY 'ECH911   - RESUMING AFTER TABLE REF OOB'
```

Resume with Move Cursor Relative...

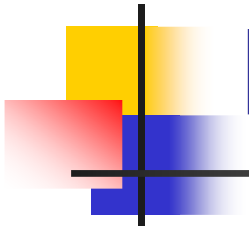
Program Output:

```
ECHMAIN  - CALLING ECHOUTBD
ECHOUTBD - STARTING
ECHOUTBD - ATTEMPTING AN OUT OF BOUNDS REFERENCE
ECH911   - ENTERED
ECH911   - ENTERED AFTER TABLE REF OOB
ECH911   - CEEMRCR SUCCESSFUL
ECH911   - RESUMING AFTER TABLE REF OOB
ECHMAIN  - RESUMED AFTER OOB ERROR
```

Resume with Move Cursor Explicit

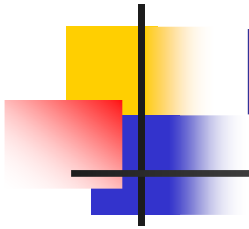
- A Move Cursor Explicit requires the use of two different callable services:
  - CEE3SRP – Set Resume Point
    - Called from mainline to establish a location to which a condition handler can resume
  - CEEMRCE – Move Resume Cursor Explicit
    - Called from a condition handler to set the resume cursor to the location set by a CEESRP call

Resume with Move Cursor Explicit...



- Set Resume Point
- CEE3SRP(resume_token, fc)
  - resume_token – a token for the machine state block built by CEE3SRP representing the location to where a condition handler can resume

Resume with Move Cursor Explicit...



- Move Resume Cursor Explicit
- CEEMRCE(resume_token, fc)
  - resume_token – a token from CEE3SRP that represents the resume point in the application

Resume with Move Cursor Explicit...

Code from main routine ECHMAIN:

```
*********************************************************
* USE CEE3SRP TO SET UP A RESUME POINT AT THE NEXT
* STATEMENT AFTER THE CALL. FORCE A PROTECTION
* EXCEPTION. WHEN ECH911 DETECTS THIS CONDITION, IT WILL
* USE CEEMRCE TO MOVE THE RESUME CURSOR TO THE SAVED
* RESUME POINT.
*********************************************************
     MOVE 'N' TO ERROR-INDICATOR
     CALL 'CEE3SRP' USING RECOVER-ADDR, FEEDBACK.

     IF ERROR-INDICATOR = 'N'
       DISPLAY "ECHMAIN  - ATTEMPTING PROTECTION EXCP"
       SET SS-POINTER TO NULL
       SET ADDRESS OF SIMPLE-STRUCTURE TO SS-POINTER
       MOVE 'A' TO SS-CHAR
     ELSE
       DISPLAY "ECHMAIN  - RESUMED AFTER PROTECTION EXCP"
     END-IF.
```

Resume with Move Cursor Explicit...

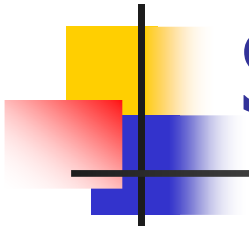
Code in condition handler ECH911 recognizes the Protection Exception (symbolic feedback code CEE344), uses CEEMRCE to move the resume cursor to a location previously established using CEE3SRP, and sets the result_code to indicate 'resume':

```
WHEN CEE344
        DISPLAY 'ECH911   - ENTERED FOR PROTECTION EXCP'
        MOVE 'Y' TO ERROR-INDICATOR
        CALL 'CEEMRCE' USING RECOVER-ADDR, FEEDBACK
        IF FB-SEV = ZEROS
          DISPLAY 'ECH911   - CEEMRCE SUCCESSFUL'
        ELSE
          DISPLAY 'ECH911   - CEEMRCE FAILED'
          DISPLAY 'FB-MSG = ' FB-FAC, FB-MSG
        END-IF
        SET RESUME TO TRUE
        DISPLAY 'ECH911   - RESUMING AFTER PROTECTION EXCP'
```

Resume with Move Cursor Explicit...

Program Output:

```
ECHMAIN  - ATTEMPTING PROTECTION EXCP
ECH911   - ENTERED
ECH911   - ENTERED FOR PROTECTION EXCP
ECH911   - CEEMRCE SUCCESSFUL
ECH911   - RESUMING AFTER PROTECTION EXCP
ECHMAIN  - RESUMED AFTER PROTECTION EXCP
```



Simple Percolate

- Condition Handler sets the result_code to indicate that the handler wants to percolate, allowing other condition handlers, if any, to process the condition

Simple Percolate...

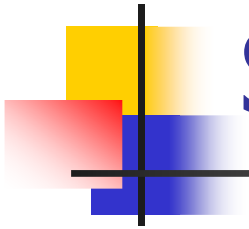
Code from main routine ECHMAIN:

```
*****************************************************
* FORCE A WILD BRANCH TO ADDRESS ZERO, WHICH WILL RESULT
* IN AN OPERATION EXCEPTION. WHEN ECH911 DETECTS THIS
* EXCEPTION, IT WILL REQUEST PERCOLATION.
*****************************************************
     MOVE 'N' TO ERROR-INDICATOR
     DISPLAY "ECHMAIN  - ATTEMPTING WILD BRANCH"
     SET FP TO NULL.
     CALL FP.
* EXECUTION SHOULD NOT REACH HERE
     DISPLAY "ECHMAIN  - AFTER WILD BRANCH"
```

Simple Percolate...

Code in condition handler ECH911 recognizes the Operation Exception (symbolic feedback code CEE341), and sets the result_code to indicate 'percolate':

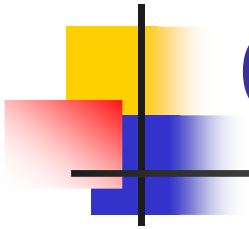
```
WHEN CEE341
  DISPLAY 'ECH911 - ENTERED FOR OPERATION EXCP'
  MOVE 'Y' TO ERROR-INDICATOR
  SET PERCOLATE TO TRUE
  DISPLAY 'ECH911 - PERCOLATING AFTER OPERATION EXCP'
```



Simple Percolate...

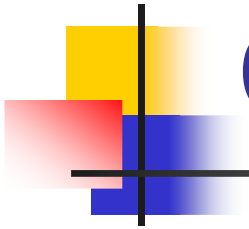
Program Output:

```
ECHMAIN  - ATTEMPTING WILD BRANCH
ECH911   - ENTERED
ECH911   - ENTERED FOR OPERATION EXCP
ECH911   - PERCOLATING AFTER OPERATION EXCP
CEE3201S The system detected an operation exception (System
  Completion Code=0C1)
         From compile unit ECHMAIN at entry point ECHMAIN at
  compile unit offset -20F00000 at entry offset -20F00000
         at address 00000000.
         Possible Bad Branch:  Statement: 113  Offset: +0000099C
ECH911   - ENTERED
ECH911   - CONDITION NOT RECOGNIZED, PERCOLATE
```



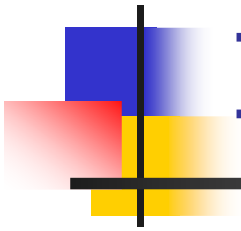
Other Useful Services

- CEE3CIB – Returns pointer to current CIB, Condition Information Block
  - Mapped by CEEBALCI, CEEIBMCI, CEEIGZCI, leawi.h
  - Fields described in LE Vendor Interfaces book
- CEE3GRN - Get the routine name of the offender
- CEE3GRO - Get the offset of the condition
- CEEMOUT - Output a message

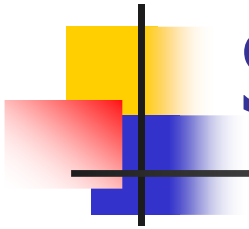


Other Useful Services...

- CEEMSG – get, format, and dispatch a message
- CEE3SPM – query or modify hardware condition
- CEEGQDT – get the q_data token from the ISI
- CEEITOK – return "initial" condition token from current CIB
- CEE3DMP – Ask LE to produce a DUMP

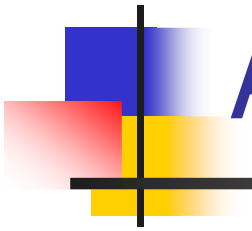


Sources of Additional Information

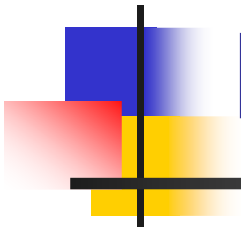


Sources of Additional Info

- All Language Environment documentation is available on the z/OS DVD collection and on the Language Environment website
  - Language Environment Debug Guide
  - Language Environment Runtime Messages
  - Language Environment Programming Reference
  - Language Environment Programming Guide
  - Language Environment Customization
  - Language Environment Migration Guide
  - Language Environment Writing ILC Applications
- Language Environment Web site
  - http://www-03.ibm.com/systems/z/os/zos/features/lang_environment/



Appendix



Related Run-time Options

Related Run-time Options

- ABPERC(NONE) - Percolates (removes from LE condition handling) a single abend code you specify via this option (or CEEBXITA)
- DEPTHCONDLMT(10) – Indicates how deep conditions can be "nested" (how many conditions inside a condition you will tolerate)
- ERRCOUNT(0) – Number of sev 2/3/4 conditions before LE terminates the enclave. Depends on the language (COBOL, PL/1, C/C++)

Related Run-time Options...

- TRAP(ON) – Best to be ON unless instructed otherwise by IBM support!
- XUFLOW –Should exponent underflow cause an interrupt? (PL/I)
- TERMTHDACT(TRACE) –if condition goes unhandled, tells LE the diagnostic documentation to be produced (CEEDUMP, SYSUDUMP, SYSMDUMP)
- ABTERMENC(ABEND) – Indicates whether to end the application with an ABEND or with a return code