

IBM Software Group

IBM Developer for z Systems – for ISPF Developers

Module 9 – Debugging COBOL with IDz



DevOps

Jon Sayles, IBM z Products - jsayles@us.ibm.com

© 2019 IBM Corporation

@Copyright March, 2019

IBM Trademarks and Copyrights

© Copyright IBM Corporation 2008 through 2019.

All rights reserved – including the right to use these materials for IDz instruction.

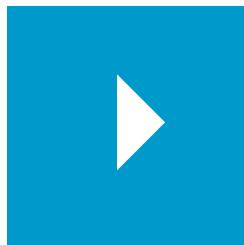
The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo, the on-demand business logo, Rational, the Rational logo, and other IBM Rational products and services are trademarks or registered trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

UNIT

The IDz Workbench



Topics:

- **Debugging z/OS COBOL Batch Applications**
- Debugging z/OS COBOL Online Applications
- Appendix

Topic Considerations



Note: In this topic you will learn how to debug a COBOL program running on a z/OS mainframe. The screen captures all describe connecting to a public z/OS machine that IBM makes available – during classes.

If you are taking this course through standard IBM services delivery you should be able to use the properties (I/P address, port#s, etc.), logon IDs and passwords that your instructor provides you with.

But you may also be taking this course standalone – and in that case, you will need to speak to your company's Systems Programming staff to learn how to connect and logon.

It goes without saying that the actual file names in the screen captures of mainframe libraries and datasets will vary. So you should focus on the process and steps and "how to" – and don't be perplexed at differences in screen captures.

You also may be using your company's own Source Control Management system – to do things like builds, compiles, etc. In that case much of the remote functionality in IDz will be customized and tailored to your company's unique and idiosyncratic procedures and protocols.

Topic Objectives

After completing this unit, you should be able to:

- Describe the concept of source code debugging
 - List the run-times that the IDz Integrated Debugger supports
 - List the steps in preparing a program for debugging
 - Debug a mainframe batch job
 - Describe the run/step/animate options
 - List PF-Keys associated with them
 - Set/unset/inspect conditional and unconditional break-points
 - Set "watch" break-points that halt execution when a value in a variable changes
 - Show how to access the LPEX editor functionality during debugging (such as Perform Hierarchy)
 - Be able to Jump to any given line, and run to a line
 - Show how to change variable values dynamically during debug
 - Show how to set different levels of variable display
 - Monitor specific variables you are interested in
 - Debug a CICS online transaction
 - Discuss the Debug Option setup and configuration requirements for Online Debugging
 - CADP Profile/View
 - CADP Transaction
 - Launch a CICS transaction that invokes the IDz Integrated Debugger

Debugging Overview

Face facts: **No one gets it right the first time.**



- ▶ Not at the level of production business logic

That's why IBM invented source-level application debuggers, so that you can:

- ▶ View program execution, line-by-line
- ▶ Verify the value of a variable – during program execution
- ▶ Stop and start program execution, and analyze results at the speed that our procedural understanding of the application's execution flow can handle

Debug Product Packaging Options

1. IDz Integrated Debugger - Packaged and free with IDz/IDz
 - ▶ Became **IBM z/OS Debugger** with IDz v14 and later
2. IBM Debug Tool:
 - ▶ Became **IBM Debug for z Systems** with IDz v14 and later
 - ▶ Integrated with ADFz and Standalone as PD Tools Replacement
 - ▶ Must own IBM Debug Tool on the host
 - ▶ Includes Green-Screen (3270 Debugging)
3. Compuware's Xpeditor product line:
 - ▶ Requires Xpeditor host software + plug-ins to IDz (from Compuware)
4. CA-Intertest
 - ▶ Requires Intertest host software + plug-ins to IDz (from Computer Associates)
5. ASG SmartTest
 - ▶ Requires Intertest host software + plug-ins to IDz (from Allen Systems Group)

It is up to you to know – or find out which if the above debugging products is installed on your mainframe. Typically a tech-lead or Project Manager knows – but your Systems Programming staff? Always know what's installed.

IBM Debuggers – Runtime & Feature Support

- Online
 - ▶ CICS & IMS TM
- Batch including Batch DB2 and Batch IMS (DL/I and BMP, QBMP)
- Enterprise COBOL (5.1, 4, 3.4) – Debug Tool supports OS/VS and VS COBOL II
- Enterprise PL/I
- C
- HLASM and BAL

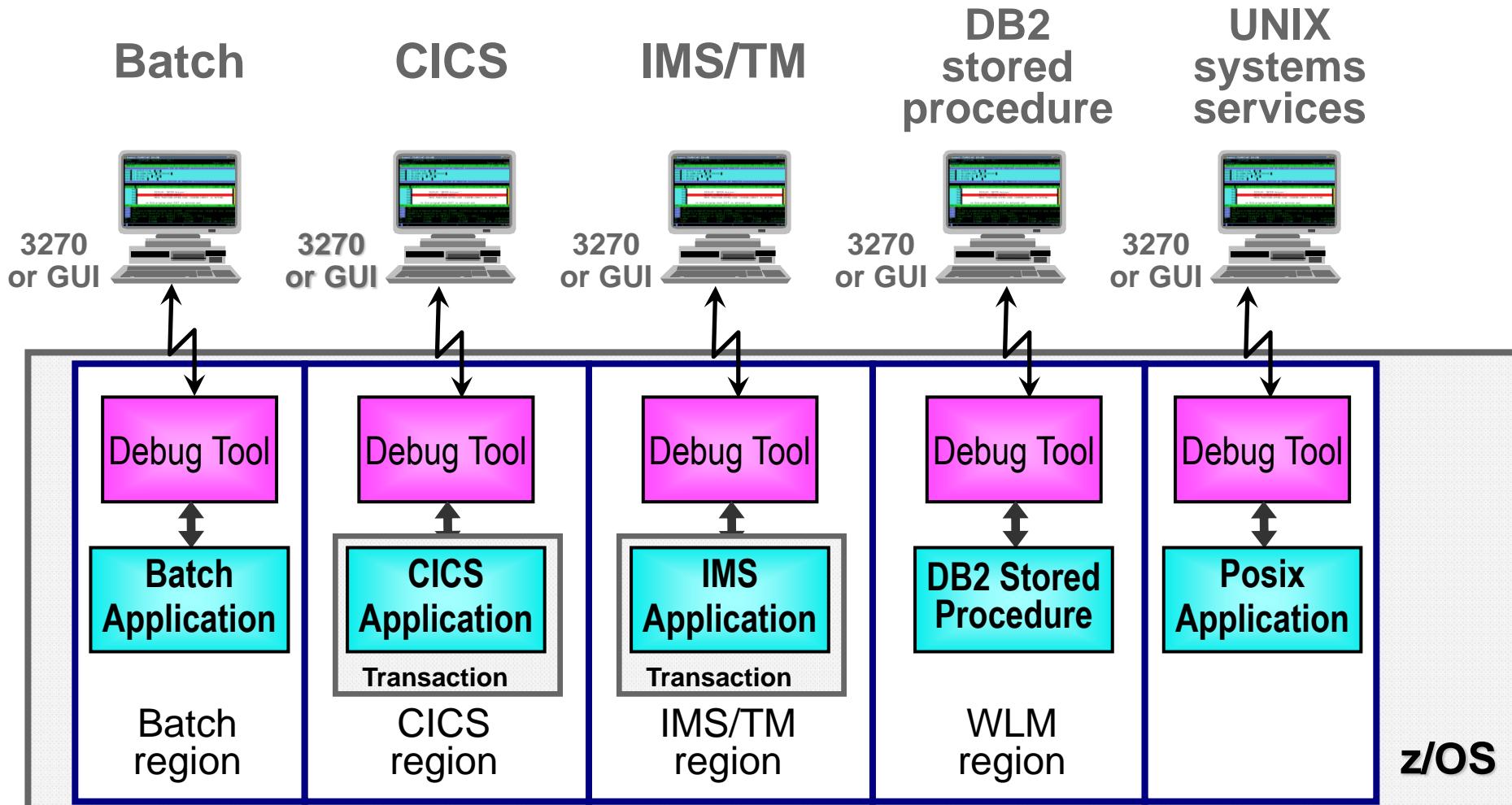
Consult User Documentation for complete list of language and language version runtimes supported

Features (partial list):

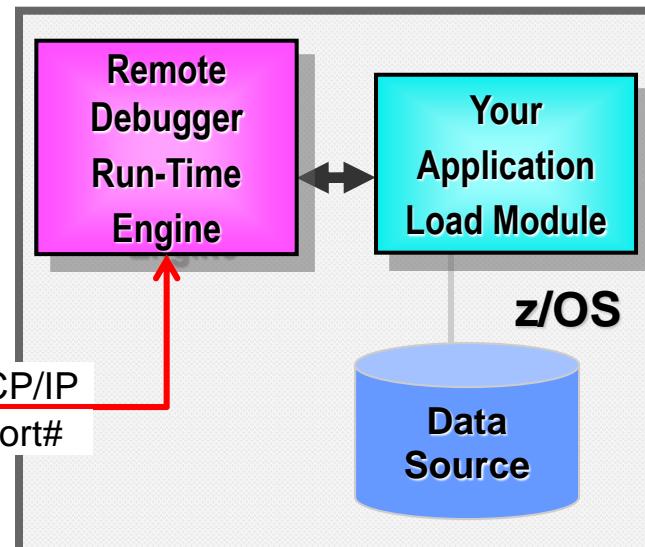
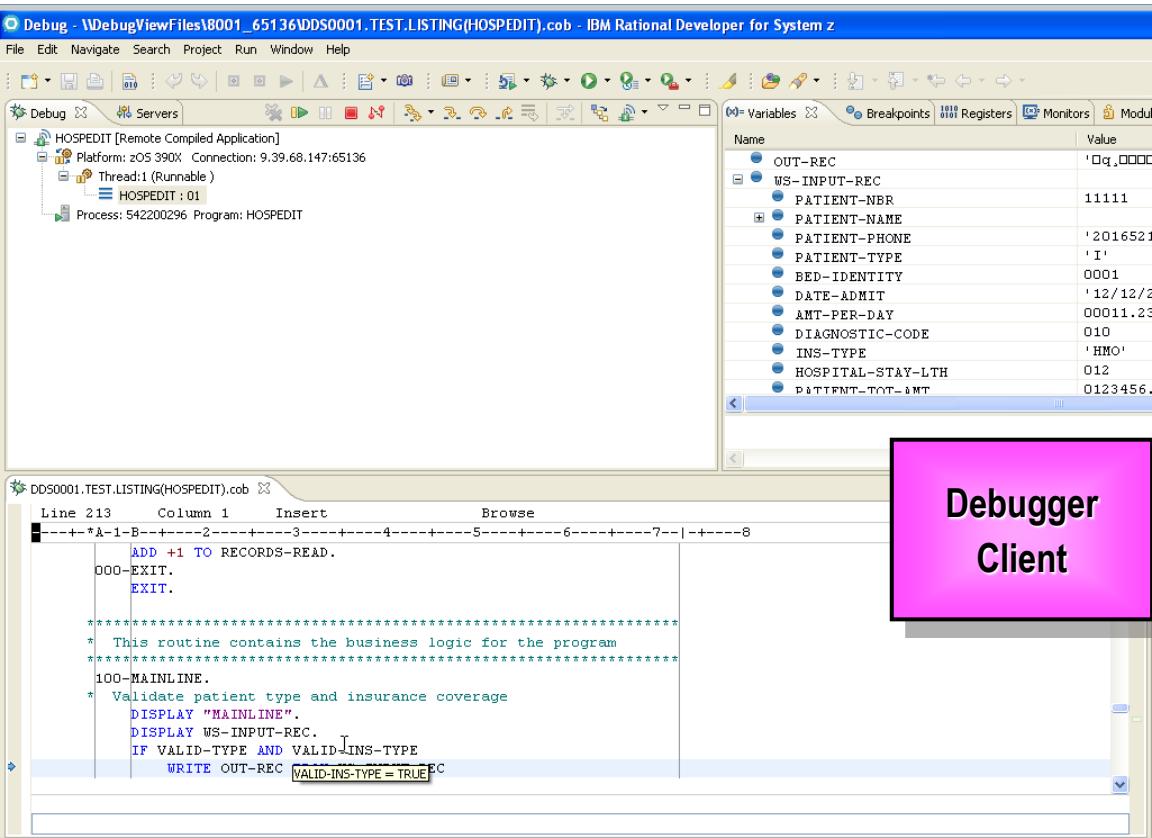
- ▶ Debug mixed-language/cross-platform applications
- ▶ Interactive, source-level debugging in IDz with program running on z/OS
- ▶ Display, monitor and alter program variables
- ▶ Breakpoints – all shapes and sizes – including the ability to set breakpoints in the program source file
- ▶ Watch monitors – Integrated Debugger requires @v9.5 and later
- ▶ Record/Playback of instructions – Debug Tool only
- ▶ View data in Hex (EBCDIC) or string values
- ▶ Visual Debugging – Integrated Debugger @v95 and later
- ▶ Full asynchronous mode

Debug Tool - Application Environments

One debugging engine, with support for many environments:



Interfacing IDz with the Debugger



Architectures:

- ▶ Client software is installed with IDz on your workstation
- ▶ Communicates with the Debugger run-time engine on the mainframe
 - Establishes handshake using combination of: TCP/IP address + Port#
 - You can “reuse” IDz Remote Systems connection

Steps for Batch Application Debug Session

1. Ensure that your compile proc has the necessary **TEST** parameter, Compile/Link options and DD cards to create a debug-ready load module – and Compile/Link your program
2. *Discover workstation TCP/IP parameters (Only needed for RDz v9 releases – or – if your shop requires):*
 - ▶ *IP Address ... Listener port#*
3. Specify TCP/IP address of workstation in run JCL for Debug Tool DD statement – or if you're using IDz v14 specify a TEST literal and Submit the JCL
 - *TEST literals are covered in this deck*
4. Submit the JCL to Debug the application

Debug Tool – Compiler Requirements

- Use the **TEST** compiler option to prepare your executable COBOL program for use with the debugger.
 - ▶ The **TEST** option is *required* for remote debugging. It produces symbol and statement information that enables the debugger to perform symbolic source-level debugging
 - ▶ Include the DD card for your **SYSDEBUG** dataset in the COBOL Compile step
 - In traditional compile JCL – this would be in the **IGYCTRL** step of your Compile PROC
 - If you are not using the IBM/IDz compile PROCs for building your applications, you'll need to add **TEST** - as shown above

```
//STP0000 EXEC PROC=ELAXFCOC,
// CICS=,
// DB2=,
// COMP=,
// PARM.COBOL=( 'SQL',
// 'LIB', 'TEST(NONE,SYM,SEP') ')
//COBOL.SYSPRINT DD DISP=SHR,
// DSN=DDS0001.TEST.LISTING(TRTMNT)
//COBOL.SYSDEBUG DD DISP=SHR,
// DSN=DDS0001.TEST.SYSDEBUG(TRTMNT)
//COBOL.SYSLIN DD DISP=SHR,
// DSN=DDS0001.TEST.OBJ(TRTMNT)
//COBOL.DBRMLIB DD DISP=SHR,
// DSN=DDS0001.TEST.DBRMLIB(TRTMNT)
//COBOL.SYSLIB DD DISP=SHR,
// DSN=DDS0001.TEST.COPYLIB
// DD DISP=SHR
//COBOL.SYSXMLSD DD DUMMY
//COBOL.SYSIN DD DISP=SHR,
// DSN=DDS0001.TEST.COBOL(TRTMNT)
//*
//LKED EXEC PROC=ELAXFLNK
//LINK.SYSLIB DD DSN=DDS0001.TEST.OBJ,
// DISP=SHR
// DSN=CEE.SCEELKED,
// DISP=SHR
//LINK.OBJ0000 DD DISP=SHR,
// DSN=DDS0001.TEST.OBJ(TRTMNT)
//LINK.SYSLIN DD *
// INCLUDE OBJ0000
/*
```

Integrated Debugger – COBOL V4

▪ Build and Run-time JCL

► COBOL compile options: **SOURCE, LIST, XREF, MAP, LIB**

► Run-time JCL:

