SHARE in Orlando 2015

Lab 17314

IBM PD Tools Hands-On Lab: Dive into Increased Programmer Productivity

IBM Debug Tool for z/OS

Eclipse interface

Hands-on Lab Exercises

IBM Debug Tool for z/OS v13

Lab Exercises

Contents

Overview

Debug Tool

Debug Tool is a single tool you use to debug application programs across z/OS environments, including batch, CICS, IMS, TSO, DB2 compiled stored procedures, Unix System Services, about anywhere an application program can run on a z/OS system.

It is an interactive debugger, and has the features you need to step through a program, set breakpoints, and monitor and change variables. It also has powerful debugging features, including automation with scripts, playback, statement tracing and more.

Debug Tool can be accessed using a 3270 terminal interface. There is also an eclipse GUI interface that is available in the PD Tools Studio, and will also run in RDz, z/OS Explorer and CICS Explorer.

This workbook

This workbook contains instructions for lab exercises that are designed to give you hands-on experience for the eclipse interface of IBM Debug Tool for z/OS.

Reference

Product manuals and other information about IBM Debug Tool for z/OS, and the other IBM problem determination tools are available on the Web at URL:

http://www.ibm.com/software/awdtools/deployment

Lab Exercise 1

Getting started with the Debug Tool perspective

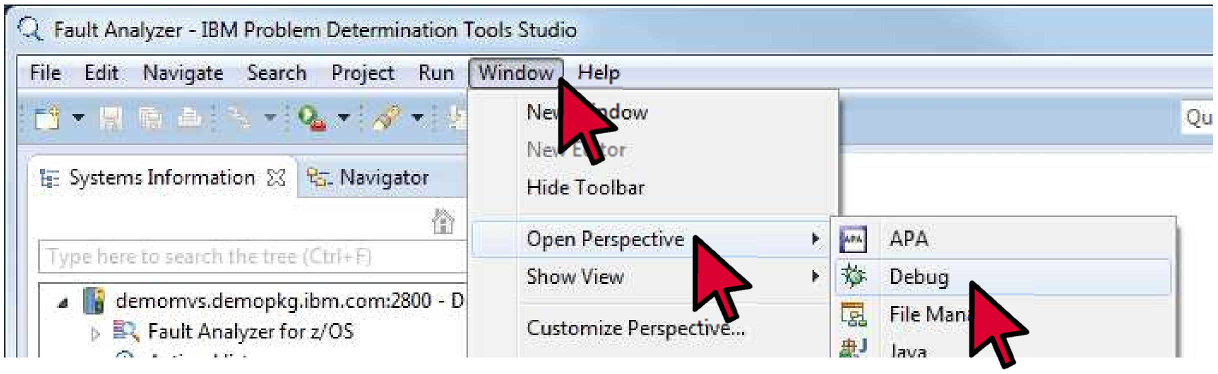
In this exercise you will:

- Open the Debug Tool perspective in the Eclipse workbench.
- Learn how to display help information for Debug Tool.

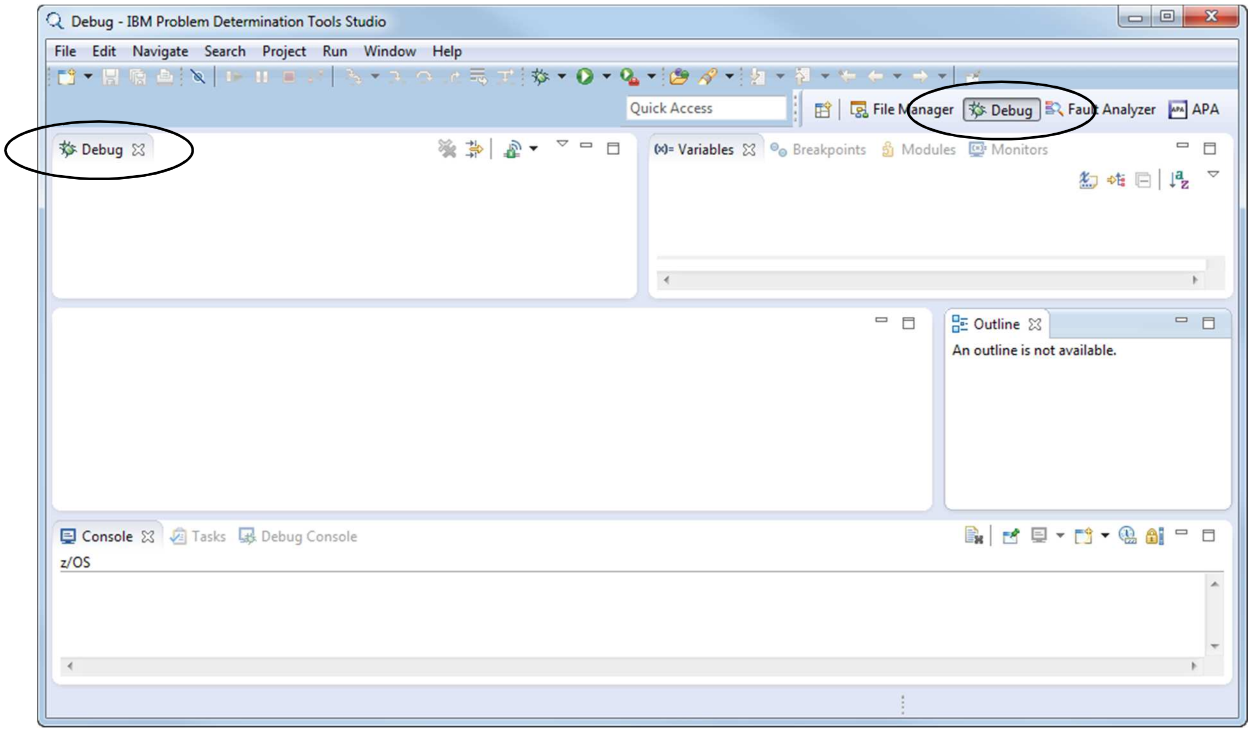
1. Before you begin, you must have the Eclipse interface open on your workstation. If you aren't sure how to open it, please contact your instructor.

In eclipse, a perspective is a set of views (windows), menus, and options that provide a set of functions. The Debug Tool interface is a perspective. Before you can use Debug Tool, you need to open its perspective.

2. Opening the Debug Tool perspective.
   a. From the menu bar near the top of the eclipse workbench, select **<u>Window</u>** > **<u>Open Perspective</u>** > **<u>Debug</u>**.
      - Notes:
        - Depending on your eclipse workbench, you need to select <u>Window</u> > <u>Open Perspective</u> > ***<u>Other</u>*** > <u>Debug</u>.
        - If "Debug Tool" isn't shown as a selection, then the Debug Tool perspective may already be open, and you can proceed to the next step.

