UNIT

COBOL for z/OS - Module 10

"It's almost a given and an unavoidable fact of life that there is some amount of invalid data floating around in the files and data bases." z/OS Architect, 2020

ABEND Resolution

- Terms and Concepts
- Types of ABENDs
- Defensive Programming
- Specific ABENDs
- ABEND on purpose

Sources for MVS Completion Codes (related to ABENDs) that you can find on the web:

http://www.jaymoseley.com/hercules/sabends.htm
http://ibmmainframes.com/references/a29.html
http://ibmmainframes.com/topic-42-0-250.html

COBOL Program Big Picture - Topics in Module 10

Identification	Name the executable	Program-ID. PAYROL03.
Environment	Statements that connect the program to Indexed and Sequential data sets.	SELECT <internal file name> ASSIGN TO JCL-DDNAME
Data	Variable declarations - Fields that contain values to be processed in the program's PROCEDURE DIVISION	FILE SECTION FD 77 Standalone variable declaration 01 Data Hierarchy variable definition 05 10 Binary Data: COMP, COMP-3, DISPLAY EBCDIC values REDEFINES 88 Named condition Signed Numeric PIC FILLER VALUE Output/Report Formatting: Z, $, * Suppression, Comma/Decimal Point, BLANK WHEN ZERO
Procedure	Executable statements that process the variable values in the DATA DIVISION	IF/ELSE; How IF tests are evaluated: PIC X fields, PIC 9 fields Compound IF, Nested IF EVALUATE Signed Conditions, Class Conditions MOVE: PIC X MOVE behavior, PIC 9 MOVE behavior Compute/ADD/SUBTRACT/DIVIDE ROUNDED, ON SIZE ERROR DISPLAY GOBACK Code Paragraph PERFORM Paragraph UNTIL <condition> OPEN <Filename>, READ <Filename> AT END, WRITE <Recordname>, CLOSE <Filename> INITIALIZE Counters, Accumulators, Flags Reference Modification Figurative Constants

z/OS ABENDS

- Understanding
- Safeguarding
- Resolving

COBOL Divisions

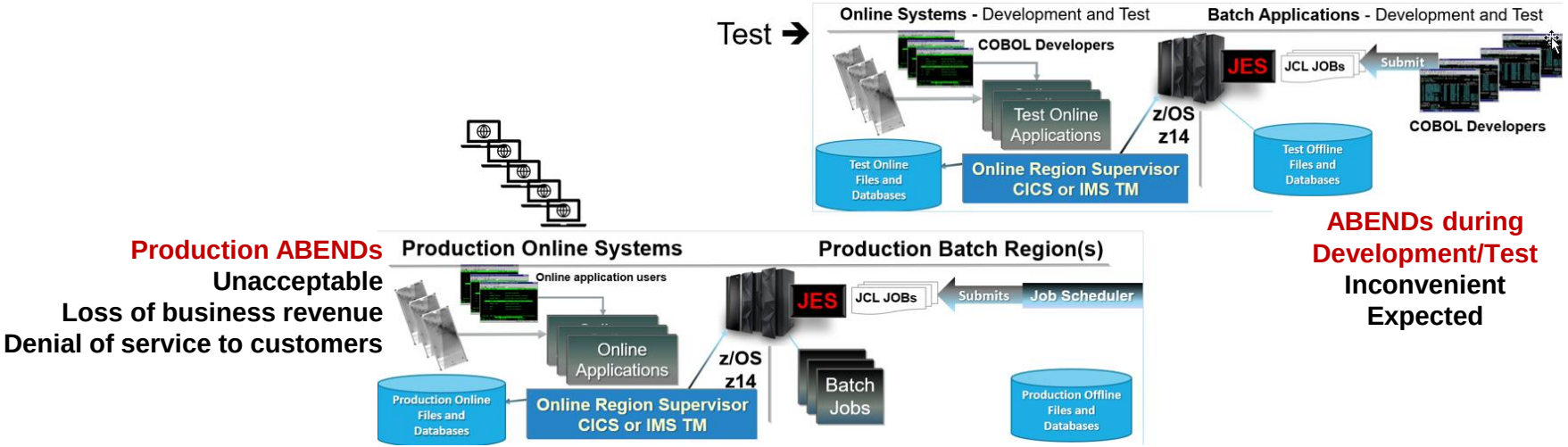
z/OS ABEND (**AB**normal **END** of Task)

An ABEND is a mainframe business application "Blue Screen"

- **Production** business application software errors are costly:
  - While they are nowhere near as expensive as mistakes on an operating table
  - They're more expensive than mixing up the 1% vs. 2% milk in the dairy cabinets…or hitting Reply All when you actually meant to hit Reply 😯
- There are ~ a dozen <u>reasons for</u> COBOL errors which produce ABENDS. Including but are not limited to:
  - **Incorrect data typing of field definitions**
  - **Incorrect subprogram parameter passing order**
  - **Invalid data within files**
    - **Values out of range**
    - **Specific bad values … missing values**
  - **Incorrect record-layout offset definitions**
  - **Programmer/Analyst/Developer errors**
    - **Misunderstanding of the specs** – Typically the most expensive single issue
    - **Incomplete testing** – Second biggest issue
    - **Incomplete understanding of the COBOL language**

ABENDING in **Production** vs. **Test and Development**

- ABENDs during Development & Test are expected
  - Not welcome - but Expected
- ABENDs in Production not welcome, not expected & expensive
  - They can negatively impact corporate financials, market reputation, etc.



ABEND or Invalid Data - Which is worse???

- It is widely held that invalid production data is far worse than MVS ABEND situations:
  - When something ABENDS it **ABENDS**
    - Execution stops
    - z/OS tells you precisely what failed - when & where it failed (the why & how are up to you to discover)
    - Backout routines can be called automatically
    - CHECKPOINT routines can be used to provide point-in-time recovery
  - When applications "go EOJ“
    - Results may (or may not) be correct
      - Sometimes only business experts can validate
    - If results are not correct – we need to assess:
      - What's wrong - was it the data or the code?
      - If it's the code, where in the heck do you start?
        - Backtrack from the program that produced the bad data – or start from the beginning – or the middle, etc.
    - If this was production, invalid values will negatively impact the corporation - not just you or your team
  - Sometimes programs contain their own "self-balancing" defensive-programming:
    - Record in/Record out counters
    - Amounts in/Amounts out as well "trial balances"

ABENDS and COBOL Coding Errors

Typical COBOL ABEND causes for sequential batch applications:

- **Alphanumeric Data:**
  - Truncation
  - Incorrect PIC clause alignment in the record layout
- **Numeric data:**
  - Reference to numeric field that contains non-numeric data
  - Decimal place precision and rounding - esp. with internal variables
- **File Problems:**
  - Read past end of file
  - Reference to file before OPEN or after CLOSE
  - Write loop fills up an output file
- **IF Conditions**
  - Incorrect specification of True/False logic
  - References to numeric fields that contain non-numeric data
- **Programmatic "fall-thru"**
  - COBOL statements execute downwards sequentially - irrespective of paragraph boundaries
- **Unchecked PERFORM UNTIL (Iteration):**
  - Infinite Loops
- **Index issues:**
  - Typically "index out of range"
- **File Handling:**
  - Invalid ASSIGN clause
- **JCL:**
  - Incorrect module name
  - Invalid DD Name
  - Invalid DSN
  - DISP = not correct with READ/WRITE
- **Application Version Control Issues**

Avoiding ABENDS

- **Data:**
  - Truncation**: Understand the COBOL MOVE instruction**
  - Incorrect PIC clause alignment in the record layout: **Align the actual data file to the record layout**
- **Numeric data:**
  - Reference to numeric field that contains non-numeric data: **Liberal use of IF … NOT NUMERIC tests**
  - Decimal place precision and rounding - esp. with internal variables: **Understand the underlying accounting - and use ROUND**
- **File Problems:**
  - Read past end of file: **Debugging, Desk-Checking and Peer Reviews**
  - Reference to file before OPEN or after CLOSE: **Ditto**
  - Write loop fills up the output file: **Understand the record capacity and file Space Allocation. Debug for Infinite Loop**
- **IF Conditions**
  - Incorrect specification of True/False logic**: Debug with "Jump To" function, Flow Charting, Clear understanding of the COBOL semantics and business spec.**
- **Program "fall-thru":**
  - Paragraph Fall Thru: **Debug with "Conditional Watch Monitors and/or code a DISPLAY statement at the top of the paragraph - which names the paragraph.**
  - IF/Conditional Fall /thru: **Ditto**
- **Iteration:**
  - Infinite Loops: **Check for numeric truncation in loop counters**
- **File Handling**
  - Invalid ASSIGN clause: **Vertical split screen, view JCL & Program ENVIRONMENT DIVISION**
- **JCL**
  - Incorrect module name: **Typically easy (JCL Error)**
  - **Invalid DD Name - View ENVIRONMENT DIVISION and batch JCL side-by-side**
  - Invalid DSN: **JCL Error**
  - **File not the correct DCB: Debug with "Conditional Watch Monitors.**
  - **DISP= not correct with READ/WRITE: ABEND upon OPEN <file>.** In general: OPEN INPUT assumes that the file contains data (DISP=SHR) and OPEN OUTPOUT assumes that the file is empty (DISP=NEW). OPEN OUTPUT will over-write the content of a file.

