IBM Developer for z Systems – for ISPF Developers

Module 9 – Debugging COBOL with IDz

DevOps

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

IBM Trademarks and Copyrights

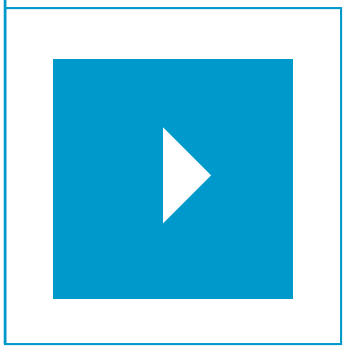
The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these 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**

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

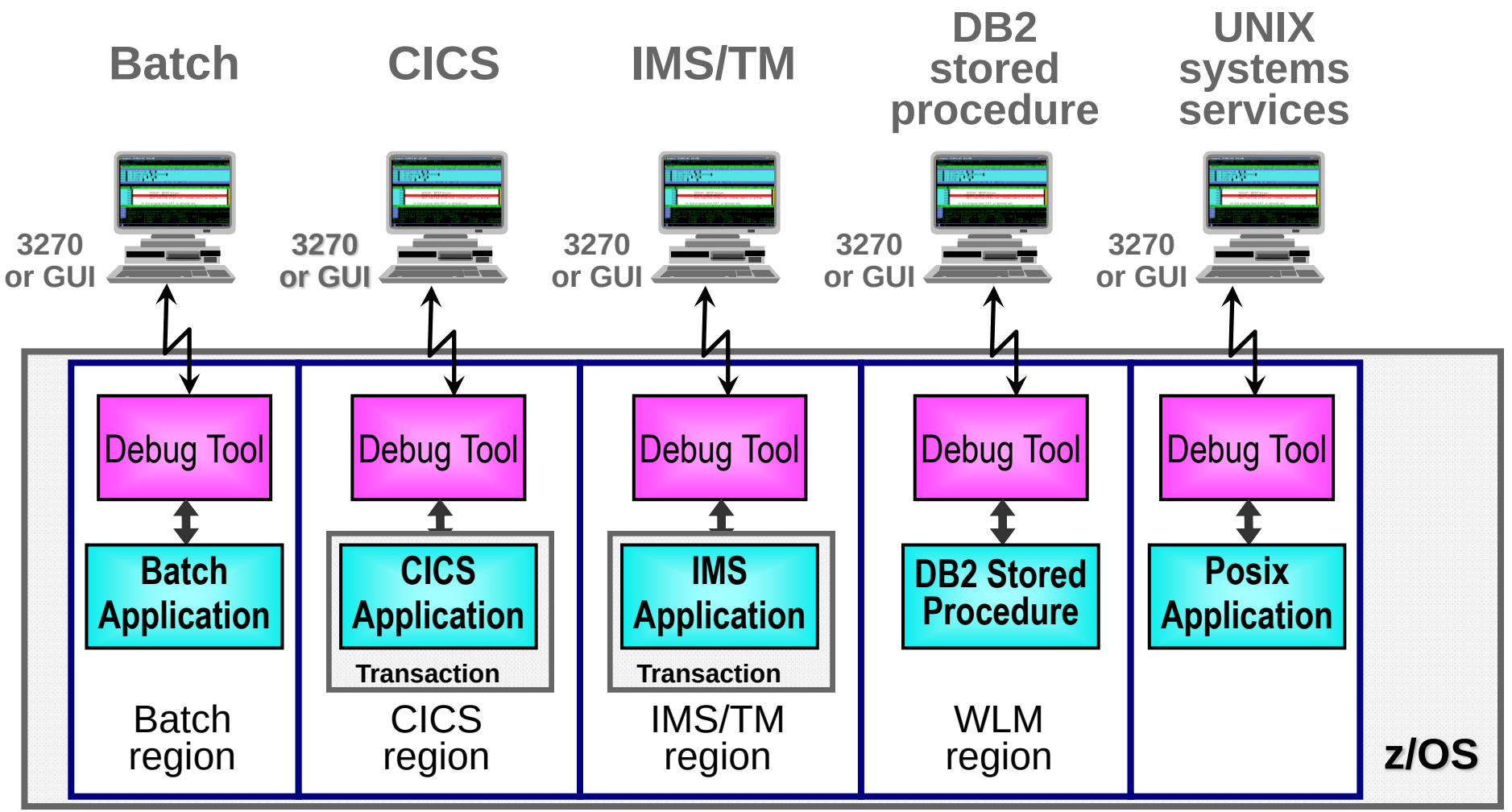
- Batch including Batch DB2 and Batch IMS (DL/I and BMP, QBMP)
- Enterprise COBOL (5.1, 4, 3.4) – Debug Tool supports OS/VS and VS COBOL II
- Enterprise PL/I
- C
- HLASM and BAL

Features (partial list):

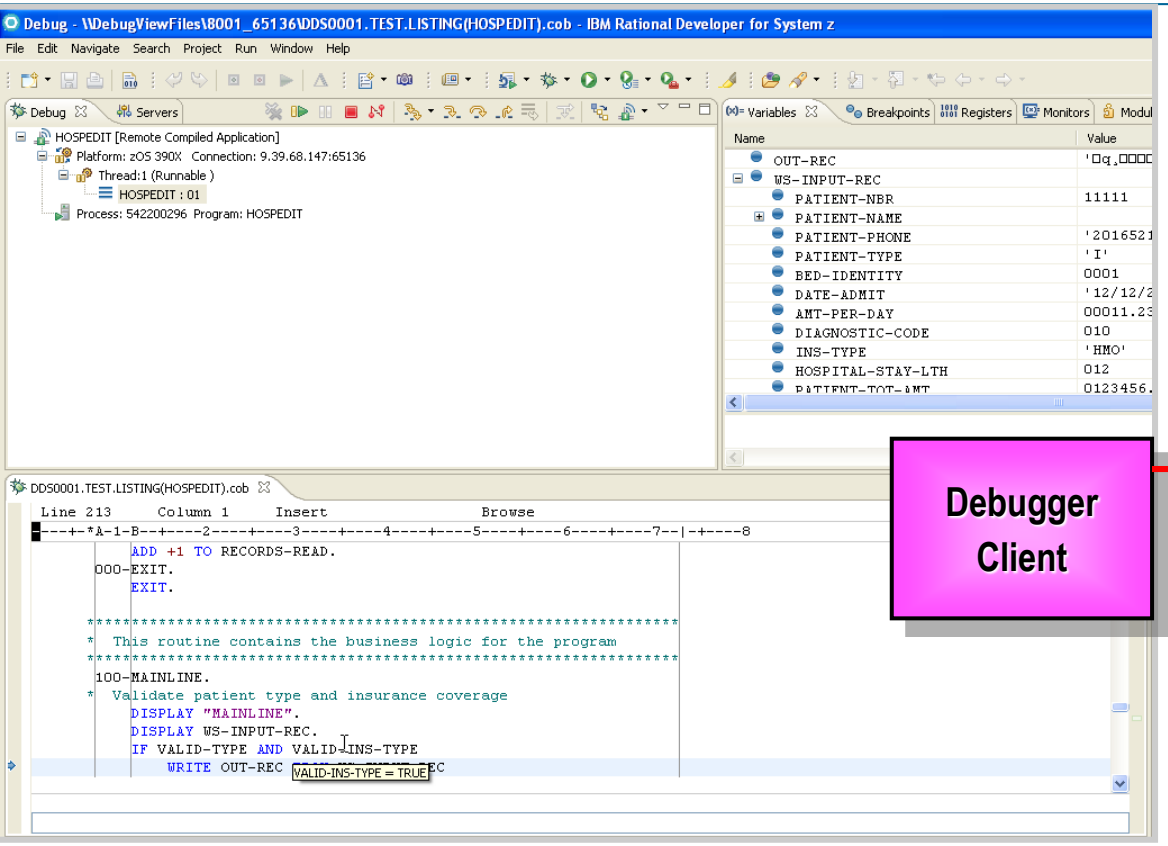
- Debug mixed-language/cross-platform applications
- Interactive, source-level debugging in IDz with program running on z/OS
- Display, monitor and alter program variables
- Breakpoints – all shapes and sizes – including the ability to set breakpoints in the program source file
- Watch monitors – Integrated 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

```
//STP00000 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
//            DD 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=*
//CUSTRPT  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:

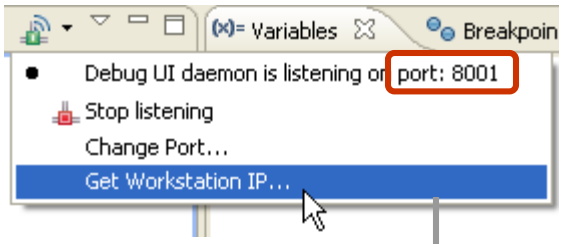
  //STEPLIB – DD Cards for the IDz & Debug Run-Time Libraries –
  - Unless Run-Time installed in Link Pack

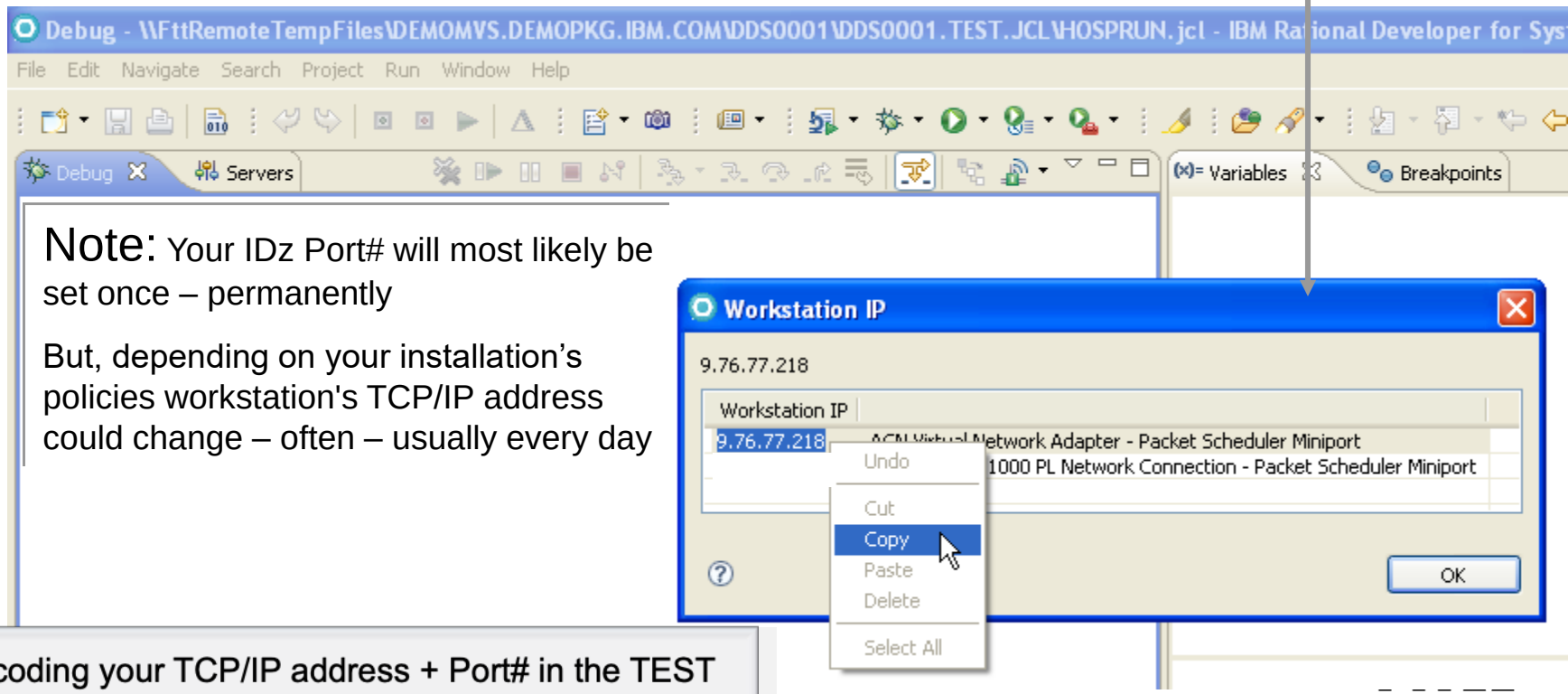
  //CEEOPTS
  - TEST(,,,DBM)
    - Invokes/connects to the debug listener on z/OS
  - ENVAR("EQA_STARTUP_KEY=CC")
    - Invokes **Code Coverage**

```
//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=*
//CUSTRPT  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**



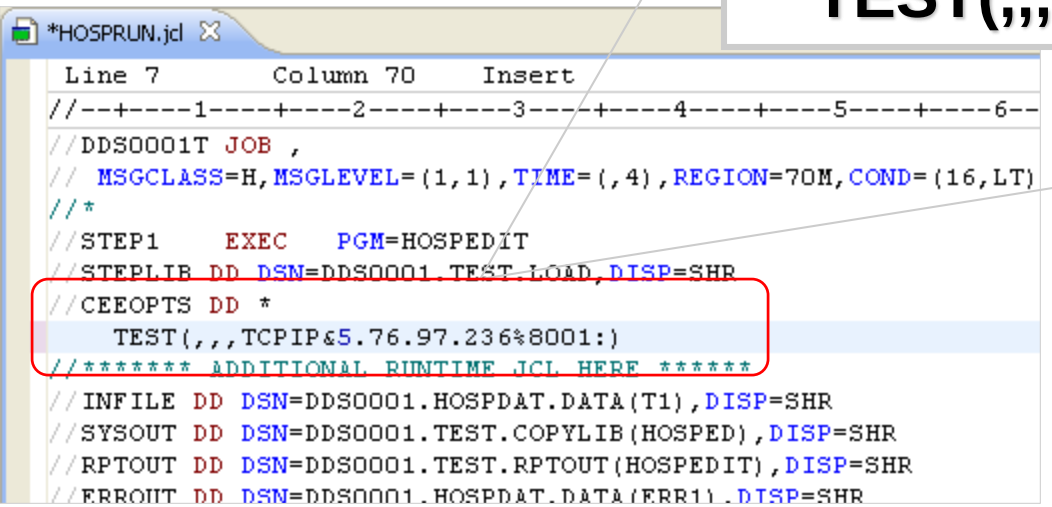


*** Hardcoding your TCP/IP address + Port# in the TEST statement may not be necessary. Check with your Sysprog

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

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

- **TEST(,,,DBM)** launches the "Internal Debugger"

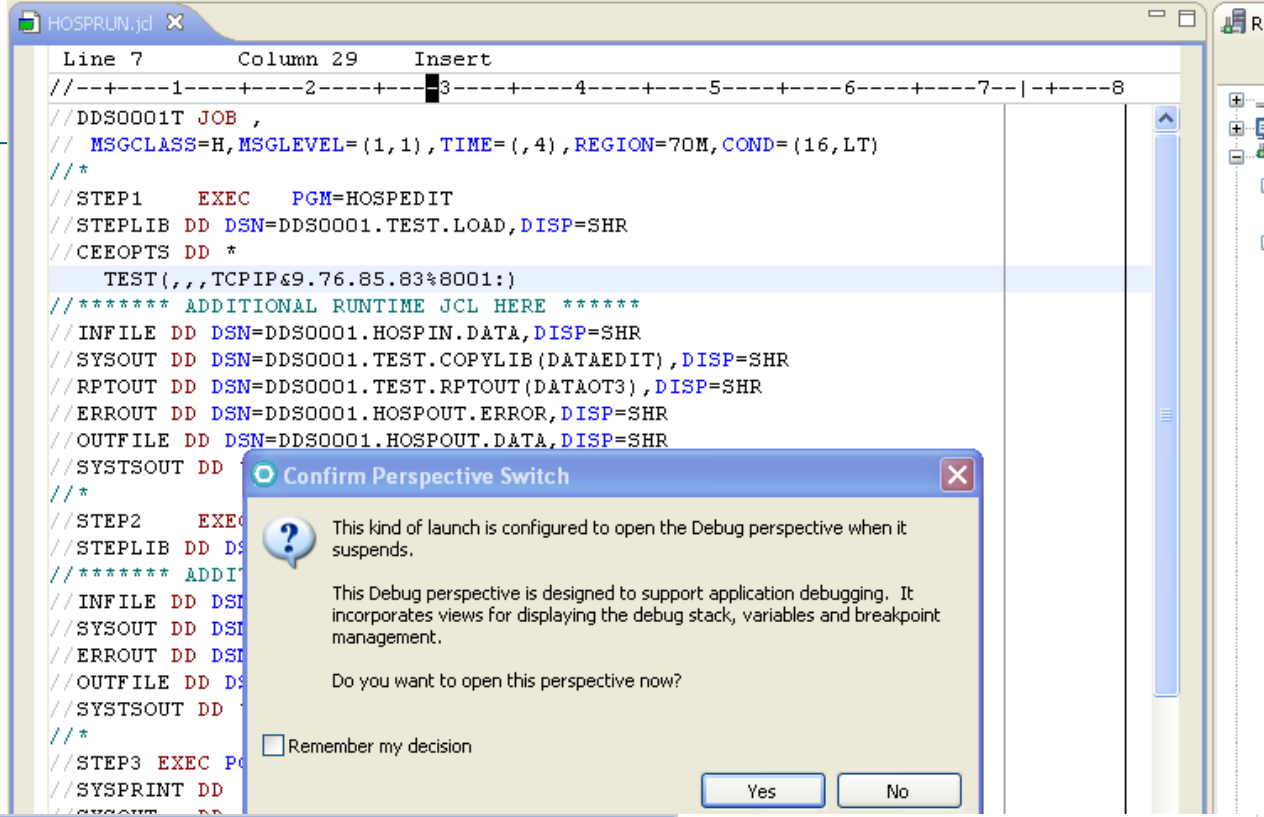
```
//CEEOPTS   DD   *
     TEST(,,,DBM)
```