//STEPLIB – DD Cards for the IDz
Run-Time Libraries – unless Run-Time installed in Link Pack

//AQEV4LST - Points to your COBOL compiler listing library

▪ //CEEOPTS

- TEST(,,DBM)
 - Invokes/connects to the debug listener on z/OS
- ENVAR("EQA_STARTUP_KEY=CC")
 - Invokes Code Coverage – instead of Debug

```
//DDS0001L JOB REGION=4M,CLASS=A,  
// TIME=(1),MSGCLASS=H,NOTIFY=&SYSUID,MSGLEVEL=(1,1)  
//*****  
//RUNSAM1 EXEC PGM=SAM3  
//STEPLIB DD DSN=DDS0001.TEST.LOAD,DISP=SHR  
//          DD DSN=RATRDZ.V9R5.SFEKLOAD,DISP=SHR  
//          DD DSN=RATRDZ.V9R5.SFEKAUTH,DISP=SHR  
//AQEV4LST DD DISP=SHR,DSN=DDS0001.TEST.LISTING  
//CEEOPTS DD *  
TEST(,,DBM)  
/*           ,ENVAR("AQE_STARTUP_KEY=CC")  
//CUSTFILE DD DSN=DDS0001.TEST2.SAMFILE,DISP=SHR  
//SYSPRINT DD SYSOUT=*  
//SYSOUT DD SYSOUT=*  
//CUSRPT DD SYSOUT=*  
//CUSTOUT DD SYSOUT=*  
//TRANFILE DD DSN=DDS0001.TRANFILE,DISP=SHR  
/* IDIOPTS CAN BE USED TO SPECIFY FAULT ANALYZER  
//IDIOPTS DD *  
INCLUDE,MAXMINIDUMPPAGES(1000)  
NODUP(NORMAL(0))  
/*  
//IDITRACE DD SYSOUT=*
```

Integrated Debugger – COBOL V5

▪ Build and Run-time JCL

COBOL compile options: **TEST**

Run-time JCL:

//STEPLIB – DD Cards for the IDz
& Debug Run-Time Libraries –
▪ Unless Run-Time installed in Link Pack

//CEEOPTS

- **TEST(,,DBM)**
 - Invokes/connects to the debug listener on z/OS
 - **ENVAR("EQA_STARTUP_KEY=CC")**
 - Invokes **Code Coverage**

```
//DDS0001L JOB REGION=4M,CLASS=A,  
// TIME=(1),MSGCLASS=H,NOTIFY=&SYSUID,MSGLEVEL=(1,1)  
//*****  
//RUNSAM1 EXEC PGM=SAM3  
//STEPLIB DD DSN=DDS0001.TEST.LOAD,DISP=SHR  
//          DD DSN=RATRDZ.V9R5.SFEKLOAD,DISP=SHR  
//          DD DSN=RATRDZ.V9R5.SFEKAUTH,DISP=SHR  
///*AQEV4LST DD DISP=SHR,DSN=DDS0001.TEST.LISTING  
//CEEOPTS DD *  
TEST(,,,DBM)  
/*           ,ENVAR("AQE_STARTUP_KEY=CC")  
//CUSTFILE DD DSN=DDS0001.TEST2.SAMFILE,DISP=SHR  
//SYSPRINT DD SYSOUT=*  
//SYSOUT DD SYSOUT=*  
//CUSRPT DD SYSOUT=*  
//CUSTOUT DD SYSOUT=*  
//TRANFILE DD DSN=DDS0001.TRANFILE,DISP=SHR  
//* IDIOPTS CAN BE USED TO SPECIFY FAULT ANALYZER  
//IDIOPTS DD *  
INCLUDE,MAXMINIDUMPPAGES(1000)  
NODUP(NORMAL(0))  
/*  
//IDITRACE DD SYSOUT=*
```

2. Discover TCP/IP address and Port***

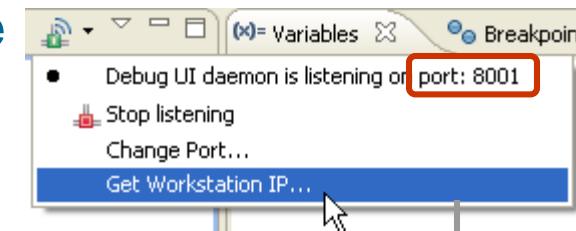
▪ Open the **Debug Perspective**

 Click the small downward pointing triangle next to the debug-daemon icon

▶ **Note the Port#**

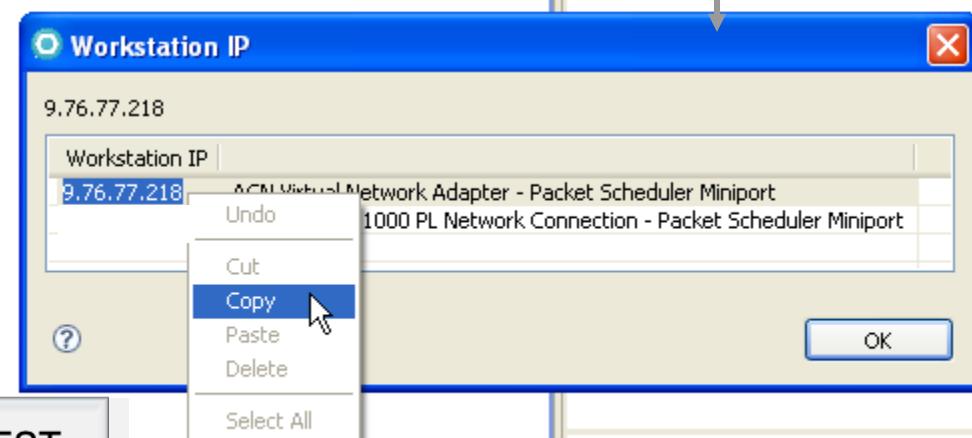
▶ Select: **Get Workstation IP...**

▶ **Copy the IP address**



Note: Your IDz Port# will most likely be set once – permanently

But, depending on your installation's policies workstation's TCP/IP address could change – often – usually every day



*** Hardcoding your TCP/IP address + Port# in the TEST statement may not be necessary. Check with your Sysprog

3a. TEST Statement – Hardcoded TCP/IP & Port#

- Configure your application to start Debug Tool by including a specific DD card in the run JCL – that includes your workstation's current Port# and TCP/IP address
 - This is an example of JCL to run a batch job
 - The CEEOPTS DD statement is the easiest way to start the Debug Tool for batch applications
 - Code as shown

```
*HOSPRUN.jcl X
Line 7      Column 70  Insert
//---+---1---+---2---+---3---+---4---+---5---+---6---
//DDSO001T JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=70M,COND=(16,LT)
//*
//STEP1      EXEC    PGM=HOSPEDIT
//STEPLIB DD DSN=DDSO001.TEST.LOAD,DISP=SHR
//CEEPTS DD *
//      TEST(,,,TCPPIP&5.76.97.236%8001:)
//***** ADDITIONAL RUNTIME JCL HERE *****
//INFILE DD DSN=DDSO001.HOSPDAT.DATA(T1),DISP=SHR
//SYSOUT DD DSN=DDSO001.TEST.COPYLIB(HOSPED),DISP=SHR
//RPTOUT DD DSN=DDSO001.TEST.RPTOUT(HOSPEDIT),DISP=SHR
//ERROUT DD DSN=DDSO001.HOSPDAT.DATA(ERR1),DISP=SHR
```

//CEEPTS DD *

TEST(,,,TCPPIP&5.76.97.236%8001:)

Save your edits, and Submit the JCL

3b. TEST Statement – Alternative Syntax

Simplified workflow to connect your Client to the listener via a TCP/IP address.
Three options:

- ▶ Hard-coded TCPIP Address + Port#

```
//CEEOPTS DD *  
TEST(,,TCPPIP&5.76.97.236%8001:)
```

- ▶ TEST(,,DBM) launches the “Internal Debugger”

```
//CEEOPTS DD *  
TEST(,,DBM)
```

- ▶ (,,,DBMDT%TSOID:) launches “Debug Tool”

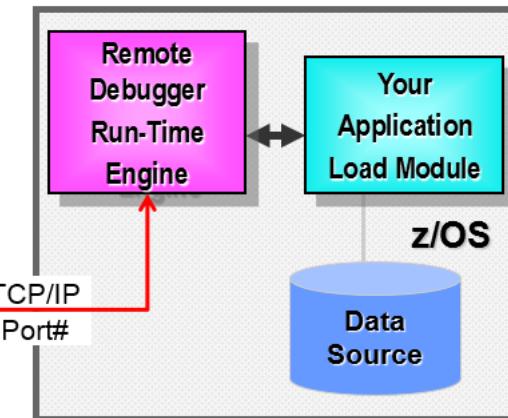
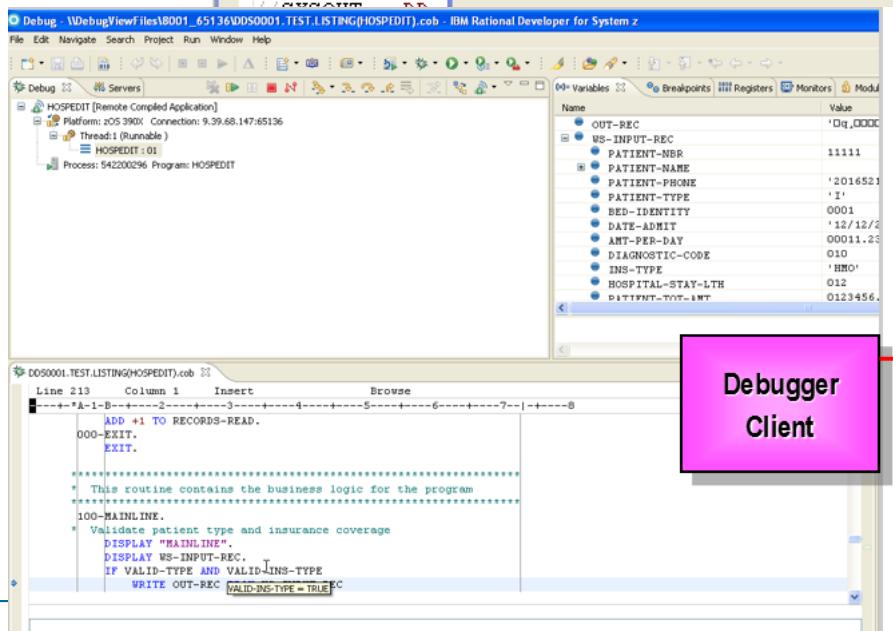
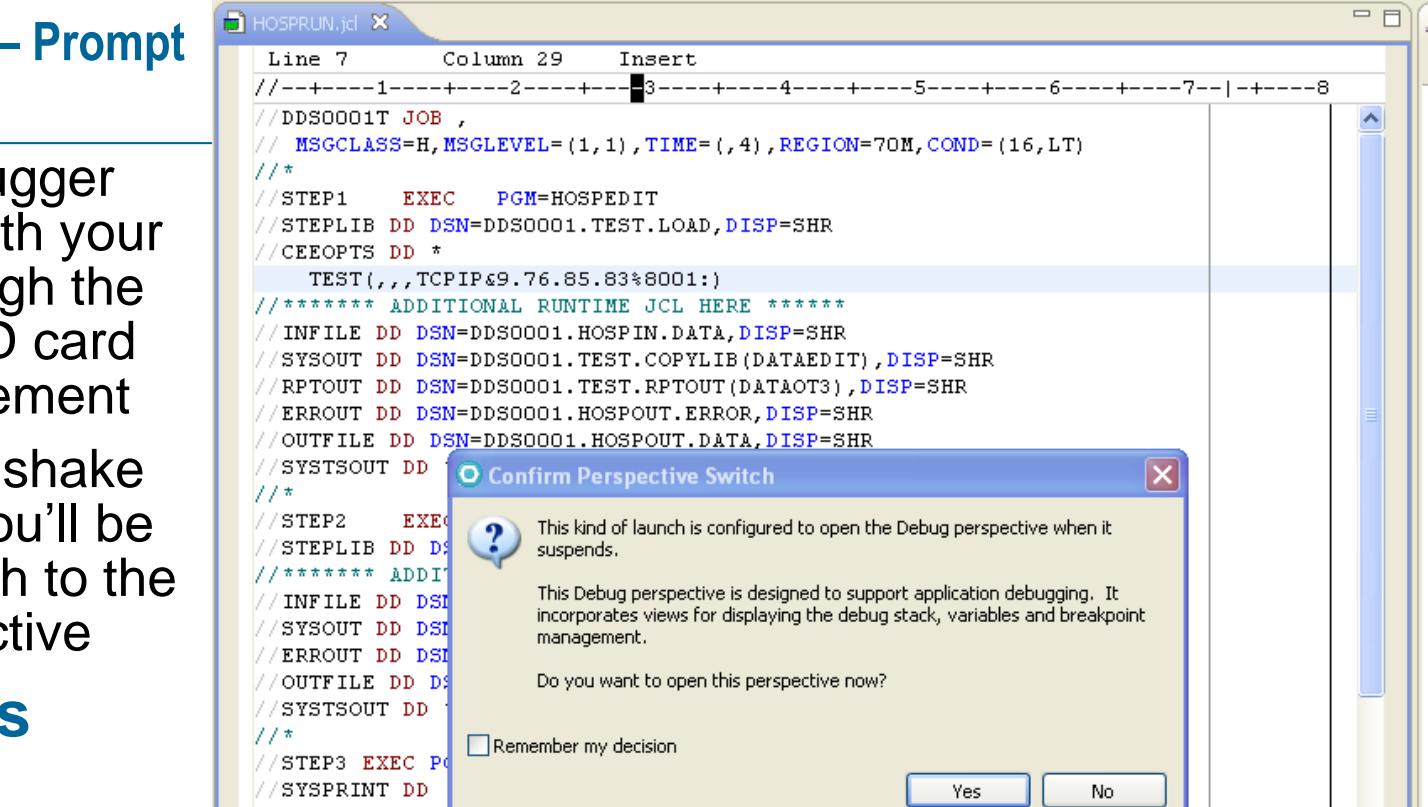
```
//CEEOPTS DD *  
TEST(,,,DBMDT%DDS0001:)
```



Perspective Switch – Prompt Upon Debug Open

- The z/OS Debugger will interface with your IDz client through the //CEEOPTS DD card and TEST statement
- When the handshake is successful you'll be prompted switch to the Debug Perspective