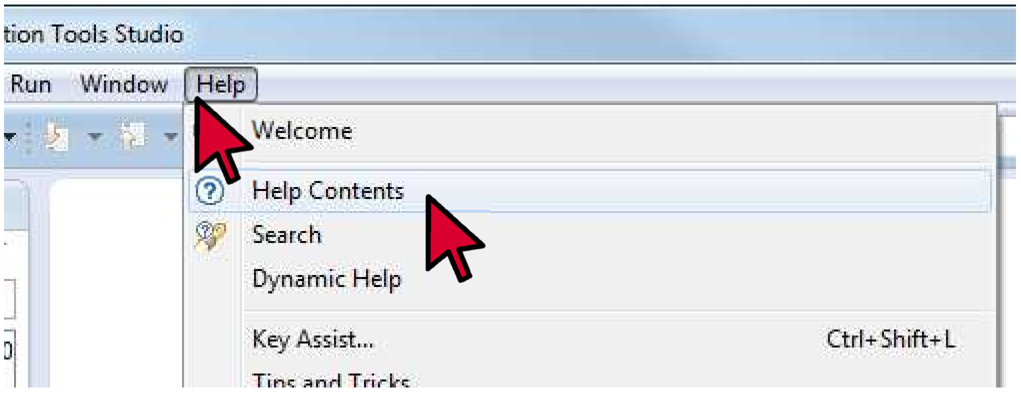


b. The Debug Tool perspective is displayed. Note that the default configuration of the views may be different on your system.


Debug - IBM Problem Determination Tools Studio
File Edit Navigate Search Project Run Window Help
Quick Access
File Manager
Debug
Fault Analyzer
APA
Debug
Variables
Breakpoints
Modules
Monitors
Outline
An outline is not available.
Console
Tasks
Debug Console
z/OS

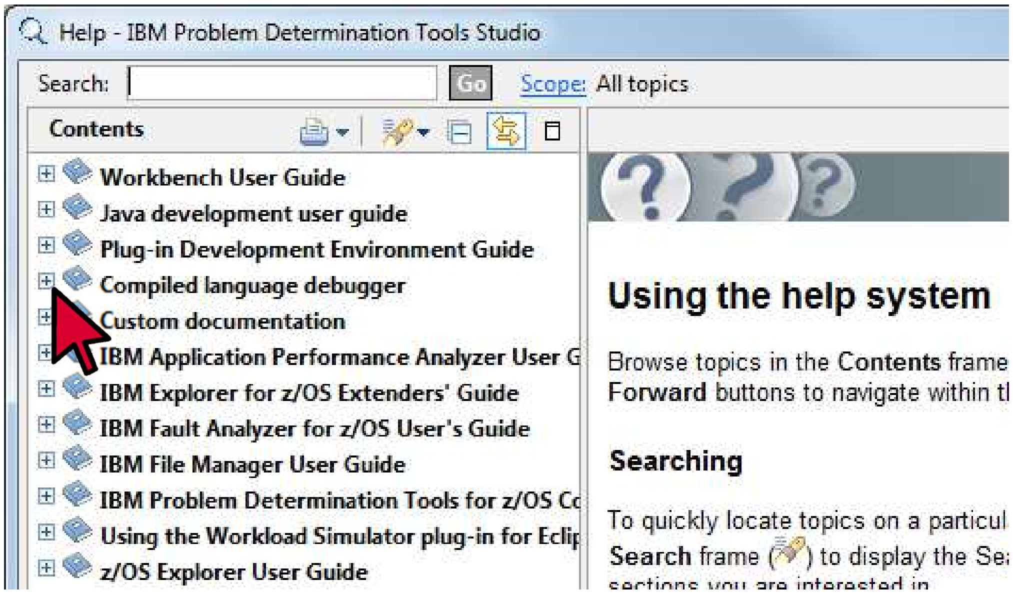
c. Notice that there are several views (windows) in the Debug Tool perspective. By default, the Debug view is displayed in the upper left.

d. You can display help information to assist with various features of Debug Tool. To open the help dialog, select **Help** > **Help Contents**.



e. The help dialog is displayed.

f. To expand the Debug Tool help topics, click the plus (+) next to ‘Compiled language debugger’.



Help - IBM Problem Determination Tools Studio

Search: Go Scope: All topics

Contents

Workbench User Guide
Java development user guide
Plug-in Development Environment Guide
Compiled language debugger
Custom documentation
IBM Application Performance Analyzer User G
IBM Explorer for z/OS Extenders' Guide
IBM Fault Analyzer for z/OS User's Guide
IBM File Manager User Guide
IBM Problem Determination Tools for z/OS Cc
Using the Workload Simulator plug-in for Eclip
z/OS Explorer User Guide

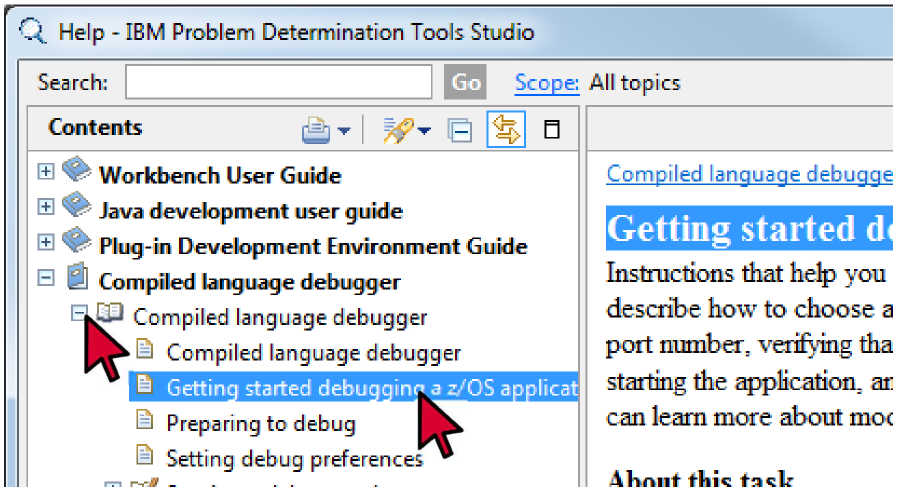
Using the help system

Browse topics in the Contents frame
Forward buttons to navigate within tl

Searching

To quickly locate topics on a particul
Search frame ( ) to display the Se:
sections you are interested in

g. Under **Compiled Language Debugger**, select **Compiled Language Debugger** > **Getting started debugging a z/OS application with the debugger**.