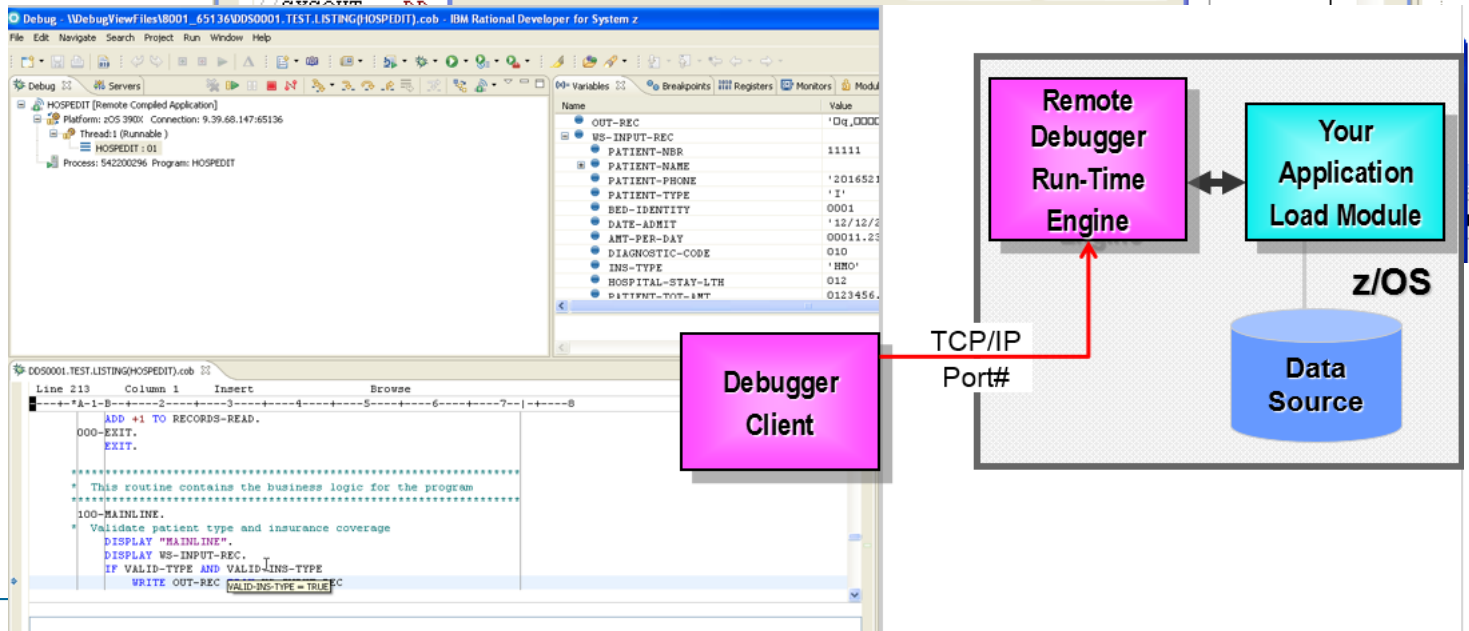
- **(,,,DBMDT%TSOID:)** launches "Debug Tool"

```
//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**

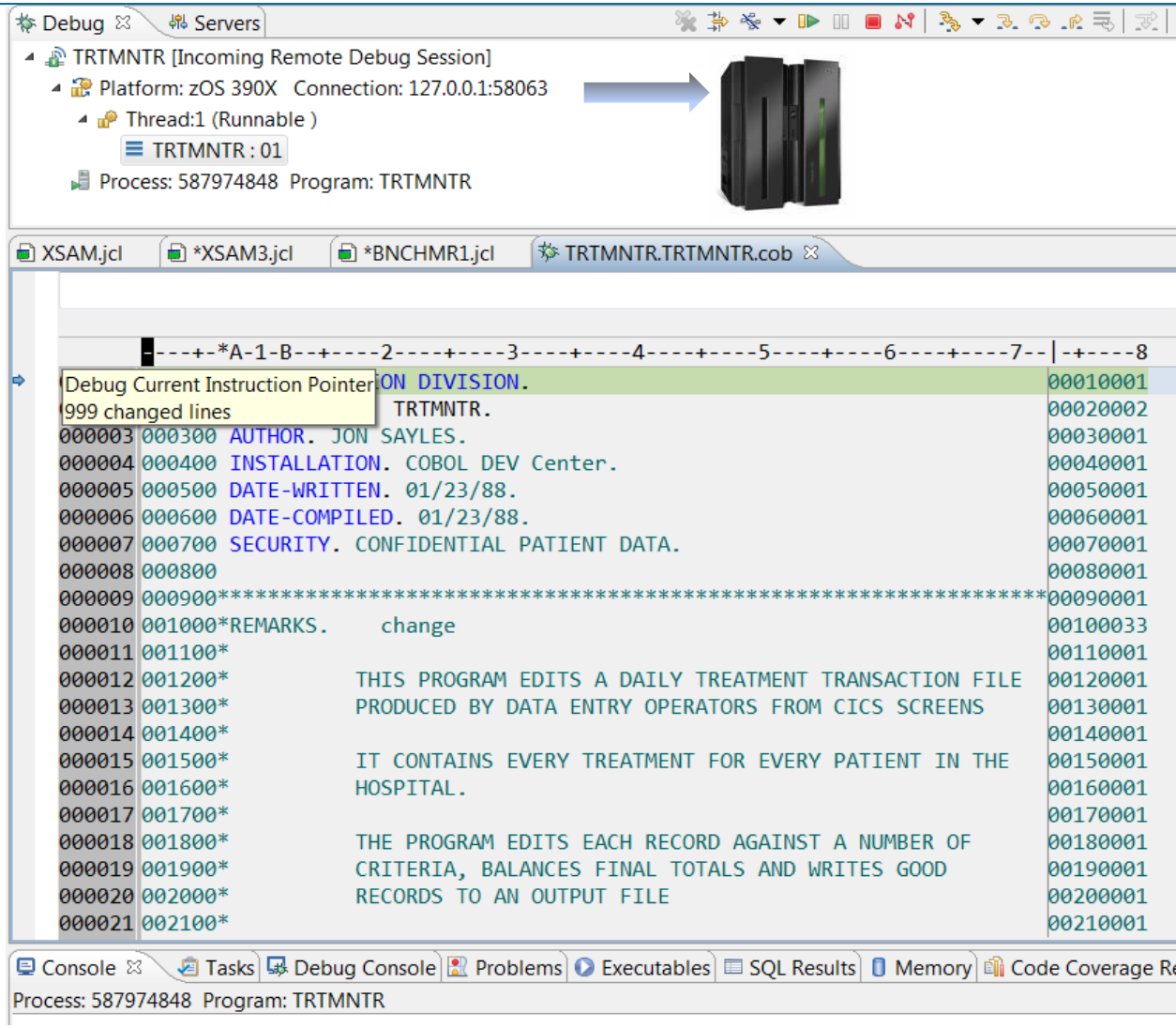




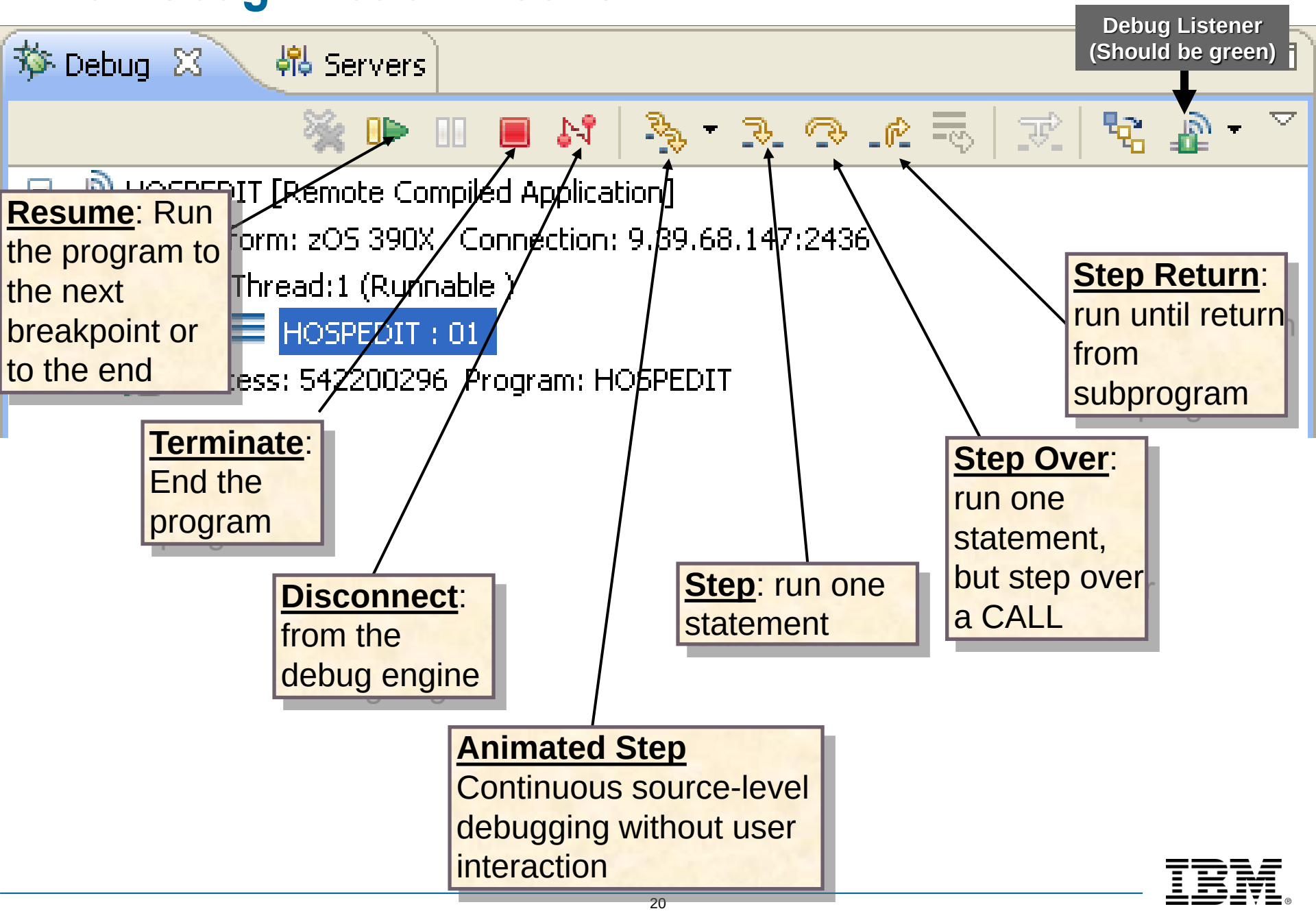
Ready to Debug

After you click YES at the Confirm Perspective switch prompt:

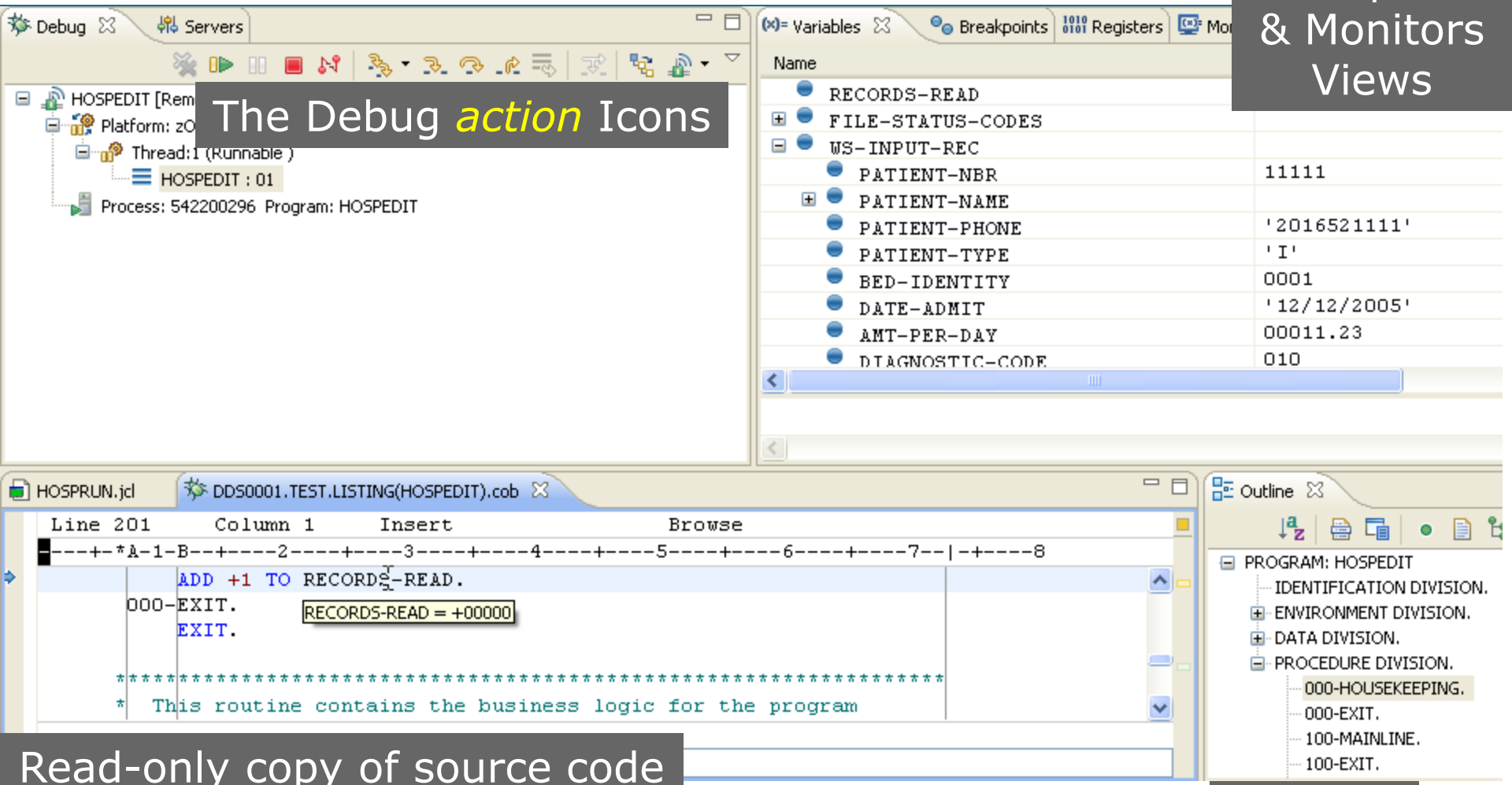
- The Debug Perspective opens
- A Read-Only copy of your source is copied down from z/OS to your IDz client & workstation
- A copy of your load module is NOT placed into an Address Space
  - The Current Instruction Pointer is **not** in the PROCEDURE DIVISION
- To open your Load Module into an Address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



The Debug "Action" Icons

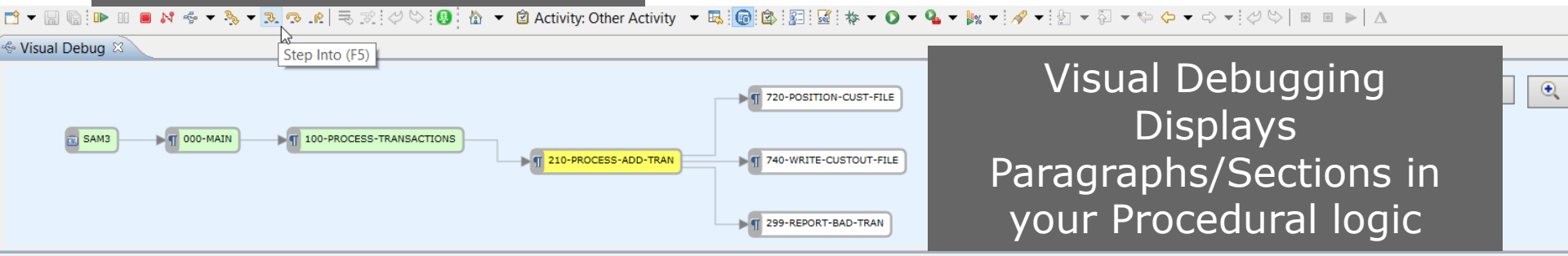


The Debug Perspective and Views

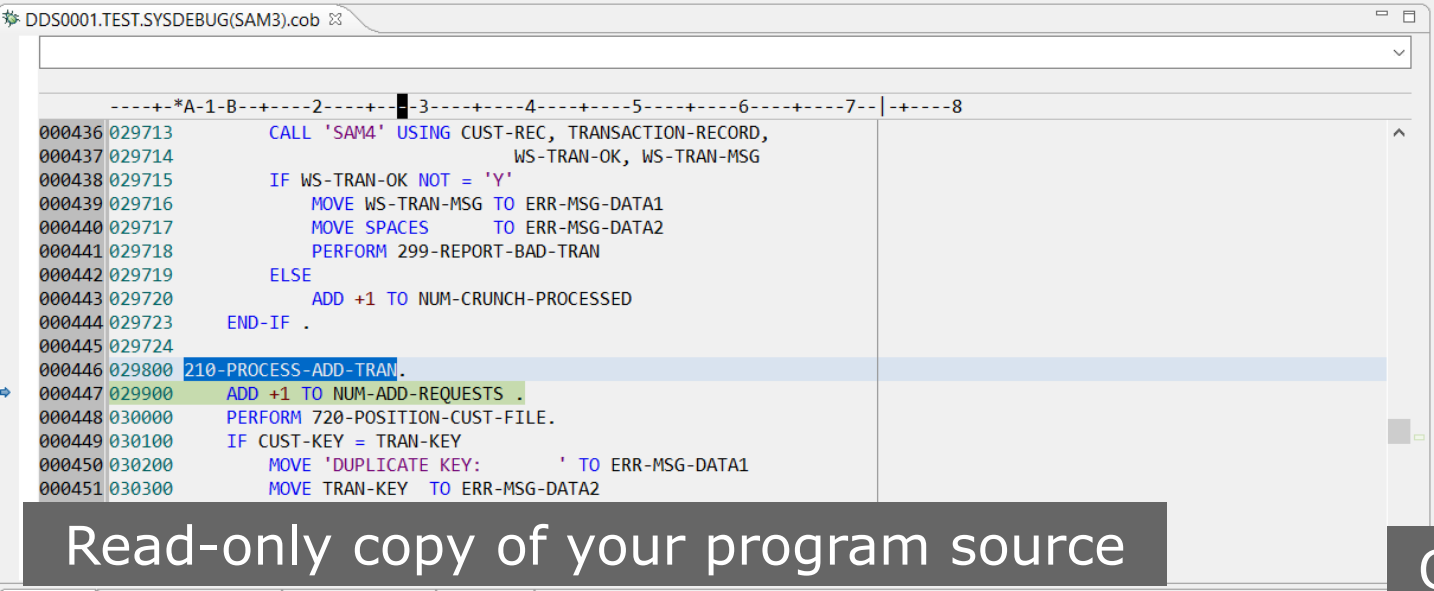


The Debug Perspective and Views – IDz v14.1.x and Later

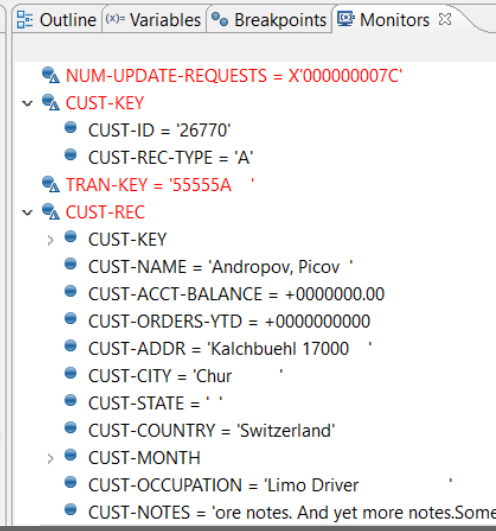
The Debug *action* Icons



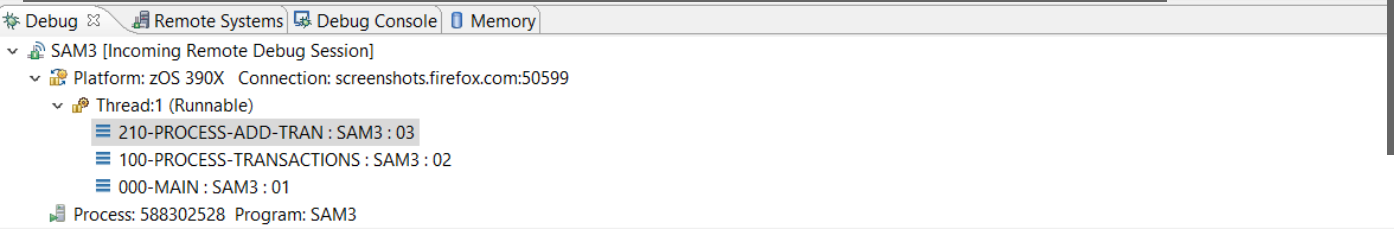
Visual Debugging Displays Paragraphs/Sections in your Procedural logic