Common COBOL Business Application ABEND Types

There are definitely more ABEND types and situations that you'll see as a COBOL coder.
But understanding these nine common ABENDS in this list will get you started

Also - z/OS will mask or return different system ABENDS than those listed below depending on whether the ABENDS occur in a layer of System Software (CICS, Language Environment)

- **S001** - Record Length/Block Size Discrepancy
- **S013** - Empty File/Record Length/Block Size Discrepancy
- **S0C1** - Invalid Instruction
- **S0C4** - Storage Protection Exception
- **S0C7** - Data Exception
- **S0CB** - Divide by Zero
- **S222/S322** - Time out/Job Cancelled - Infinite Loop
- **S806** - Module Not Found
- **B37/E37** - Out of space (output file)

S001: *Record Length/Block Size - Discrepancy*

Reason(s)

S001-0: Conflict between record length specifications (program vs. JCL vs. dataset label)

S001-2: Damaged storage media or hardware error

S001-3: Fatal QSAM error

S001-4: Conflict between Block specifications (program vs. JCL)

S001-5: Attempt to read past end-of-file

Instructions: OPEN, CLOSE, READ, WRITE

Frequent Coding Causes:

S001-0: Typos in FD or JCL

S001-2: Corrupt disk or tape dataset

S001-3: Internal z/OS problem

S001-4: Forgot to code BLOCK CONTAINS 0 RECORDS in FD (default Block is 1)

S001-5: Logic error (either forgot to close file, or end-of-file-switch not set, overwritten or ignored)

Defensive Programming:

1. Split-Screen COBOL ←→ JCL
2. From JCL: Right-Click on DSN … Open Declaration
3. Select File and verify LRECL from the Properties View

S001: Record Length/Block Size - Discrepancy

Defensive Programming:

1. Split-Screen COBOL ←→ JCL ←→ File Properties
2. From JCL: Right-Click on DSN … Open Declaration
3. Select File and verify LRECL from the Properties View

z/OS Projects

There are no projects in your workspace. To add a project:

Properties | Outline

Property	Value
Attribute	
BLKSIZE	27920
DATA CLAS	**None**
DSNTYPE	SEQ
DSORG	PS
EXTENTS	1 (maximum 16)
5. LRECL	80
MANAGEM	USRMGMT
Num of ger	
PRIMARY	1
RECFM	FB
SECONDAF	1
SPACE UNI	TRACKS
STORAGE C	USRBASE
VOLUME	DMEU24

COBUCLG.jcl

```
----+----1----+----2----+----3----+----4----+----5----+
//LINK.SYSLIB DD DSN=CEE.SCEELKED,
//         DISP=SHR
//         DD DSN=&SYSUID..LEARN.LOAD,
//         DISP=SHR
//LINK.OBJ0000 DD DISP=SHR,
//         DSN=&SYSUID..COBOBJS.OBJ(&COBPGM.)
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
/*
//LINK.SYSLMOD   DD  DISP=SHR,
//         DSN=&SYSUID..LEARN.LOAD(&COBPGM.)
//*
//** Go (Run) Step. Add //DD cards when needed ****
//GO     EXEC   PROC=ELAXFGO,GO=&COBPGM.,
//         LOADDSN=&SYSUID..LEARN.LOAD
//******* ADDITIONAL RUNTIME JCL HERE ******
3. //PAYROLL   DD DSN=&SYSUID..LEARN.PAYROL01,DISP=SHR  4.
//PAYROL3A   DD DSN=&SYSUID..LEARN.PAYROL3A,DISP=SHR
//PAYCHECK DD SYSOUT=*
//ACCTREC   DD DSN=&SYSUID..LEARN.ACCT.DATA,DISP=SHR
//INFILE    DD DSN=&SYSUID..LEARN.ACCT.DATA,DISP=SHR
//OUTFILE   DD DSN=&SYSUID..LEARN.OUTFILE,DISP=SHR
//INVALS    DD DSN=DDS0001.LEARN.INVALS,DISP=SHR
```

S001.cbl

```
----+-*A-1-B--+----2----+----3----+----4----+----5
      *-----------------------
       IDENTIFICATION DIVISION.
      *-----------------------
       PROGRAM-ID.     S001.
       AUTHOR.         ABEND-S001.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT PAYROLL                    2.
           ASSIGN TO UT-S-PAYROLL
             ORGANIZATION IS SEQUENTIAL.
       DATA DIVISION.
       FILE SECTION.
       FD  PAYROLL
           RECORDING MODE IS F
           LABEL RECORDS ARE STANDARD
           RECORD CONTAINS 59 CHARACTERS     1.
           BLOCK CONTAINS 0 RECORDS
           DATA RECORD IS PAYROLL-REC.
       01  PAYROLL-REC   PIC X(59).
       WORKING-STORAGE SECTION.
       01   MONTH-IN                  PIC S9(02)
```

S013: *Conflicting DCB Parameters*

Reason(s)

S013-10:	**Dummy data set needs buffer space; specify BLKSIZE in JCL**
S013-14:	**DD statement must specify a PDS**
S013-18:	**PDS member not found**
S013-1C:	**I/O error search PDS directory**
S013-20:	**Block size is not a multiple of the LRECL**
S013-34:	**LRECL is incorrect**
S013-50:	**Tried to open a printer for Input of I/O**
S013-60:	**Block size not equal to LRECL for unblocked file**
S013-64:	**Attempted to Dummy out indexed or relative file**
S013-68:	**Block size > 32K**
S013-A4:	**SYSIN or SYSOUT not QSAM file**
S013-A8:	**Invalid RECFM for SYSIN/SYSOUT**
S013-D0:	**Attempted to define PDS with RECFM FBS or FS**
S013-E4:	**Attempted to concatenate > 16 PDSs**

Instructions: **`OPEN, CLOSE, READ, WRITE`**

Frequent Coding Causes:
Most of these ABENDs occur running under z/OS (some may not even occur under z/OS, although older modules running on older operating systems (OSVS or VS COBOL II code) that have not been recompiled can produce them). And most are due JCL/COBOL🡐🡒 FD inconsistencies.

Tools to debug **– Static Analysis:** *S013-18*: Same technique as S001

Workshop:

- COBUCLG: **S013**
- COBUCLG: **S0001**

S013: *Block Size - Discrepancy*

Defensive Programming:

1. Delete the BLOCK CONTAINS 0 RECORDS line as shown below
2. Save and COBUCLG the S0001 program
3. Open the IDIREPORT

z/OS Projects

There are no projects in your workspace. To add a project:

Properties | Outline

Property	Value
Attribute	
BLKSIZE	27920
DATA CLASS	**None**
DSNTYPE	SEQ
DSORG	PS
EXTENTS	1 (maximum 16)
LRECL	80
MANAGEMENT	USRMGMT
Num of generat	
PRIMARY	1
RECFM	FB
SECONDARY	1
SPACE UNITS	TRACKS
STORAGE CLAS	USRBASE

*COBUCLG.jcl | S001.cbl

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6-
      AUTHOR.        ABEND-S001.
      ENVIRONMENT DIVISION.
      INPUT-OUTPUT SECTION.
      FILE-CONTROL.
          SELECT PAYROLL
          ASSIGN TO UT-S-PAYROLL
            ORGANIZATION IS SEQUENTIAL.
      DATA DIVISION.
      FILE SECTION.
      FD  PAYROLL
          RECORDING MODE IS F
          LABEL RECORDS ARE STANDARD
          RECORD CONTAINS 80 CHARACTERS
          DATA RECORD IS PAYROLL-REC.
      01  PAYROLL-REC  PIC X(80).
      WORKING-STORAGE SECTION.
      01   MONTH-IN                PIC S9(02)    COMP.
           88 VALID-MONTH VALUES ARE 1 THRU 12.
      01   WS-USER-ABEND-CODE      PIC S9(04)    COMP.
      PROCEDURE DIVISION.
*---  Cause ABEND
```

No BLOCK CONTAINS - defaults to BLOCK CONTAINS 1

DDS0001.DDS0001G.JOB09258.D0000110.?.spool

```
----+----1----+----2----+----3----+----4----+----5----+----6--
*************************************************************
* IBM Fault Analyzer for z/OS V14R1M2 (UI54624  2018/03/21
*
*     Copyright IBM Corp. 2000, 2018.  All rights reserved
*     Copyright HCL Technologies Ltd 2017, 2018.  All righ
*************************************************************

JOBNAME: DDS0001G  SYSTEM ABEND: 013              ESYSMVS


<H1> I B M   F A U L T   A N A L Y Z E R   S Y N O P S I S


A system abend 013 reason code X'18' occurred in module IG
X'15444'.

The abend occurred after executing machine instruction 0A1
IGYCSIMD CSECT IGYVCNTL at offset X'31F4'.

NOTE: Source code information for CSECT IGYVCNTL could not
      no compiler listing or side-file data sets were prov
```

Remote Systems

- DDS0001.LEARN.INVALS
- DDS0001.LEARN.LINER.NOTES
- DDS0001.LEARN.MANU
- DDS0001.LEARN.OUTFILE
- DDS0001.LEARN.PAYCHECK.D
- DDS0001.LEARN.PAYROL01
- DDS0001.LEARN.PAYROL02
- DDS0001.LEARN.PAYROL3A
- DDS0001.LEARN.RENTALS
- DDS0001.LEARN.SALES
- DDS0001.LEARN.SEARCH
- My Favorites
- TSO Commands
- JES
  - Retrieved Jobs
    - DDS0001G:JOB09258 [ABEND
      - JES2::JESMSGLG
      - JES2::JESJCL
      - JES2::JESYSMSG

SOC1: *Invalid Instruction*

Reason(s)

- SYSOUT DD statement missing

- The value in an AFTER ADVANCING clause is < 0 or > 99

- And Index or Subscript is out of range

- An I/O verb was issued against an unopened dataset

- Can also happen of CALL/ENTRY subroutine LINKAGE does not match the calling programs record definition

Instructions:

`OPEN, CLOSE, READ, WRITE`, Table handling routines

- **Note also that during Debug SYSOUT-DISPLAYs are written to the "console"**

Frequent Coding Causes:

- Incorrect logic in setting AFTER ADVANCING variable (or failure to understand 0-99 limits)

- Incorrect logic in table handling code, or number of table entries has overflowed the PIC of variable e.g. PIC 99 (two digits, max) - but there are 100 entries in the table

Tools to debug:

Static

SYSOUT problem: Open multiple windows on AD Batch Job Diagram and program Environment Division - SELECT ASSIGN.

Logic problem: Select File. Use Occurrences in Compilation to isolate statements

Dynamic:

Set Watch Breakpoint and Monitor on table index or AFTER ADVANCING variable.

Set conditional advanced break point on subscript (i.e. SUB<100).

S0C4: *Protection Exception*

Reason(s):

The program is attempting to access a memory address that is not within the application's z/OS "Address Space"

Frequent Coding Causes:

- JCL DD statement is missing or incorrectly coded:
  File Status: 47 upon READ Instruction

- Incorrect logic in table handling code (referencing a table subscript < 1 or > max-table-size)

- INITIALIZE used against a Buffer (file FD) that hasn't been opened.

- Number of table entries has outgrown PIC of variable (i.e. PIC 99, but 100 entries).

.

Tools to debug:

Static

- DD statement problem: Open multiple windows on AD Batch Job Diagram and program Environment Division - SELECT ASSIGN

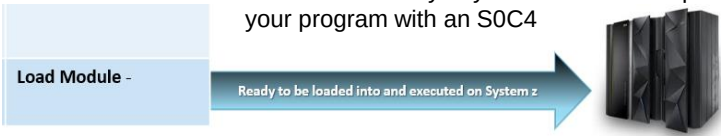
- Incorrect linkage problem:

- Open multiple windows on CALLing and CALLed programs - verify linkage declarations.

Dynamic

The problem with S0C4 ABENDS, is that once they happen - there's nothing left to capture and assist with Debugging.

An "**Address Space**" is a block of virtual memory your Load Module is assigned and runs in, when executing on z/OS. If your program attempts to reference memory beyond the Address Space assigned, z/OS ABENDS your program with an S0C4



Important Note: Compile parameters influence what statements will and won't S0C4

S0C7: *Data Exception*

Reason:

A machine instruction expecting numeric data found invalid data

Instructions:

Arithmetic, IF MOVE (if receiving field is numeric) and **PERFORM VARYING** statements

Your application can S0C7 <u>if the sending field</u> is numeric and contains non-numeric data (**MOVE** pic9field TO picXfield).

Frequent Coding Causes:

- Incorrectly initialized, or uninitialized variable
- Missing or incorrect data edit
- 01 to 01 level **MOVE** if sending field is shorter than receiving field
- Move of Zeros to Group-level numeric fields
- **MOVE CORRESPONDING** incorrect
- **MOVE** field1 to field2 incorrect assignment statements.

Tools to debug:

Static

Occurrences in Compilation Unit on numeric fields

Isolate all PIC 9 Fields

Dynamic

Set Watch points and Monitor on field.

Record the Debug session - Run through to S0C7 and Playback from the ABEND

Locate the field definition - and use client data analysis tools

Solutions:

Add edit checks for valid data in **all numeric fields**

Define all numeric data that does do participate in arithmetic as PIC X

Important Note: <u>Compile parameters influence what statements will and won't S0C7</u>

S0CB: *Divide by Zero*

Reason:

CPU attempted to divide a number by 0.

Instructions:

```
DIVIDE, COMPUTE with / operation
```

Frequent Coding Causes:

- Incorrectly initialized, or un-initialized variable

- Missing or incorrect data edits (i.e. failed to check divisor for zero value)

Tools to debug:

Static

Search for all **DIVIDE** and **COMPUTE** instructions – or using IDz double-click on these verbs and select Filter from the Context Menu

Dynamic

Run through to the S0CB

Locate to field definitions of the offending fields

Solution:

Add edit to check for zero divide:

```
IF divisor > ZERO
THEN
      COMPUTE ...
ELSE
      PERFORM error-processing routine
```

Add **ON SIZE ERROR** to all arithmetic verbs.

S222/S322: *Timeout … Endless Loop*

Reason:

Timeout due to program logic caught in "loop" through instruction set with no exit.

- ***S322 = Timeout***
- ***S222 = Job Cancelled***

Frequent Coding Causes:

- Invalid logic or fall-through logic

- Invalid end-of-file logic

- End-of-file switch overlaid

- Subscript not large enough

- Perform Thru wrong Exit

- PERFORM UNTIL "End-Of-File", but not performing "READ" routine to reach EOF condition

Tools to debug:

Static

Perform Hierarchy/Program Control Flow on logic in PERFORM chain

Desk-Checking for other loop possibilities

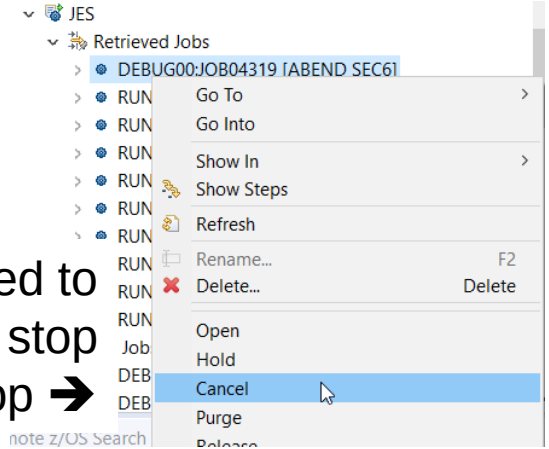
Dynamic tools.

Debug to Loop
Query and Monitor on subscript
Set an Advanced Break Point - Conditional on count

Solution:

For S322 - you may need to increase the TIME=(,n) value in the JCL Job Card
For S222 - you will need to read the code carefully to find one of the Frequent Coding Causes

Note: You will need to Cancel the job to stop the Endless Loop ➔



S806: *Module Not Found*

Reason:

CALL made to program which could not be located along normal search path - which is:

//STEPLIB

//JOBLIB

LINKPACK

Instructions:

Fix the program CALL keyword or the JCL EXEC PGM=XXXX

Frequent Coding Causes:

- Module deleted from library, or never compiled to library

- Module name spelled incorrectly

- STEPLIB does not contain load library with module

- I/O error occurred while z/OS searched the directory of the library

Tools to debug:

Static

Build (Link) Map
Do Remote Systems search on module name – in the Load Libraries

Dynamic

Set Program Advanced Break Point (Entry) to set program break before entry to system.

Solution:

Spell name correctly
Check for 4 or 8 return code from Link Edit (Build step)

Change &COBPGM. ➔ XXXX