Help - IBM Problem Determination Tools Studio

Search: Go Scope: All topics

Contents

Workbench User Guide
Java development user guide
Plug-in Development Environment Guide
Compiled language debugger
Compiled language debugger
Compiled language debugger
Getting started debugging a z/OS applicat
Preparing to debug
Setting debug preferences

Compiled language debugge

Getting started d

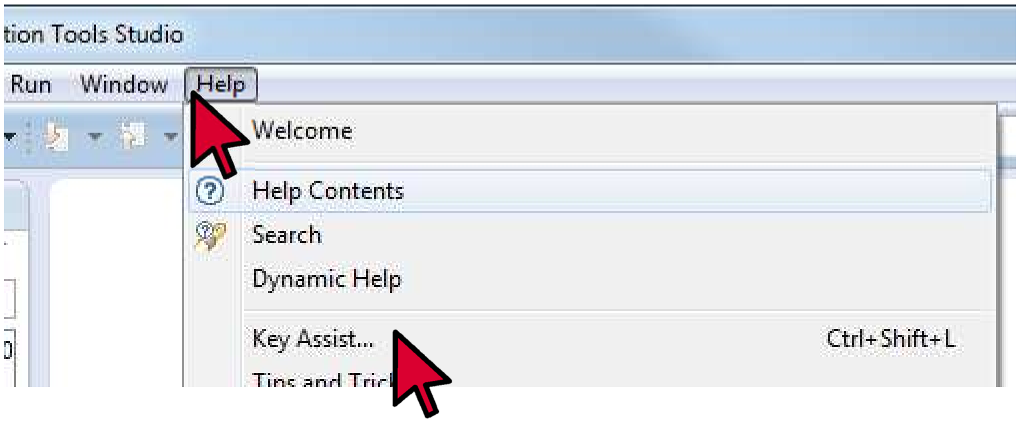
Instructions that help you
describe how to choose a
port number, verifying tha
starting the application, ar
can learn more about moc

About this task

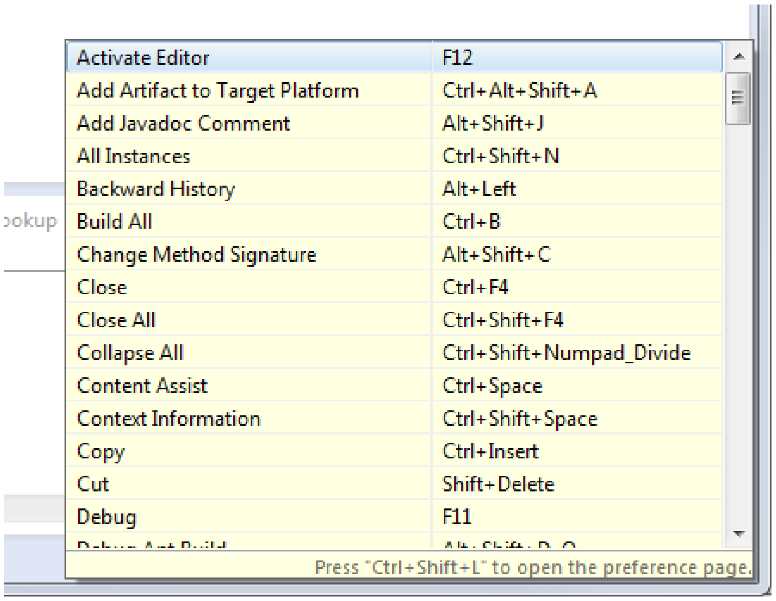
h. The selected topic is displayed.

i. Now you have seen how to display general help information. Close the help dialog: click the **X** (close) icon in the upper right corner of the window.

j. You can also display a list of keyboard commands. In the eclipse workbench Select **Help** > **Key Assist**.



k. A list of keyboard commands is displayed.



Activate Editor	F12
Add Artifact to Target Platform	Ctrl+Alt+Shift+A
Add Javadoc Comment	Alt+Shift+J
All Instances	Ctrl+Shift+N
Backward History	Alt+Left
Build All	Ctrl+B
Change Method Signature	Alt+Shift+C
Close	Ctrl+F4
Close All	Ctrl+Shift+F4
Collapse All	Ctrl+Shift+Numpad_Divide
Content Assist	Ctrl+Space
Context Information	Ctrl+Shift+Space
Copy	Ctrl+Insert
Cut	Shift+Delete
Debug	F11

Press "Ctrl+Shift+L" to open the preference page.

l. Now that the Debug Tool perspective is open, and you know where to find help, you are ready to start using Debug Tool in the following exercises.

Lab Exercise 2

Submitting an example batch application and starting the debugger

In this exercise you will:

- Use tools in the Debug Tool perspective to determine the IP address of your workstation
- Edit JCL that will be used to submit a batch job to run an example application
- Code a 'TEST' option in the JCL that will start the debugger when the job runs
- Submit the job and start the debugger

Introduction to starting the debugger with a batch application

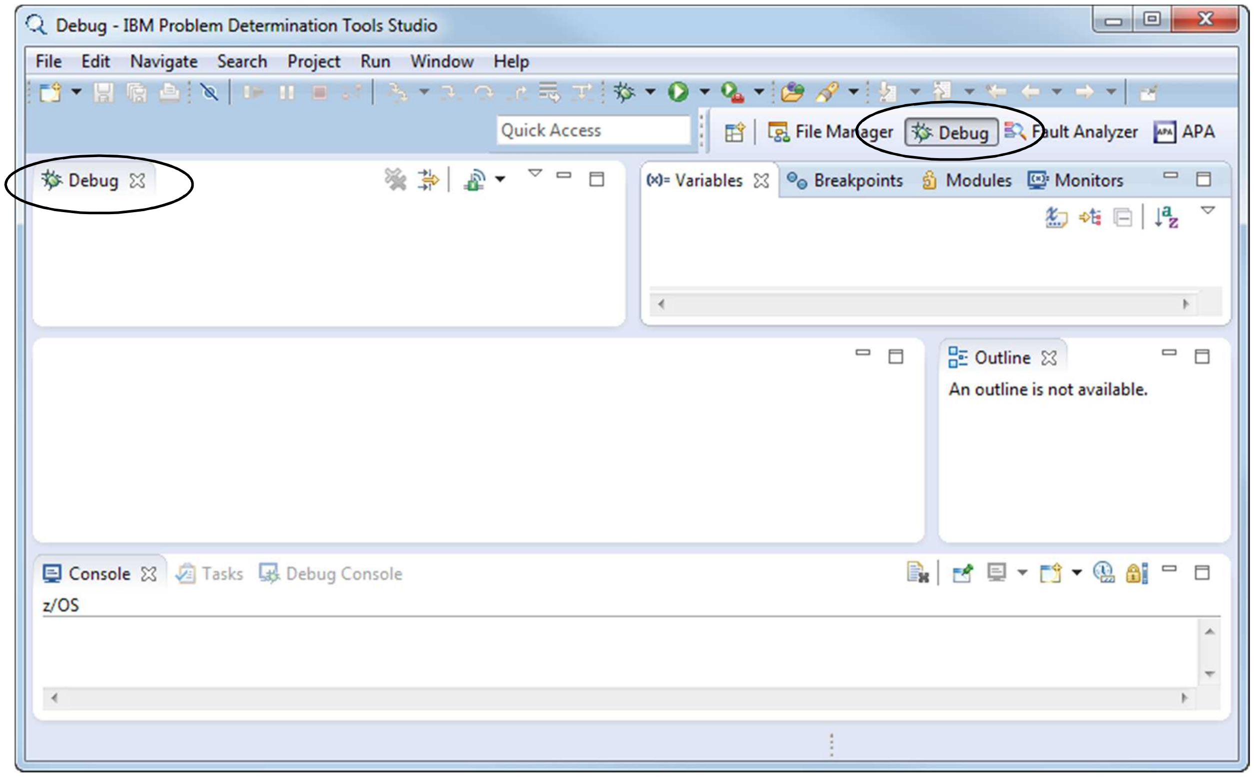
Debug Tool can be used to debug applications running in various environments on a z/OS system, including batch jobs, TSO, CICS transactions, IMS/TM transactions, DB2 external stored procedures, USS and other environments.