► Click Yes



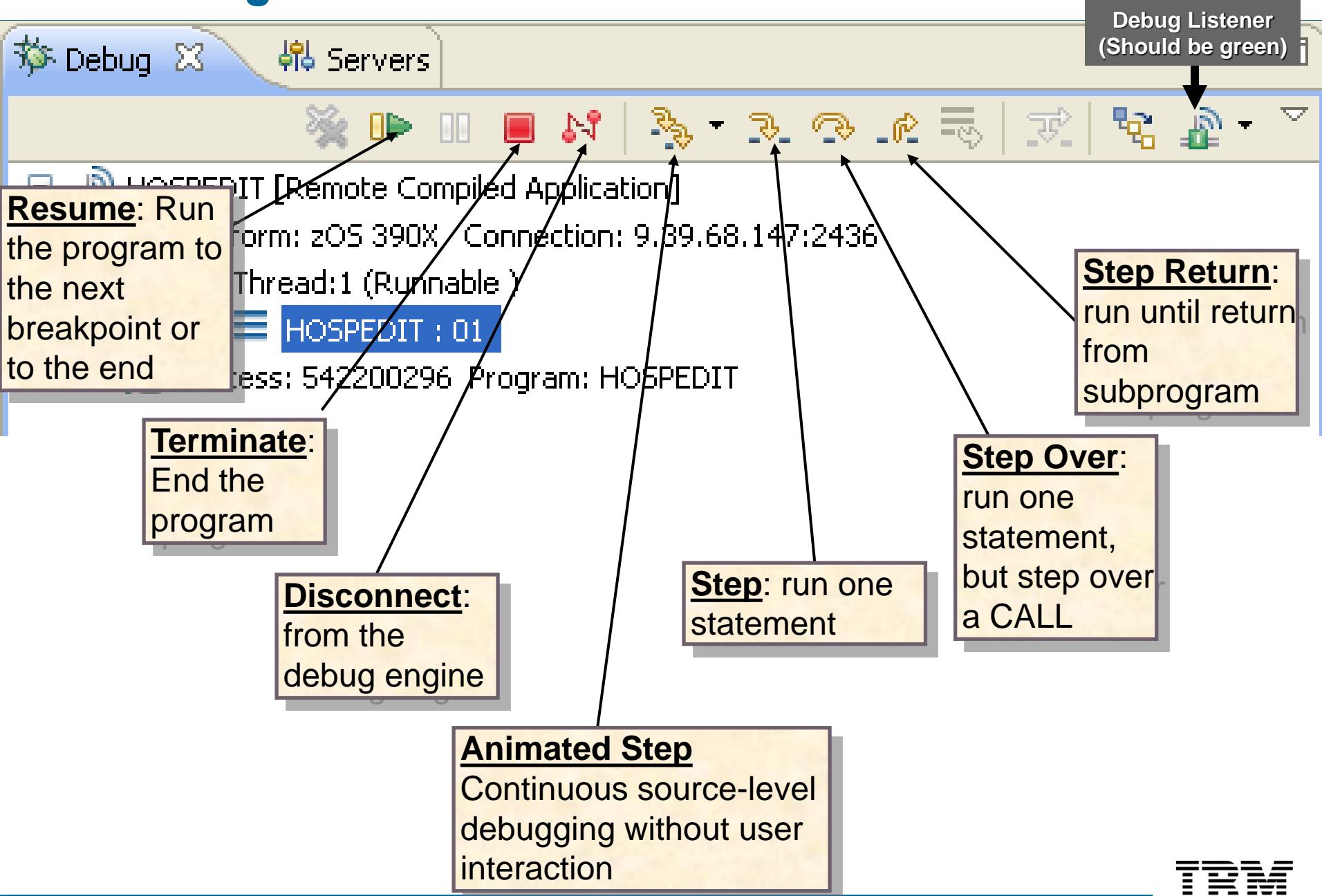
Ready to Debug

After you click YES at the Confirm Perspective switch prompt:

- The Debug Perspective opens
- A Read-Only copy of your source is copied down from z/OS to your IDz client & workstation
- A copy of your load module is NOT placed into an Address Space
 - ▶ The Current Instruction Pointer is not in the PROCEDURE DIVISION
- To open your Load Module into an Address Space and begin debugging you press F5 or click the Step or Resume action icons

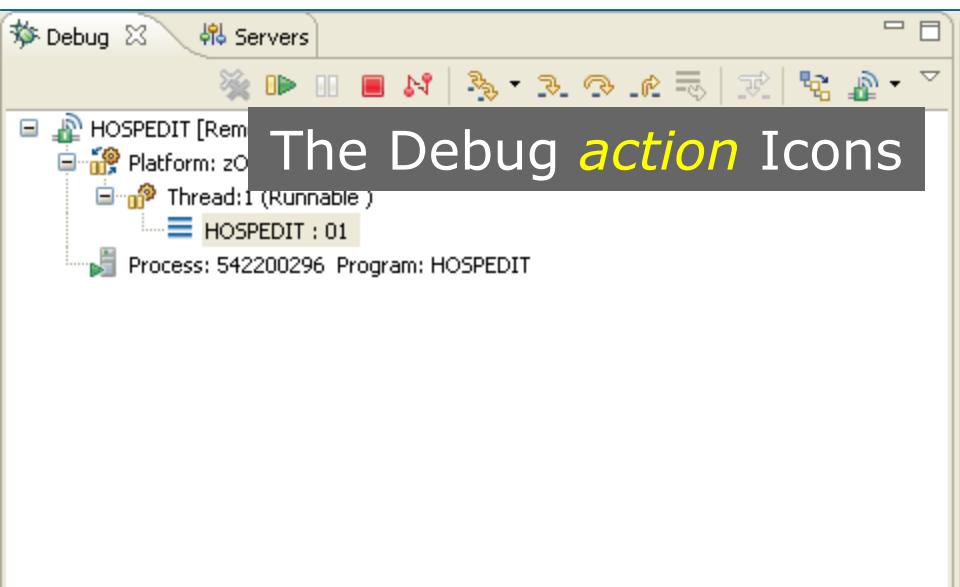
The screenshot shows the RDz interface with the Debug perspective selected. The top navigation bar includes tabs for Debug, Servers, and other development tools. A tree view on the left shows a connection to 'TRTMNTR [Incoming Remote Debug Session]' on 'Platform: zOS 390X' with 'Connection: 127.0.0.1:58063'. A blue arrow points from this connection to a small icon of two server racks. Below the tree view is a tab bar with 'XSAM.jcl', '*XSAM3.jcl', '*BNCHMR1.jcl', and 'TRTMNTR.TRTMNTR.cob'. The main window displays the source code for 'TRTMNTR.cob'. The code is a COBOL program with the following highlights:
- Line 000001: 00010001
- Line 000002: 00020002
- Line 000003: 00030001
- Line 000004: 00040001
- Line 000005: 00050001
- Line 000006: 00060001
- Line 000007: 00070001
- Line 000008: 00080001
- Line 000009: 00090001
- Line 000010: 00100001
- Line 000011: 00110001
- Line 000012: 00120001
- Line 000013: 00130001
- Line 000014: 00140001
- Line 000015: 00150001
- Line 000016: 00160001
- Line 000017: 00170001
- Line 000018: 00180001
- Line 000019: 00190001
- Line 000020: 00200001
- Line 000021: 00210001
The code starts with the division header 'ON DIVISION.' and includes various comments and section descriptions. The current instruction pointer is highlighted at the beginning of the PROCEDURE DIVISION.

The Debug “Action” Icons



The Debug Perspective and Views

Variables Breakpoints & Monitors Views



Variables	
Breakpoints	
RECORDS-READ	1010
FILE-STATUS-CODES	0101
WS-INPUT-REC	
PATIENT-NBR	11111
PATIENT-NAME	'2016521111'
PATIENT-PHONE	'I'
PATIENT-TYPE	0001
BED-IDENTITY	'12/12/2005'
DATE-ADMIT	00011.23
AMT-PER-DAY	010
DTAGNOSTIC-CODE	



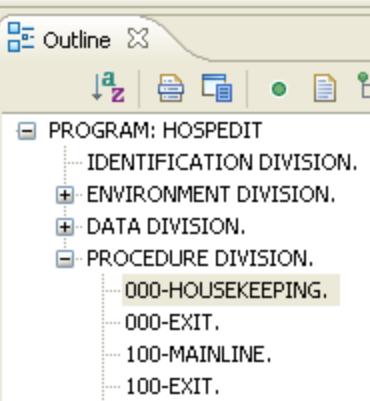
The screenshot shows a Cobol editor window titled "HOSPRUN.jcl DDS0001.TEST.LISTING(HOSPEDIT).cob". The code is as follows:

```
Line 201 Column 1 Insert Browse
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+*A-1-B---+---2---+---3---+---4---+---5---+---6---+---7--| +---8
ADD +1 TO RECORDS-READ.
000-EXIT.      RECORDS-READ = +00000
EXIT.

***** This routine contains the business logic for the program
```

A tooltip "Read-only copy of source code" is visible at the bottom left.

Read-only copy of source code



Program Outline View



The Debug Perspective and Views – IDz v14.1.x and Later

The Debug *action* Icons

Visual Debugging Displays Paragraphs/Sections in your Procedural logic

Read-only copy of your program source

Outline, Variables Breakpoints & Monitors Views

Step Into (F5)

Activity: Other Activity

Visual Debug

DDS0001.TEST.SYSDEBUG(SAM3).cob

```
--+-*A-1-B---2---+---3---+---4---+---5---+---6---+---7---+---8
000436 029713 CALL 'SAM4' USING CUST-REC, TRANSACTION-RECORD,
000437          WS-TRAN-OK, WS-TRAN-MSG
000438 029714 IF WS-TRAN-OK NOT = 'Y'
000439 029715      MOVE WS-TRAN-MSG TO ERR-MSG-DATA1
000440 029716      MOVE SPACES      TO ERR-MSG-DATA2
000441 029717      PERFORM 299-REPORT-BAD-TRAN
000442 029718 ELSE
000443 029719      ADD +1 TO NUM-CRUNCH-PROCESSED
000444 029720 END-IF .
000445 029724
000446 029800 210-PROCESS-ADD-TRAN.
000447 029900      ADD +1 TO NUM-ADD-REQUESTS .
000448 030000      PERFORM 720-POSITION-CUST-FILE.
000449 030100      IF CUST-KEY = TRAN-KEY
000450 030200          MOVE 'DUPLICATE KEY:'      TO ERR-MSG-DATA1
000451 030300          MOVE TRAN-KEY      TO ERR-MSG-DATA2
```

Debug

Remote Systems

Debug Console

Memory

SAM3 [Incoming Remote Debug Session]

Platform: zOS 390X Connection: screenshots.firefox.com:50599

Thread:1 (Runnable)

210-PROCESS-ADD-TRAN : SAM3 : 03

100-PROCESS-TRANSACTIONS : SAM3 : 02

000-MAIN : SAM3 : 01

Process: 588302528 Program: SAM3

IBM

Debugging your code

The Action Icons

Actions to execute program statements:

Step (F5)

- Executes the current instruction
- Positions the pointer at the Next Sequential Instruction

Step Over (F6)

- Executes the current instruction
- Except on a CALL statement – where Step Over runs (executes) all of the logic in the Called subroutine stack
- Positions the pointer at the Next Sequential Instruction

Step Return (F7)

- Executes all of the statements thru to the end (GOBACK) of the program

Resume (F8)

- Executes (runs) all statements until:
 - ▶ ABEND
 - ▶ Breakpoint
 - ▶ Normal EOJ

```
-----*A-1-B-----2-----3-----4-----5-----6-----7-----8
000336 021500
000337 021600 01 CUST-KEY-DIAG      PIC X(6).
000338 021601 01 REF-MOD-VAL        PIC X(4).
000339 021602 01 REF-MOD-1         PIC S9(1) VALUE 1.
000340 021603 01 REF-MOD-2         PIC S9(1) VALUE 4.
000341 021604 01 ABEND-TEST       PIC X(2).
000342 021605 01 TEST-N REDEFINES ABEND-TEST PIC S9(3) COMP-3.
000343 021606 01 TEST-N VISION.
000344 021607 01 TEST-N VISION.
000345 021608 01 TEST-N VISION.
000346 021609 01 TEST-N VISION.
000347 021610 01 TEST-N VISION.
000348 022300 000-MAIN.
000349 022400    ACCEPT CURRENT-DATE FROM DATE.
000350 022500    ACCEPT CURRENT-TIME FROM TIME.
000351 022600    DTCR1 AV 'SAM1 STARTED DATE - ' CURRENT MONTH //
```

Additional Debug actions:

Terminate (Ctrl+F2)

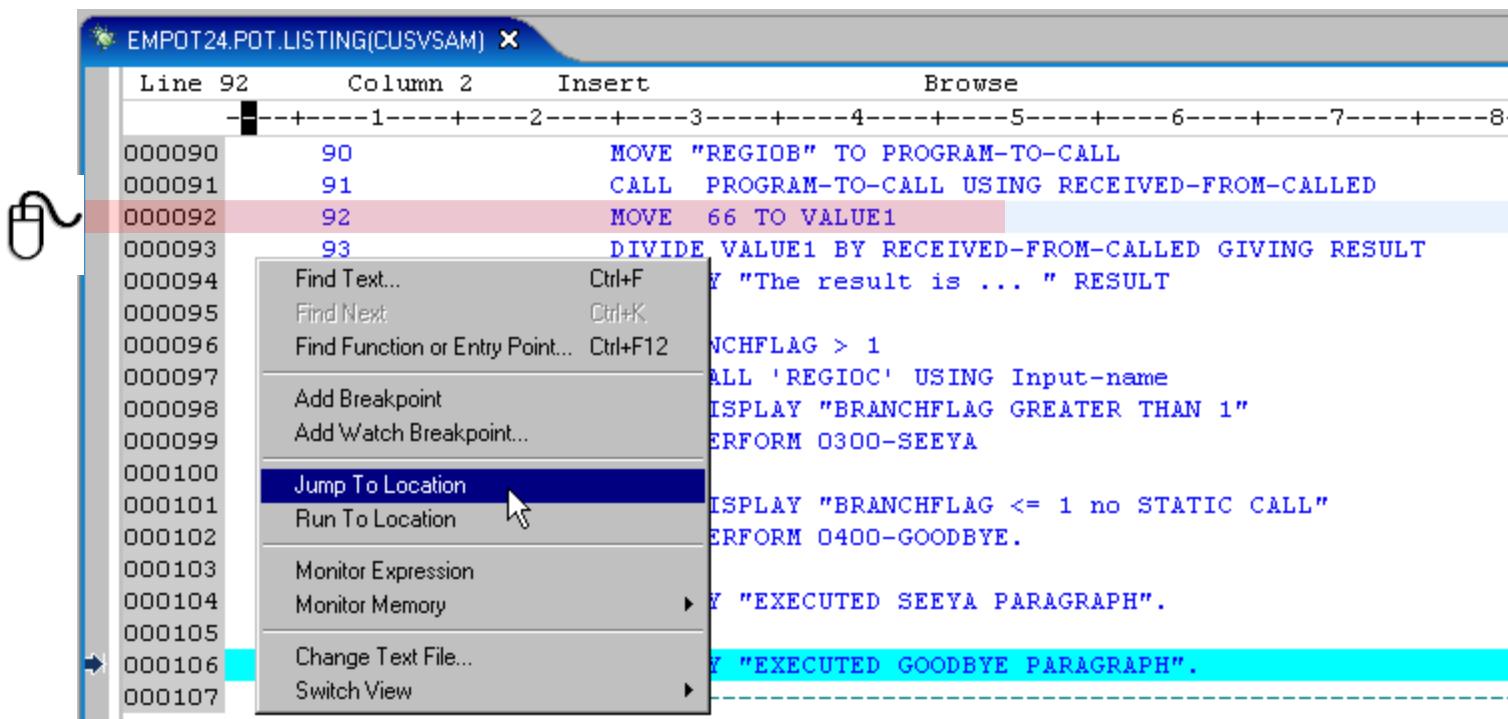
- Halts the Debug session

Open Visual Debug

- Opens a view that keeps a running visual depiction of the program's business logic procedural flow

Additional Debug Actions – Jump to & Run to Cursor Location

- **Jump to Location** - skip over sections of code to avoid executing certain statements or move to a position where certain statements can be executed again. Useful:
 - To avoid called programs or I/OS to a not available dataset
 - Or to iteratively execute some statements of interest
- **Run to Location** - executes all statements between the current location and the run-to location.