Read-only copy of your program source



Outline, Variables Breakpoints & Monitors Views



Debugging your code

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

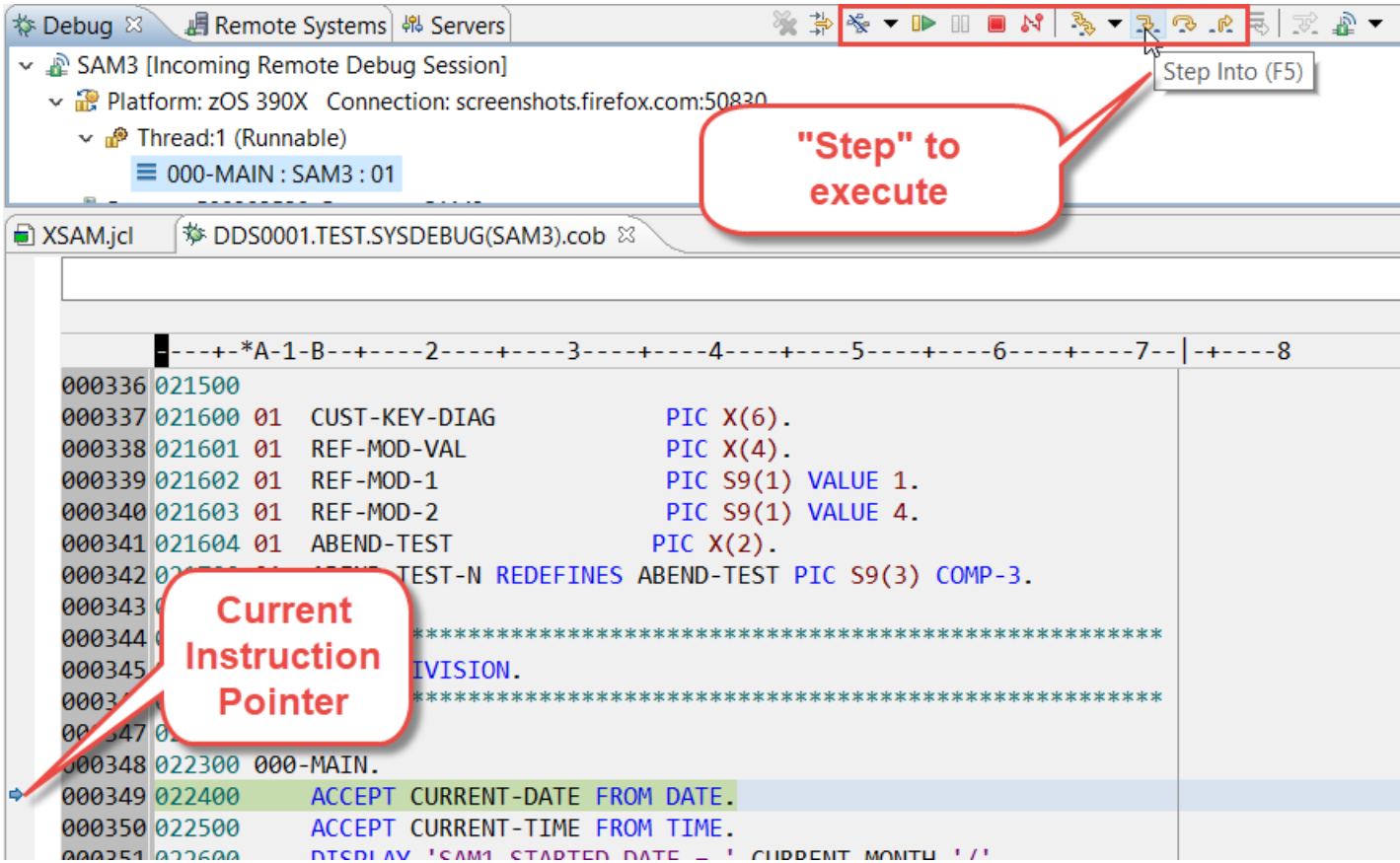
Step Return (F7)

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

Resume (F8)

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

The Action Icons



Additional Debug actions:

Terminate (Ctrl+F2)

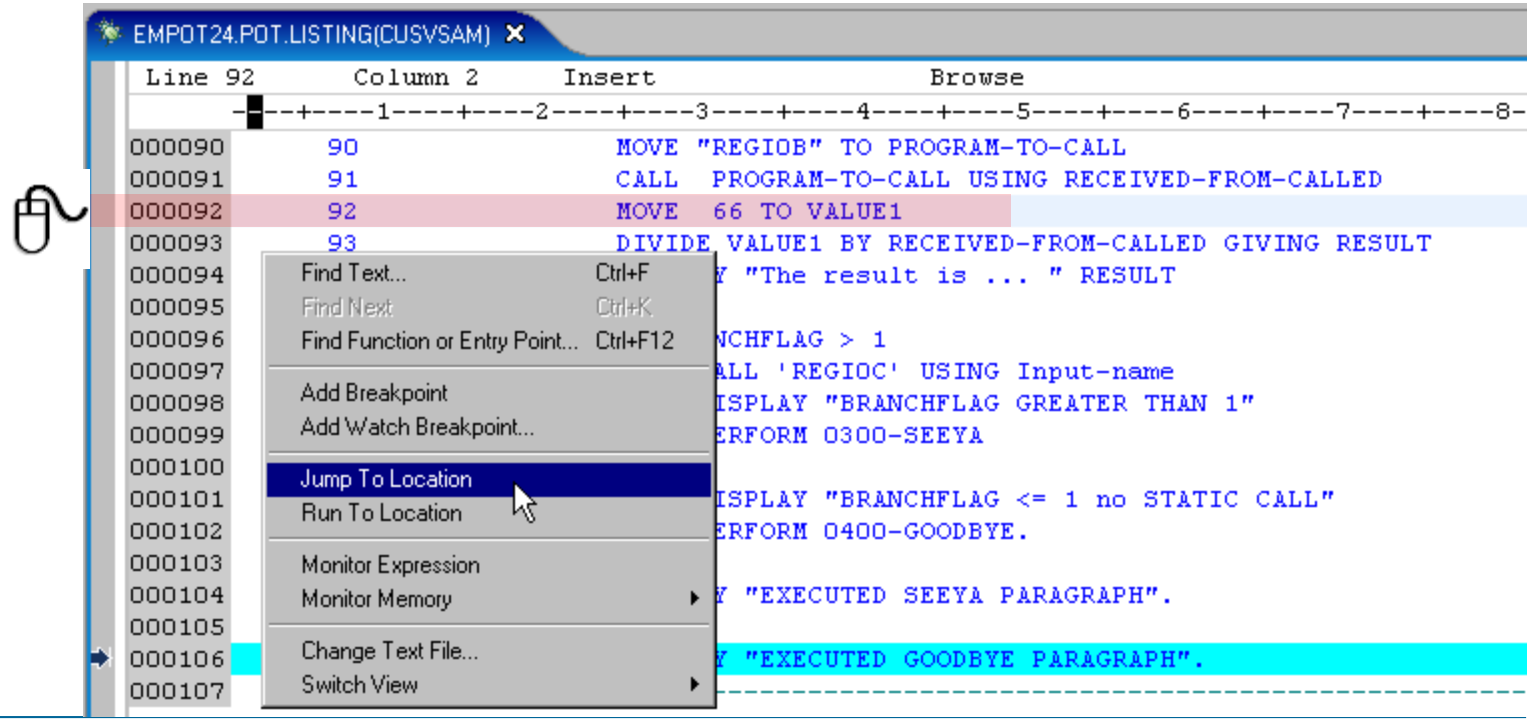
- Halts the Debug session

Open Visual Debug

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

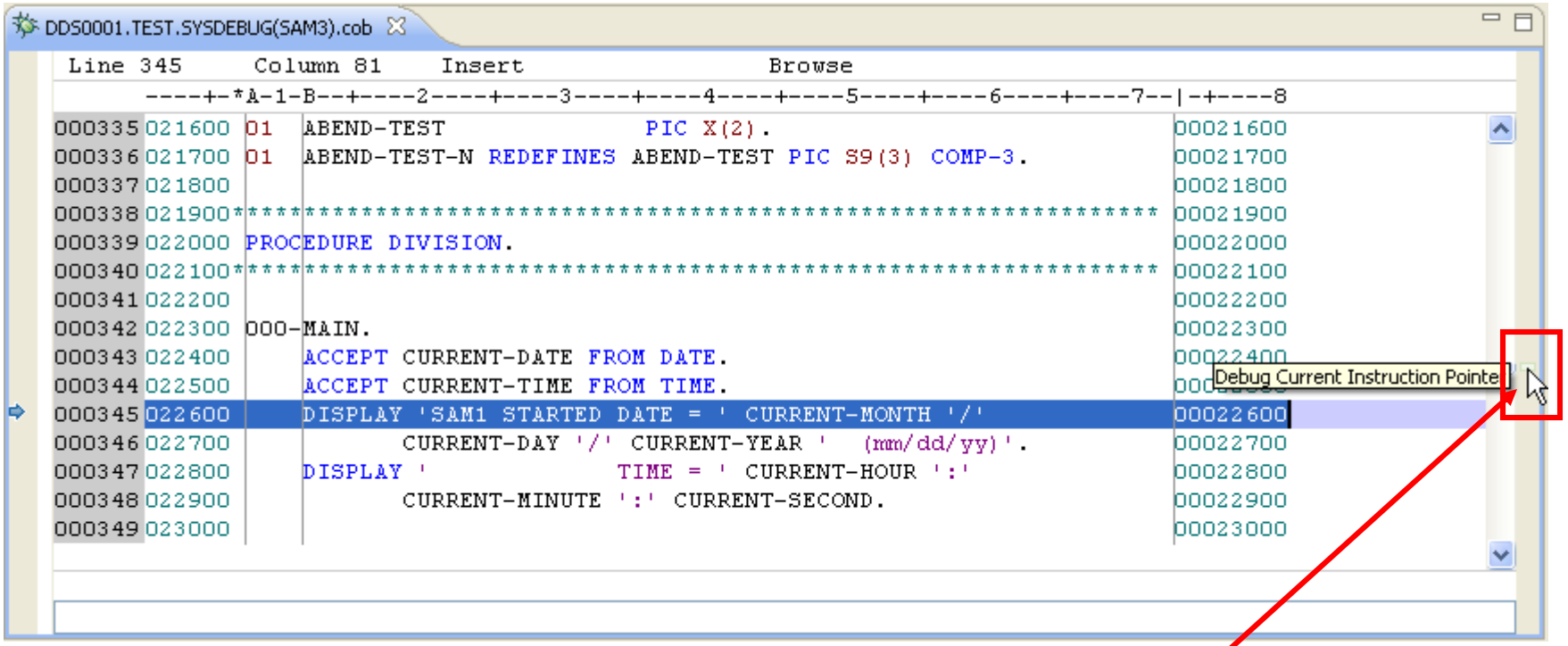
Additional Debug Actions – **Jump to** & **Run to Cursor Location**

- **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”

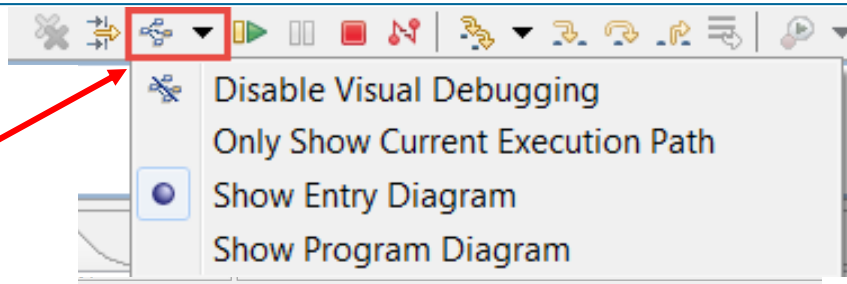
- To get back to the Current Instruction Pointer (the "next sequential instruction") – if you've navigated away within the source:
  - **Click the small blue rectangle in the right-hand margin of your source code**



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**



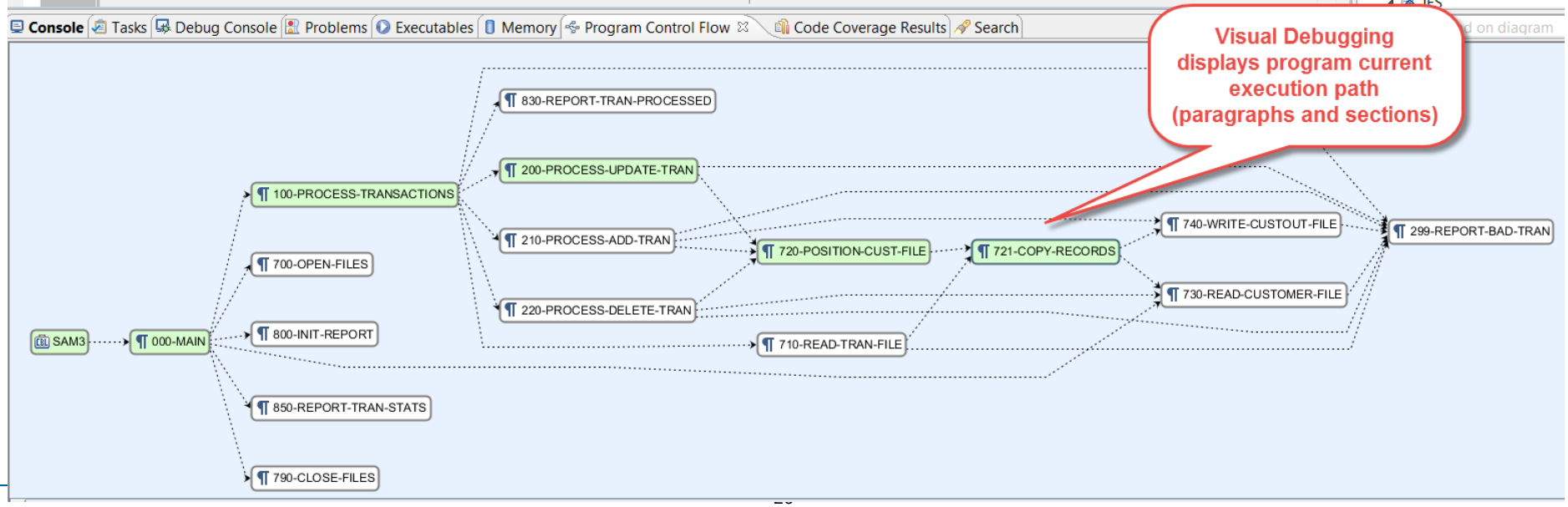
BNCHMR1.jcl | XSAM3.jcl | XSAM.jcl | SAM3.list.cob