In this exercise, you will submit JCL to run an example batch application on a z/OS system. Before submitting the job, you will code a special 'TEST' option in the JCL. TEST causes a system feature called Language Environment to trigger the debugger when the job runs. When the debugger starts, it displays in your eclipse workbench. You can then use the debugger in eclipse to control and examine the application as it runs as a batch job on a z/OS system.

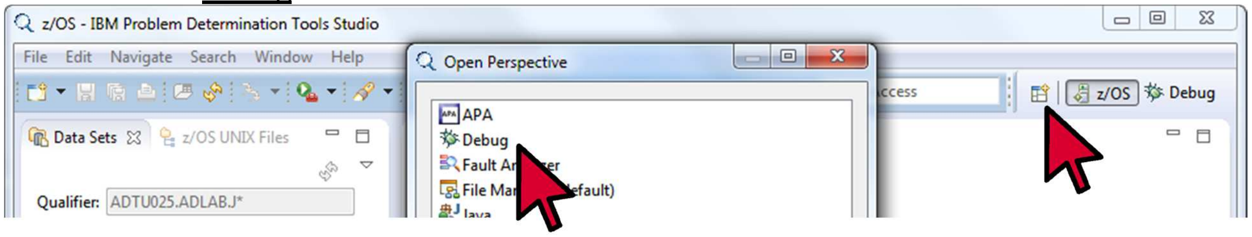
This is the general process you will perform:

1) In the Debug Tool eclipse perspective:
    a. turn on the Debug Tool IP listener
    b. determine the IP address of your workstation, and the Debug Tool listener IP port number
2) In TSO (or other development environment):
    a. edit and prepare JCL to run a provided sample application
    b. code a special 'CEEOPTS' DD in the JCL with a certain 'TEST' option. The IP address and listener port are coded as parameters of the TEST option, so the debugger knows where to connect.
3) Submit the JCL to run the sample application as a batch job on a z/OS system.
4) When the job runs, Debug Tool is triggered, and appears in the eclipse interface automatically.

1. First you will start the Debug Tool listener, and determine the IP address of your workstation.
   a. Determine if the Debug Perspective is already open. If the 'Debug' view is displayed, then the Debug perspective is already open.
   b. If it is not already open, open it:
      - From the menu near the top of the eclipse workbench, select **Window** > **Open Perspective** > **Debug**.
   c. The Debug perspective is open.

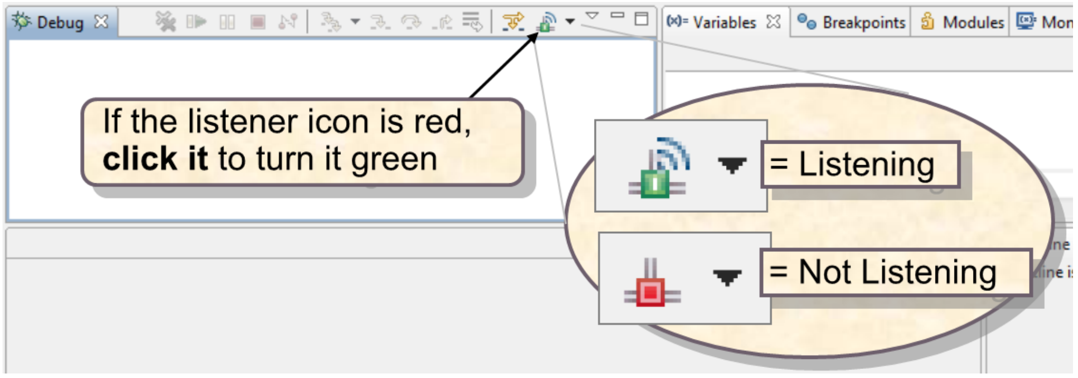

Debug - IBM Problem Determination Tools Studio
File Edit Navigate Search Project Run Window Help
Quick Access
File Manager
Debug
Fault Analyzer
APA
Debug
Variables
Breakpoints
Modules
Monitors
Outline
An outline is not available.
Console
Tasks
Debug Console
z/OS

      - Note: Another way to open the Debug Perspective is to click the 'Open Perspective' icon and then select **Debug**.


z/OS - IBM Problem Determination Tools Studio
File Edit Navigate Search Window Help
Data Sets
z/OS UNIX Files
Qualifier: ADTU025.ADLAB.J*
Open Perspective
APA
Debug
Fault Analyzer
File Manager (default)
Java
z/OS
Debug

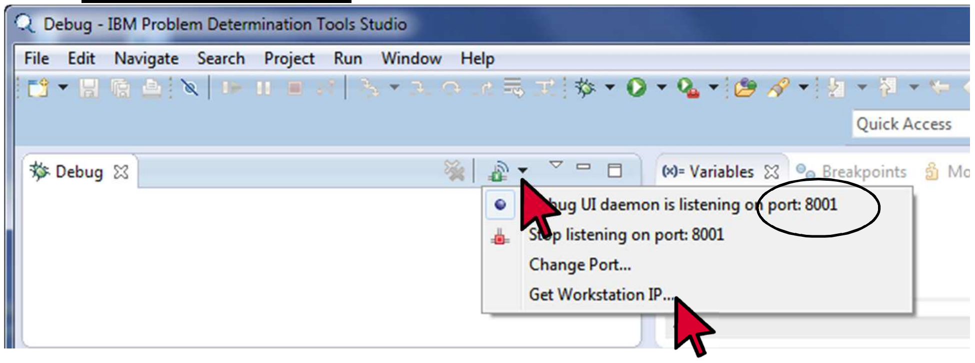
   d. Note: Debug Tool listens for your job to talk to the Debug Perspective. The Debug Tool listener must be on.

a. Ensure that the listener is on (green).
   - If the listener icon is red, then **click it** to turn it green.