```
//
//** Go (Run) Step. Add //DD cards when needed ******
//GO     EXEC    PROC=ELAXFGO,GO=&COBPGM.,
//          LOADDSN=&SYSUID..LEARN.LOAD
//******* ADDITIONAL RUNTIME JCL HERE ******
```

Note: If the initial EXEC PGM=XXXX is incorrect and causes the S806 - the Debugger will **not** start

B37/D37/E37: Dataset or PDS Index Space Exceeded

ABENDS - B37/D37/E37 (RTS-028)

B37: Disk volume out of space.

D37: Primary space exceeded, no secondary extents defined.

E37: Primary and secondary extents full. In TSO, PDS directory needs compress.

E37-04: Disk volume table of contents (VTOC) is full.

Reason:

MVS could not find space for output WRITE to disk

Instructions:

WRITE

Frequent Coding Causes:

- Not enough space initially allocated to output file(s).

- (more likely) Logic error - program in (infinite) loop writing output file(s)
- see S222/S322 reasons.

Tools to debug:

Static – Fault Analyzer will show the DSNs of the out-of-space dataset. As will the JES Output messages

On the host the JCL will show the DDNAME and z/OS filespec of the dataset in question

Dynamic
Set an advanced conditional break point to break on a certain number on iterations

See S222/S322 reasons and solutions
Also, set break point on file WRITE statements

COBOL Program Big Picture - Topics in Module 10

Identification	Name the executable	Program-ID. PAYROL03.
Environment	Statements that connect the program to Indexed and Sequential data sets.	SELECT <internal file name> ASSIGN TO JCL-DDNAME
Data	Variable declarations - Fields that contain values to be processed in the program's PROCEDURE DIVISION	FILE SECTION FD 77 Standalone variable declaration 01 Data Hierarchy variable definition 05 10 Binary Data: COMP, COMP-3, DISPLAY EBCDIC values REDEFINES 88 Named condition Signed Numeric PIC FILLER VALUE Output/Report Formatting: Z, $, * Suppression, Comma/Decimal Point, BLANK WHEN ZERO
Procedure	Executable statements that process the variable values in the DATA DIVISION	IF/ELSE; How IF tests are evaluated: PIC X fields, PIC 9 fields Compound IF, Nested IF EVALUATE Signed Conditions, Class Conditions MOVE: PIC X MOVE behavior, PIC 9 MOVE behavior Compute/ADD/SUBTRACT/DIVIDE ROUNDED, ON SIZE ERROR DISPLAY GOBACK Code Paragraph PERFORM Paragraph UNTIL <condition> OPEN <Filename>, READ <Filename> AT END, WRITE <Recordname>, CLOSE <Filename> INITIALIZE Counters, Accumulators, Flags Reference Modification Figurative Constants

z/OS ABENDS

- Understanding
- Safeguarding
- Resolving

COBOL Divisions

Avoiding ABENDS and COBOL Logic Errors **- Coding Best Practices**

Defensive Programming

- **INITIALIZE** fields at the beginning of a routine
  - Pay particular attention to flags and accumulators
- **I/O Statements:**
  - Use a **FILE STATUS** variable, and always check it
  - Always check for Empty Input files and other possible I/O exceptions
- **Numeric Fields:**
  - Never trust a numeric field that you're doing math on (never assume the data is good)
  - Understand the use of **ROUNDED**
  - Always include **ON OVERFLOW** and **ON SIZE ERROR**
  - If a numeric data item is NOT being used in a calculation, declare it as PIC X
- **Format your code**
- **Consistent use of Scope Terminators:** `END-IF, END-COMPUTE, END-PERFORM, etc.`
- Run your code thru Software Analysis (**Code Review**) scanners
- **Desk Check** and peer-review - i.e. find programmatic & business logic errors
- **Testing -** Methodical, comprehensive Quality Assurance measures are the best defense against ABENDs

COBOL language error trapping clauses

- `ON OVERFLOW` in `STRING` and `UNSTRING` operations
- `ON SIZE ERROR` in arithmetic operations
- Elements for handling input or output errors
- `ON EXCEPTION` or `ON OVERFLOW` in `CALL` statements
- User-written routines for handling errors

Avoiding ABENDs and Software Logic Errors **- ABEND Routines**

ABEND on Purpose

- Data Integrity is ultimately the responsible of your COBOL program. And even if a system ABEND does not occur, there are typically many situations or data results that are unexpected or unacceptable, and you will be expected to call an "ABEND Routine"
  - Values outside allowable ranges i.e. ON OVERFLOW
  - Database/IO calls that detect an error-condition
    - Empty input file
    - Specific record(s) not found
    - Negative return-code from a call
- ABEND routines are typically supplied by corporations - including directions on how to "call" them, when anything that warrants ending your program happens.
- The logic would look something like this:
  - IF ABEND-Condition
    - PERFORM ABEND-ROUTINE
  - ABEND-ROUTINE
    - Set error-message and displays
    - Take steps to end the program

Workshop 10 - Using the Debugger to Experience the Common ABENDs

1. Run **COBUCLD** on the following LEARN.COBOL(Sxxx) programs. Each program will produce the ABEND condition associated with its name.

ABEND Code	Description	Programs that will ABEND
S001	Record Length/Block Size Discrepancy	**S001**
S013	Record Length/Block Size Discrepancy	**S013**
S0C1	Invalid Instruction	**S0C1**
S0C4	Storage Protection Exception	**S0C4** - There are multiple scenarios in the program comments
S0C7	Numeric Data Exception	**S0C7** - There are multiple scenarios in the program comments
S0CB	Divide by Zero	**S0CB**
S222/S322	Time out/Job Cancelled	N/A
S806	Module Not Found	**Mistype the name of: PGM=xxxx in the JCL**
B37/E37	Out of space (output file)	**B37**

2. If you're using zserveros, run these ABEND programs again using **COBUCLG**. From JES, open the file: **GO:RUN:IDIREPRT**. Read the first page of SYSOUT and validate the report with the errant program logic.

3. Using defensive programming techniques - add COBOL code into each of the Sxxx programs so that they do not ABEND

Online sources for MVS ABEND Code research

Online sources for MVS ABEND Code research:

- http://www.jsayles.com/tech/cobol/ABEND.htm
- http://ibmmainframes.com/references/a29.html

Wikipedia DB2/SQLCODES Site: http://en.wikipedia.org/wiki/DB2_SQL_return_codes

IBM SQLCODE and SQLSTATE Analysis:

- http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=%2Frzala%2Frzalaco.htm
- http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.codes%2Fsrc%2Ftpc%2Fdb2z_sqlcodes.htm

Miscellaneous site: http://theamericanprogrammer.com/programming/sqlcodes.shtml

MS ABEND Code Sites:

- http://mainframe230.blogspot.com/2011/04/ims-dbdc-return-codes.html
- http://www.felgall.com/ims1.htm
- http://theamericanprogrammer.com/programming/abend-codes.shtml

Note that these site links are not IBM endorsements - they are simply lists of internet resources

UNIT

COBOL for z/OS - Module 11

Coding Standards and Best Practices - COBOL Quality Measures:

- Capitalization
- Naming conventions
- Coding Style
  - Sentences
  - End-if
  - Use of Periods
  - Input/Output FILE-STATUS Checking
- Grouping W-S functionally
- COBOL Comments

COBOL Program Big Picture - Topics in Module 11

COBOL Divisions		
Identification	Name the executable	Program-ID. PAYROL03.
Environment	Statements that connect the program to Indexed and Sequential data sets.	SELECT <internal file name> ASSIGN TO JCL-DDNAME
Data	Variable declarations - Fields that contain values to be processed in the program's PROCEDURE DIVISION	FILE SECTION FD 77 Standalone variable declaration 01 Data Hierarchy variable definition 05 10 COMP, COMP-3, DISPLAY EBCDIC values REDEFINES 88 Named condition Signed Numeric PIC FILLER, VALUE Output/Report Formatting: Z, $, * Suppression, Comma/Decimal Point, BLANK WHEN ZERO
Procedure	Executable statements that process the variable values in the DATA DIVISION	IF/ELSE; How IF tests are evaluated: PIC X fields, PIC 9 fields Compound IF, Nested IF EVALUATE Signed Conditions, Class Conditions MOVE: PIC X MOVE behavior, PIC 9 MOVE behavior Compute/ADD/SUBTRACT/DIVIDE ROUNDED, ON SIZE ERROR DISPLAY GOBACK Code Paragraph PERFORM Paragraph UNTIL <condition> OPEN <Filename>, READ <Filename> AT END, WRITE <Recordname>, CLOSE <Filename> INITIALIZE Counters, Accumulators, Flags Reference Modification Figurative Constants

COBOL Coding Standards

- Quality
- Consistency
- Compliance

z/OS ABENDS

- Understanding
- Safeguards
- Resolutions

COBOL Coding - Styles and Standards

- Software programming - *and this is especially true of COBOL whose language design was codified in the late 1950's* - is more "art" than "engineering"

- A fundamental mechanism used by corporations to build application portfolios that run their business reliably and efficiently are "Coding Standards"

- **What are Coding Standards?**
  - **Coding Standards are programming policies, guidelines and best practices**
- Different software languages and different application domains have different types of Coding Standards and different levels of required compliance.

- Every shop has its own set of Coding Standards

- For COBOL Business Applications - there are two primary benefits:

  1. Code Quality … 2. Code Consistency

Code Quality

- During this course you will write/test programs measured in the dozens to (low) hundreds of lines
- In the real-world of production business AppDev, you will find no such programs. Real-world COBOL programs are measured in the thousands to tens-of-thousands of lines.**
- To code/test efficiently and effectively, you should learn and adhere to your shop's Code Quality rules.

- **NET:** The next generation of COBOL programmers assigned to maintain and support your code will need to be able to **1;** Understand the details of your program design and **2;** Be able to make changes to your programs with confidence.

** We have actually seen single Business Application programs in excess of 200,000 lines. Coding programs this large is **not** considered a "Best Practice".

Code Consistency

Code Consistency means creating uniform COBOL routines that developers recognize and can; **Fix** when broken, **Maintain** efficiently over time and **Upgrade** with confidence that unforeseen errors will be minimized. Note that COBOL Comments are a major part of what makes programs easy to maintain……..or not.

Some of the categories of Coding Standards for Quality and Consistency:

1. Capitalization
2. Naming Conventions
3. Scope terminators
4. Sentences vs. Statement code-authoring style
5. File-Status
6. Grouping variable types in WORKING-STORAGE
7. READ INTO/WRITE FROM
8. Use of GO TO and other COBOL language features
9. Reusable code libraries
10. Structured COBOL Coding

Capitalization

There are several approaches to capitalizing code, and six COBOL language categories to consider:

Language Categories	UPPER CASE	InitCap (Mixed Case)	lower case
Reserved Words	IF, MOVE, PERFORM, WRITE, READ, SECTION	If, Move, Perform, Write, Read Procedure Division	if, move, perform procedure division.
Variable Names	WS-LINE-COUNTER, LAST-NAME, EMP-SALARY	Ws-Line-Counter, Last-Name, Emp-Salary	ws-line-counter, last-name emp-salary
Paragraph and Section Names	400-READ-FILE 100-HOUSEKEEPING 999-ABEND-ROUTINE	400-Read-File 100-Housekeeping 999-Abend-Routine	400-read-file 100-housekeeping 999-abend-routine
Comments	THIS IS A COMMENT	This Is A Comment	this is a comment
Intrinsic Functions	UPPER-CASE SQRT	Upper-Case Sqrt	upper-case sqrt
Literals / VALUE Clause	TODAY'S DATE IS:	Today's Date Is:	today's date is

Note that you can mix & match the above options and language categories
i.e. Reserved Words and Paragraph Names will be Upper-Case, Comments and Value Clauses should be Mixed Case.

Capitalization - Examples

```cobol
    05  FILLER              PIC X(9) VALUE " VALUE-3:".
    05  VALUE-3             PIC 9(6).
    05  FILLER              PIC X(9) VALUE "S0CB VALS".
    05  ONE-VAL             PIC 9 VALUE 1.
    05  ZERO-VAL            PIC 9 VALUE 0.

PROCEDURE DIVISION.
    PERFORM 000-HOUSEKEEPING THRU 000-EXIT.
    PERFORM 100-MAINLINE THRU 100-EXIT
            UNTIL NO-MORE-TRANSRCH-RECS OR TRAILER-REC.
    PERFORM 400-APPLY-UPDATES THRU 400-EXIT.
    PERFORM 900-CLEANUP THRU 900-EXIT.
    MOVE ZERO TO RETURN-CODE.
    GOBACK.

000-HOUSEKEEPING.
    MOVE "000-HOUSEKEEPING" TO PARA-NAME.
    DISPLAY "HOUSEKEEPING".
    ACCEPT  WS-DATE FROM DATE.
    OPEN INPUT TRMTSRCH-FILE.
    OPEN I-O PATMSTR.
    OPEN OUTPUT SYSOUT.

    READ TRMTSRCH-FILE INTO INPATIENT-TREATMENT-REC
        AT END
        MOVE 'N' TO MORE-TRANSRCH-SW
        GO TO 000-EXIT
    END-READ

    INITIALIZE  COUNTERS-AND-ACCUMULATORS, WS-TRAILER-R
    ADD +1 TO RECORDS-READ.
    MOVE 1 TO ROW-SUB.
    MOVE PATIENT-ID IN INPATIENT-TREATMENT-REC TO
            HOLD-PATIENT-ID.
000-EXIT.
    EXIT.
```

```cobol
    05 Filler                   Pic X(9)      Value " VALUE-3:".
    05 Value-3                  Pic 9(6).
    05 Filler                   Pic X(9)      Value "S0CB VALS".
    05 One-Val                  Pic 9         Value 1.
    05 Zero-Val                 Pic 9         Value 0.

Procedure Division.
    Perform 000-Housekeeping Thru 000-Exit.
    Perform 100-Mainline Thru 100-Exit
       Until No-More-Transrch-Recs Or Trailer-Rec.
    Perform 400-Apply-Updates Thru 400-Exit.
    Perform 900-Cleanup Thru 900-Exit.
    Move Zero To Return-Code.
    Goback.

000-Housekeeping.
    Move "000-HOUSEKEEPING" To Para-Name.
    Display "HOUSEKEEPING".
    Accept Ws-Date From Date.
    Open Input Trmtsrch-File.
    Open I-O Patmstr.
    Open Output Sysout.

    Read Trmtsrch-File Into Inpatient-Treatment-Rec
    At End
       Move 'N' To More-Transrch-Sw
       Go To 000-Exit
    End-Read

    Initialize Counters-And-Accumulators, Ws-Trailer-Rec.
    Add +1 To Records-Read.
    Move 1 To Row-Sub.
    Move Patient-Id In Inpatient-Treatment-Rec To
       Hold-Patient-Id.
000-Exit.
    Exit.

100-Mainline.
    Move "100-MAINLINE" To Para-Name.
```

```cobol
    05 filler                   pic x(9)      value " 
    05 value-3                  pic 9(6).
    05 filler                   pic x(9)      value "s
    05 one-val                  pic 9         value 1.
    05 zero-val                 pic 9         value 0.

procedure division.
    perform 000-housekeeping thru 000-exit.
    perform 100-mainline thru 100-exit
       until no-more-transrch-recs or trailer-rec.
    perform 400-apply-updates thru 400-exit.
    perform 900-cleanup thru 900-exit.
    move zero to return-code.
    goback.