Returning to the “Current Instruction”

- To get back to the Current Instruction Pointer (the "next sequential instruction") – if you've navigated away within the source:
 - Click the small blue rectangle in the right-hand margin of your source code

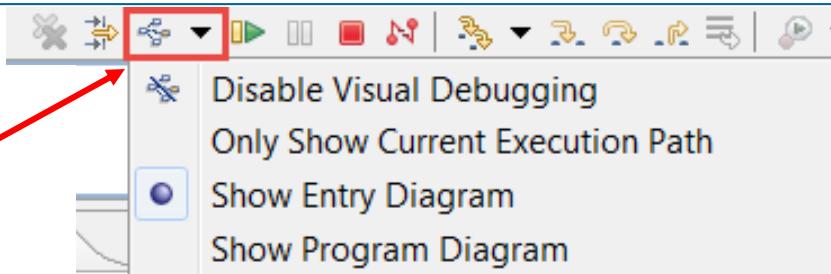
The screenshot shows a window titled "DDS0001.TEST.SYSDEBUG(SAM3).cob". The main area displays a COBOL program with various lines of code. A red arrow points from the text "Note that this icon can be hard to see – especially in large source files" to a small blue rectangular icon located in the right-hand margin of the code editor, just before the line number 000345. This icon represents the "Current Instruction Pointer". The line 000345 contains the instruction "DISPLAY 'SAM1 STARTED DATE = ' CURRENT-MONTH '/'". The code editor has a toolbar at the top with buttons for Line, Column, Insert, and Browse.

```
Line 345      Column 81      Insert      Browse
-----+---*A-1-B---+---2---+---3---+---4---+---5---+---6---+---7---+---8
000335 021600 01 ABEND-TEST          PIC X(2).
000336 021700 01 ABEND-TEST-N REDEFINES ABEND-TEST PIC S9(3) COMP-3.
000337 021800
000338 021900 ****
000339 022000 PROCEDURE DIVISION.
000340 022100 ****
000341 022200
000342 022300 000-MAIN.
000343 022400    ACCEPT CURRENT-DATE FROM DATE.
000344 022500    ACCEPT CURRENT-TIME FROM TIME.
000345 022600    DISPLAY 'SAM1 STARTED DATE = ' CURRENT-MONTH '/'
000346 022700          CURRENT-DAY '/' CURRENT-YEAR '(mm/dd/yy)' .
000347 022800    DISPLAY '                TIME = ' CURRENT-HOUR ':'
000348 022900          CURRENT-MINUTE ':' CURRENT-SECOND.
000349 023000
```

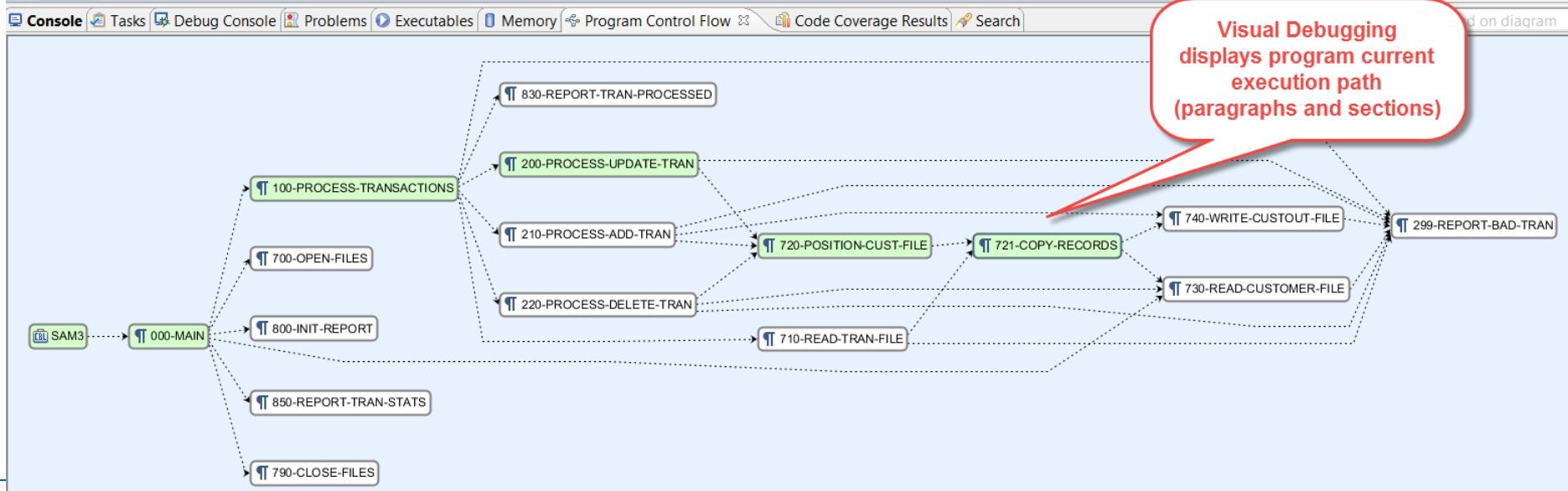
Note that this icon can be hard to see – especially in large source files

Visual Debug – IDz v14 and later

- Visually track your program's progress through the PROCEDURE DIVISION.
- Disable/Enable Visual Debugging



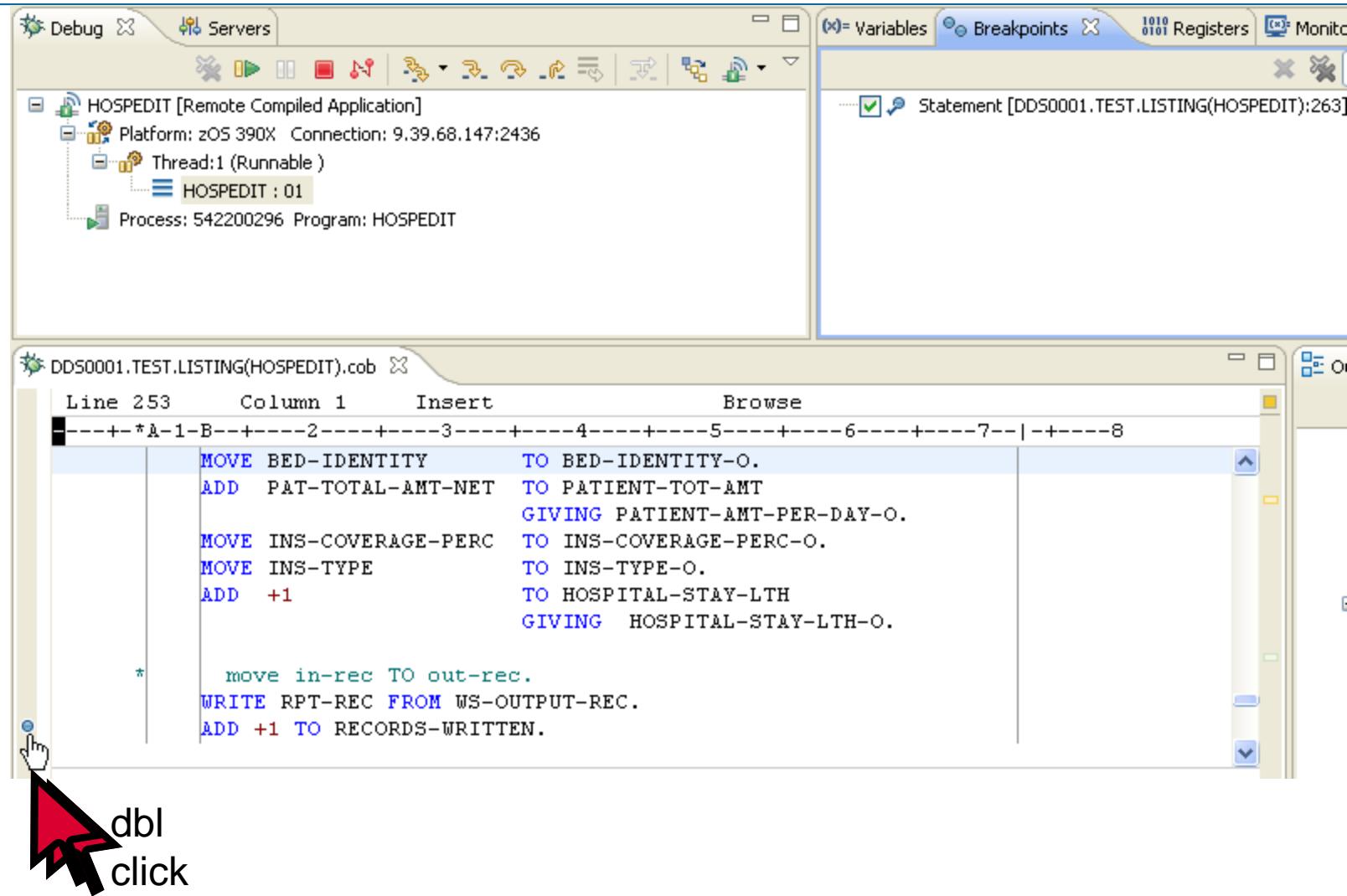
The image shows the main workspace of IDz. At the top, there are tabs for BNCHMR1.jcl, XSAM3.jcl, XSAM.jcl, and SAM3.list.cob. Below the tabs is a COBOL source code editor window displaying a program. A red arrow points from the text "Visually track your program's progress through the PROCEDURE DIVISION." to the second icon from the left in the toolbar above the editor. The code editor shows several lines of COBOL code, with some lines highlighted in green. On the right side of the screen is a navigation pane titled "Outline" which lists various MVS files and z/OS UNIX resources.



Statement Breakpoints

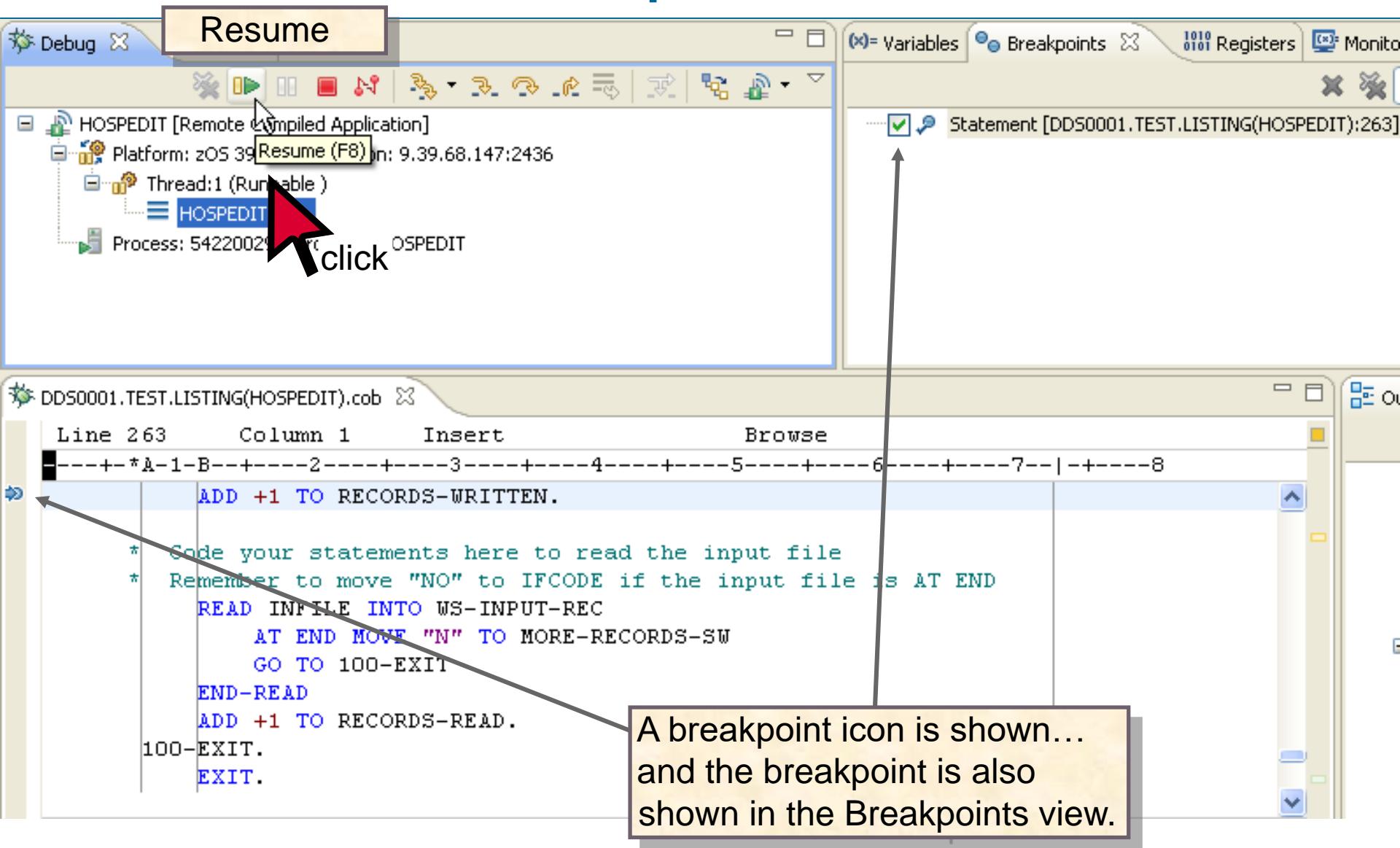
- A statement breakpoint stops the program when it reaches a statement:
 - ▶ It stops **before** the statement runs
- A breakpoint can optionally be made conditional
 - ▶ A simple condition may be specified such as:
 - **VariableX > 999**
...or...
 - **VariableY = 'Abc'**
- A breakpoint can be based on a frequency:
 - ▶ Stop the Nth time a statement runs
- A breakpoint can stop at the Entry to a Load Module

Setting Statement Breakpoints



Set a statement breakpoint by double-clicking in the gray area next to a statement

Run to a Statement Breakpoint



See Slide Notes

Edit/Create Conditional Statement Breakpoints

The screenshot shows a debugger's Breakpoints window with a context menu open over a selected breakpoint. The menu items include 'Add Breakpoint' and 'Edit Breakpoint...', with 'Edit Breakpoint...' being the current selection. A tooltip above the menu says: 'Select the Breakpoint. Right-click and select: Edit Breakpoint...'. Below the menu, the 'Edit a Statement Breakpoint' dialog is open, showing fields for 'Load Module/DLL/Executable' (set to 'HOSPEDIT'), 'Object/Program/CSECT' (set to 'HOSPEDIT'), 'Source(optional)' (set to 'DDS0001.TEST.LISTING(HOSPEDIT)'), and 'Statement:' (set to '263'). A tooltip at the bottom left says: 'Can set to different statement/line Or click Next > to specify conditional breakpoint logic'. To the right, the 'Edit a Line Breakpoint' dialog is open, showing 'Thread: Every', 'Frequency' settings ('From: 1', 'To: Infinity', 'Every: 1'), and an 'Expression' field containing 'NUM-UPDATE-REQUESTS = 4'. A tooltip for this dialog says: 'A breakpoint can trigger the Nth time the statement runs... ... and breakpoints can be conditional.' The IBM logo is visible in the bottom right corner.

Select the Breakpoint.
Right-click and select: Edit Breakpoint...

Add Breakpoint
Edit Breakpoint... **Edit Breakpoint...**

Enable
Disable
Remove
Remove All
Select
Copy
Paste
Export
Import

Statement [DDS0001.TEST.LISTING(HOSPEDIT):263]

Go to File

Required information
Sets a breakpoint to stop execution at a specific source line.