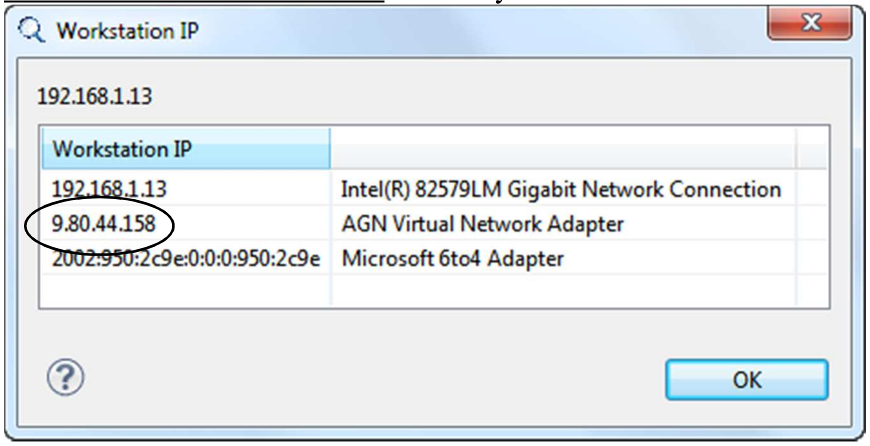


   - Ensure that the listener icon is green before continuing.

b. Next, note the Workstation IP address, and the listener port number.
   i. Click the small, black triangle next to the green listener icon.
   ii. The port is shown on the line: 'Debug UI daemon is listening on port ….'.
      - Make a note of the port number.
   iii. Click **Get Workstation IP**..


Debug - IBM Problem Determination Tools Studio
File Edit Navigate Search Project Run Window Help
Quick Access
Debug
Variables
Breakpoints
Debug UI daemon is listening on port: 8001
Stop listening on port: 8001
Change Port...
Get Workstation IP...

   iv. Make a note of the IP address. This is your workstation's network address.



Workstation IP

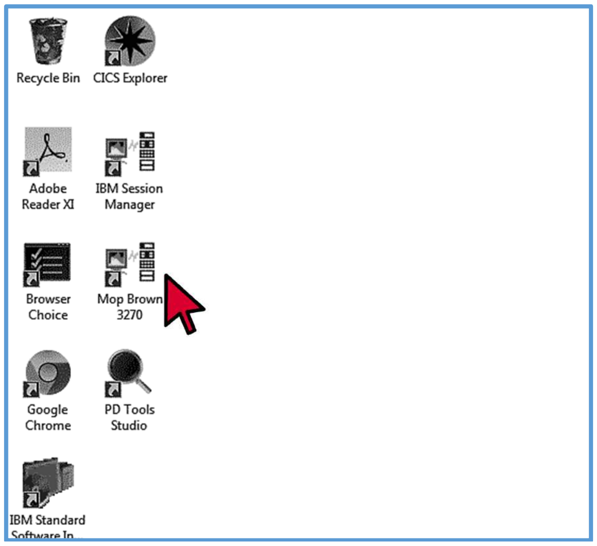
192.168.1.13

Workstation IP	
192.168.1.13	Intel(R) 82579LM Gigabit Network Connection
9.80.44.158	AGN Virtual Network Adapter
2002:950:2c9e:0:0:0:950:2c9e	Microsoft 6to4 Adapter

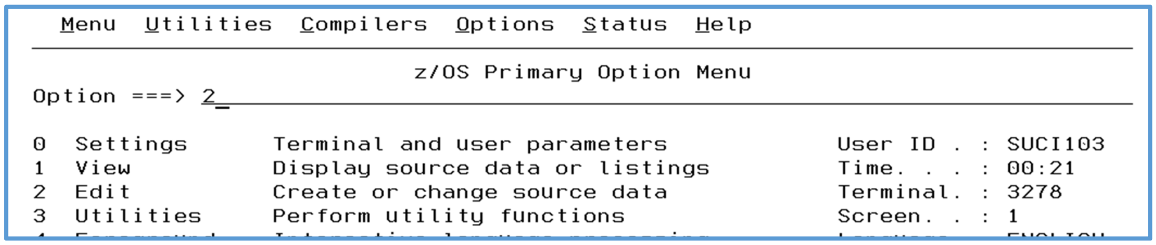
OK

   - Tip: You can copy the Workstation IP address by right-clicking the address, then selecting Copy.

2. Next, you will connect to a z/OS host system so you can edit JCL and submit a job.
   a. Double-click the icon on the desktop called **<u>MOP Brown 3270</u>**

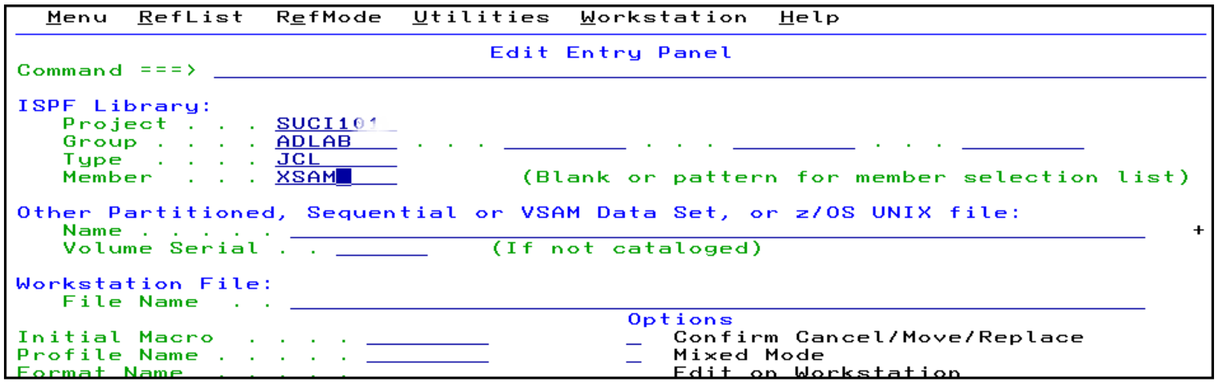


   b. A 3270 terminal window opens. Log on with the User ID and password you were provided.
      - Note: the 'Enter' key may be mapped to the right Ctrl key on your keyboard.
   c. After logging on, from the main menu, select option 2 (edit).



```
  Menu  Utilities  Compilers  Options  Status  Help
 ------------------------------------------------------------------------------
                            z/OS Primary Option Menu
 Option ===> 2_
 
 0  Settings      Terminal and user parameters          User ID . : SUCI103
 1  View          Display source data or listings       Time. . . : 00:21
 2  Edit          Create or change source data          Terminal. : 3278
 3  Utilities     Perform utility functions             Screen. . : 1
```