000-housekeeping.
    move "000-HOUSEKEEPING" to para-name.
    display "hOUSEKEEPING".
    accept ws-date from date.
    open input trmtsrch-file.
    open i-o patmstr.
    open output sysout.

    read trmtsrch-file into inpatient-treatment-rec
    at end
       move 'n' to more-transrch-sw
       go to 000-exit
    end-read

    initialize counters-and-accumulators, ws-trailer-re
    add +1 to records-read.
    move 1 to row-sub.
    move patient-id in inpatient-treatment-rec to
       hold-patient-id.
000-exit.
    exit.
```

Variable and Paragraph **Naming Conventions**

- There are dozens of approaches/standards for COBOL program naming conventions
- The goal for all naming standards is "**meaningful**" - as in, the paragraph or variable name should be **easily understood and descriptive of its business content or purpose**
- Besides labels, shops often use COBOL-DIVISION standards such as:
  - **WORKING-STORAGE** variable names start (or end) with **WS-**
  - **LINKAGE SECTION** variable names start (or end) with **LS-**
  - Paragraph names begin with a 3-digit prefix
  - Commonly-required paragraphs should have prescribed names:
    - MAIN, READ, OPEN, CLOSE, ABEND, etc.

```
WS-LAST-NAME
WS-ERROR-REC-KTR
WS-GRAND-TOTAL
…
LS-ACCOUNT-NBR
LS-GROSS-PAY
LS-RETURN-CODE
```

- Additionally, you will find common field **abbreviations** in COBOL programs:
  - Counter ➔ **`KTR, CTR`**
  - Total ➔ **`TOT`**
  - **`NUMBER`** ➔ **`NBR, NO, NM`**
  - Last name/First name ➔ **`LNAME, FNAME, LAST-NM, FIRST-NM`**
  - Date ➔ **`DT, DTE, DAT, CC, YR, MN, MNTH, DY,`**

```
100-HOUSEKEEPING
300-MAIN
400-READ
500-WRITE
900-ERROR-
PROCESSING
…
```

Prefix Variable Naming - Examples

```
100-Main.
    MOVE FIRST-IN         TO  FIRST-OUT.
    MOVE LAST-IN          TO  LAST-OUT.
    MOVE DATE-IN          TO  DATE-OUT.
    MOVE CHECK-NBR-IN     TO  CHECK-NBR-OUT.
    MOVE CITY-STAT-ZIP-IN TO  CITY-STATE-ZIP-OUT.
    MOVE STREET-ADDR-IN   TO  STREET-ADDR-OUT.
    MOVE FUNCTION CURRENT-DATE TO DATE-OUT.
    MOVE NAME             TO NAME-OUT OF LINE1 NAME-OU
    PERFORM  700-PROCESS-CHECK.
    PERFORM 500-Write-Paycheck.
    PERFORM 400-Read-Payroll.
300-Open-Files.
    OPEN INPUT PAYROLL.
    OPEN OUTPUT PAYCHECK.
400-Read-Payroll.
    READ PAYROLL INTO PAYROLL-IN
* Set AT END Switch
        AT END MOVE "Y" TO PAYROLL-EOF
    END-READ.
500-Write-Paycheck.
    WRITE PAYCHECK-REC FROM BLANK-LINE.
    WRITE PAYCHECK-REC FROM LINE1.
    WRITE PAYCHECK-REC FROM LINE2.
    WRITE PAYCHECK-REC FROM LINE3.
    WRITE PAYCHECK-REC FROM LINE4.
    WRITE PAYCHECK-REC FROM LINE5.
600-CLOSE-FILES.
    CLOSE PAYROLL, PAYCHECK.
700-PROCESS-CHECK.
** What if a category other than M, E or H shows up?
    IF CATEGORY-IN = "M"  THEN
     COMPUTE GROSS-PAY-OUT =
                 SALARY-IN * (1 + MANAGEMENT-BONUS-IN)
    ELSE IF CATEGORY-IN = "E"  THEN
     COMPUTE GROSS-PAY-OUT = SALARY-IN
```

```
100-Main.
    MOVE WS-FIRST-IN         TO  FIRST-OUT.
    MOVE WS-LAST-IN          TO  LAST-OUT.
    MOVE WS-DATE-IN          TO  DATE-OUT.
    MOVE WS-CHECK-NBR-IN     TO  CHECK-NBR-OUT.
    MOVE WS-CITY-STAT-ZIP-IN TO  CITY-STATE-ZIP-OUT.
    MOVE WS-STREET-ADDR-IN   TO  STREET-ADDR-OUT.
    MOVE FUNCTION CURRENT-DATE TO DATE-OUT.
    MOVE WS-NAME             TO NAME-OUT OF LINE1 NAME-OUT
    PERFORM  700-PROCESS-CHECK.
    PERFORM 500-Write-Paycheck.
    PERFORM 400-Read-Payroll.
300-Open-Files.
    OPEN INPUT PAYROLL.
    OPEN OUTPUT PAYCHECK.
400-Read-Payroll.
    READ PAYROLL INTO PAYROLL-IN
* Set AT END Switch
        AT END MOVE "Y" TO WS-PAYROLL-EOF
    END-READ.
500-Write-Paycheck.
    WRITE PAYCHECK-REC FROM BLANK-LINE.
    WRITE PAYCHECK-REC FROM LINE1.
    WRITE PAYCHECK-REC FROM LINE2.
    WRITE PAYCHECK-REC FROM LINE3.
    WRITE PAYCHECK-REC FROM LINE4.
    WRITE PAYCHECK-REC FROM LINE5.
600-CLOSE-FILES.
    CLOSE PAYROLL, PAYCHECK.
700-PROCESS-CHECK.
* What if a category other than M, E or H shows up?
    IF WS-CATEGORY-IN = "M"  THEN
     COMPUTE GROSS-PAY-OUT =
                 WS-SALARY-IN * (1 + WS-MANAGEMENT-BONUS-IN
```

+ Can see at-a-glance where the field is declared
- Makes the PROCEDURE DIVISION code bigger

Code Authoring Style - **Sentences vs. Statements**

- From Module 3
  - Paragraphs consist of one to many sentences
    - Sentences consist of one to many statements
      - Statements consist of one to many keywords and variables
- You can end statements with a period

  …or…

- End sentences with a period
  - In which case, you will use explicit scope-terminators in PROCEDURE DIVISION statements
- Pluses to using **sentence-based coding style**:
  - Less possibility of PROCEDURE DIVISION "fall-thru"
  - More of a modern COBOL coding approach
  - The more complex the code, the more beneficial sentence-based coding becomes

```
IF ITEM = "A"
    DISPLAY "THE VALUE OF ITEM IS " ITEM
    ADD 1 TO TOTAL
    MOVE "C" TO ITEM
    DISPLAY "THE VALUE OF ITEM IS NOW " ITEM.
IF ITEM = "B"
    ADD 2 TO TOTAL.
```

```
IF ITEM = "A"
    DISPLAY "THE VALUE OF ITEM IS " ITEM
    ADD 1 TO TOTAL
    MOVE "C" TO ITEM
    DISPLAY "THE VALUE OF ITEM IS NOW " ITEM
END-IF
IF ITEM = "B"
    ADD 2 TO TOTAL
END-IF
```

https://www.ibm.com/support/knowledgecenter/en/SS6SG3_4.2.0/com.ibm.entcobol.doc_4.2/PGandLR/concepts/cppgm42.htm

Explicit Scope Terminators

- A *scope terminator* marks the end of certain PROCEDURE DIVISION statements.
- Explicit Scope Terminators are used in place of periods as a means of documenting the end-scope of COBOL keywords:
  - **IF … … END-IF**
  - **PERFORM … END-PERFORM**
  - **COMPUTE (and other math statements) COMPUTE … … END-COMPUTE**
  - **READ/WRITE - READ … … END-READ WRITE … … END-WRITE**
  - **SEARCH … END-SEARCH**
  - **STRING/UNSTRING - UNSTRING … … END-UNSTRING**
- They are considered a "Best Practice" for COBOL coding - as they explicitly define the extent of the keyword **operation**.

Links to additional scope terminator documentation and learning content:

- http://www.ibm.com/support/knowledgecenter/SS6SG3_4.2.0/com.ibm.entcobol.doc_4.2/PGandLR/ref/rlpdsste.htm
- http://www.naspa.net/magazine/1999/January/T9901012.PDF

- END-ADD
- END-CALL
- END-COMPUTE
- END-DELETE
- END-DIVIDE
- END-EVALUATE
- END-IF
- END-INVOKE
- END-MULTIPLY
- END-PERFORM
- END-READ
- END-RETURN
- END-REWRITE
- END-SEARCH
- END-START
- END-STRING
- END-SUBTRACT
- END-UNSTRING
- END-WRITE

Scope Terminator Examples

```cobol
READ PATMSTR INTO PATIENT-MASTER-REC.
IF PATMSTR-STATUS = "23"
   MOVE INPATIENT-TREATMENT-REC-SRCH TO SYSOUT-REC
   WRITE SYSOUT-REC
   GO TO 400-EXIT.

ADD WS-ANCILLARY-CHARGES, WS-MEDICATION-CHARGES,
    WS-PHARMACY-CHARGES TO PATIENT-TOT-AMT.

PERFORM 425-POSITION-PAT-TABLE-IDX THRU 425-EXIT.

ADD WS-LABTEST-CHARGES, WS-VENIPUNCTURE-CHARGES
                 GIVING TEST-CHARGES (PAT-SUB).

PERFORM 500-RECONCILE-DIAGNOSTIC-CODES THRU 500-EXIT.

MOVE HOLD-LAB-TEST-ID          TO LAB-TEST-S-ID(PAT-SUB).
MOVE HOLD-SHORT-DESC-ID        TO TEST-SHORT-S-DESC(PAT-SUB).
MOVE WS-DATE                   TO LAB-TEST-DATE(PAT-SUB).