Defer breakpoint until executable is loaded

Load Module/DLL/Executable
HOSPEDIT

Object/Program/CSECT
HOSPEDIT

Source(optional):
DDS0001.TEST.LISTING(HOSPEDIT)

Statement:
263

Optional parameters
Make the breakpoint conditional

Thread: Every

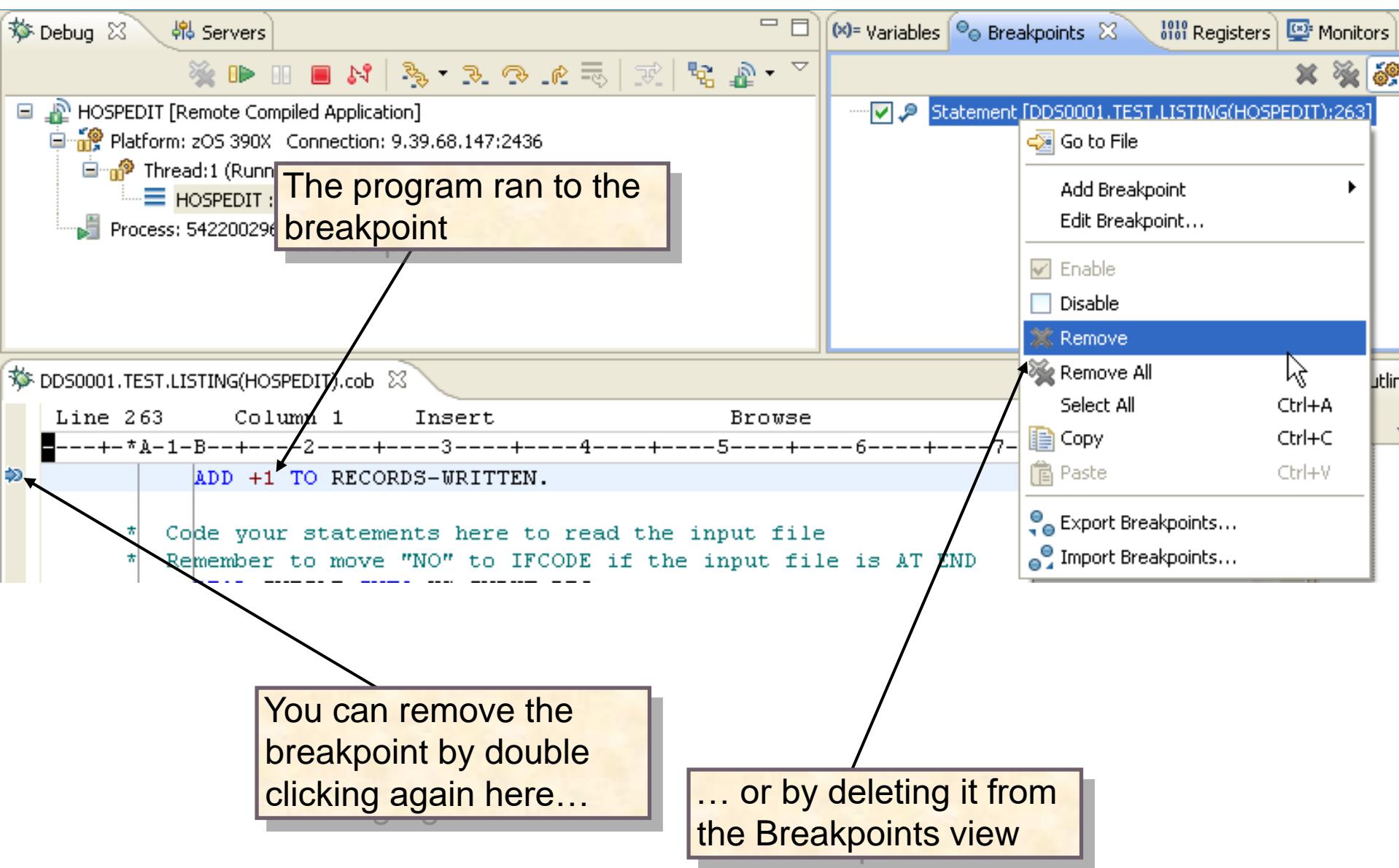
Frequency
From: 1
To: Infinity
Every: 1

Expression: NUM-UPDATE-REQUESTS = 4

A breakpoint can trigger the Nth time the statement runs...
... and breakpoints can be conditional.

Can set to different statement/line
Or click Next > to specify
conditional breakpoint logic

Additional Breakpoint Options



Entry Breakpoints – to stop in a specific program within a series of subroutine calls

File Edit Navigate Search Project Run Window Help

Servers

- SAM3 [Incoming Remote Debug Session]
 - Platform: zOS 390X Connection: demomvs.cc9.pok.ibm.com:31467
 - Thread:1 (Runnable)
 - SAM3 : 01
 - Process: 587253952 Program: SAM3

XSAM.jcl DDS0001.TEST.SYSDEBUG(SAM3).cob

```
-----+-----+-----+-----+-----+
000338 021603 01 REF-MOD-2          PIC S9(1) VALUE 4.
000339 021604 01 ABEND-TEST        PIC X(2).
000340 021700 01 ABEND-TEST-N REDEFINES ABEND-TEST RTC S9(2) COMP-5
000341 021800
000342 021900*****
000343 022000 PROCEDURE DIVISION.
000344 022100*****
000345 022200
000346 000-MAIN.
000347 022400 ACCEPT CURRENT-DATE FROM DATE.
000348 022500 ACCEPT CURRENT-TIME FROM TIME.
000349 022600 DISPLAY 'SAM1 STARTED DATE = ' CURRENT-MONTH '
000350 022700           CURRENT-DAY '/' CURRENT-YEAR ' (mm/dd/
000351 022701*           UPON CONSOLE.
000352 022800 DISPLAY '           TIME = ' CURRENT-HOUR ' :
```

-----+-----+-----+-----+-----+

XSAM.jcl DDS0001.TEST.SYSDEBUG(SAM3).cob

Variables Breakpoints Registers Modules Monitors

Entry [SAM4] [deferred]

Statement [DDS0001.TEST.SYSDEBUG(SAM3).272]

Edit an Entry Breakpoint

Required information

Sets a breakpoint to stop execution at a specific function or entry point

Defer breakpoint until executable is loaded

Show items with debug info only

Load Module/DLL/Executable (Optional)

SAM4

Object/Program/CSECT

SAM4

Function or Entry point

SAM4

Case sensitive

User label (optional)

The Breakpoint causes execution to stop at Program Entry

File Edit Navigate Search Project Run Window Help

Servers

 - Platform: zOS 390X Connection: demomvs.cc9.pok.ibm.com:31467
 - Thread:1 (Runnable)
 - SAM4 : 02
 - SAM3 : 01
 - Process: 587253952 Program: SAM3

XSAM.jcl DDS0001.TEST.SYSDEBUG(SAM3).cob

```
* parameters:
*   1: Customer Record (passed)
*   2: Transaction Record (passed)
*   3: tran-ok flag (returned)
*           Return values: Y = transaction was processed
*                           N = error processing transaction
*   4: message (returned)
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. SAM4.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
*****
DATA DIVISION.
```

Stopping on a line in a Subroutine

Steps:

- ▶ Set a Statement Breakpoint
- ▶ Over-ride the Load Module/CSECT values
- ▶ Add the Statement line# to break at

Debug - RemoteSystemsTempFiles/DebugViewFiles/4/DDS0001.TEST.SYSDEBUG(SAM4).cob - IBM Developer for z Systems

File Edit Navigate Search Project Data Run Window Help

Servers Debug Thread:1 (Runnable) SAM4 : 02 SAM3 : 01

XSAM3.jcl DDS0001.TEST.SYSDEBUG(SAM4).cob

000119
000120 01 TRAN-OK PIC X.
000121 01 TRAN-MSG PIC X(50).

000124 PROCEDURE DIVISION USING CUST-REC,
TRANSACTION-RECORD,
TRAN-OK,
TRAN-MSG .
000129 000-MAIN.
000130 MOVE 'Y' TO TRAN-OK.
MOVE SPACES TO TRAN-MSG.
000133 IF TRAN-CODE = 'CRUNCH '
PERFORM 300-PROCESS-CPU-CRUNCH
000134 ELSE
PERFORM 100-VALIDATE-TRAN
000136

Edit a Statement Breakpoint

Required information

Sets a breakpoint to stop execution at a specific source line

Defer breakpoint until executable is loaded

Load Module/DLL/Executable
SAM4

Object/Program/CSECT
SAM4

Source(optional): DDS0001.TEST.SYSDEBUG(SAM4)

Statement: 130

User label (optional):

← You need to select the "Defer breakpoint until executable is loaded" if the subroutine is in a different load mod

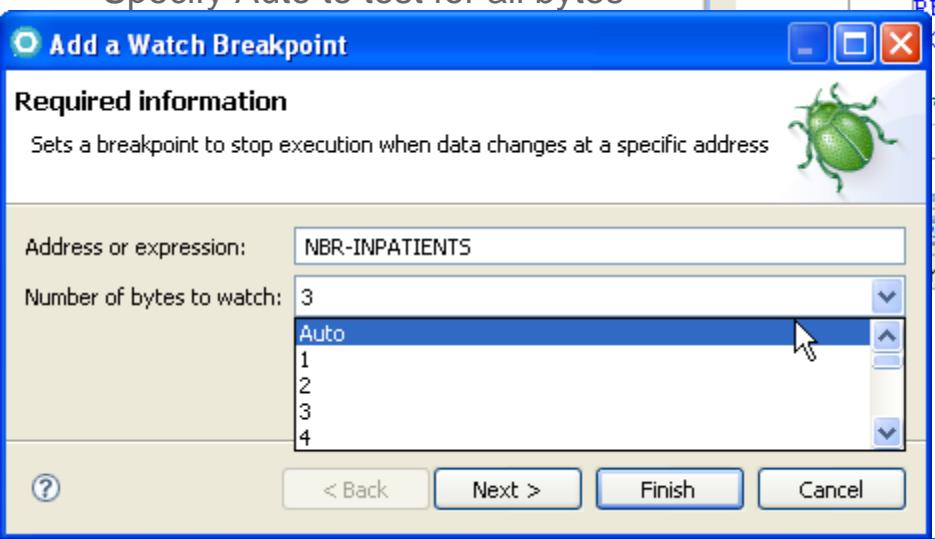
Console Tasks Bookmarks Debug Console Executables SQL Results

EQA2261E An error occurred while opening file: INSPPREF. The file EQA2458I SVC Screening is disabled by EQAOPTS. Handling of non-LE events is not available. Debugging of non-LE programs will be Program was stopped due to line/statement breakpoint at statement 130.

Debug Tool

Watch Monitor Breakpoints

- Can have breakpoints occur conditionally, when:
 - ▶ The value in a field changes
 - ▶ Some portion (# of bytes) of a field changes
 - ▶ A logical condition tests true for the value in the field
 - Steps:
 - ▶ Select a variable
 - ▶ Right-click, and select: Add Watch Breakpoint...
 - ▶ Select Number of bytes to watch – or add a simple condition
 - Specify Auto to test for all bytes



Line 164 Column 29 Insert Browse

----+*A-1-B---+---2---

77 WS-DATE
77 MORE-RECORDS-SW
88 NO-MORE-RECORD:

01 COUNTERS-AND-ACCUMULATORS
05 RECORDS-READ
05 RECORDS-WRITTEN
05 ERROR-RECS
05 NBR-INPATIENTS
05 NBR-OUTPATIENTS
05 NBR-HMO
05 NBR-STATE-FED
05 NBR-NO-COVERAGE
05 PAT-TOTAL-AMT-1
05 TOTAL-AMT-GROSS
05 TOTAL-AMT-NET

PROCEDURE DIVISION.
PERFORM 000-HOUSEKEEPING.
PERFORM 100-MAINLINE
 UNTIL NO-MORE-RECORDS.
PERFORM 200-CLEANUP.
GOBACK.

Cut Ctrl+X
Copy Ctrl+Insert
Paste Ctrl+V
Select ►
Selected ►
Deselect Alt+U
Filter view ►
Show all Ctrl+W
Source ►
Refactor ►
Open Declaration ►
Open Perform Hierarchy ►
Asset Analyzer Program Tree ►
View ►
Run As ►
Debug As ►
Profile As ►
Validate ►
Software Analyzer ►
Team ►
Compare With ►
Replace With ►
Local Syntax Check ►
Browse Copy Member ►
Open Copy Member ►
Content assist Ctrl+Space
Start Flagging Changed Lines
Add Breakpoint
Add Watch Breakpoint... Add Watch Breakpoint...

Debug Tool

Variables View During Remote Debug – Additional Options

The screenshot shows the Eclipse IDE's Variables View during a remote debug session. The view lists various COBOL variables with their current values. A context menu is open over the variable 'DATE-ADMIT' (value: '12/12/2005'). The menu includes options like 'Monitor Local Variable', 'Monitor Memory', 'Change representation', 'Select All', 'Copy Variables', 'Find...', and 'Change Value...'. Below the menu, a 'Filter Locals' dropdown is open, showing filter categories: '0 All', '1 Automonitor Current', '2 Automonitor Previous', '3 COBOL File Section', '4 COBOL Working-Storage Section', '5 COBOL Linkage Section', and '6 COBOL Local-Storage Section'. The '4 COBOL Working-Storage Section' option is checked. In the bottom right corner of the Variables View, there is an 'Outline' view showing some COBOL structure details.

Many options for working with variables

- ← Add to Monitor – for permanent viewing
- ← Monitor internal memory values
- ← Show value in hex – (EBCDIC internal) or string value representation
- ← Copy variable and value to Notepad
- ← Find specific variable in filtered Locals
- ← Show specific variables in the view

With large COBOL programs can "filter" and display only certain categories of variables

Variables View – EBCDIC (Hex) Data Representation

The screenshot shows the 'Variables' tab of a debugger interface. A context menu is open over the entry 'PAT-TOTAL-AMT-NET'. The menu items are: 'Monitor Local Variable', 'Monitor Memory', 'Change representation', 'Select All', and 'Ctrl+A'. The 'Change representation' item has a submenu with '1 Hexadecimal' and '2 Decimal'. The '2 Decimal' option is highlighted with a blue background and white text. A callout bubble points to this option with the text 'Value – in string or numeric display format'.

Name	Value
RECORDS-WRITTEN	+000000
ERROR-RECS	+000000
NBR-INPATIENTS	+00001
NBR-OUTPATIENTS	+00000
NBR-HMO	+00001
NBR-STATE-FED	+00000
NBR-NO-COVERAGE	+00000
PAT-TOTAL-AMT-NET	100145700.00
TOTAL-AMT-GROSS	+000000
TOTAL-AMT-NET	+000000

+0123466.88

Value – in string or numeric display format

Monitor Local Variable
Monitor Memory
Change representation
Select All Ctrl+A

1 Hexadecimal
2 Decimal

The screenshot shows the 'Variables' tab of a debugger interface. The entry 'PAT-TOTAL-AMT-NET' is selected. A context menu is open over it, with the 'Change representation' item highlighted. The submenu shows '1 Hexadecimal' and '2 Decimal'. The '1 Hexadecimal' option is highlighted with a blue background and white text. A callout bubble points to this option with the text 'Value in EBCDIC internal display → very useful for debugging data exceptions'.

Name	Value
RECORDS-WRITTEN	+000000
ERROR-RECS	+000000
NBR-INPATIENTS	+00001
NBR-OUTPATIENTS	+00000
NBR-HMO	+00001
NBR-STATE-FED	+00000
NBR-NO-COVERAGE	+00000
PAT-TOTAL-AMT-NET	X'012346688C'
TOTAL-AMT-GROSS	Monitor Local Variable
TOTAL-AMT-NET	Monitor Memory

1 Hexadecimal
2 Decimal

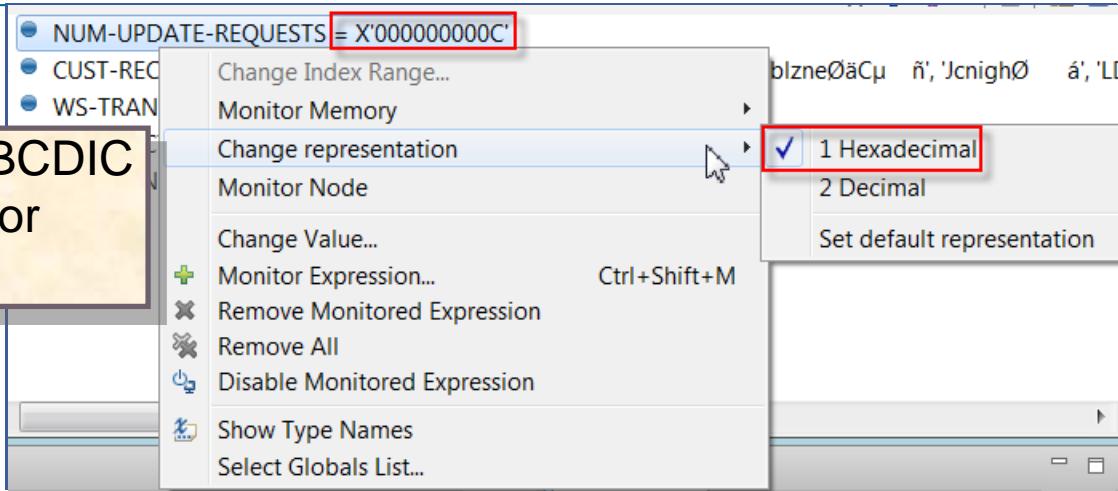
Change representation
Select All Ctrl+A

You can create invalid binary values in numeric fields for type-check (0C7) testing

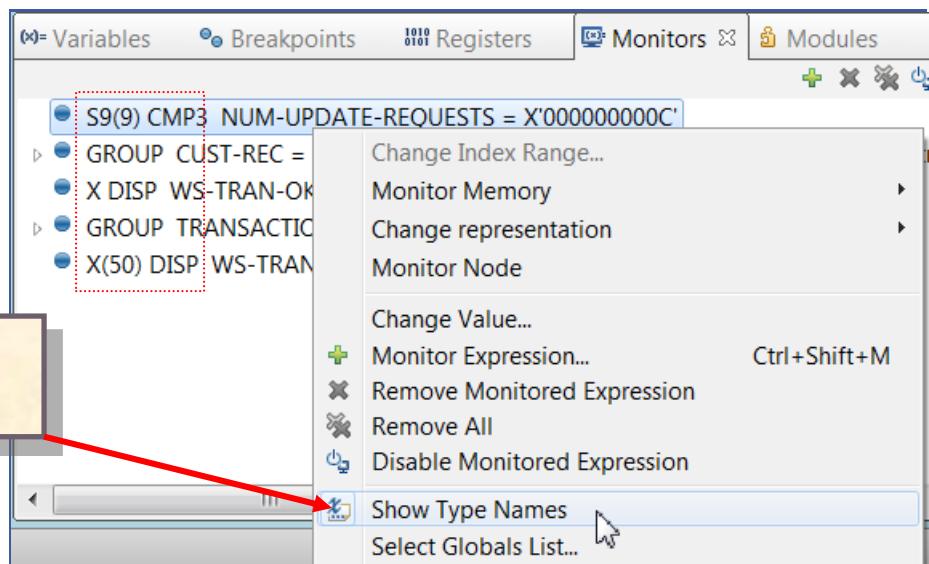
1. Change representation to Hex.
2. Type invalid numeric EBCDIC values
3. Save

Monitors View – Options

Monitored variable value – in EBCDIC internal display → very useful for debugging data exceptions



Add datatype prefix to variable data-name displayed – still retain variable value)



Best Practice – Detach the Eclipse View

Drag the Monitors View out of the Workbench and onto a Dual Monitor or onto your desktop...