   d. The Edit Entry Panel is displayed. Next you will edit JCL that has been prepared for you.
      - Specify file name: your-id.ADLAB.JCL and member name XSAM.
      - Use the User ID that you were provided as the first qualifier.



```
  Menu  RefList  RefMode  Utilities  Workstation  Help
 ------------------------------------------------------------------------------
                              Edit Entry Panel
 Command ===> ____________________________________________________________

 ISPF Library:
    Project . . . SUCI101
    Group . . . . ADLAB     . . . ________ . . . ________ . . . ________
    Type  . . . . JCL
    Member  . . . XSAM        (Blank or pattern for member selection list)

 Other Partitioned, Sequential or VSAM Data Set, or z/OS UNIX file:
    Name . . . . . . ________________________________________________________ +
    Volume Serial . . ______    (If not cataloged)

 Workstation File:
    File Name  . . ________________________________________________________
                                         Options
 Initial Macro  . . . . ________         _  Confirm Cancel/Move/Replace
 Profile Name . . . . . ________         _  Mixed Mode
 Format Name  . . . . .                     Edit on Workstation
```

e. The JCL in your-id.ADLAB.JCL(XSAM) is displayed in the editor.

3. Customize the JCL.

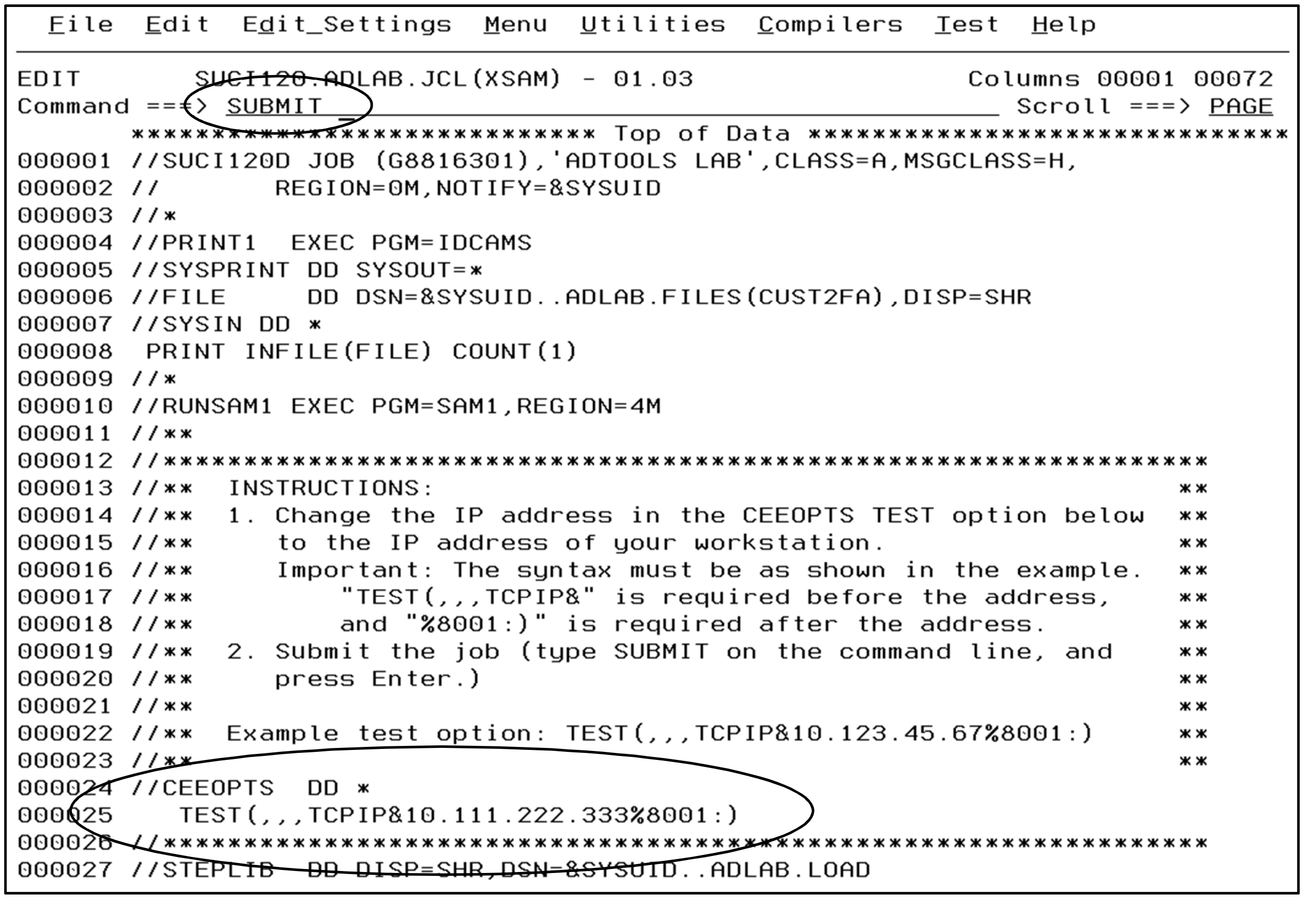
   a. Only one change is needed in the JCL: you must update the IP address coded in the TEST option.
      - Locate the CEEOPTS DD statement.
      - There is a TEST option just below it. The IP address is coded in the TEST string after the & (ampersand) and before the % (percent).
      - In an earlier step, you noted the IP address of your workstation. Replace the IP address in the TEST option with your IP address.
         - For example, if your IP address were 11.22.33.44, then you would code the CEEOPTS DD and TEST option like this:

```
//CEEOPTS  DD  *
TEST(,,,TCPIP&11.22.33.44%8001:)
```

      - After updating the JCL with your IP address, you are ready to submit it.