REWRITE PATMSTR-REC FROM PATIENT-MASTER-REC
  INVALID KEY
      MOVE "** PROBLEM REWRITING PATMSTR" TO ABEND-REASON
      GO TO 1000-ABEND-RTN.
```

COBOL Statement (period) Coding Style

```cobol
READ PATMSTR INTO PATIENT-MASTER-REC
END-READ

IF PATMSTR-STATUS = "23"
   MOVE INPATIENT-TREATMENT-REC-SRCH TO SYSOUT-REC
   WRITE SYSOUT-REC
   GO TO 400-EXIT
END-IF

ADD WS-ANCILLARY-CHARGES, WS-MEDICATION-CHARGES,
    WS-PHARMACY-CHARGES TO PATIENT-TOT-AMT
END-ADD

PERFORM 425-POSITION-PAT-TABLE-IDX THRU 425-EXIT

ADD WS-LABTEST-CHARGES, WS-VENIPUNCTURE-CHARGES
                 GIVING TEST-CHARGES (PAT-SUB)
END-ADD

PERFORM 500-RECONCILE-DIAGNOSTIC-CODES THRU 500-EXIT

MOVE HOLD-LAB-TEST-ID          TO LAB-TEST-S-ID(PAT-SUB)
MOVE HOLD-SHORT-DESC-ID        TO TEST-SHORT-S-DESC(PAT-SUB)
MOVE WS-DATE                   TO LAB-TEST-DATE(PAT-SUB)

REWRITE PATMSTR-REC FROM PATIENT-MASTER-REC
  INVALID KEY
      MOVE "** PROBLEM REWRITING PATMSTR" TO ABEND-REASON
      GO TO 1000-ABEND-RTN
END-REWRITE
.
```

Scope Terminators - **CONTINUE** vs. **NEXT SENTENCE**

Both NEXT SENTENCE and CONTINUE can be used in an IF statement as a "no-op" - basically a do-nothing IF branch

The distinction between NEXT SENTENCE vs. CONTINUE is critical - particularly if Scope Terminators are a shop standard.

- NEXT SENTENCE will execute the COBOL verb **following the next period (.)**
- CONTINUE will execute the next verb **after the explicit scope terminator** (END-IF)

If you are you are using Scope Terminators, it's safer to use CONTINUE rather than NEXT SENTENCE.

```
MOVE 5 TO AMT-1.
MOVE 9 TO AMT-2.
…
IF AMT-1 IS LESS THAN AMT-2
   NEXT SENTENCE
   DISPLAY 'AMT-1: ' AMT-1
   DISPLAY 'AMT-2: ' AMT-2.
   DISPLAY 'AMT-3: ' AMT-3.
```

If AMT-1 IS LESS THAN AMT-2 NEXT SENTENCE will DISPLAY **only AMT-3**.

Challenge Question - **Why does NEXT SENTENCE work this way?**

```
IF AMT-1 IS LESS THAN AMT-2
   CONTINUE
END-IF
   DISPLAY 'AMT-1: ' AMT-1
   DISPLAY 'AMT-2: ' AMT-2.
   DISPLAY 'AMT-3: ' AMT-3.
```

If AMT-1 IS LESS THAN AMT-2 CONTINUE will DISPLAY **AMT-1, AMT-2 and AMT-3.**

I/O File Status Checking

As a Best Practice, always define and verify every I/O operation in your program:

- QSAM and VSAM files
  - **FILE STATUS field - defined in WORKING-STORAGE**
- IMS 'CBLTDLI' Calls
  - **DL/I Status Code - defined in the LINKAGE SECTION**
- SQL queries
  - **SQLCODE - part of an SQLCA WORKING-STORAGE record structure**

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTERFILE ASSIGN TO AS-MASTERA
    FILE STATUS IS MASTER-CHECK-KEY
    . . .
DATA DIVISION.
. . .
WORKING-STORAGE SECTION.
01  MASTER-CHECK-KEY        PIC X(2).
. . .
PROCEDURE DIVISION.
    OPEN INPUT MASTERFILE
    IF MASTER-CHECK-KEY NOT = "00"
        DISPLAY "Nonzero file status returned from OPEN " MASTER-CHECK-KEY
```

QSAM/VSAM File-Status Checking Pattern:

1. Define a 2-byte PIC X field for each file in WORKING-STORAGE
2. Reference the field with a FILE STATUS IS… statement in the ENVIRONMENT DIVISION
3. Test the file status variable after each I/O operation in the PROCEDURE DIVISION

Follow these rules for each file:

- Define a different file status key for each file.
- Check the file status key after each input or output request.
  - If the file status key contains a value other than 0, your program can issue an error message or can take action based on that value.
- You do not have to reset the file status key code, because it is set after each input or output attempt.
- Comprehensive web-page for File Status code: http://ibmmainframes.com/references/a27.html

File Status - Examples

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SYSOUT
    ASSIGN TO UT-S-SYSOUT
      ORGANIZATION IS SEQUENTIAL.

    SELECT PATSRCH
    ASSIGN TO UT-S-PATSRCH
      ACCESS MODE IS SEQUENTIAL
      FILE STATUS IS PATCODE.

    SELECT WARDFILE
    ASSIGN TO UT-S-WARDRPT
      ACCESS MODE IS SEQUENTIAL
      FILE STATUS IS WARDCODE.

    SELECT PATERR
    ASSIGN TO UT-S-PATERR
      ACCESS MODE IS SEQUENTIAL
      FILE STATUS IS ERRCODE.

    SELECT PATMSTR
           ASSIGN       TO PATMSTR
           ORGANIZATION IS INDEXED
           ACCESS MODE  IS RANDOM
           RECORD KEY   IS PATMSTR-KEY
           FILE STATUS  IS PATMSTR-STATUS.
```

ENVIRONMENT DIVISION

```
           ASSIGN       TO PATPERSN
           ORGANIZATION IS INDEXED
           ACCESS MODE  IS RANDOM
           RECORD KEY   IS PATPERSN-KEY
           FILE STATUS  IS PATPERSN-STATUS.
```

```
01  FILE-STATUS-CODES.
    05  PATMSTR-STATUS          PIC X(2).
        88 PATMSTR-FOUND     VALUE "00".
    05  PATPERSN-STATUS          PIC X(2).
        88 PATPERSN-FOUND     VALUE "00".
    05  WARDCODE                  PIC X(2).
        88 CODE-WRITE     VALUE SPACES.
    05  ERRCODE                  PIC X(2).
        88 CODE-WRITE     VALUE SPACES.
    05  PATCODE                  PIC X(2).
        88 CODE-WRITE     VALUE SPACES.
```

Working-Storage Definitions

```
800-OPEN-FILES.
    MOVE "800-OPEN-FILES" TO PARA-NAME.
    OPEN INPUT PATSRCH
    IF PATCODE NOT EQUAL ZERO
       DISPLAY 'PATSRCH FILE PROBLEM'
       GO TO 1000-ABEND-RTN

    OPEN INPUT PATPERSN
    IF PATPERSON-STATUS NOT EQUAL ZERO
       DISPLAY 'PATPERSON FILE PROBLEM'
       GO TO 1000-ABEND-RTN

    OPEN INPUT PATMSTR
    IF PATMASTER-STATUS NOT EQUAL ZERO
       DISPLAY 'PATPERSON FILE PROBLEM'
       GO TO 1000-ABEND-RTN

    OPEN OUTPUT WARDFILE
    IF WARDCODE NOT EQUAL ZERO
       DI
       GO TO 1000-ABEND-RTN
```

Procedure Division
File OPEN

```
900-READ-WARD-DATA.
    READ PATSRCH INTO INPATIENT-DAILY-REC
        AT END MOVE "N" TO MORE-WARD-DATA-SW
        GO TO 900-EXIT
    IF PATCODE NOT EQUAL ZERO
       DISPLAY 'PATPERSON FILE PROBLEM'
       GO TO 1000-ABEND-RTN
    END-READ

    ADD +1 TO PAT-RECORDS-READ.
900-EXIT.
    EXIT.
```

Procedure Division
File READ

Formatting - "Pretty Printing"

```
01 CLAIM-RECORD-WS.
    05 INSURED-DETAILS.
        10 INSURED-POLICY-NO      PIC S9(7).
        10 INSURED-LAST-NAME      PIC X(15).
        10 INSURED-FIRST-NAME     PIC X(10).
    05 POLICY-DETAILS.
        10 POLICY-TYPE            PIC 9.
            88 PRIVATE            VALUE 1.
            88 MEDICARE           VALUE 2.
            88 AFFORDABLE-CARE    VALUE 3.
        10 POLICY-BENEFIT-DATE-NUM PIC S9(8).
        10 POLICY-BENEFIT-DATE-X REDEFINES
                POLICY-BENEFIT-DATE-NUM PIC X(8).
        10 POLICY-BENEFIT-PERIOD REDEFINES
                POLICY-BENEFIT-DATE-NUM.
            15 POLICY-YEAR        PIC S9(4).
            15 POLICY-MONTH       PIC S9(2).
            15 POLICY-DAY         PIC S9(2).
        10 POLICY-AMOUNT          PIC S9(7)V99.
        10 POLICY-DEDUCTIBLE-PAID PIC S9(4).
        10 POLICY-COINSURANCE     PIC V99.
    05 CLAIM-DETAILS.
        10 CLAIM-AMOUNT           PIC S9(7)V99.
        10 CLAIM-AMOUNT-PAID      PIC S9(7)V99.
```

Line up DATA DIVISION variables based on Level numbers

```
01 - column 8
     05 - column 12
          10 - column 16
```

- VALUE clauses and 88-levels line up
- Indent conditional logic
- "Spacer lines" between statements.

Line up PROCEDURE DIVISION "TO" statements:

Unformatted

```
INSPECT DET-POLICY-NO REPLACING ALL ' ' BY '-'.
MOVE 1 TO INSURED-SUB.
MOVE SPACES TO DET-NAME.
MOVE INSURED-LAST-NAME TO DET-LAST-NAME.
MOVE INSURED-FIRST-NAME TO DET-FIRST-NAME.
MOVE POLICY-BENEFIT-DATE-X TO DET-RENEW-DATE.
MOVE POLICY-DEDUCTIBLE-MET-WS TO DET-DEDUCTIBLE-MET.
MOVE DEDUCTIBLE-PERC TO DET-DEDUCTIBLE-PERC.
MOVE POLICY-DEDUCTIBLE-PAID TO DET-COINSURANCE.
MOVE CLAIM-AMOUNT-PAID TO DET-CLAIM-PAID.
MOVE CLAIM-AMOUNT TO DET-CLAIM-AMOUNT.
 MOVE ALLOWED-AMT TO DET-INSURANCE-TOTAL.
MOVE POLICY-AMOUNT TO DET-CLAIM-AMOUNT.
WRITE PRINT-LINE FROM DETAIL-LINE
   AFTER ADVANCING 2 LINES.
ADD 1 TO LINE-COUNT.
```

Formatted

```
INSPECT DET-POLICY-NO REPLACING ALL ' ' BY '-'.
MOVE 1                        TO INSURED-SUB.
MOVE SPACES                   TO DET-NAME.
MOVE INSURED-LAST-NAME        TO DET-LAST-NAME.
MOVE INSURED-FIRST-NAME       TO DET-FIRST-NAME.
MOVE POLICY-BENEFIT-DATE-X    TO DET-RENEW-DATE.
MOVE POLICY-DEDUCTIBLE-MET-WS TO DET-DEDUCTIBLE-MET.
MOVE DEDUCTIBLE-PERC          TO DET-DEDUCTIBLE-PERC.
MOVE POLICY-DEDUCTIBLE-PAID   TO DET-COINSURANCE.
MOVE CLAIM-AMOUNT-PAID        TO DET-CLAIM-PAID.
MOVE CLAIM-AMOUNT             TO DET-CLAIM-AMOUNT.
MOVE POLICY-AMOUNT            TO DET-CLAIM-AMOUNT.