The screenshot shows the SAP ABAP Workbench interface. On the left, the ABAP code editor displays a program with several sections of code, including an IF block for VALID-TYPE and a EVALUATE block for INS-TYPE. A callout box highlights the IF block with the text: "Can view a large of variable values during debug, animated debug or Resume to breakpoints". On the right, a separate window titled "Monitors" lists numerous variable names and their current values, such as RECORDS-READ, NBR-HMO, PAT-TOTAL-AMT-NET, and various patient and hospital-related variables.

```
-- *A-1-B--+
DISPLAY "MAINLINE".
DISPLAY WS-INPUT-REC.
IF VALID-TYPE AND VALID-INS-TYPE
  VALID-TYPE = TRUE
ELSE
  MOVE WS-INPUT-REC TO ERR-REC
  WRITE ERR-REC
  ADD +1 TO ERROR-RECS
  READ INFILE INTO WS-INPUT-REC
    AT END MOVE "N" TO MORE-RECORDS-SW
    GO TO 100-EXIT
  END-READ
  ADD +1 TO RECORDS-READ
  GO TO 100-EXIT
END-IF

* Add to counters and total amounts
EVALUATE INS-TYPE
  WHEN "HMO" ADD +1 TO NBR-HMO
  WHEN "GOV" ADD +1 TO NBR-STATE-FED
  WHEN OTHER ADD +1 TO NBR-NO-COVERAGE
END-EVALUATE

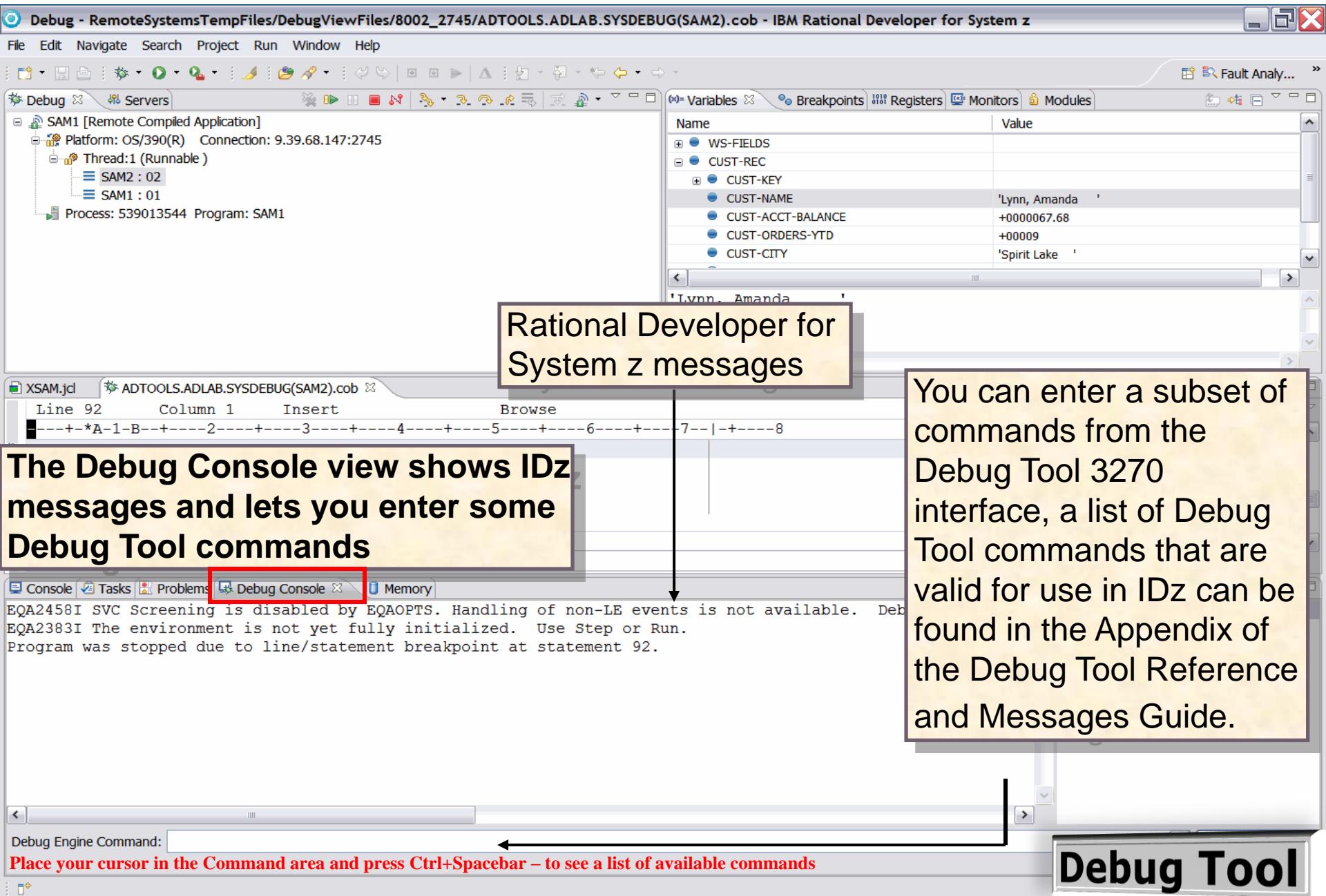
IF INP
  AD
ELSE
  AD
```

Can view a large of variable values during debug, animated debug or Resume to breakpoints

Monitors

- RECORDS-READ = +00009
- NBR-HMO = +00003
- PAT-TOTAL-AMT-NET = +0035572.21
- WS-INPUT-REC
- WS-INPUT-REC
 - PATIENT-NBR = 99999
 - PATIENT-NAME
 - PATIENT-PHONE = '9098883213'
 - PATIENT-TYPE = 'I'
 - BED-IDENTITY = 0002
 - DATE-ADMIT = '06/30/2006'
 - AMT-PER-DAY = 00023.23
 - DIAGNOSTIC-CODE = 483
 - INS-TYPE = 'HMO'
 - HOSPITAL-STAY-LTH = 020
 - PATIENT-TOT-AMT = 0000475.74
 - PCP-ID = 'I9D2E2'
 - IN-OUT-NETWORK = 'I'
 - COPAY = +020
 - DEDUCTIBLE = +0085
- RECORDS-WRITTEN = +00008
- COUNTERS-AND-ACCUMULATORS
- RECORDS-WRITTEN = +00009

The Debug Console



Debug Console Commands – Tracing Statement Execution

This is another very popular command:

SET AUTOMONITOR ON LOG

It forces Debug Tool to track each statement as it's executed and write it to the Debug Console

Using this technique you can copy and paste your program's dynamic execution and trace forward and backward through any portion of your code

You can also copy all of the statements to hard-copy :

1. Right-click
2. Select Export History
3. Specify a file – preferably an RTF or MS-Word doc, as formatting will be retained

000508 038600 END-IF .
000509 038700
000510 038800 720-POSITION-CUST-FILE.
000511 038900 IF CUST-KEY < TRAN-KEY
000512 039000 IF WS-CUST-FILE-EOF NOT = 'Y'
000513 039100 PERFORM 721-COPY-RECORDS
000514 039200 UNTIL CUST-KEY >= TRAN-KEY

WS-TRAN-EOF = ' '
The current location is SAM3 at line 368
WS-TRAN-EOF = ' '
The current location is SAM3 at line 369
NUM-TRAN-RECS = +000000006
The current location is SAM3 at line 370
WS-TRAN-OK = 'Y'
The current location is SAM3 at line 371
TRAN-KEY = '11204A '
WS-PREV-TRAN-KEY = '11204A '
The current location is SAM3 at line 376
TRAN-CODE = 'UPDATE'
The current location is SAM3 at line 378
The current location is SAM3 at line 399
NUM-UPDATE-REQUESTS = +000000004
The current location is SAM3 at line 400
The current location is SAM3 at line 511
CUST-KEY =
TRAN-KEY = '11204A '

Debug Engine Command: **set automonitor on log**

Set Automonitor On Log – To help solve Infinite Loop ABENDs

Infinite Loops can be difficult to solve for.

A “Best Practice” would be to:

1. SET AUTOMONITOR ON LOG

2. Resume ➤ to the S222 ABEND
- you will stop at some part of the code that looping

3. From the Debug Console Log →

- Note the line # in the program that is inside the Infinite Loop
- Set a Breakpoint at one of the lines
- Resubmit your JCL
...or...
- Re-run your online transaction

The screenshot shows the IBM i debugger interface. At the top, assembly code is displayed with several lines highlighted in purple, indicating they are part of the infinite loop. Below the code is a stack trace window showing a series of memory locations and their corresponding line numbers from 368 to 511. At the bottom, a command line window shows the command "set automonitor on log" entered by the user.

Line #	Stack Trace
368	WS-TRAN-EOF = ' '
369	The current location is SAM3 at line
370	NUM-TRAN-RECS = +000000006
371	The current location is SAM3 at line
372	WS-TRAN-OK = 'Y'
373	The current location is SAM3 at line
374	TRAN-KEY = '11204A'
375	WS-PREV-TRAN-KEY = '11204A'
376	The current location is SAM3 at line
377	TRAN-CODE = 'UPDATE'
378	The current location is SAM3 at line
379	NUM-UPDATE-REQUESTS = +000000004
380	The current location is SAM3 at line
381	TRAN-KEY = '11204A'
382	The current location is SAM3 at line
383	CUST-KEY =
384	TRAN-KEY = '11204A'
385	The current location is SAM3 at line
386	TRAN-KEY = '11204A'
387	The current location is SAM3 at line
388	CUST-KEY =
389	TRAN-KEY = '11204A'
390	The current location is SAM3 at line
391	TRAN-KEY = '11204A'
392	The current location is SAM3 at line
393	CUST-KEY =
394	TRAN-KEY = '11204A'
395	The current location is SAM3 at line
396	TRAN-KEY = '11204A'
397	The current location is SAM3 at line
398	CUST-KEY =
399	TRAN-KEY = '11204A'
400	The current location is SAM3 at line
401	TRAN-KEY = '11204A'
402	The current location is SAM3 at line
403	CUST-KEY =
404	TRAN-KEY = '11204A'
405	The current location is SAM3 at line
406	TRAN-KEY = '11204A'
407	The current location is SAM3 at line
408	CUST-KEY =
409	TRAN-KEY = '11204A'
410	The current location is SAM3 at line
411	TRAN-KEY = '11204A'
412	The current location is SAM3 at line
413	CUST-KEY =
414	TRAN-KEY = '11204A'
415	The current location is SAM3 at line
416	TRAN-KEY = '11204A'
417	The current location is SAM3 at line
418	CUST-KEY =
419	TRAN-KEY = '11204A'
420	The current location is SAM3 at line
421	TRAN-KEY = '11204A'
422	The current location is SAM3 at line
423	CUST-KEY =
424	TRAN-KEY = '11204A'
425	The current location is SAM3 at line
426	TRAN-KEY = '11204A'
427	The current location is SAM3 at line
428	CUST-KEY =
429	TRAN-KEY = '11204A'
430	The current location is SAM3 at line
431	TRAN-KEY = '11204A'
432	The current location is SAM3 at line
433	CUST-KEY =
434	TRAN-KEY = '11204A'
435	The current location is SAM3 at line
436	TRAN-KEY = '11204A'
437	The current location is SAM3 at line
438	CUST-KEY =
439	TRAN-KEY = '11204A'
440	The current location is SAM3 at line
441	TRAN-KEY = '11204A'
442	The current location is SAM3 at line
443	CUST-KEY =
444	TRAN-KEY = '11204A'
445	The current location is SAM3 at line
446	TRAN-KEY = '11204A'
447	The current location is SAM3 at line
448	CUST-KEY =
449	TRAN-KEY = '11204A'
450	The current location is SAM3 at line
451	TRAN-KEY = '11204A'
452	The current location is SAM3 at line
453	CUST-KEY =
454	TRAN-KEY = '11204A'
455	The current location is SAM3 at line
456	TRAN-KEY = '11204A'
457	The current location is SAM3 at line
458	CUST-KEY =
459	TRAN-KEY = '11204A'
460	The current location is SAM3 at line
461	TRAN-KEY = '11204A'
462	The current location is SAM3 at line
463	CUST-KEY =
464	TRAN-KEY = '11204A'
465	The current location is SAM3 at line
466	TRAN-KEY = '11204A'
467	The current location is SAM3 at line
468	CUST-KEY =
469	TRAN-KEY = '11204A'
470	The current location is SAM3 at line
471	TRAN-KEY = '11204A'
472	The current location is SAM3 at line
473	CUST-KEY =
474	TRAN-KEY = '11204A'
475	The current location is SAM3 at line
476	TRAN-KEY = '11204A'
477	The current location is SAM3 at line
478	CUST-KEY =
479	TRAN-KEY = '11204A'
480	The current location is SAM3 at line
481	TRAN-KEY = '11204A'
482	The current location is SAM3 at line
483	CUST-KEY =
484	TRAN-KEY = '11204A'
485	The current location is SAM3 at line
486	TRAN-KEY = '11204A'
487	The current location is SAM3 at line
488	CUST-KEY =
489	TRAN-KEY = '11204A'
490	The current location is SAM3 at line
491	TRAN-KEY = '11204A'
492	The current location is SAM3 at line
493	CUST-KEY =
494	TRAN-KEY = '11204A'
495	The current location is SAM3 at line
496	TRAN-KEY = '11204A'
497	The current location is SAM3 at line
498	CUST-KEY =
499	TRAN-KEY = '11204A'
500	The current location is SAM3 at line
501	TRAN-KEY = '11204A'
502	The current location is SAM3 at line
503	CUST-KEY =
504	TRAN-KEY = '11204A'
505	The current location is SAM3 at line
506	TRAN-KEY = '11204A'
507	The current location is SAM3 at line
508	CUST-KEY =
509	TRAN-KEY = '11204A'
510	The current location is SAM3 at line
511	TRAN-KEY = '11204A'

Debug Console View – Other Debug Console Options

There are a number of useful Debug Console commands that work with IDz

SET INTERCEPT ON - allows you to see your program's DISPLAY statement output
that ordinarily goes to //SYSOUT