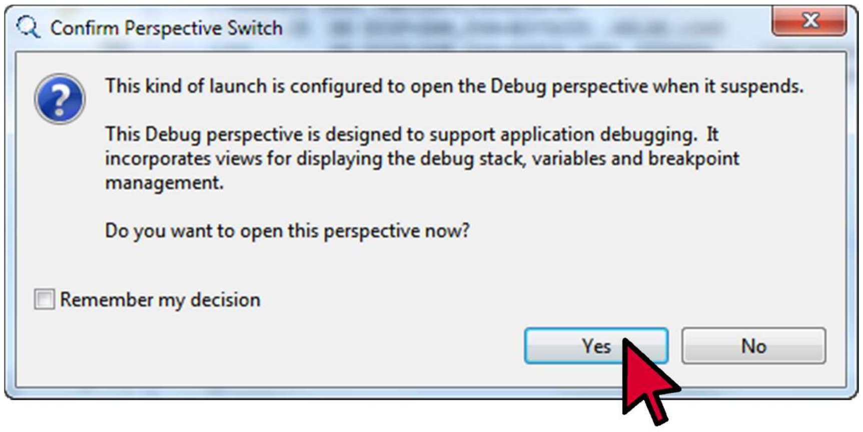
4. Submit the job.

   a. Type SUBMIT on the command line, and press Enter.



```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
 ──────────────────────────────────────────────────────────────────────────────
EDIT       SUCI120.ADLAB.JCL(XSAM) - 01.03                Columns 00001 00072
Command ===> SUBMIT _                                        Scroll ===> PAGE
       ****************************** Top of Data ******************************
000001 //SUCI120D JOB (G8816301),'ADTOOLS LAB',CLASS=A,MSGCLASS=H,
000002 //        REGION=0M,NOTIFY=&SYSUID
000003 //*
000004 //PRINT1  EXEC PGM=IDCAMS
000005 //SYSPRINT DD SYSOUT=*
000006 //FILE     DD DSN=&SYSUID..ADLAB.FILES(CUST2FA),DISP=SHR
000007 //SYSIN DD *
000008  PRINT INFILE(FILE) COUNT(1)
000009 //*
000010 //RUNSAM1 EXEC PGM=SAM1,REGION=4M
000011 //**
000012 //*****************************************************************
000013 //**  INSTRUCTIONS:                                               **
000014 //**  1. Change the IP address in the CEEOPTS TEST option below   **
000015 //**     to the IP address of your workstation.                   **
000016 //**     Important: The syntax must be as shown in the example.   **
000017 //**         "TEST(,,,TCPIP&" is required before the address,     **
000018 //**         and "%8001:)" is required after the address.         **
000019 //**  2. Submit the job (type SUBMIT on the command line, and     **
000020 //**     press Enter.)                                            **
000021 //**                                                              **
000022 //**  Example test option: TEST(,,,TCPIP&10.123.45.67%8001:)      **
000023 //**                                                              **
000024 //CEEOPTS  DD *
000025    TEST(,,,TCPIP&10.111.222.333%8001:)
000026 //*****************************************************************
000027 //STEPLIB  DD DISP=SHR,DSN=&SYSUID..ADLAB.LOAD
```

5. When the job runs, Debug Tool will automatically attempt to connect to your session.
   a. If the PD Tools Studio is not on top, click it's icon to select it.
   b. If you receive this'Confirm Perspective Switch' message, click **Yes**.



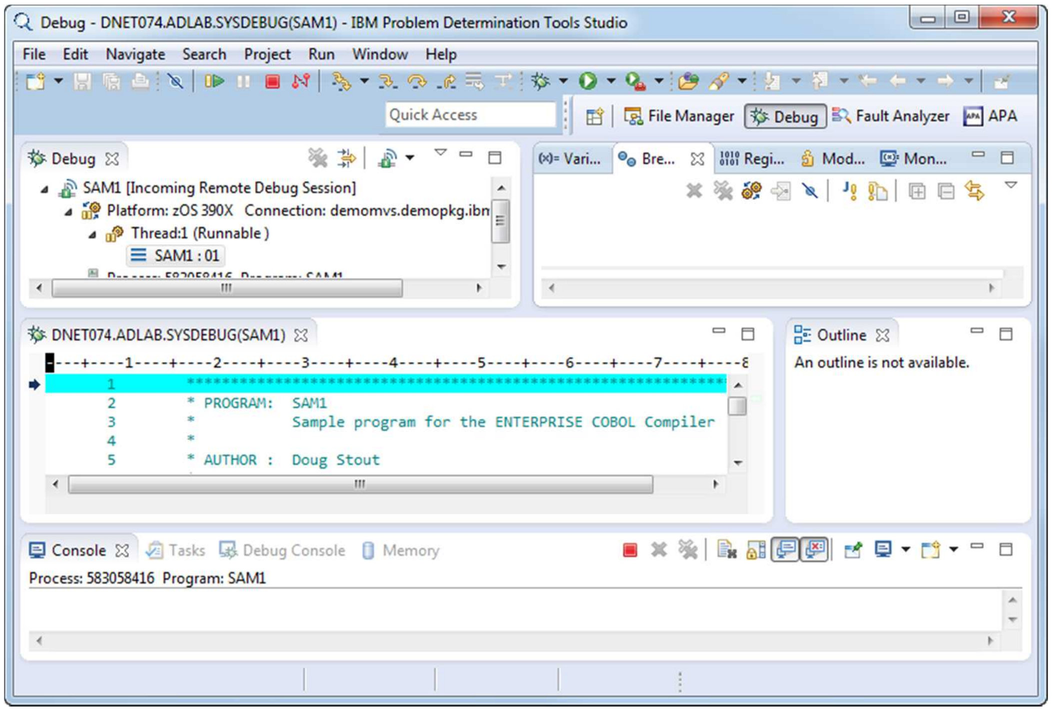
Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

   c. The debugger is displayed, and the source of the program is shown.



Debug - DNET074.ADLAB.SYSDEBUG(SAM1) - IBM Problem Determination Tools Studio

```
SAM1 [Incoming Remote Debug Session]
  Platform: zOS 390X  Connection: demomvs.demopkg.ibn
    Thread:1 (Runnable )
      SAM1 : 01
```

DNET074.ADLAB.SYSDEBUG(SAM1)