WRITE PRINT-LINE FROM DETAIL-LINE
   AFTER ADVANCING 2 LINES
ADD 1 TO LINE-COUNT.
```

Formatting - Visual-alignment of procedural flow-dependencies

Indent/Outdent conditional and iterative/UNTIL logic

```
IF  (PHARMACY-COST IN INPATIENT-TREATMENT-REC > 990)
IF  (MEDICATION-COST > 9900.0
OR  MEDICATION-COST < 1.01)
    MOVE "*** INVALID MEDICATION COST" TO
    ERR-MSG IN INPATIENT-TREATMENT-REC-ERR.
    MOVE "Y" TO ERROR-FOUND-SW
    PERFORM 710-WRITE-TRMTERR THRU 710-EXIT
    GO TO 400-EXIT
IF  (PHARMACY-COST IN INPATIENT-TREATMENT-REC > 880)
IF  (ANCILLARY-CHARGE > 900 AND ERROR-FOUND-SW = 'N')
IF  LAB-TEST-ID(ROW-SUB) AND NOT VALID-CATEGORY(ROW-SUB)
  OR  PHARMACY-COST IN INPATIENT-TREATMENT-REC < .88
MOVE "*** INVALID PHARMACY COSTS" TO
     ERR-MSG IN INPATIENT-TREATMENT-REC-ERR.
     IF  (ANCILLARY-CHARGE > 100)
     NEXT SENTENCE ELSE
     IF  (ANCILLARY-CHARGE > 1000
         OR  ANCILLARY-CHARGE < 1.01)
             MOVE "Y" TO ERROR-FOUND-SW
             GO TO 400-EXIT.
     IF VALID-RECORD
         PERFORM 450-CROSS-FIELD-EDITS THRU 450-EXIT.
```

```
IF (PHARMACY-COST IN INPATIENT-TREATMENT-REC > 990)
   IF (MEDICATION-COST > 9900.0
      OR MEDICATION-COST < 1.01)
      MOVE "*** INVALID MEDICATION COST" TO
         ERR-MSG IN INPATIENT-TREATMENT-REC-ERR.
MOVE "Y" TO ERROR-FOUND-SW
PERFORM 710-WRITE-TRMTERR THRU 710-EXIT
GO TO 400-EXIT
IF (PHARMACY-COST IN INPATIENT-TREATMENT-REC > 880)
   IF (ANCILLARY-CHARGE > 900 AND ERROR-FOUND-SW = 'N')
      IF LAB-TEST-ID(ROW-SUB) AND NOT VALID-CATEGORY
         (ROW-SUB)
         OR PHARMACY-COST IN INPATIENT-TREATMENT-REC < .88
         MOVE "*** INVALID PHARMACY COSTS" TO
            ERR-MSG IN INPATIENT-TREATMENT-REC-ERR.
IF (ANCILLARY-CHARGE > 100)
NEXT SENTENCE
ELSE
   IF (ANCILLARY-CHARGE > 1000
      OR ANCILLARY-CHARGE < 1.01)
      MOVE "Y" TO ERROR-FOUND-SW
      GO TO 400-EXIT.
IF VALID-RECORD
   PERFORM 450-CROSS-FIELD-EDITS THRU 450-EXIT.
```

Coding Styles - **Group Related Fields**

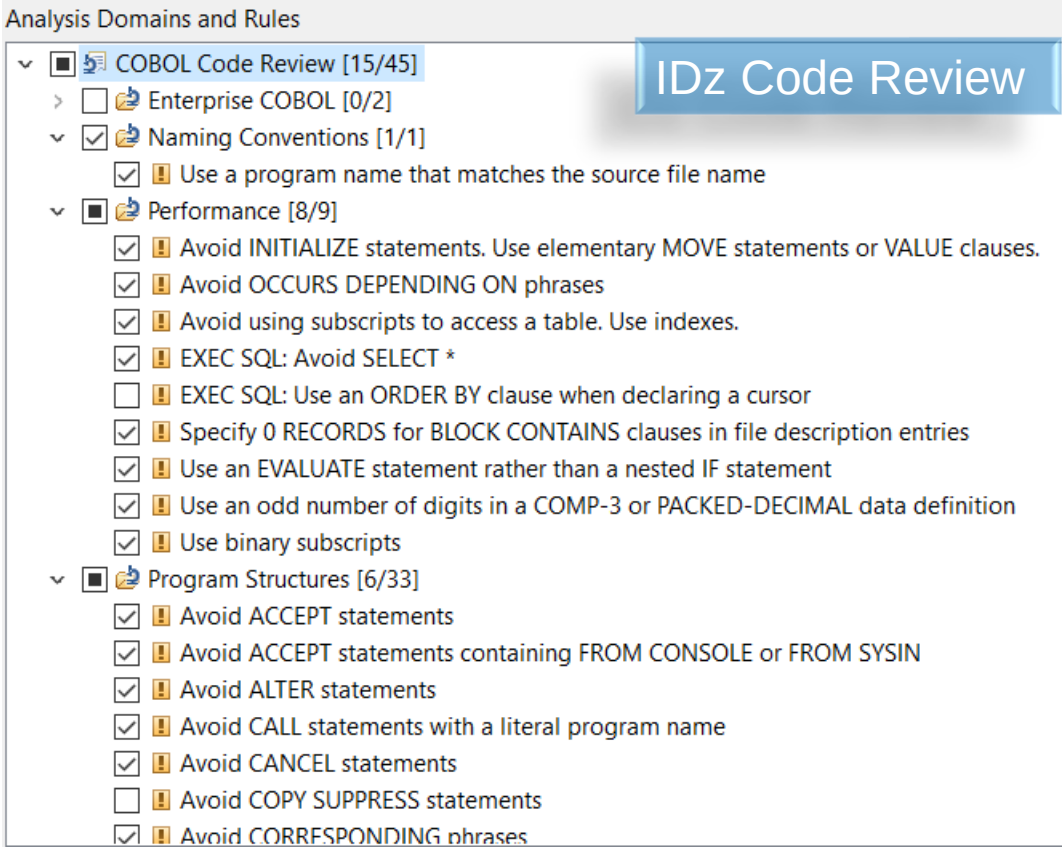
- **Group related Working-Storage fields:**
  - File-Status variables
  - Flags and Switches
  - Counters-and-Accumulators
  - Variables of a given domain
  - ABEND variables
    - What paragraph did the ABEND occur in
    - Any other salient information that can be dynamically carried throughout the code
  - …

```cobol
01 PROGRAM-SWITCHES.
    05 REINSURANCE                PIC XX          VALUE SPACES.
    05 INSURED-SUB                PIC 999         VALUE 1.
    05 CLAIMFILE-EOF              PIC X(1)        VALUE 'N'.
        88 NO-MORE-CLAIMS                          VALUE 'Y'.
    05 CLAIMFILE-ST               PIC X(2).
        88 CLAIMFILE-OK                            VALUE '00'.
    05 PRINTFILE-ST               PIC X(2).
        88 PRINTFILE-OK                            VALUE '00'.
    05 BENEFIT-PERIOD             PIC X(1).
        88 BENEFIT-PERIOD-OK                       VALUE 'Y'.
    05 POLICY-DEDUCTIBLE-MET-WS   PIC X(1).
        88 DEDUCTIBLE-MET                          VALUE 'Y'.
    05 PAY-THE-CLAIM-WS           PIC X(1).
        88 PAY-THE-CLAIM                           VALUE 'Y'.

01 COUNTERS-AND-ACCUMULATORS-WS.
    05 DEDUCTIBLE-WS              PIC S9(4).
    05 CLAIM-PAID-WS              PIC S9(7)V99.

01 DATE-FIELDS-WS.
    05 CURR-DATE-OUT              PIC X(10).
    05 CURR-DATE-WS               PIC S9(8).
    05 CURR-DATE-WS-X REDEFINES CURR-DATE-WS.
        10 WS-YEAR                PIC X(4).
        10 WS-MONTH               PIC X(2).
        10 WS-DAY                 PIC X(2).

01 REPORT-FIELDS.
    05 LINE-COUNT                 PIC S9(2)       VALUE +6.
    05 PAGE-COUNT                 PIC S9(2)       VALUE ZEROS.
    05 LINES-PER-PAGE             PIC S9(2)       VALUE +5.
```

Use of GO TO and other COBOL Language Options

Most COBOL shops have an extensive list of COBOL language do's and don'ts:

- **Do use Scope Terminators**
- **Do use Sentence Structure**
- **Do not use a number of COBOL verbs:**
  - ALTER
  - GO TO
  - ACCEPT
  - PERFORM THRU
  - OCCURS DEPENDING ON
  - …



Compliance with shop rules is enforced by; **Peer Reviews, REXX/CLIST** scripts or **IDE-based Tools**

COBOL Comments

The semantics of a COBOL program are captured in the code**. And before changing one sentence, it's your responsibility to understand the semantics of the code. Comments are the only mechanism by which you document **WHAT** a program does as well as **HOW** a program works.

Typically you will see comments in three places:

1. At the top of the program
   - A general description of the entire program. And other specifics such as a maintenance log, change dates, reasons, names, files, etc.
2. At the top of each paragraph:
   - Describing the key functionality in the paragraph - along with any useful business-level doc, and a maintenance log, etc.
3. Inline with the code
   - The Compiler will ignore whatever is to the right of *>
   - So developers can craft annotations, reminders, etc. embedded in the code itself

```
PERFORM C000-GENERATE-REPORT2 THRU  *> Update to the reports
   C010-GENERATE-REPORT2-EXIT
     UNTIL SQLCODE NOT EQUAL TO ZERO.
```

3.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  TRTMNT.
000300 AUTHOR. JON SAYLES.
000900*****************************************************************
001000*REMARKS.
001100*
001200*        THIS PROGRAM EDITS A DAILY TREATMENT TRANSACTION FILE
001300*        PRODUCED BY DATA ENTRY OPERATORS FROM CICS SCREENS
001400*
001500*        IT CONTAINS EVERY TREATMENT FOR EVERY PATIENT IN THE
001600*        HOSPITAL.
001700*
001800*        THE PROGRAM EDITS EACH RECORD AGAINST A NUMBER OF
001900*        CRITERIA, BALANCES FINAL TOTALS AND WRITES GOOD
002000*        RECORDS TO AN OUTPUT FILE
002100*
002100*        NOTE: Y2K WINDOWING USED ON ALL DATE FIELDS
002100*
002200*****************************************************************
002300
002400        INPUT FILE              -   DDS0001.TRMTDATA
002500
002600        VSAM MASTER FILE        -   DDS0001.PATMASTR
```

1.

```
A100-DONE1.
*****************************************************************
*  For all PROJs anding at a date later than the RAISE-         *
*  DATE ( i.e. those PROJs potentially affected by the          *
*  salary raises generate a report containing the PROJ          *
*  PROJ number, PROJ name, the count of EMPs                    *
*  participating in the PROJ and the total salary cost          *
*  for the PROJ                                                 *
*****************************************************************
    MOVE SPACES TO PRINT-RECORD.
    WRITE PRINT-RECORD BEFORE ADVANCING 2 LINES.
    WRITE PRINT-RECORD FROM RPT2-HEADER1
          BEFORE ADVANCING 2 LINES.
    WRITE PRINT-RECORD FROM RPT2-HEADER2
          BEFORE ADVANCING 1 LINE.
    WRITE PRINT-RECORD FROM RPT2-HEADER3
```

2.