The screenshot shows the IDz interface with the 'XSAM.jcl' file open. The code editor displays several DISPLAY statements:

```
Line 353      Column 1      Insert      Browse
-----+-----+-----+-----+-----+-----+-----+-----+
000345 022600 | DISPLAY 'SAM1 STARTED DATE = ' CURRENT-MONTH //'
000346 022700 |           CURRENT-DAY // CURRENT-YEAR '(mm/dd/yy)' .
000347 022701*| UPON CONSOLE.
000348 022800 | DISPLAY '          TIME = ' CURRENT-HOUR ':'
000349 022900 |           CURRENT-MINUTE ':' CURRENT-SECOND .
000350 022901*| UPON CONSOLE.
000351 022902 |
000352 023000 |
000353 023100 | PERFORM 700-OPEN-FILES .
000354 023200 | PERFORM 800-INIT-REPORT .
000355 023300 |
000356 023400 | PERFORM 730-READ-CUSTOMER-FILE .
000357 023500 | PERFORM 100-PROCESS-TRANSACTIONS
000358 023600 |           UNTIL WS-TRAN-EOF = 'Y' .
000359 023700 |
```

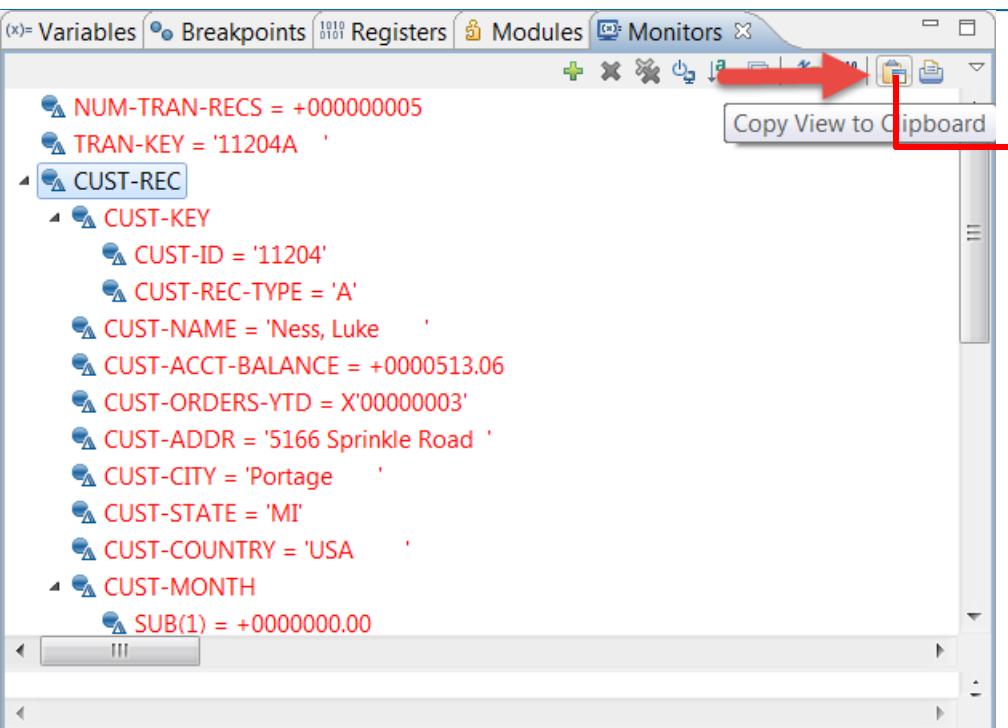
The screenshot shows the IDz interface with the 'Debug Console' tab selected. The console window displays the following messages:

```
EQA2261E An error occurred while opening file: INSPPREF. The file may not exist, or is not accessible.
EQA2458I SVC Screening is disabled by EQAOPTS. Handling of non-LE events is not available. Debugging of no
EQA2383I The environment is not yet fully initialized. Use Step or Run.
SET INTERCEPT ON
SAM1 STARTED DATE = 05/31/12 (mm/dd/yy)
TIME = 09:07:40
Program was stopped due to line/statement breakpoint at statement 353.
```

At the bottom, the 'Debug Engine Command' field contains 'SET INTERCEPT ON'.

Hint: Ctrl+Spacebar with your cursor in the Command area lists the Console commands that work with IDz

Copy the contents of a monitor to the Windows-Clipboard



Copy View to Clipboard
Ctrl+V
(Paste)

The screenshot shows a Microsoft Notepad window titled "Untitled - Notepad". The window contains the same list of variables and their values that were copied from the Eclipse Monitors View. The text in the Notepad is:

```
NUM-TRAN-RECS = +000000005
TRAN-KEY = '11204A'
CUST-REC
CUST-KEY
  CUST-ID = '11204'
  CUST-REC-TYPE = 'A'
CUST-NAME = 'Ness, Luke'
CUST-ACCT-BALANCE = +0000513.06
CUST-ORDERS-YTD = X'00000003'
CUST-ADDR = '5166 Sprinkle Road'
CUST-CITY = 'Portage'
CUST-STATE = 'MI'
CUST-COUNTRY = 'USA'
CUST-MONTH
  SUB(1) = +0000000.00
  SUB(2) = +0000004.84
  SUB(3) = +0000004.84
  SUB(4) = +0000000.00
  SUB(5) = +0000007.26
  SUB(6) = +0000000.00
  SUB(7) = +0000010.89
  SUB(8) = +0000009.68
  SUB(9) = +0000000.00
  SUB(10) = +0000000.00
  SUB(11) = +0000010.89
  SUB(12) = +0000007.26
CUST-OCCUPATION = 'Paranormal Investigator'
CUST-NOTES = 'ample notes. More notes. And yet more notes. Some sample notes. More notes. And yet more notes. Some sample notes. M'
```

CUST-DATA-1 = '
CUST-DATA-2 = '

Sometimes you'll want to save the contents of a variable (including a File/Record) to some external source – like an ASCII/Notepad file, spreadsheet, etc.

1. Monitor the variable – and from the Monitors View
2. Expand the “nodes” (Group fields)
3. Click Copy view to Clipboard
 - This copies the contents of the View to your Windows paste buffer
4. Paste the copied contents →

Using Automonitor to Capture Test Run “State”

The screenshot shows the IBM Developer for z Systems interface. The top menu bar includes File, Edit, Navigate, Search, Project, Data, Run, Window, Help. The toolbar below has various icons for file operations and navigation. The main window has tabs for 'XSAM3.jcl' and 'DDS0001.TEST.SYSDEBUG(SAM4).cob'. The code editor displays COBOL code with several lines highlighted in green, indicating they have been executed. The 'Console' tab at the bottom left shows log messages related to variable values and line numbers. A modal dialog box titled 'Event Occurred' is open, stating 'The following event has occurred: CEE3207S The system detected a data exception (System Completion Code=0C7). Select which action to take: Step into handler (radio button selected), Run, Examine. Process: 587253952 Program: SAM3 Thread Name/TID: 1'. The bottom right corner contains a bulleted list of tips for using Automonitor.

- Automonitor captures both executed lines and variable “state” (value changes over time)
- Use Automonitor in combination with Statement Recording (next topic) to solve ABENDs
- Note that Automonitor does not function during Resume
 - Use Step or Animated Step in conjunction with Automonitor + Recording (next topic)

Record and Playback Test Run

- Debug Tool allows you to record and then playback recorded statements during Debug

Steps:

- From the Debug toolbar

▶ Click the white downward-pointing triangle, and select:

✓ **Show Playback Toolbar**

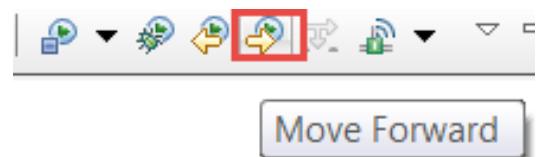
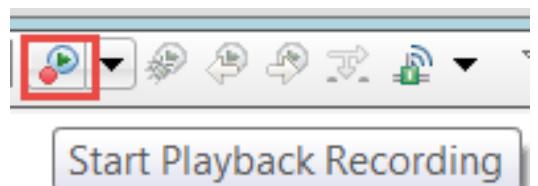
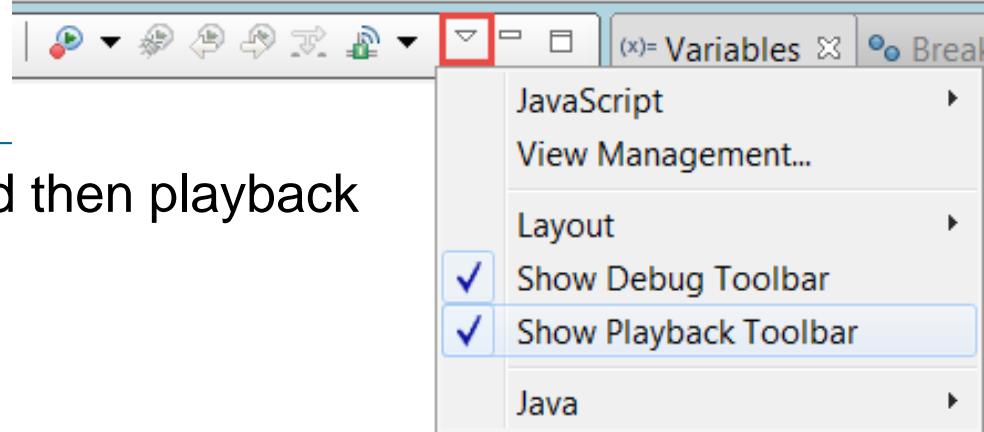
▶ From the Playback toolbar, click the green-go button, to start playback recording

▪ All of your statements are being recorded from that point until you:

- Stop recording
- End the Debug Session (ABEND or normal EOJ)

▶ If your program pauses (Breakpoint, etc.) you can backtrack through the recorded statements by pressing the **Move Back** icon on the toolbar

▶ You can also play the recorded statements forward, by clicking **Move Forward** on the toolbar



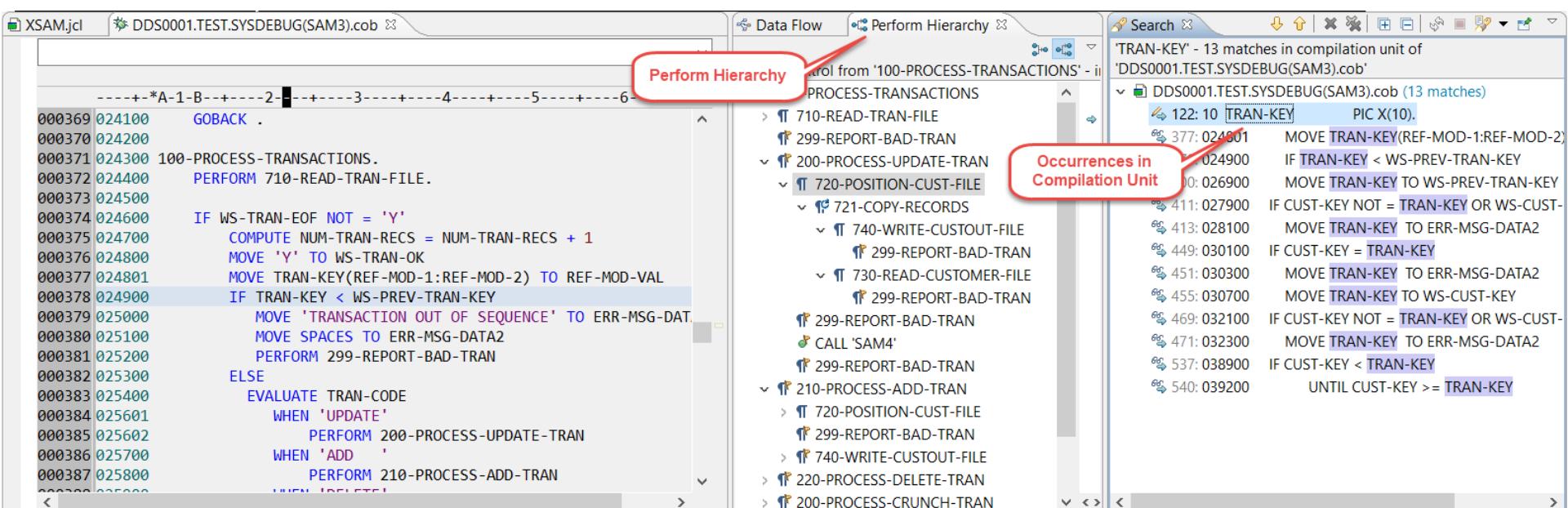
Handling Program ABENDs – Best Practice

- Debug Tool can receive control back from the system after an ABEND occurs
 - ▶ The program will be stopped at the ABENDING statement
- You can:
 - ▶ Allow the application to ABEND and terminate
 - Capture ABEND info with a product such as Fault Analyzer
 - Terminate the application and prevent further processing
 - ▶ Or continue running the program
- Usage note:
 - ▶ The **LE TRAP(ON)** option must be active

Source Context Menu Options – During Debug

During Debug (and without interrupting/influencing your running executable) you can stop at any moment and use the Context menu to access source navigation tools and program understanding tools such as:

- ▶ Perform Hierarchy, Program Control Flow, Data Element analysis, Filtering, Code Review, Show unreachable code, Occurrences in Compilation Unit, etc.
- Combining Static Analytics using the Context Menu options and Dynamic Analysis (from Debugging) you can get a very complete picture of your program's semantics - its operations and business logic



Integrating Context Menu options – Data Flow

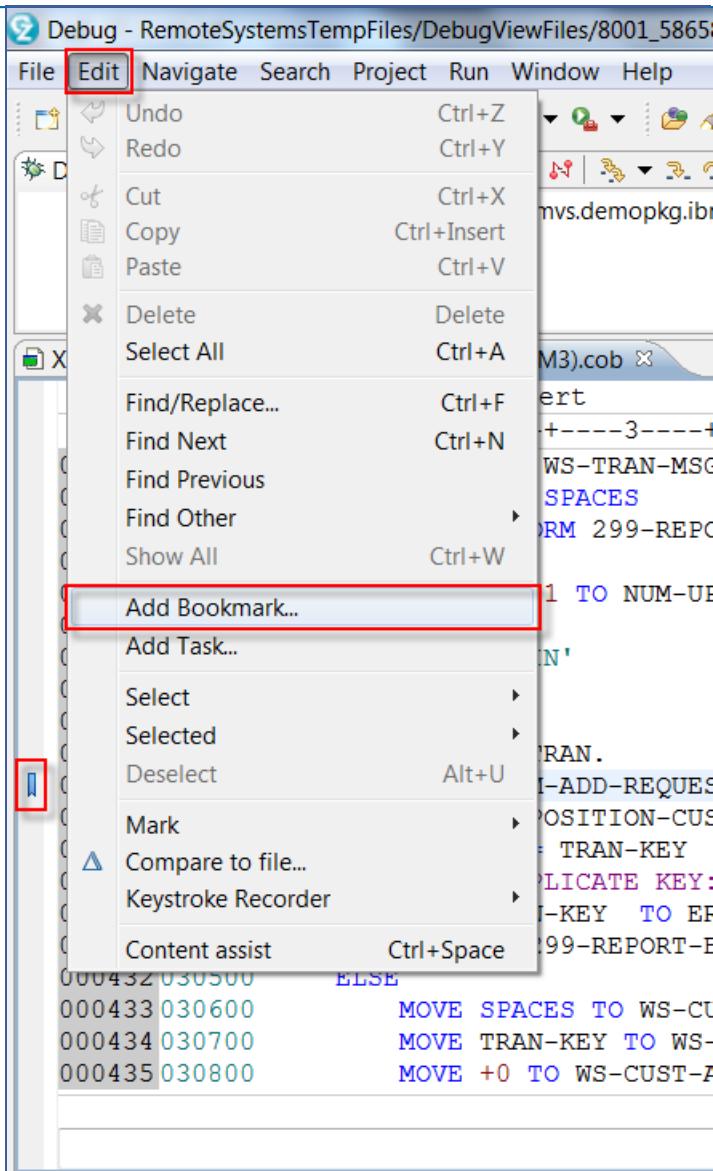
- The Static Analysis tools that you learned about in **Module 3 – Analyzing Your COBOL Programs** are all available while Debugging
- Show In > Data Flow** can be extremely useful during Debug

The screenshot shows the IBM Rational Application Developer interface with several windows open:

- XSAM.jcl DDS0001.TEST.SYSDEBUG(SAM3).cob**: The main editor window showing COBOL source code. A context menu is open over the code, with the "Show In" option highlighted.
- Data Flow**: A separate window displaying a complex data flow diagram with various nodes and connections. A red callout bubble points to the "Data Flow" option in the context menu, which is also highlighted.
- Visual Debug**: Another window showing a control flow graph with nodes like 000-MAIN, 790-CLOSE-FILES, 700-OPEN-FILES, 800-INIT-REPORT, and 730-READ-CUSTOMER-FILE. A red callout bubble points to the "Current Paragraph" option in the context menu.