```
      ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
000520 039000         IF WS-CUST-FILE-EOF NOT = 'Y'                          00039000
000521 039100             PERFORM 721-COPY-RECORDS                           00039100
000522 039200               UNTIL CUST-KEY >= TRAN-KEY                       00039200
000523 039300                  OR WS-CUST-FILE-EOF = 'Y'                     00039300
000524 039400         END-IF                                                 00039400
000525 039500     END-IF .                                                   00039500
000526 039600                                                                00039600
000527 039700 721-COPY-RECORDS.                                             00039700
000528 039800     MOVE CUST-REC TO WS-CUST-REC .                             00039800
000529 039900     PERFORM 740-WRITE-CUSTOUT-FILE .                           00039900
000530 040000     PERFORM 730-READ-CUSTOMER-FILE .                           00040000
000531 040100                                                                00040100
```

Outline | Remote Systems

- z/OS UNIX Files
- z/OS UNIX Shells
- MVS Files
  - Retrieved Data Set
  - My Data Sets (DDS
  - DNET045.B*.COBO
  - My JCL Datasets
  - My REXX Files
  - Test COBOL Source
  - Test Source Librari
  - Diagnostic Codes -
  - My Favorites
- TSO Commands
- IES

Console | Tasks | Debug Console | Problems | Executables | Memory | Program Control Flow | Code Coverage Results | Search

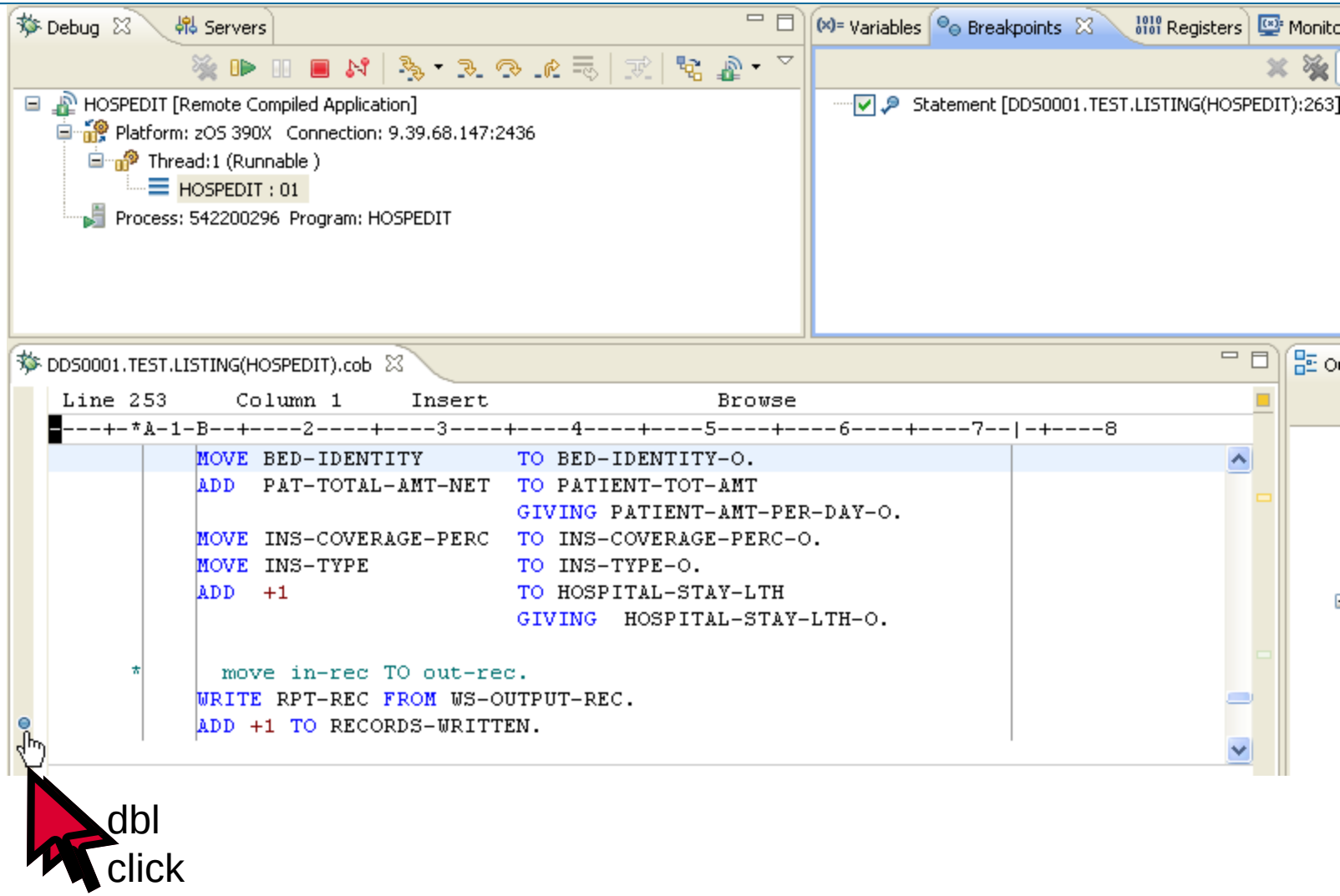


Statement Breakpoints

- 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`**

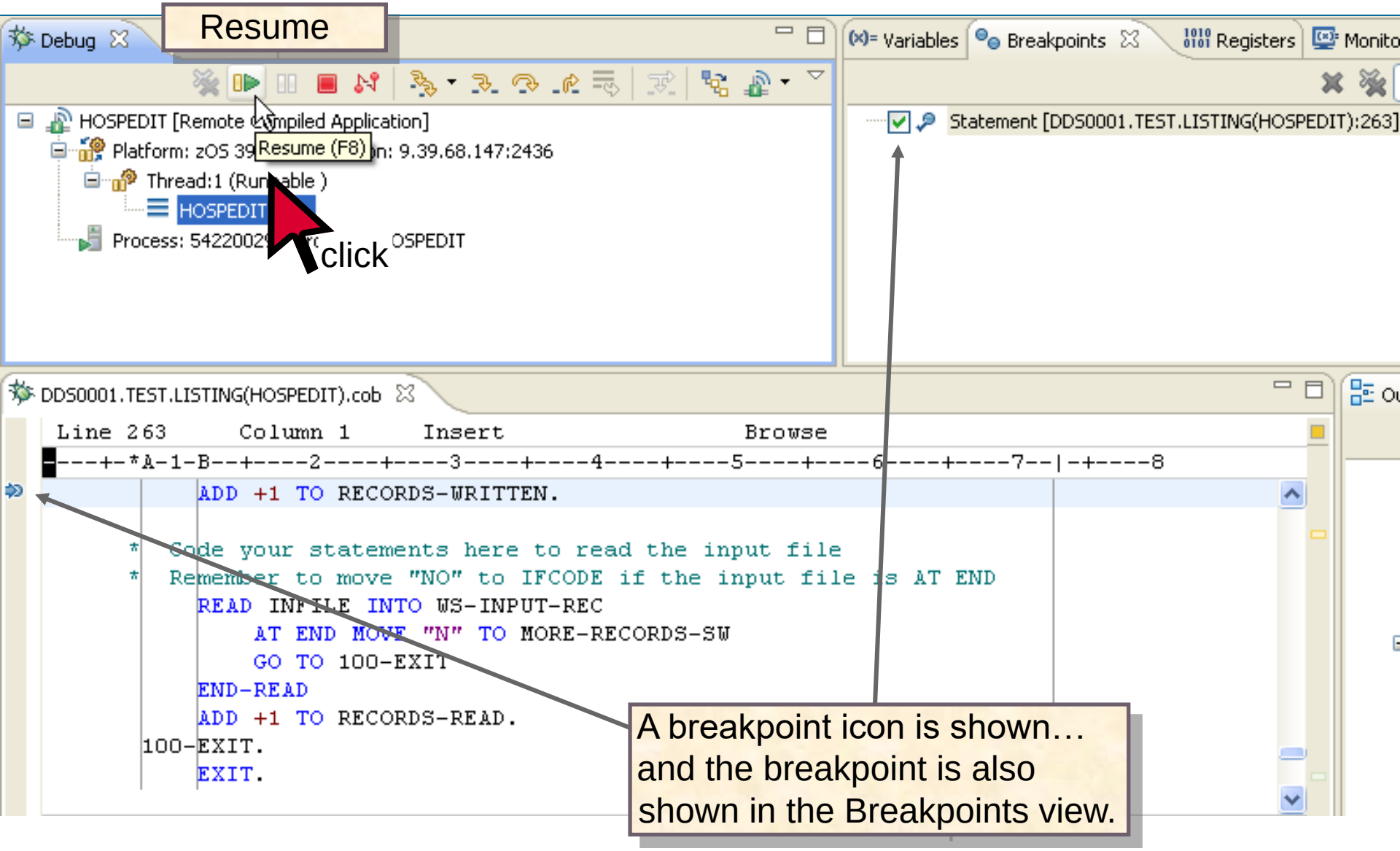
      …or…
    - **`VariableY = 'Abc'`**
- A breakpoint can be based on a frequency:
  - Stop the Nth time a statement runs
- A breakpoint can stop at the Entry to a Load Module

Setting Statement Breakpoints



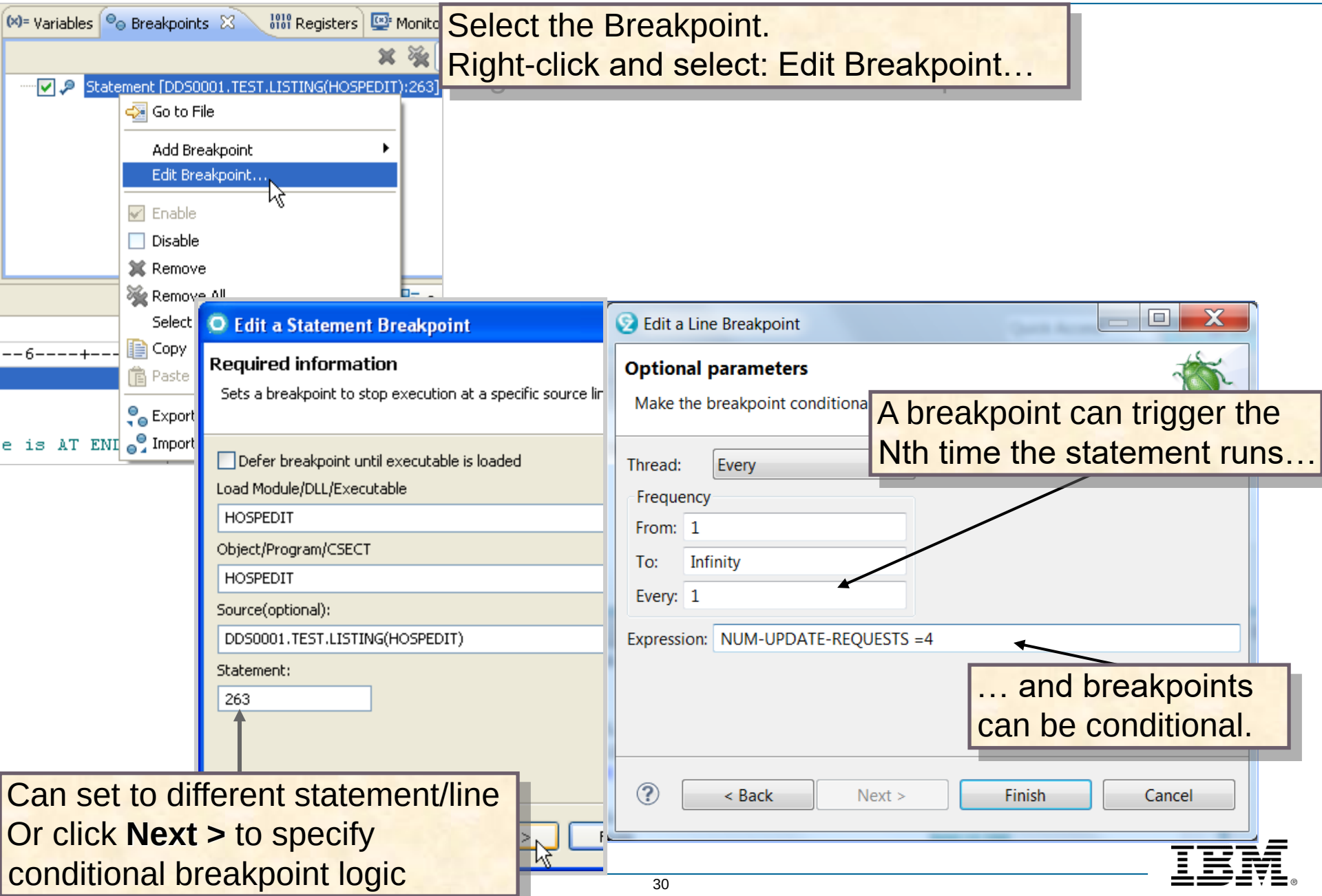
Set a statement breakpoint by double-clicking in the gray area next to a statement

Run to a Statement Breakpoint

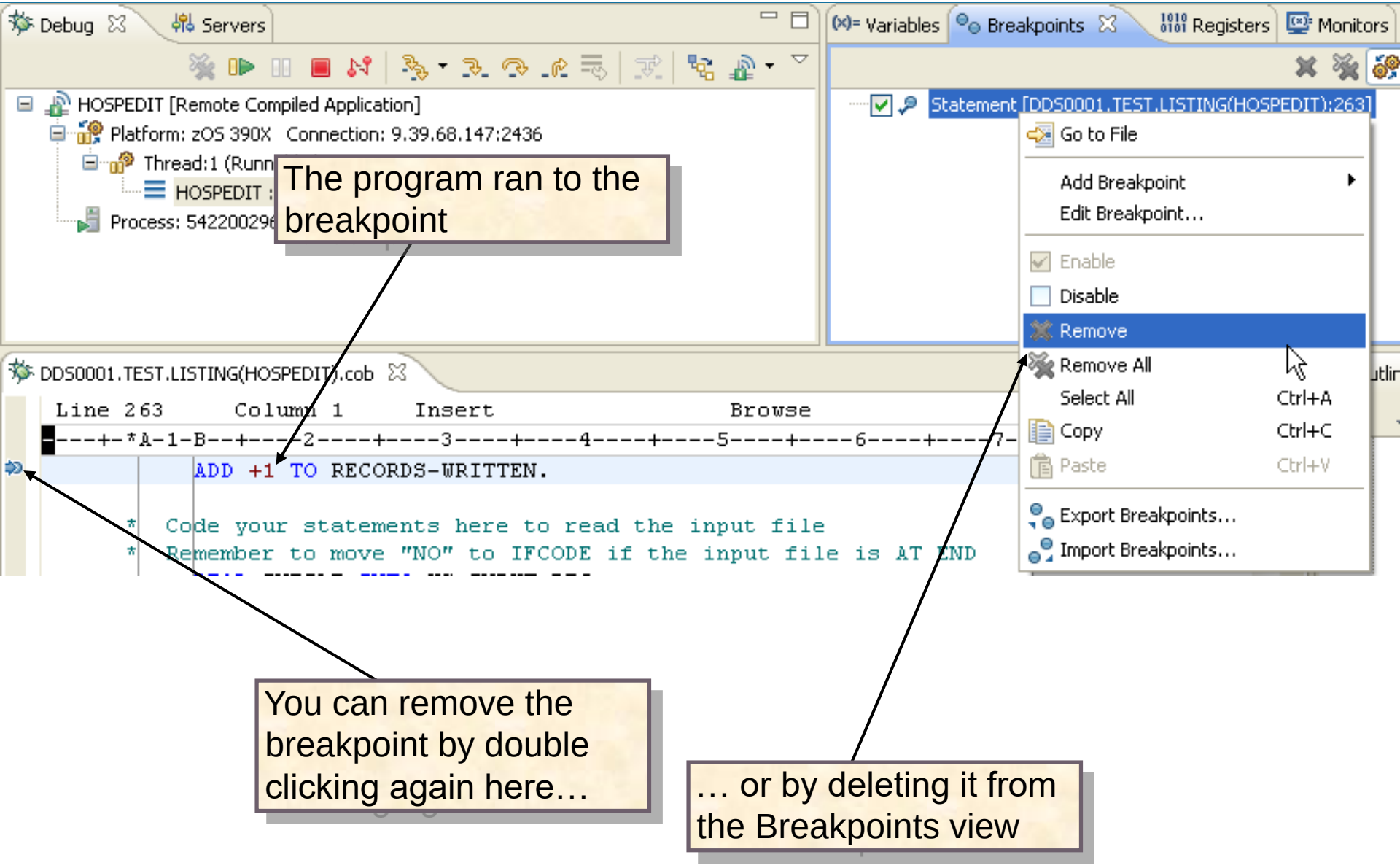


See Slide Notes