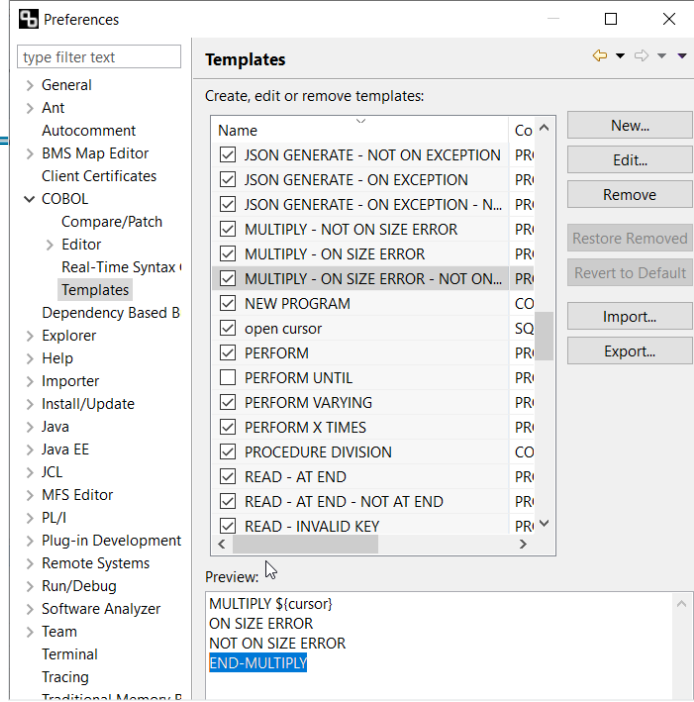
** Critically, program semantics are not only defined by variables in the DATA DIVISION, they are buried in the procedural structure of the performed paragraphs that process the data

Approaches to Coding Standards Compliance

- **Tools-based approach:**
  - There are a number of tools on the market from different software vendors
    - **Application Discovery** - An Enterprise-wide COBOL application analysis product
    - **IDz Code Review** - A collection of coding rules and Best Practices packaged with IDz
    - Sonarsource: https://www.sonarsource.com/cobol/
- **Manual Desk-Checking/Peer Reviews:**
  - Many shops have created a formal list of coding standards in MS-Excel, MS-Word, etc.
  - COBOL code is compared to, or reviewed against standards, line-by-line
- **In-house developed "Applets":**
  - There are two z/OS "command languages" - **REXX** and **CLIST** language that are used to parse application programs
    - The parsing typically based on the coding standard documents
  - The REXX or CLIST Applets compare COBOL code electronically

Reusable Code Libraries

- IDz provide a collection of sample statement templates ➔
- The templates can be **Ctrl+Spacebar** inserted into your COBOL program. The code in the templates follows current Best Practices standards - including Scope Terminators. You can modify the provided Templates - creating your own library of reusable statements based on your shops Code Review Standards
- You're also encouraged to create your own reusable code library consisting of tested COBOL routines/examples:
  - **Numeric calculation**
  - **Nested IF/Evaluate**
  - **Date & Time handling snippets**
  - **Sequential File Handling examples**
  - **ABEND routines**



- Shops using TSO/ISPF often implement the same "Reusable Code Library" principle - by creating PDS (Libraries) that contain:
  - Program skeletons for different kinds of projects
  - "Copybook" and "Include" libraries that contain:
    - Predefined WORKING-STORAGE record definitions
    - PROCEDURE DIVISION code for common routines (I/O, ABEND handling, Complex computations, etc.)
- The COBOL content in these resources can be either copied in on-demand, using ISPF Command Line "Copy" - or using the COBOL: COPY keyword - which we'll learn about later in this course

COBOL Program Big Picture - Topics in Module 11

Identification	Name the executable	Program-ID. PAYROL03.
Environment	Statements that connect the program to Indexed and Sequential data sets.	SELECT <internal file name> ASSIGN TO JCL-DDNAME FILE SECTION FD
Data	Variable declarations - Fields that contain values to be processed in the program's PROCEDURE DIVISION	77 Standalone variable declaration 01 Data Hierarchy variable definition 05 10 Binary Data: COMP, COMP-3, DISPLAY EBCDIC values REDEFINES 88 Named condition Signed Numeric PIC FILLER VALUE Output/Report Formatting: Z, $, * Suppression, Comma/Decimal Point, BLANK WHEN ZERO
Procedure	Executable statements that process the variable values in the DATA DIVISION	IF/ELSE; How IF tests are evaluated: PIC X fields, PIC 9 fields Compound IF, Nested IF EVALUATE Signed Conditions, Class Conditions MOVE: PIC X MOVE behavior, PIC 9 MOVE behavior Compute/ADD/SUBTRACT/DIVIDE ROUNDED, ON SIZE ERROR DISPLAY GOBACK Code Paragraph PERFORM Paragraph UNTIL <condition> OPEN <Filename>, READ <Filename> AT END, WRITE <Recordname>, CLOSE <Filename> INITIALIZE Counters, Accumulators, Flags Reference Modification Figurative Constants

z/OS ABENDS
- Understanding
- Safeguards
- Resolutions

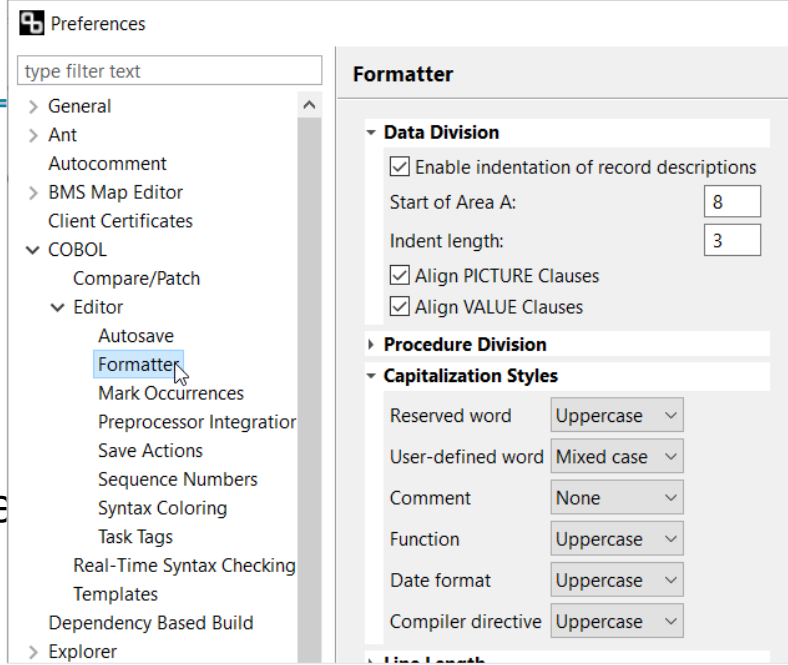
COBOL Coding Standards
- Quality
- Consistency
- Compliance

COBOL Divisions

Workshop 11.1

Steps:

1. If using IDz - Set your own custom Formatter options for;
   - Capitalization Styles
   - Procedure Division
2. Open LEARN.COBOL(FORMATER) and browse around inside the source file
   - From the Context Menu, right-click and select:
     - Source ➔ Format ➔ Yes
   - Browse again throughout the file, and note the differences
   - Repeat this exercise with CBL0009
3. Open PAYROL3A.
   - Add Scope Terminators to paragraphs 300, 600 and 700
   - Re-write the PROCEDURE DIVISION, turning all of the COBOL statements ending with periods into Sentence structure code.



COBOL Source Code Standards Online

https://www.ibm.com/support/knowledgecenter/SSUFAU_1.0.0/com.ibm.rsar.analysis.codereview.cobol.doc/topics/cac_anconfig_describerules.html - Comprehensive Coding Standards documentation

http://mainframe-tips-and-tricks.blogspot.com/2012/11/sample-cobol-coding-standard.html - Naming Conventions and Coding Practices

https://www.unf.edu/~broggio/cop3531/standard.html - coding standards broken out by COBOL Divisions

https://www.tonymarston.net/cobol/cobolstandards.html - the COBOL language per se

https://www.csus.edu/indiv/c/christenson/DL_Files/CobolStandards.pdf - Business COBOL coding standards. Unsparing details

https://ags.hawaii.gov/wp-content/uploads/2012/09/ITS_1110.pdf - Business COBOL coding standards. Unsparing details

https://www.cusys.edu/pubs/1cobol.html - COBOL Programming Language break-down of coding standards

http://ibmmainframes.com/references/a7.html - Detailed (COBOL statement-level) collection of practical coding "Do's & Don'ts"

Common COBOL Coding Errors

http://www.mainframes360.com/2009/08/cobol-tutorial-compiling-linking-and.html

Note that these site links are not IBM endorsements - they are simply lists of internet resources

Coding Styles - **Tools**

- IDz provids automated code formatting capabilities:
  - Source ➔ Format
- The rules can be customized from:
  - Window ➔
    - Preferences