Integrating Editor Features – Bookmarks

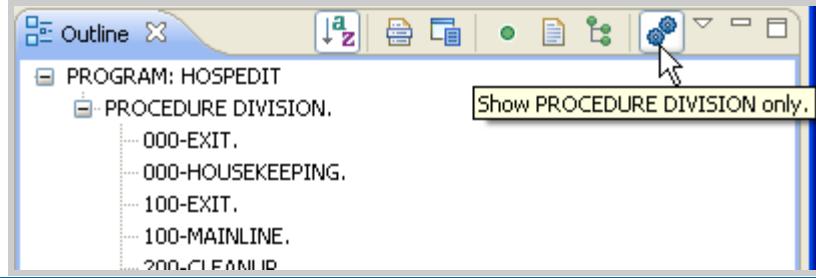
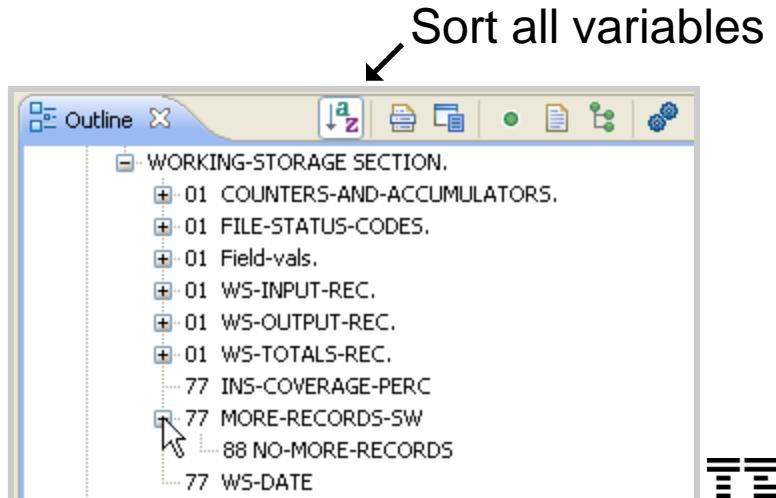
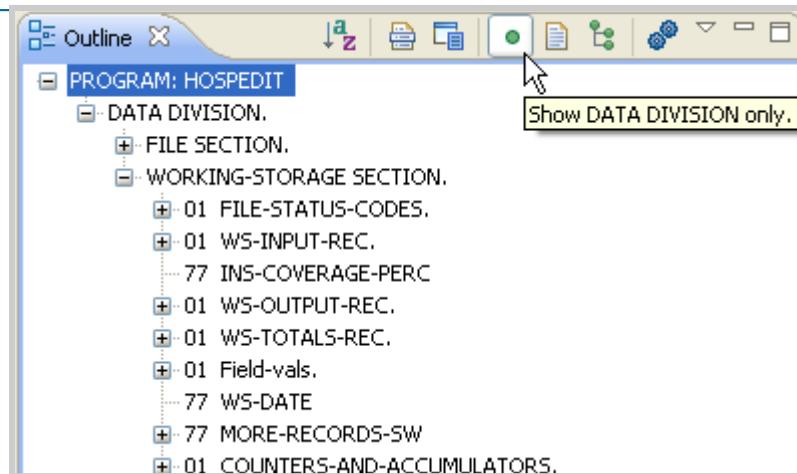
- It can be useful to set Bookmarks (and/or Tasks) into your code as:
 - ▶ Reminders of something to do
 - ▶ Navigation labels of where certain processing is
- To do this:
 - ▶ Navigate to the line you're interested in
 - Note – you can just scroll, you don't have to execute code
 - ▶ From the Edit menu, select:
 - Add Bookmark...
- Note that the Bookmarks are temporary – and only available during your Debug session



Outline View Options

For large programs, several additional Outline view features are available as toggled icons:

- Click to Toggle function on
- Click again to Toggle function off



Outline View – Code Sync Option

DDS0001.TEST.LISTING(HOSPEDIT).cob

Line 267 Column 1 Insert Browse

```
-----+*A-1-B---+---2---+---3---+---4---+---5---+
      READ INFILE INTO WS-INPUT-REC
          AT END MOVE "N" TO MORE-RECORDS-SW
          GO TO 100-EXIT
      END-READ
      ADD +1 TO RECORDS-READ.
100-EXIT.
      EXIT.

*****
* This routine should perform file close operations
*****
```

The Outline View on the right shows sections like PROCEDURE DIVISION, 000-HOUSEKEEPING, etc.

While debugging through PROCEDURE DIVISION Outline view synchronizes with code in paragraphs and sections

But if you scroll in the code while stepping, the Outline View syncs with your browsing activity

DDS0001.TEST.LISTING(HOSPEDIT).cob

Line 158

```
-----+*A-1-B---+---2---+---3---+---4---+---5---+---6---+---7---+
      88 NO-MORE-RECORDS VALUE 'N'.
01 COUNTERS-AND-ACCUMULATORS.
  05 RECORDS-READ           PIC S9(4) COMP.
  05 RECORDS-WRITTEN        PIC S9(4) COMP.
  05 ERROR-RECS             PIC S9(4) COMP.
  05 NBR-INPATIENTS         PIC S9(4) COMP.
  05 NBR-OUTPATIENTS        PIC S9(4) COMP.
  05 NBR-HMO                PIC S9(4) COMP.
  05 NBR-STATE-FED          PIC S9(4) COMP.
  05 NBR-NO-COVERAGE        PIC S9(4) COMP.
```

The Outline View on the right shows sections like 05 Field4, 05 Field5, etc., and highlights the 88 NO-MORE-RECORDS entry under the 01 COUNTERS-AND-ACCUMULATORS section.

If your program ABENDS during Debug

- The Debugger can receive control back from the system after an ABEND occurs
 - ▶ The program will be stopped at the ABENDING statement
- You can:
 - ▶ Allow the application to ABEND and terminate
 - Capture ABEND info with a product such as Fault Analyzer
 - Terminate the application and prevent further processing
 - ▶ Or continue running the program – to examine & research the ABEND situation with access to variable values at the time of the ABEND
- Usage notes:
 - ▶ The **LE TRAP(ON)** option must be active
 - ▶ Check out the Lookup View

Terminating the application

- **There are several options for terminating your application:**
 - ▶ **Remain in the debugger, and RESUME until the program runs to completion**
 - The program will terminate normally or with an abend
 - The return code is controlled by the program
 - ▶ **Disconnect the debugger, and allow the program to run to completion**
 - The program will terminate normally or with an abend
 - The return code is controlled by the program

Termination action buttons

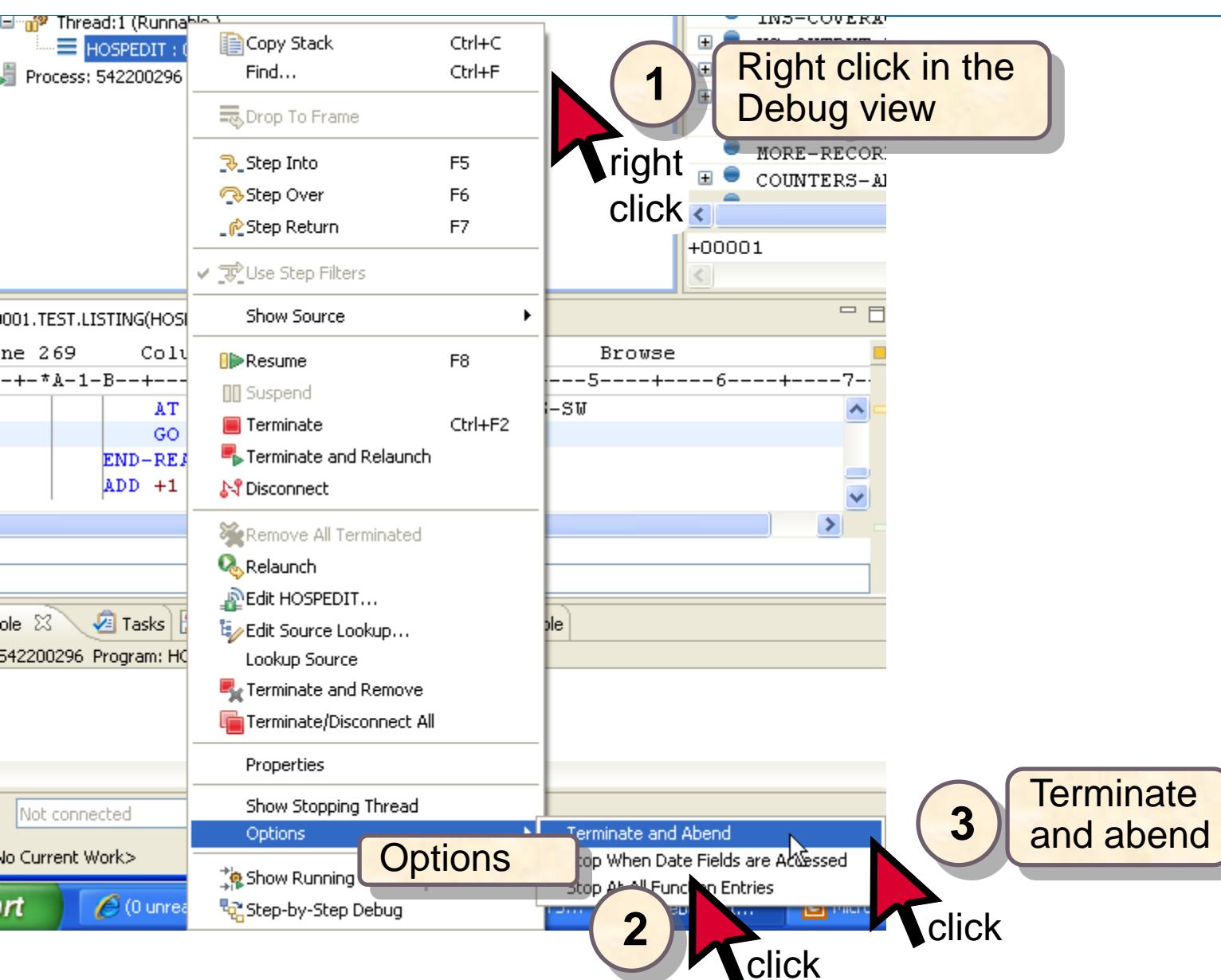
You can immediately terminate the application using action buttons

The screenshot shows the IDz Integrated Debugger interface. The title bar reads "Debug - \DebugViewFiles\8001_2436\DDS0001.TEST.LISTING(HOSPEDIT).cob - IBM Rational Dev". The toolbar has several icons, including a red square for "Terminate". A tooltip for this icon says "Terminate (Ctrl+F2)". Below the toolbar is a tree view showing "HOSPEDIT [Remote Compiled Application]", "Platform: zOS 390X", "Thread:1 (Runnable)", and "HOSPEDIT : 01". The main window displays COBOL source code with comments like "ENVIRONMENT DIVISION." and "SECURITY. NON-CONFIDENTIAL.". A status bar at the bottom says "The quick mark was set at the cursor location." Two callout boxes point to the "Terminate" button in the toolbar. The left callout box is titled "Terminate" and describes it as immediate termination with RC=0. The right callout box is titled "Disconnect" and describes it as disconnecting the debugger while allowing the program to run independently.

Terminate: Immediate termination of the application. No more program statements run. RC=0 is returned to the environment.

Disconnect: Disconnect the IDz Integrated Debugger from the application. The program continues to run from the current location without the debugger. And subsequent batch job steps can finish as well.

Force an immediate termination with abend



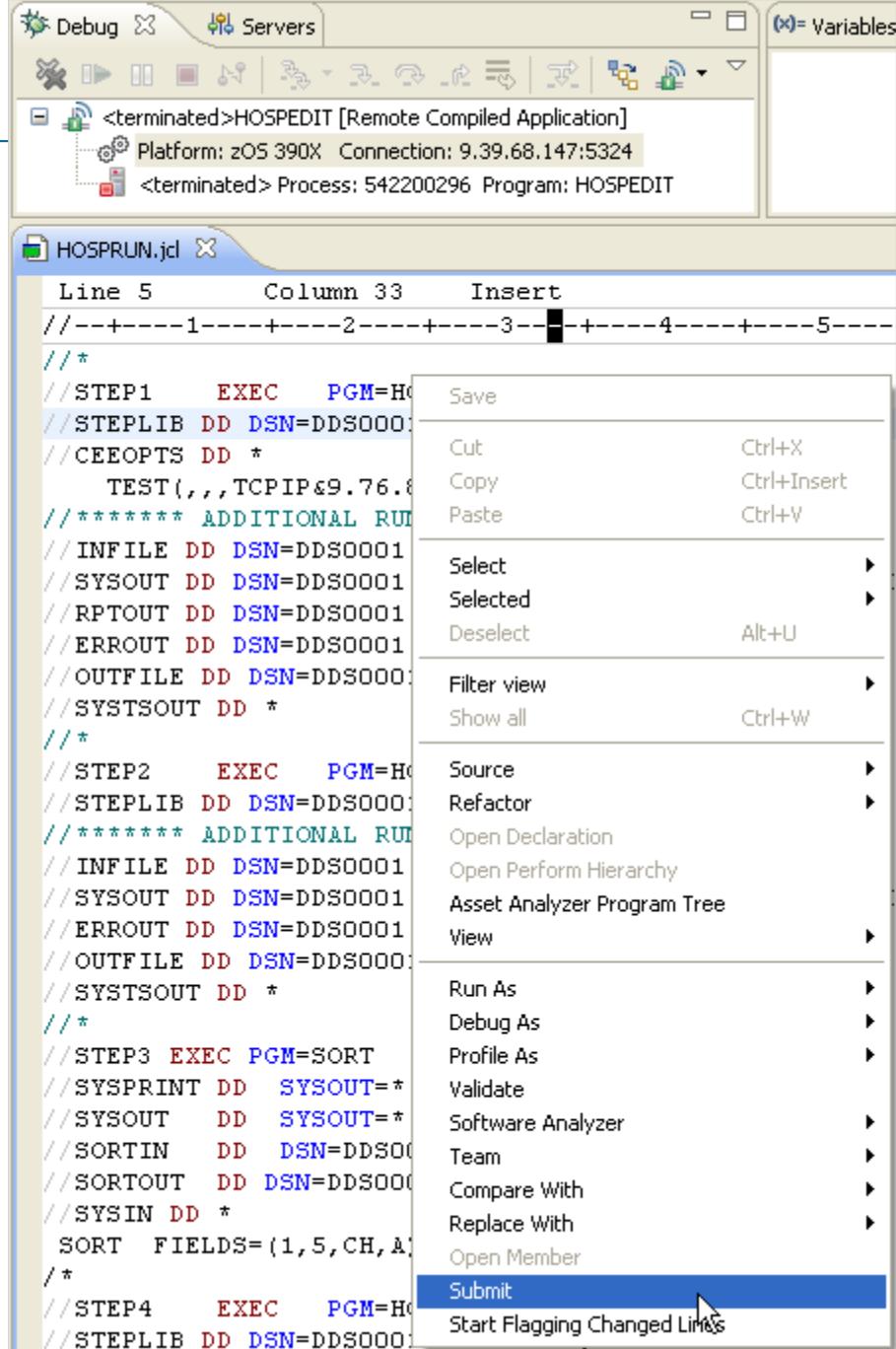
Restart Your Debugging Session

For batch debugging

- ▶ If your submitted JCL is still in the code (Content) area

- No need to return to the z/OS Projects perspective

- ▶ Right-click
- ▶ Select: Submit



Summary

Having completed this unit, you should now be able to:

- ▶ Describe where the debug engines are located
- ▶ Show how to set the workbench preferences for running and debugging
- ▶ Show how to invoke the debugger for local programs
- ▶ Describe the views of the Debug perspective
- ▶ Demonstrate how to set breakpoints in COBOL code
- ▶ Explain how to set up the COBOL compile options for remote debugging
- ▶ Show how to debug a remote batch COBOL program