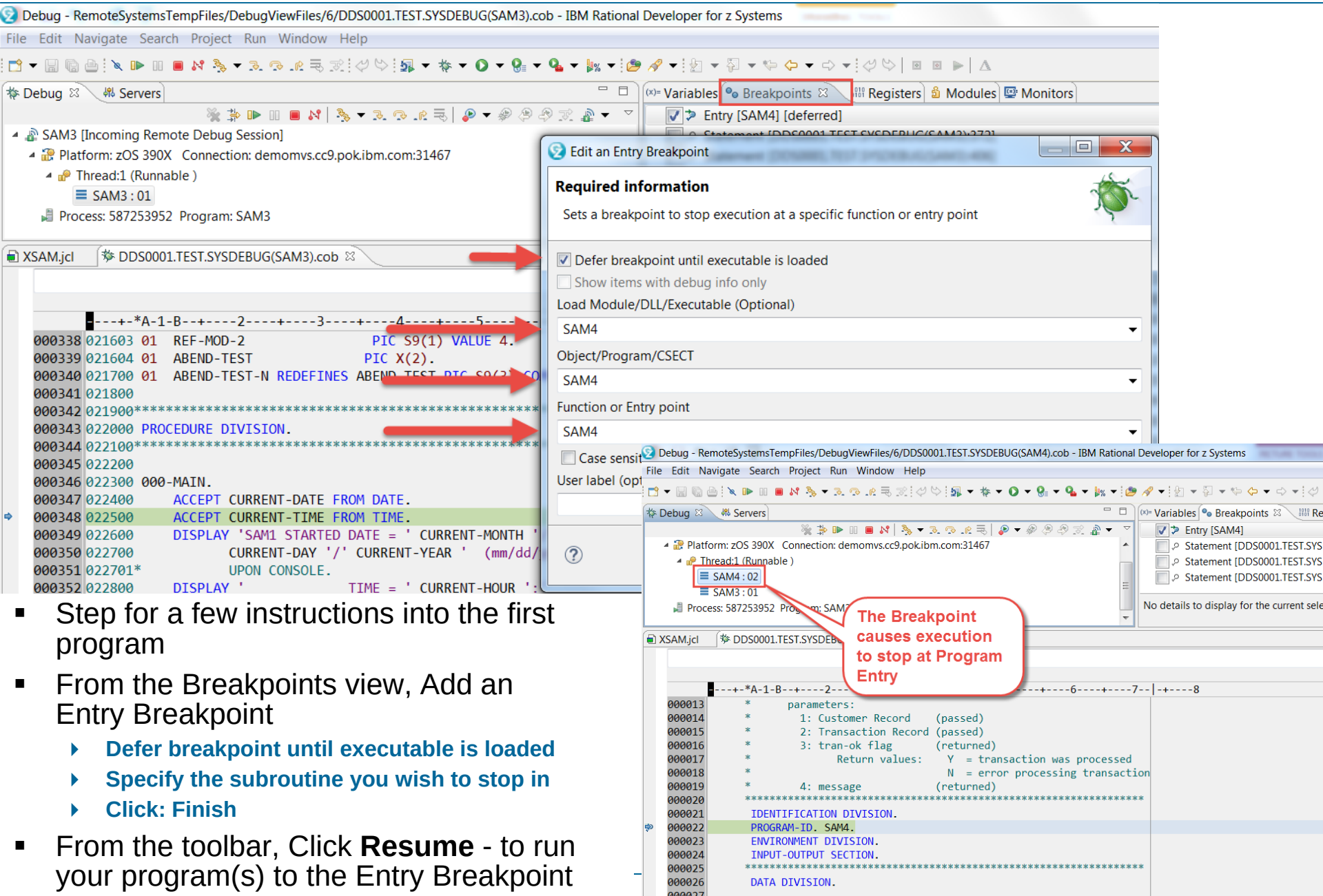
Edit/Create Conditional Statement Breakpoints



Additional Breakpoint Options



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

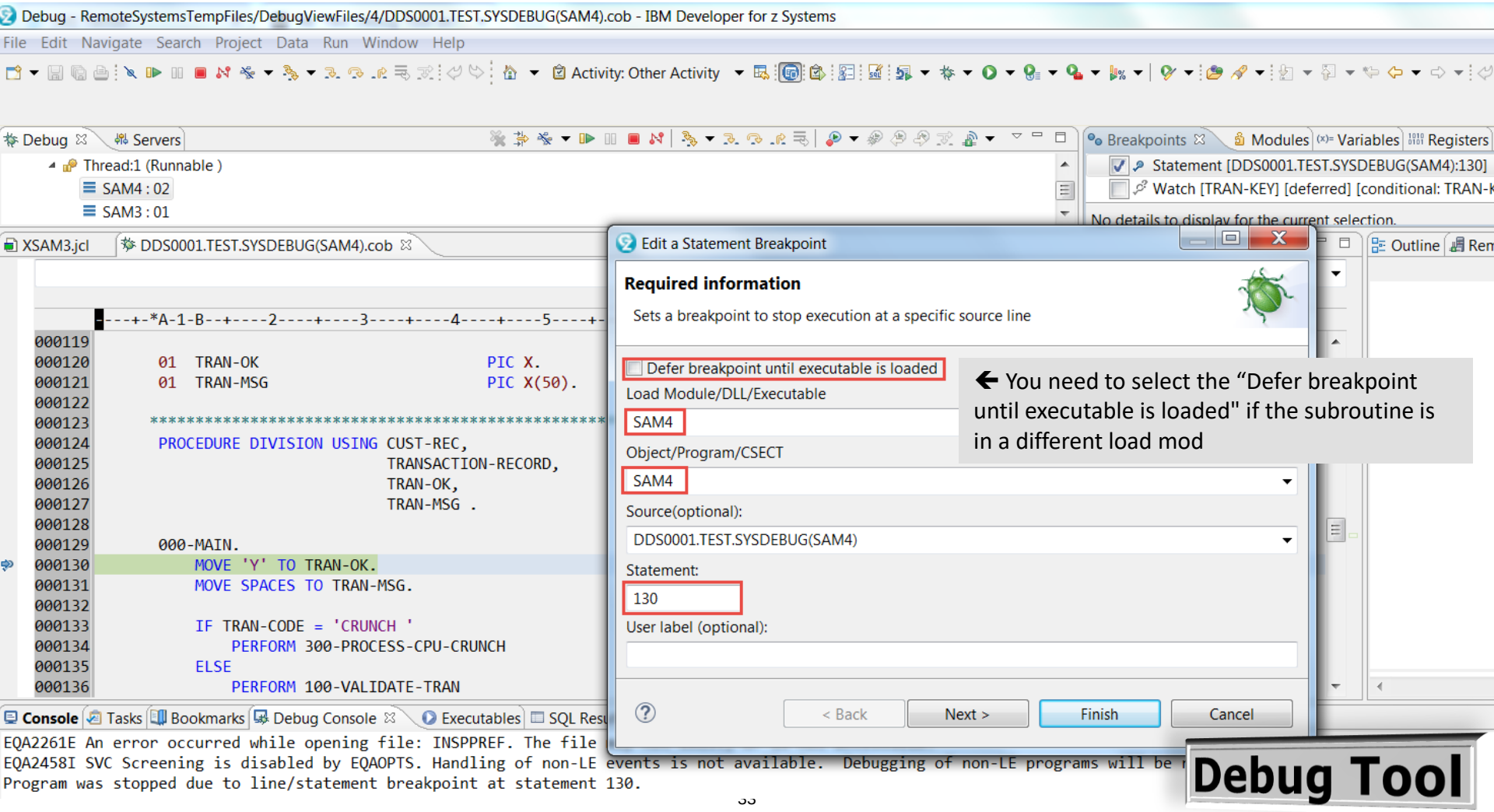


- Step for a few instructions into the first program
- From the Breakpoints view, Add an Entry Breakpoint
  - **Defer breakpoint until executable is loaded**
  - **Specify the subroutine you wish to stop in**
  - **Click: Finish**
- From the toolbar, Click **Resume** - to run your program(s) to the Entry Breakpoint

Stopping on a line in a Subroutine

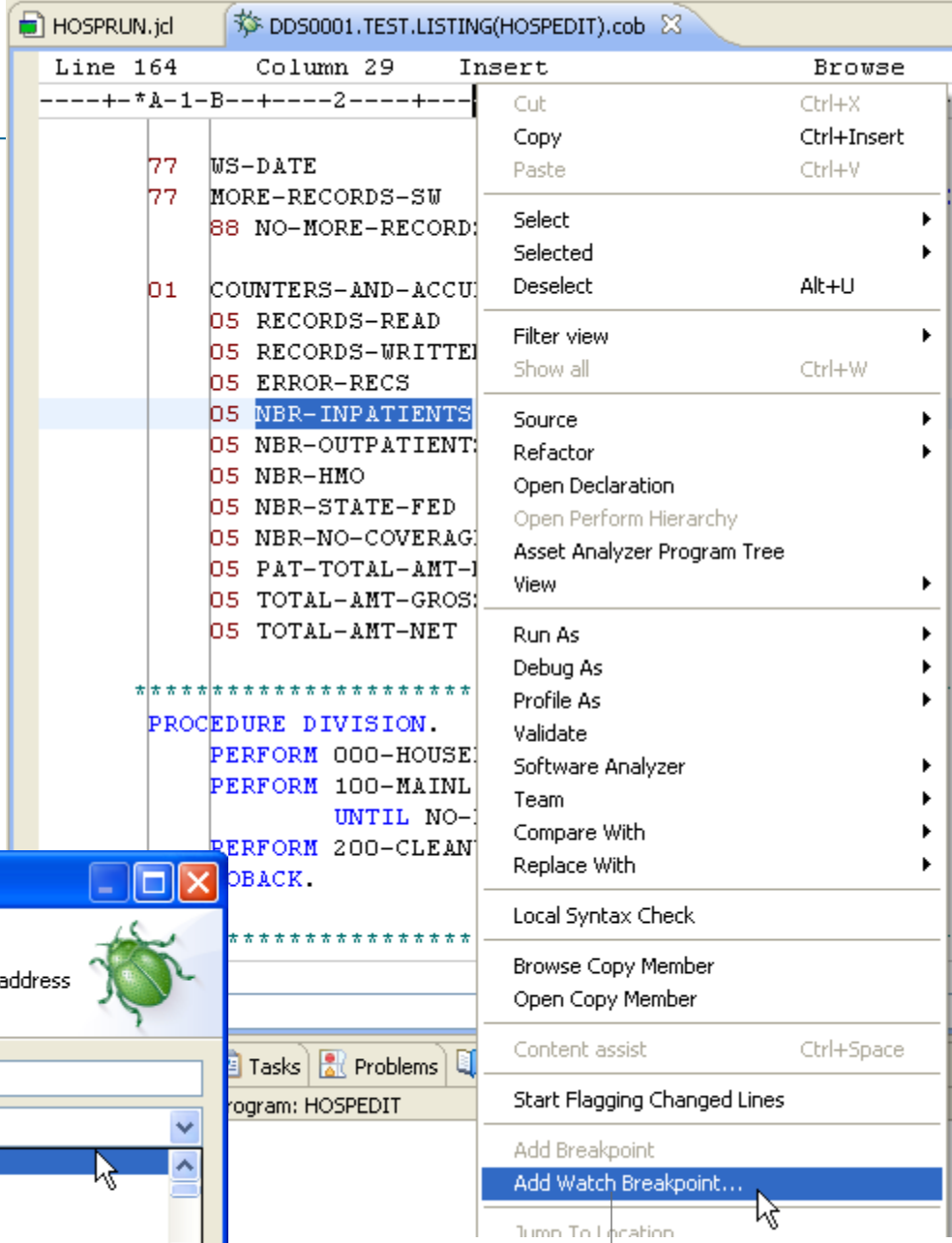
Steps:

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



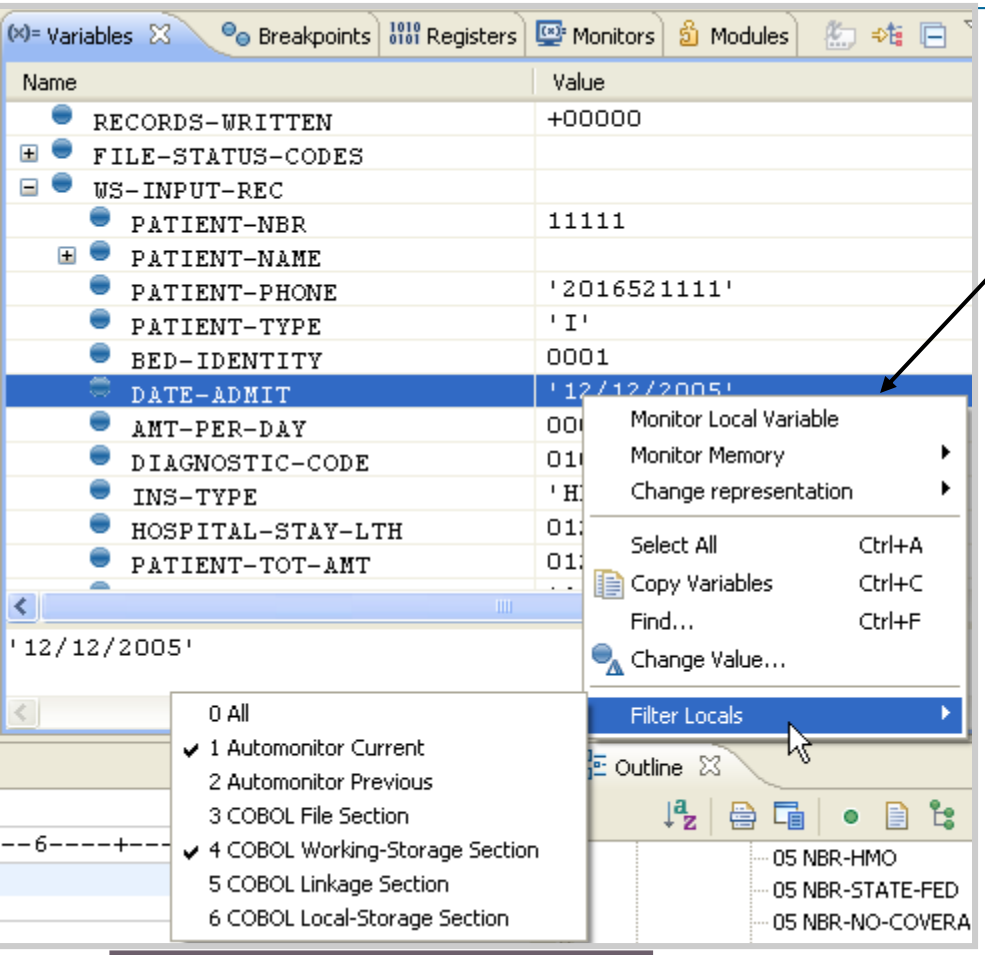
Watch Monitor Breakpoints

- 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



Debug Tool

Variables View During Remote Debug – Additional Options

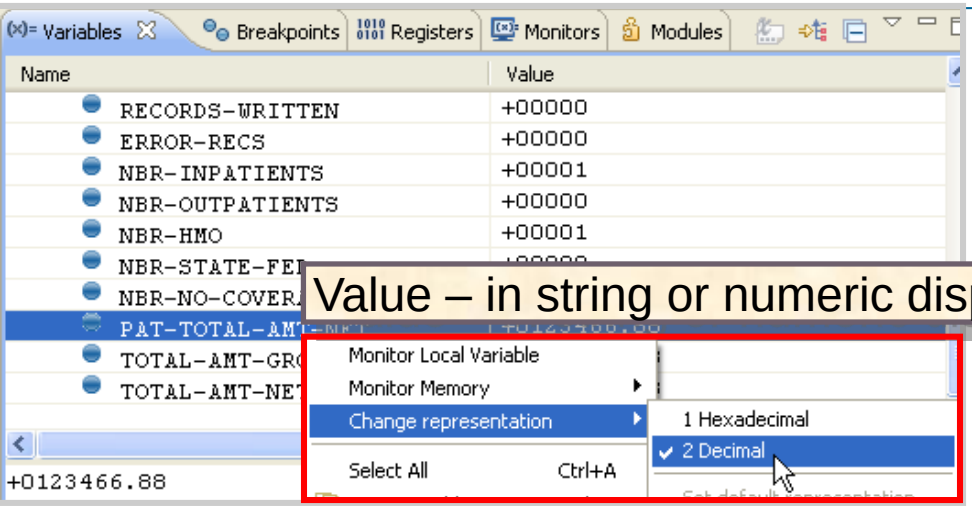


Many options for working with variables

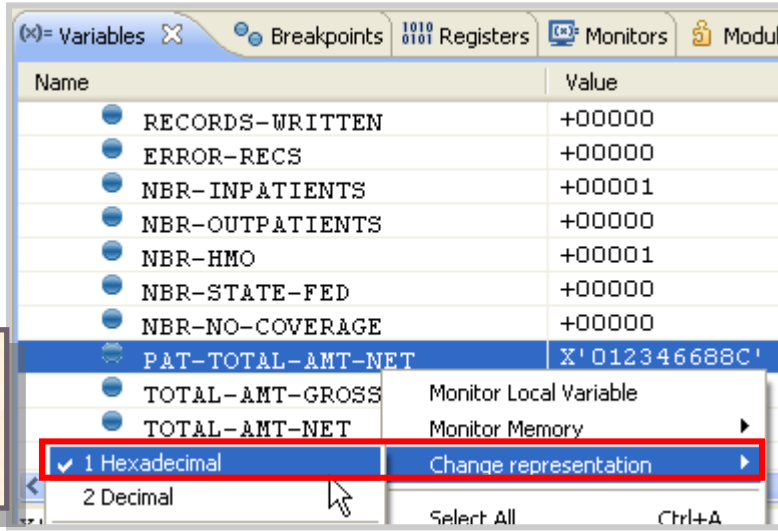
- **Add to Monitor** – for permanent viewing
- **Monitor internal memory values**
- **Show value in hex** – (EBCDIC internal) or string value representation
- **Copy variable** and value to Notepad
- **Find specific variable** in filtered Locals
- **Show specific variables** in the view

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

Variables View – EBCDIC (Hex) Data Representation



Value – in string or numeric display format



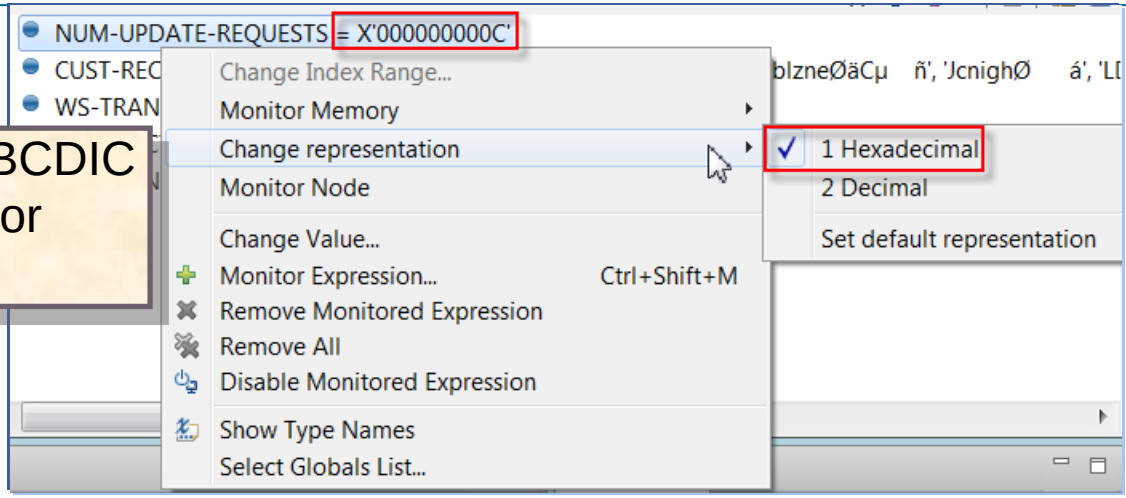
Value in **EBCDIC** internal display ➔ very useful for debugging data exceptions

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

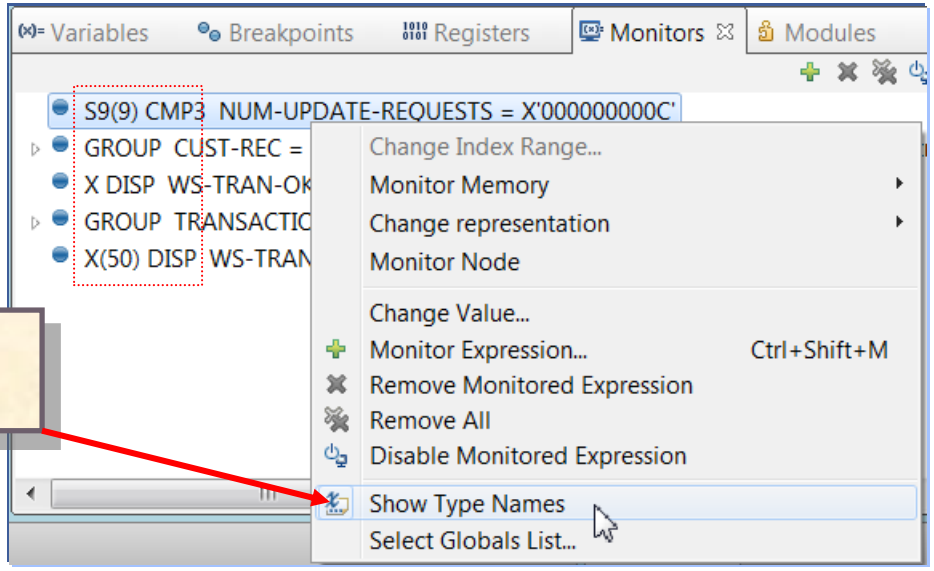
1. **Change representation to Hex.**
2. **Type invalid numeric EBCDIC values**
3. **Save**

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…

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+---
           DISPLAY "MAINLINE".
           DISPLAY WS-INPUT-REC.
           IF VALID-TYPE AND VALID-INS-TYPE
               MOVE WS-INPUT-REC TO ERR-REC FROM WS-INPUT-REC
           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
```

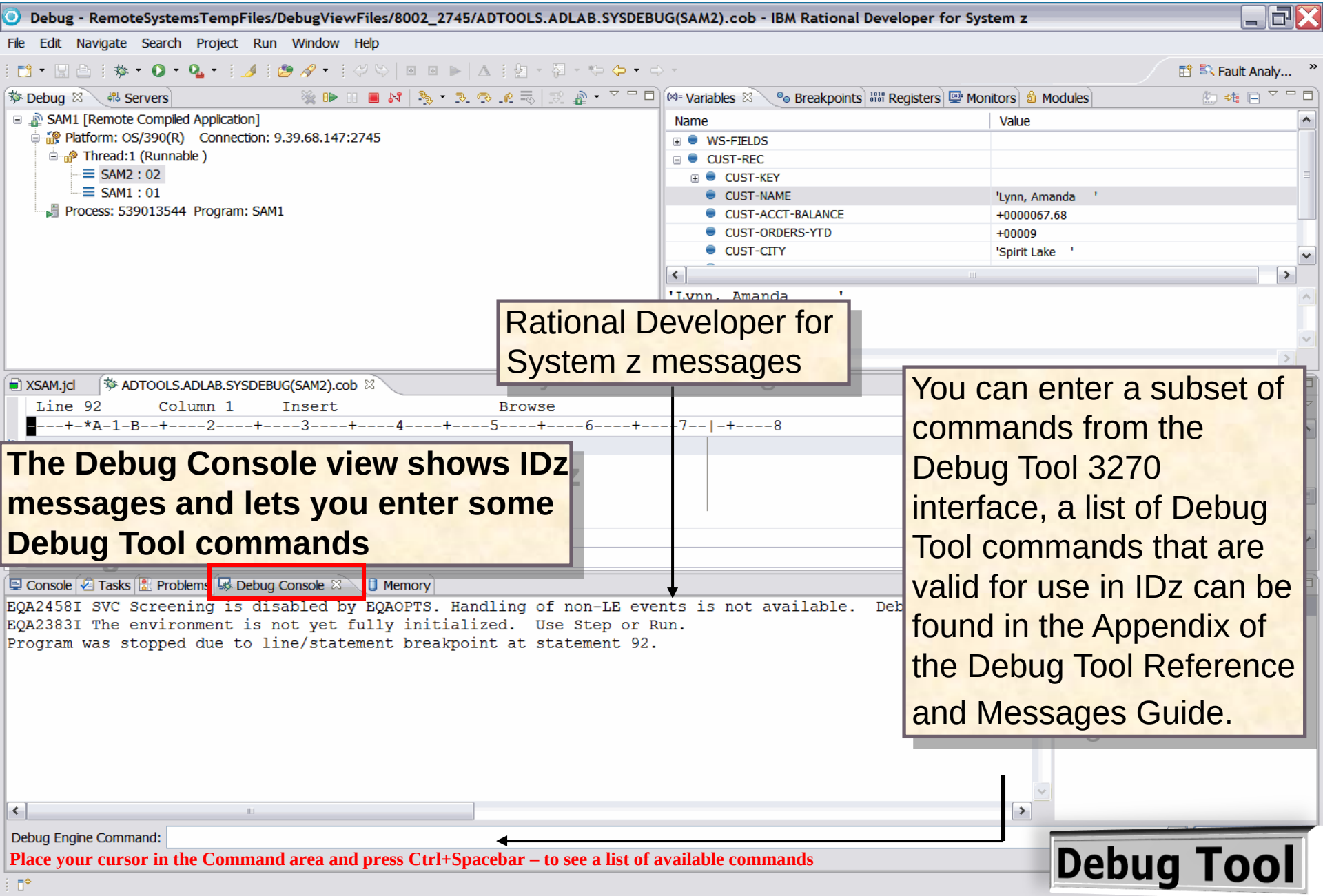
VALID-TYPE = TRUE

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-READ = +00009

The Debug Console



The Debug Console view shows IDz messages and lets you enter some Debug Tool commands

Rational Developer for System z messages

You can enter a subset of commands from the Debug Tool 3270 interface, a list of Debug Tool commands that are valid for use in IDz can be found in the Appendix of the Debug Tool Reference and Messages Guide.

Debug Tool

Debug Console Commands – Tracing Statement Execution

This is another very popular command:

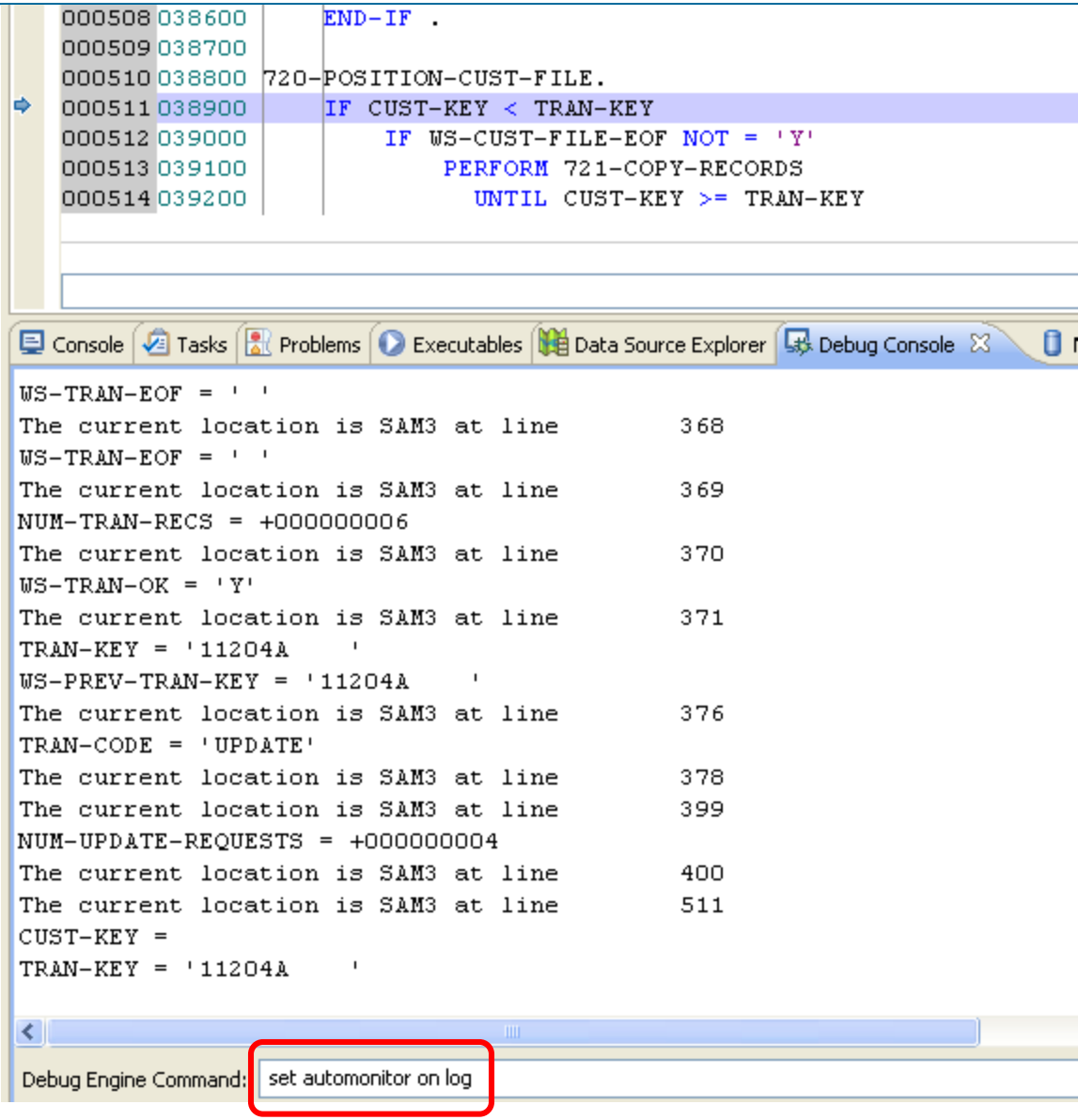
SET AUTOMONITOR ON LOG

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**



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**

```
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
```

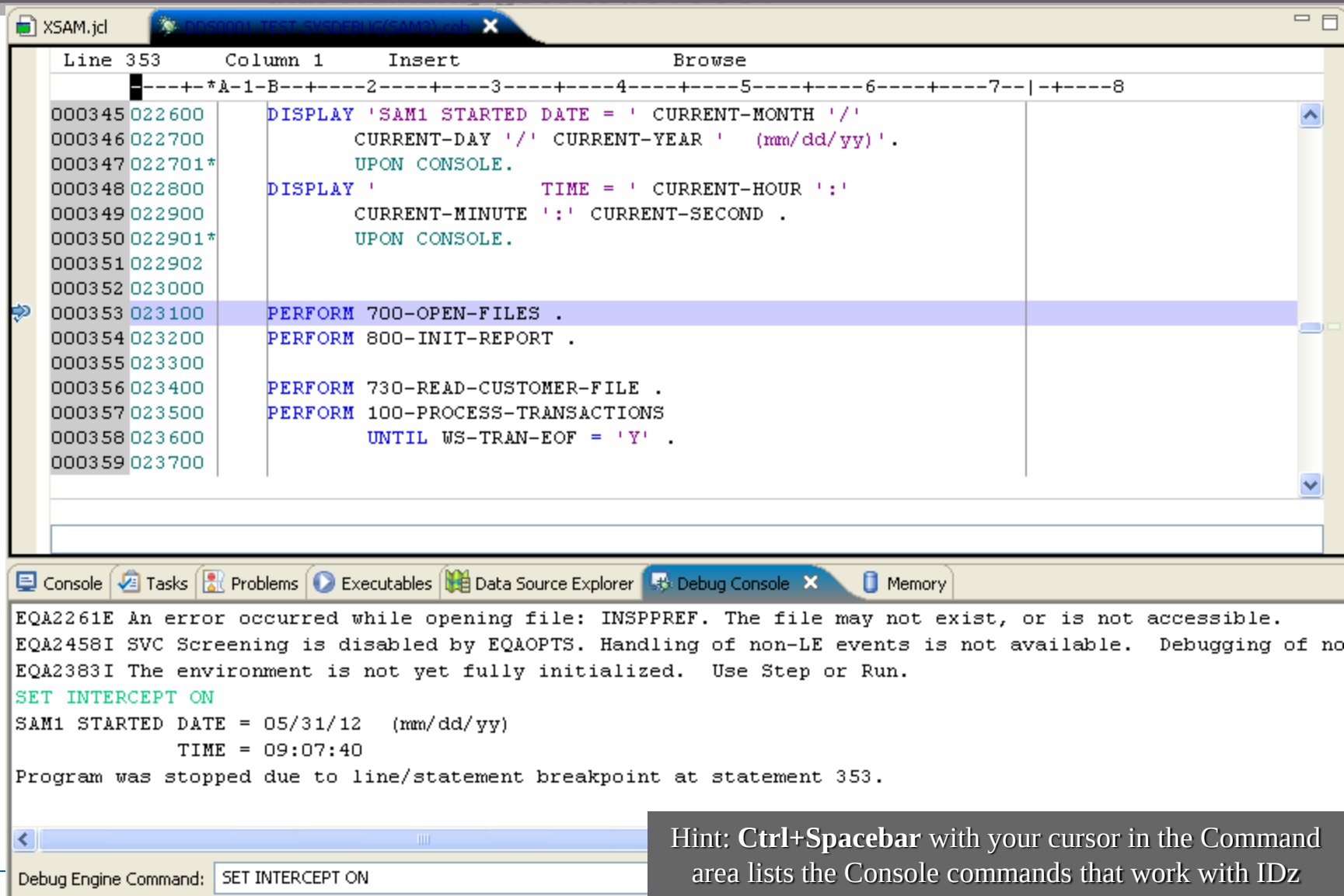
Console | Tasks | Problems | Executables | Data Source Explorer | Debug Console

```
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

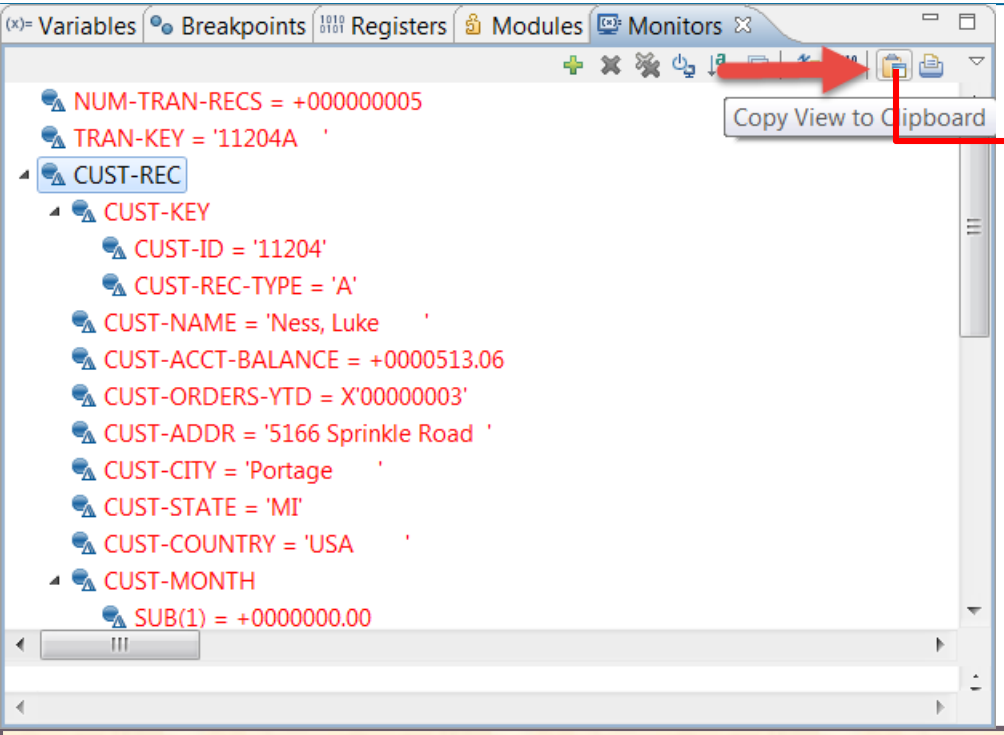
Debug Console View – Other Debug Console Options

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

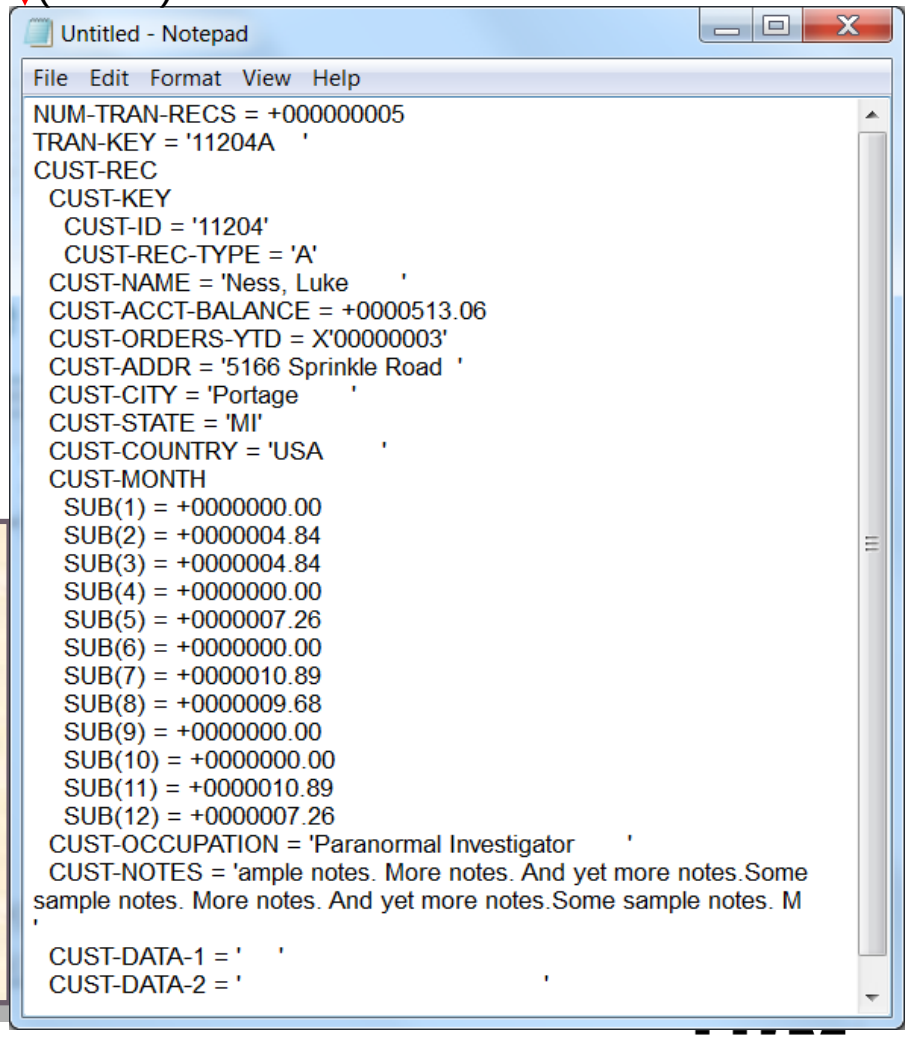


Hint: **Ctrl+Spacebar** with your cursor in the Command area lists the Console commands that work with IDz

Copy the contents of a monitor to the Windows-Clipboard



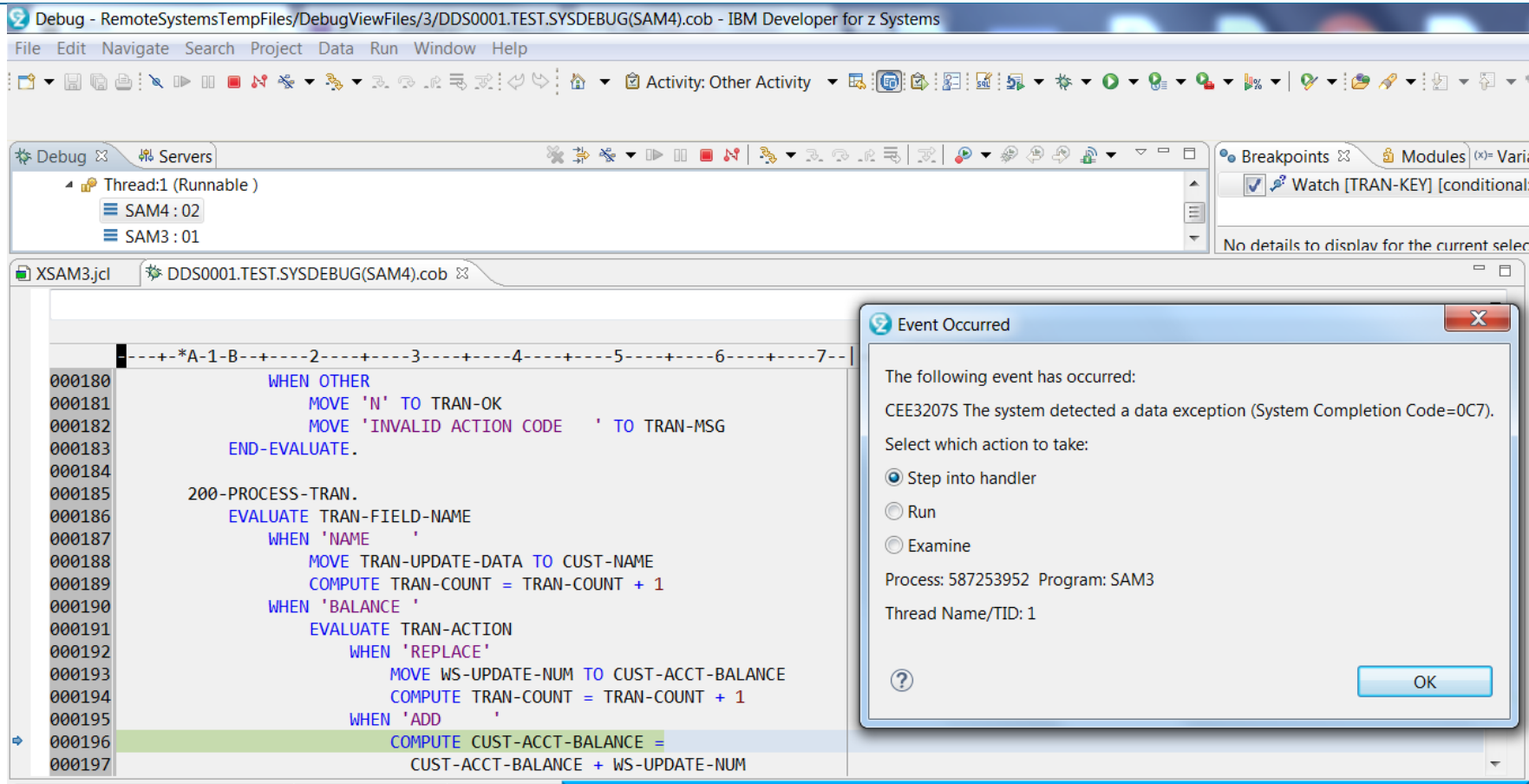
Ctrl+V
(Paste)



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.**

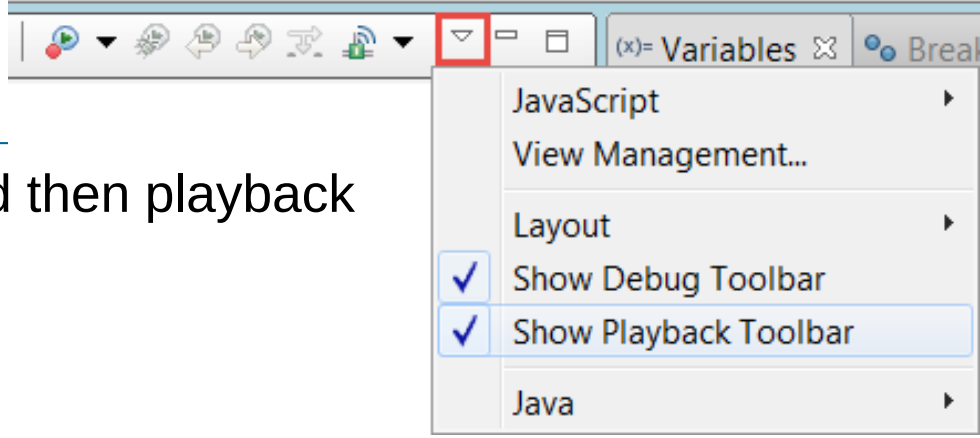
1. **Monitor the variable – and from the Monitors View**
2. **Expand the "nodes" (Group fields)**
3. **Click Copy view to Clipboard**
   - *This copies the contents of the View to your Windows paste buffer*
4. **Paste the copied contents ➔**

Using Automonitor to Capture Test Run “State”



- 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)

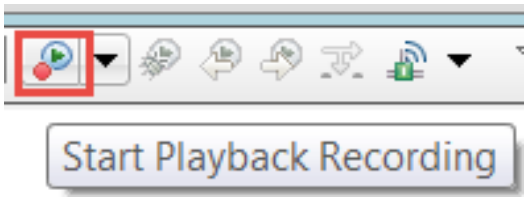
Record and Playback Test Run

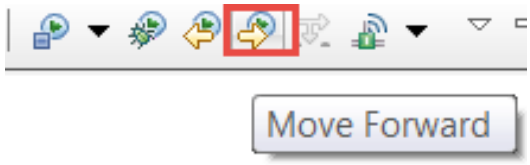


- Debug Tool allows you to record and then playback recorded statements during Debug

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**

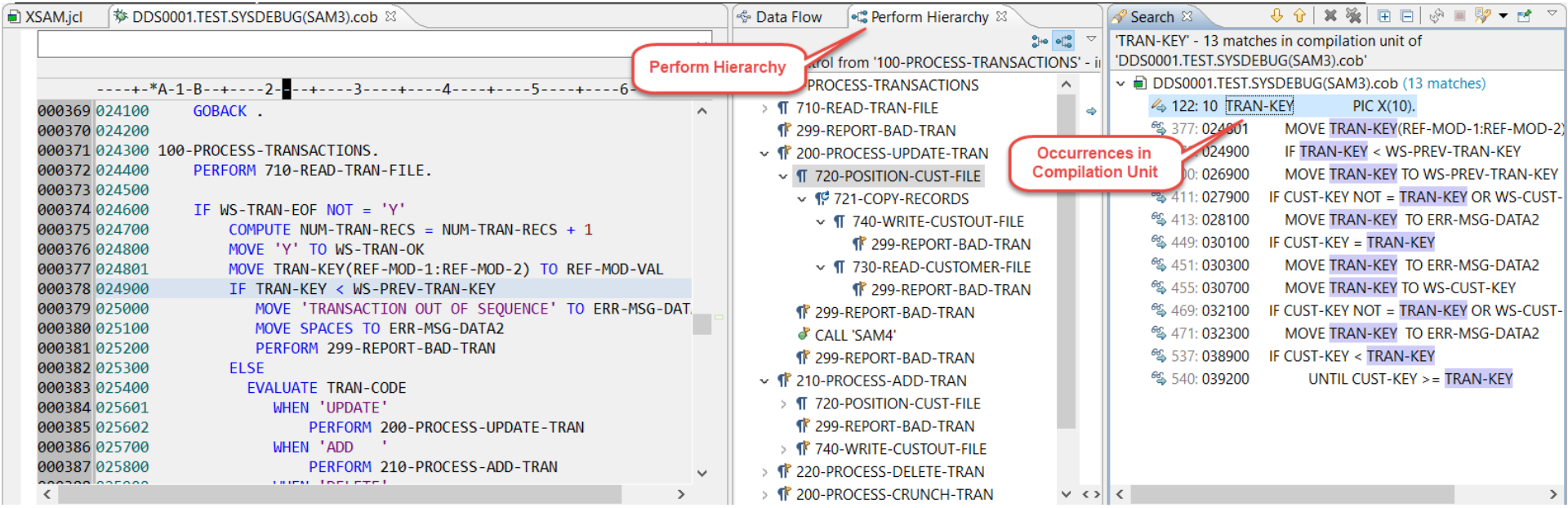
- Debug Tool can receive control back from the system after an ABEND occurs
  - The program will be stopped at the ABENDING statement
- You can:
  - Allow the application to ABEND and terminate
    - Capture ABEND info with a product such as Fault Analyzer
    - Terminate the application and prevent 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:

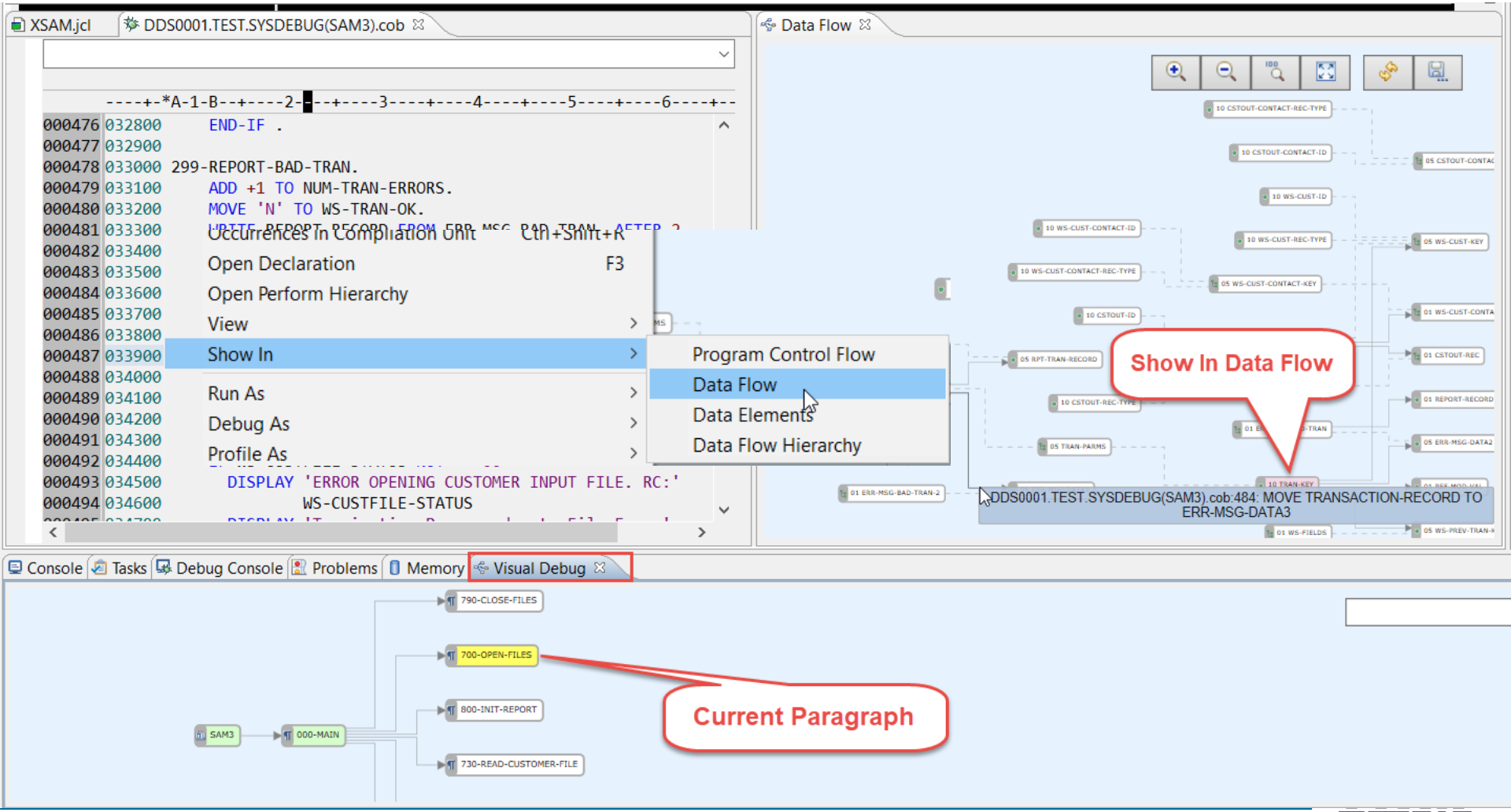
- Perform Hierarchy, Program Control Flow, Data Element analysis, Filtering, Code Review, Show unreachable code, Occurrences in Compilation Unit, etc.

- 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



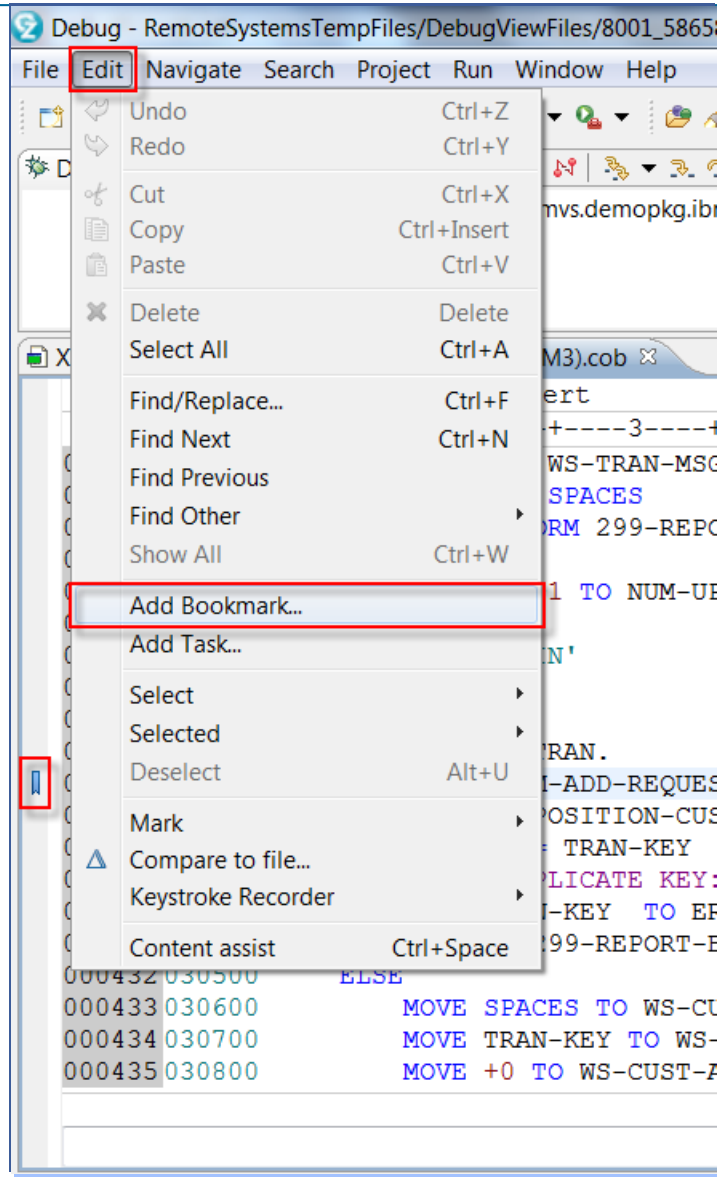
Integrating Context Menu options – Data Flow

- 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



Integrating Editor Features – Bookmarks

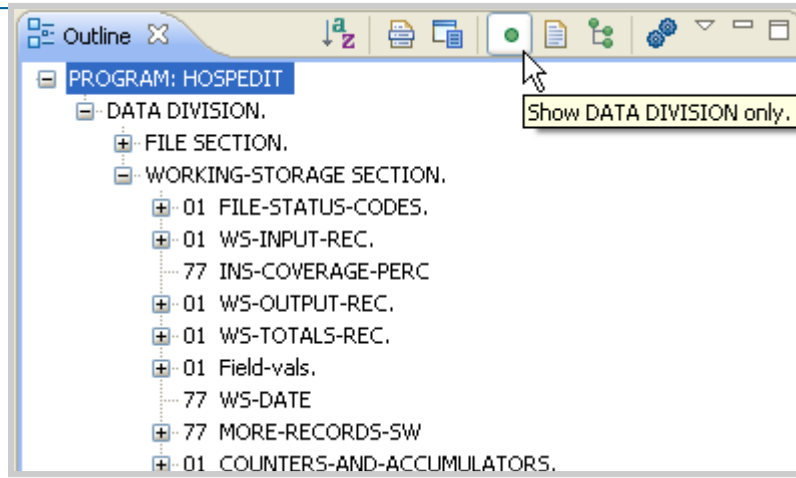
- It can be useful to set Bookmarks **(and/or Tasks)** into your code as:
  - Reminders of something to do
  - Navigation labels of where certain processing is
- To do this:
  - Navigate to the line you're interested in
    - Note – you can just scroll, you don't have to execute code
  - From the Edit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



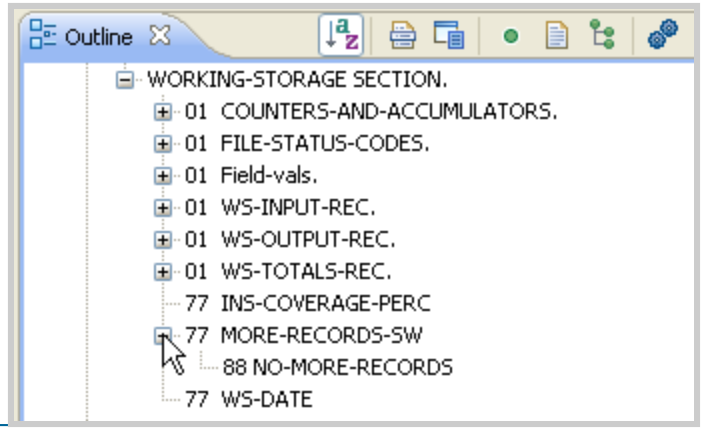
Outline View Options

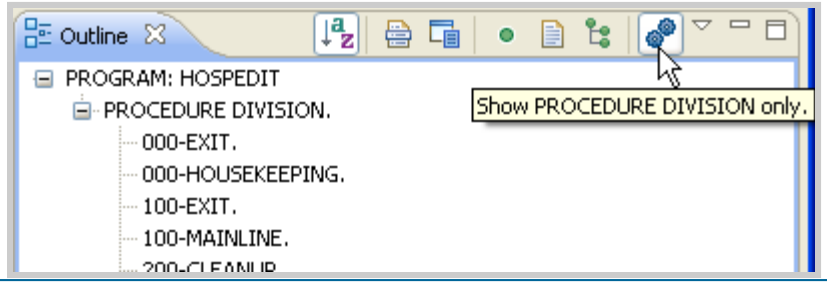
For large programs, several additional Outline view features are available as toggled icons:

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



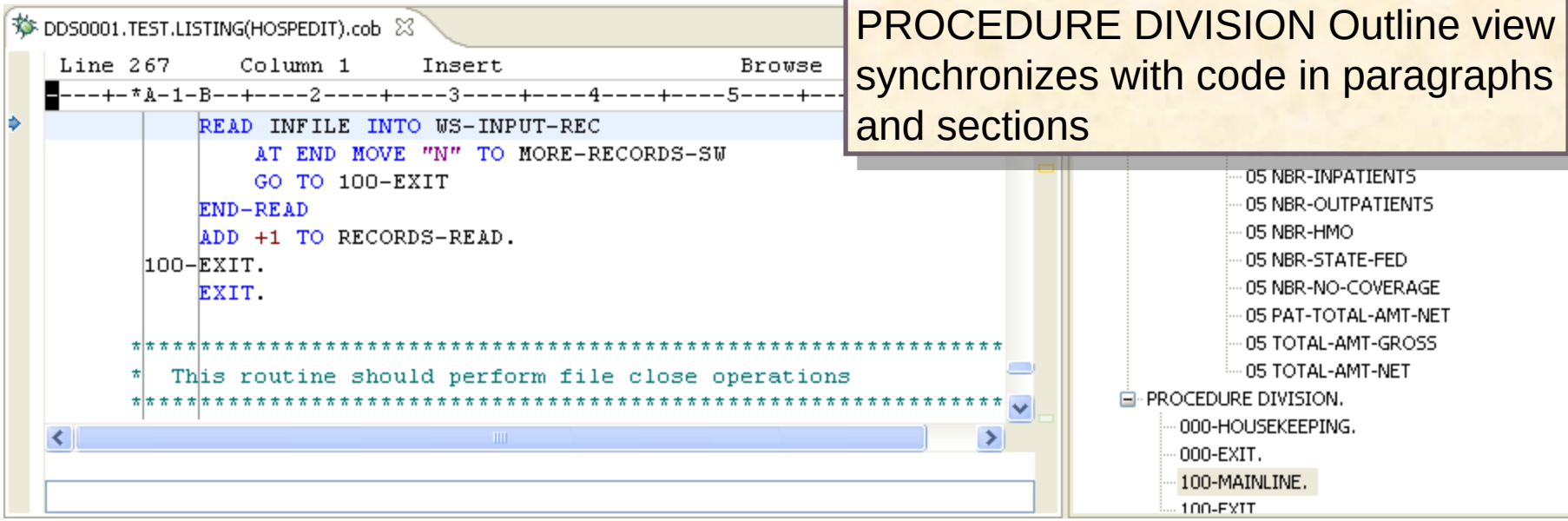
Sort all variables



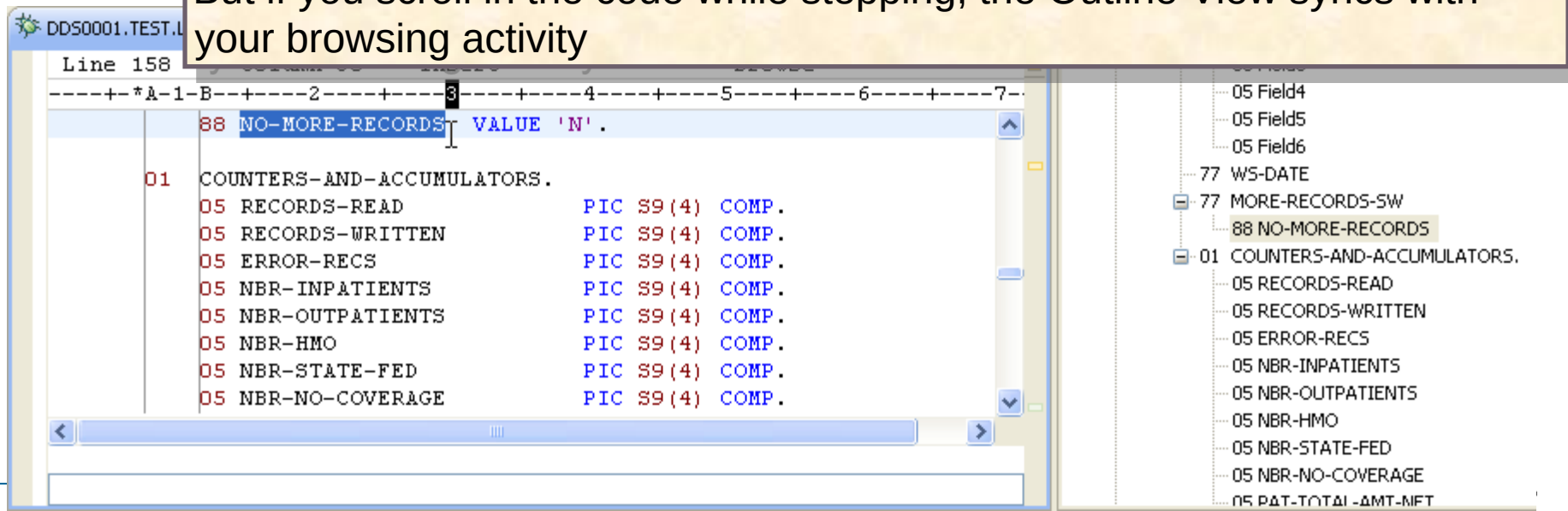


Outline View – Code Sync Option

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



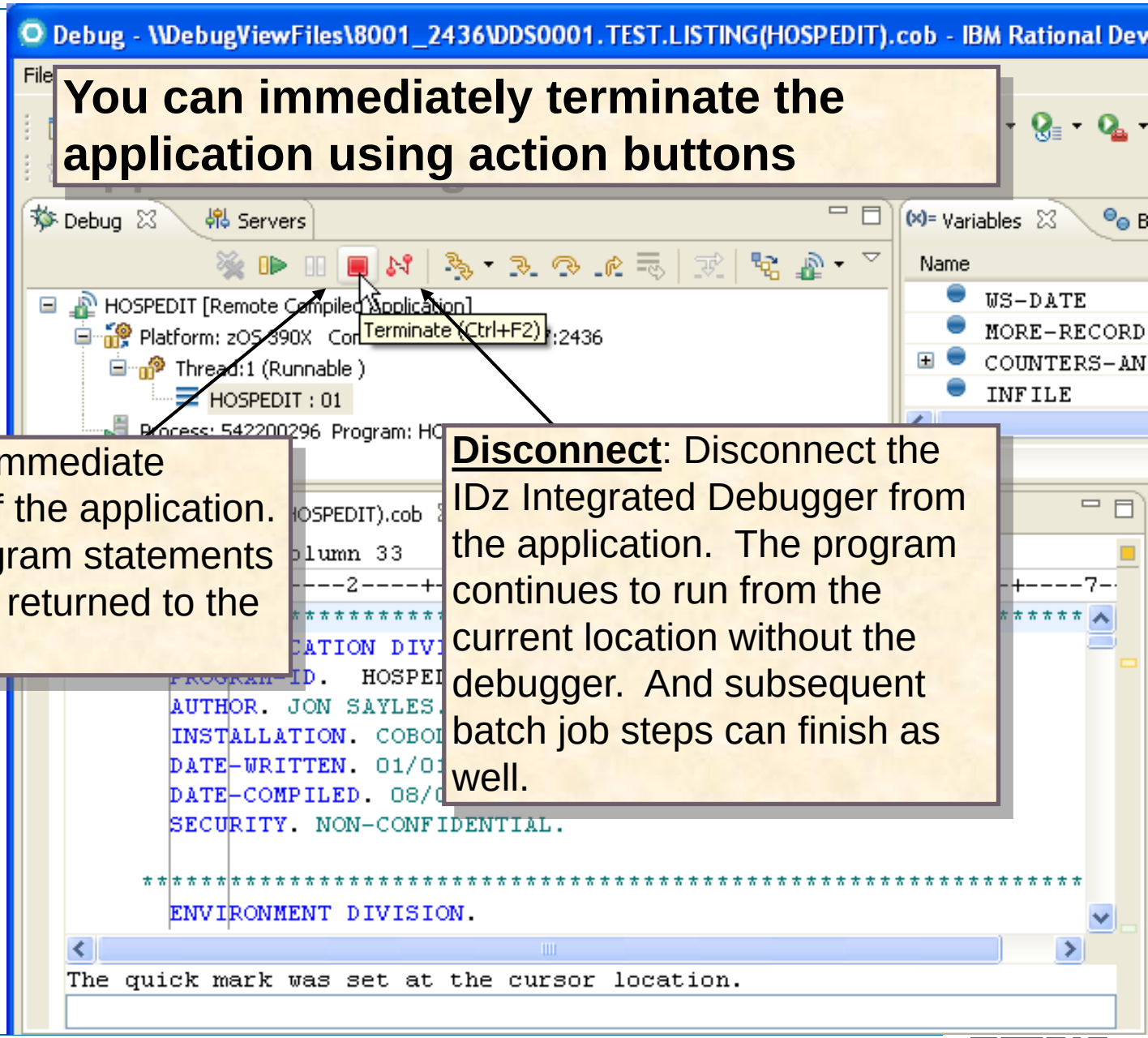
If your program ABENDS during Debug

- 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

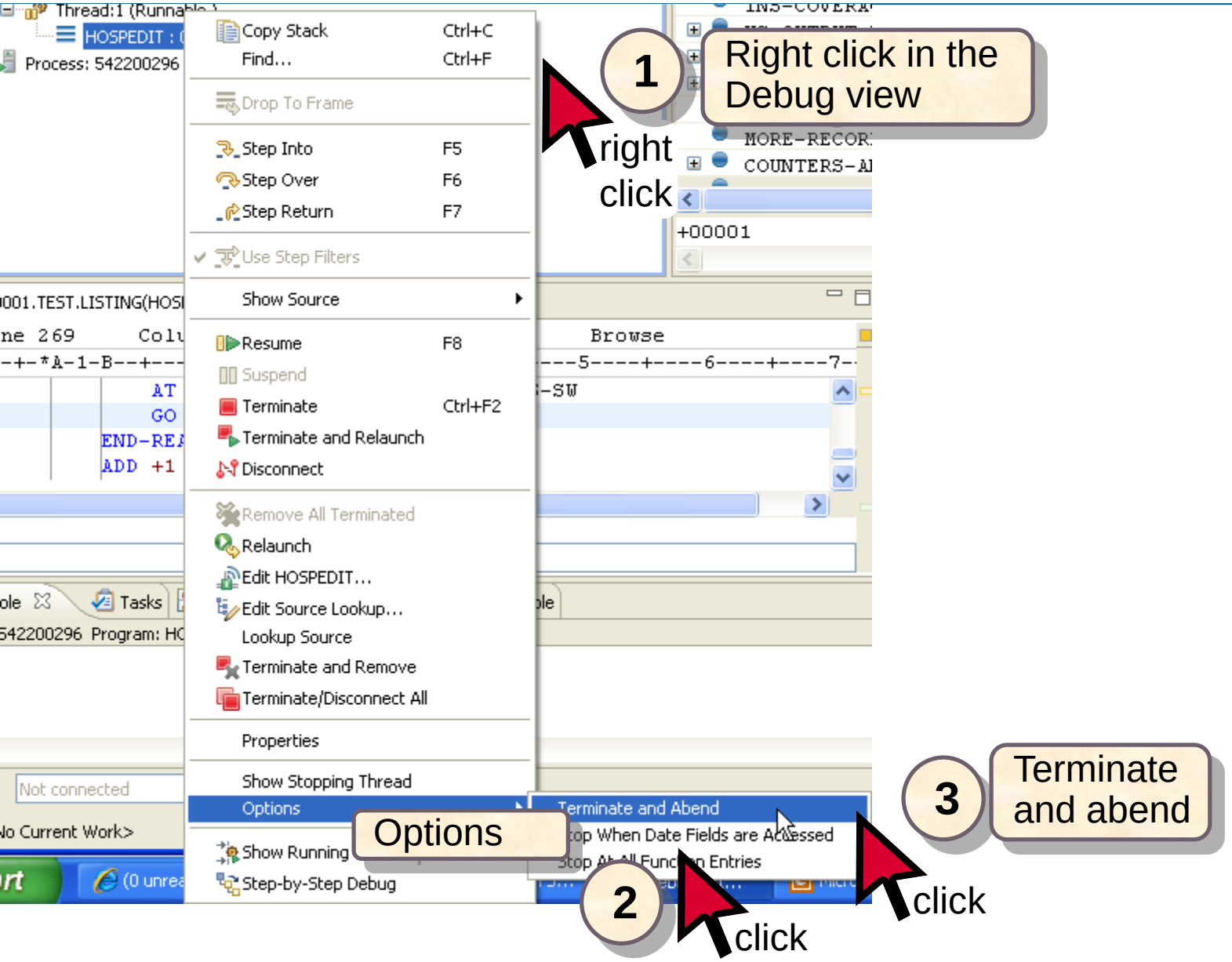
Termination action buttons



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.

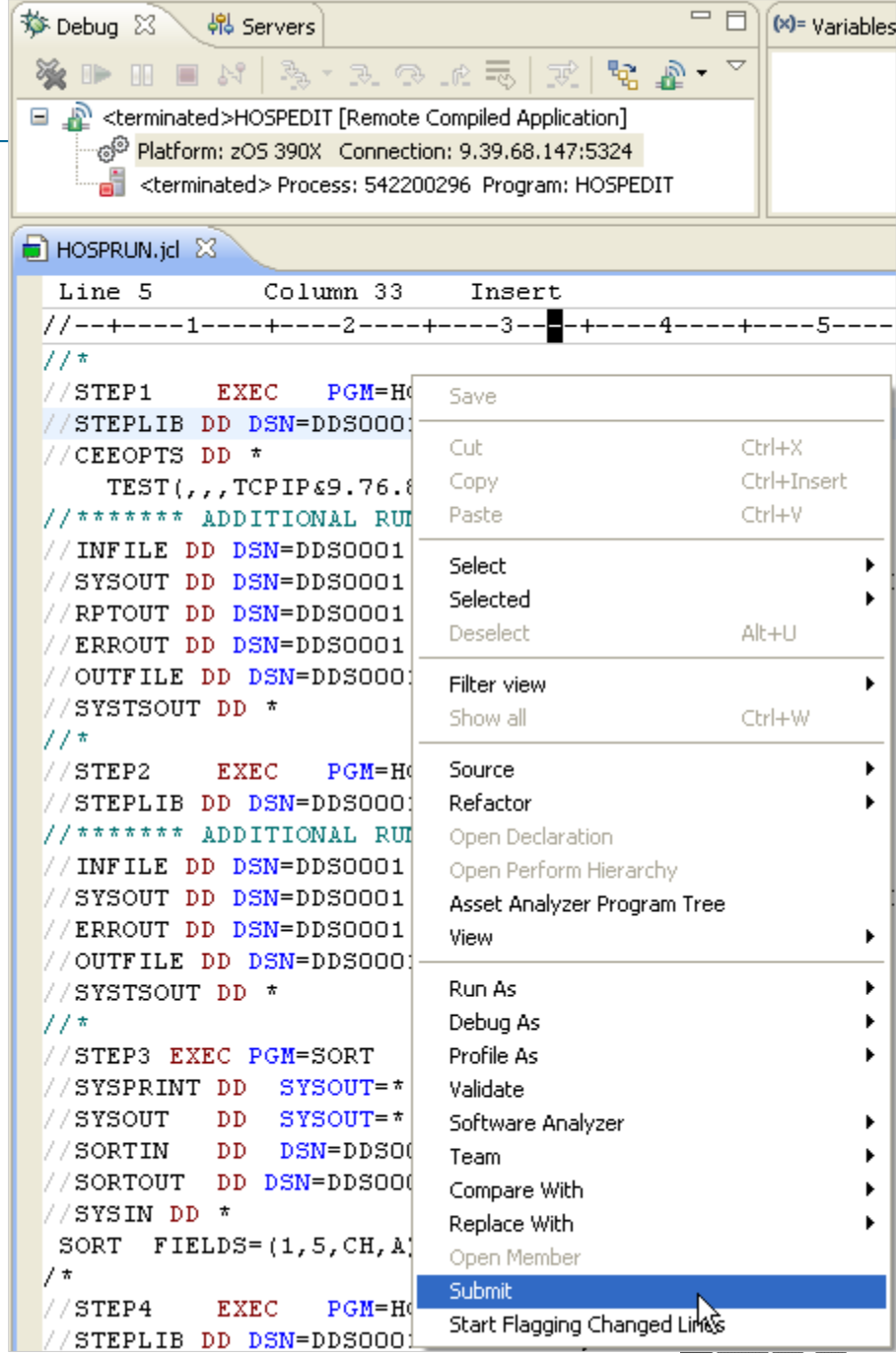
Force an immediate termination with abend



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