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
      ****************************************************************
      * PROGRAM:   SAM1
      *            Sample program for the ENTERPRISE COBOL Compiler
      *
      * AUTHOR :   Doug Stout
```

An outline is not available.

Process: 583058416 Program: SAM1

   d. If the debugger doesn't start, then …
      - Verify that the IP address and port are correct.
      - Ensure that the syntax of the TEST option is correct, based on the examples.

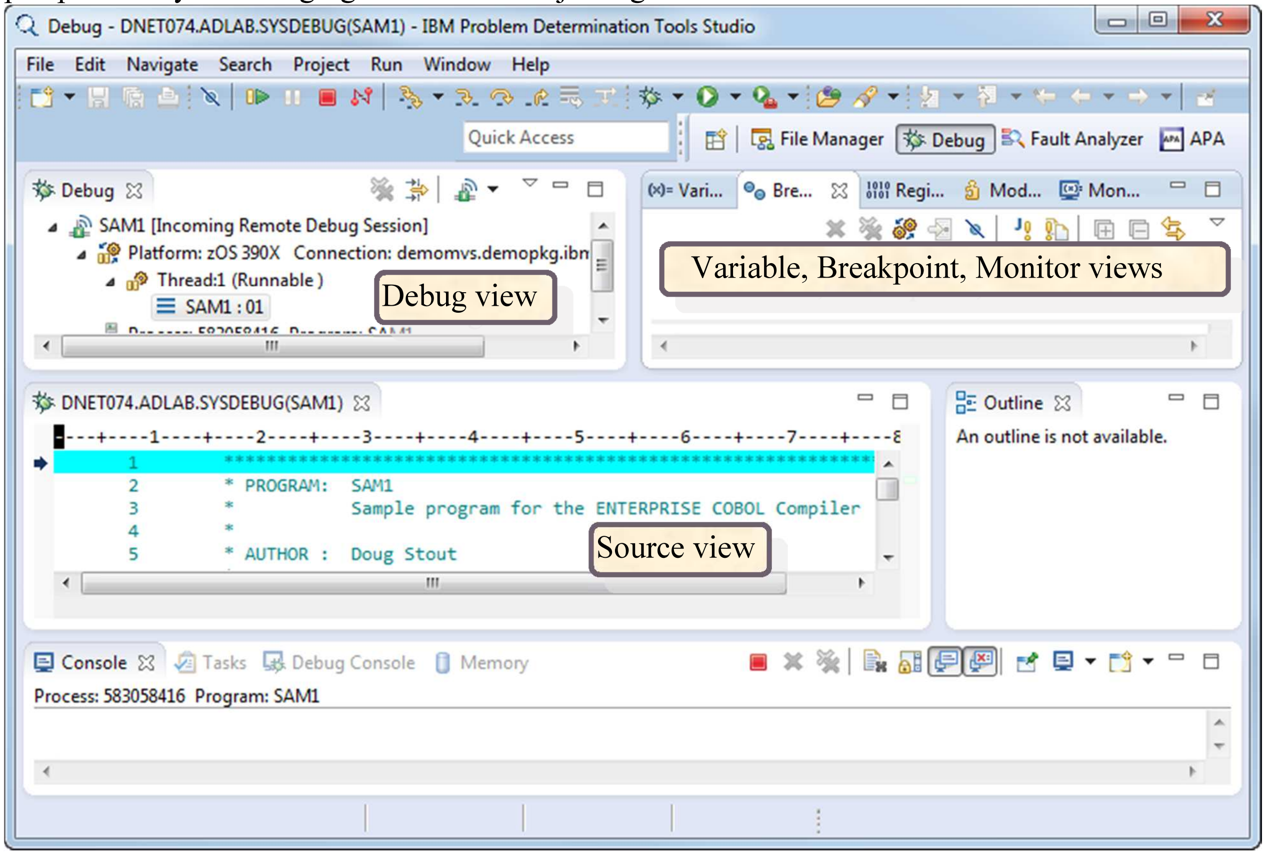
   e. If Debug Tool does not start, please ask for assistance.

6. After you have a Debug Tool session running, you are ready to proceed to the next Lab Exercise

Lab Exercise 3

Using the Debug Tool graphical user interface

1. Note: Before you do this exercise, you must first start a Debug Tool session by running sample program SAM1. If you do not already have a debug session started and you are not sure how to start it, return to the previous exercise, which describes how to start a debug session.
2. Note: Here are some things to remember during this exercise:
   a. If your debugging session ends, you can re-submit the job that you ran in the previous exercise to start a new session.
   b. You may notice that the example application will abend at a certain statement. Be aware that this is built into the example, and is expected. If you reach the abend, you can terminate the debugging session by clicking the Terminate button (the icon that looks like a red box) in the Tool Bar and then re-submit the job from the previous exercise.
3. Note: The Debug perspective is composed of several views (windows). You can customize the perspective by re-arranging the views or adjusting their sizes.

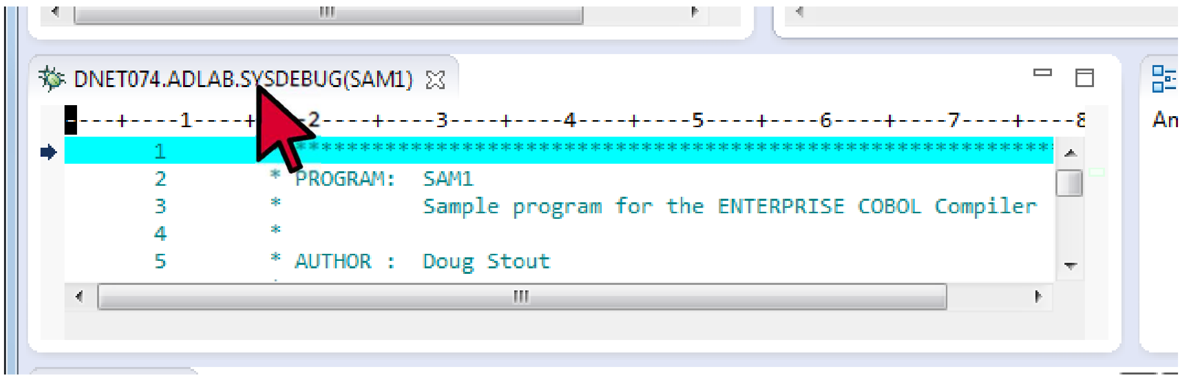

Debug - DNET074.ADLAB.SYSDEBUG(SAM1) - IBM Problem Determination Tools Studio
File Edit Navigate Search Project Run Window Help
Quick Access
File Manager
Debug
Fault Analyzer
APA
Debug
SAM1 [Incoming Remote Debug Session]
Platform: zOS 390X Connection: demomvs.demopkg.ibm
Thread:1 (Runnable)
SAM1 : 01
Debug view
Vari...
Bre...
Regi...
Mod...
Mon...
Variable, Breakpoint, Monitor views
DNET074.ADLAB.SYSDEBUG(SAM1)
Source view
Outline
An outline is not available.
Console
Tasks
Debug Console
Memory

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
      *****************************************************************
      * PROGRAM:  SAM1
      *           Sample program for the ENTERPRISE COBOL Compiler
      *
      * AUTHOR :  Doug Stout
```

```
Process: 583058416  Program: SAM1
```

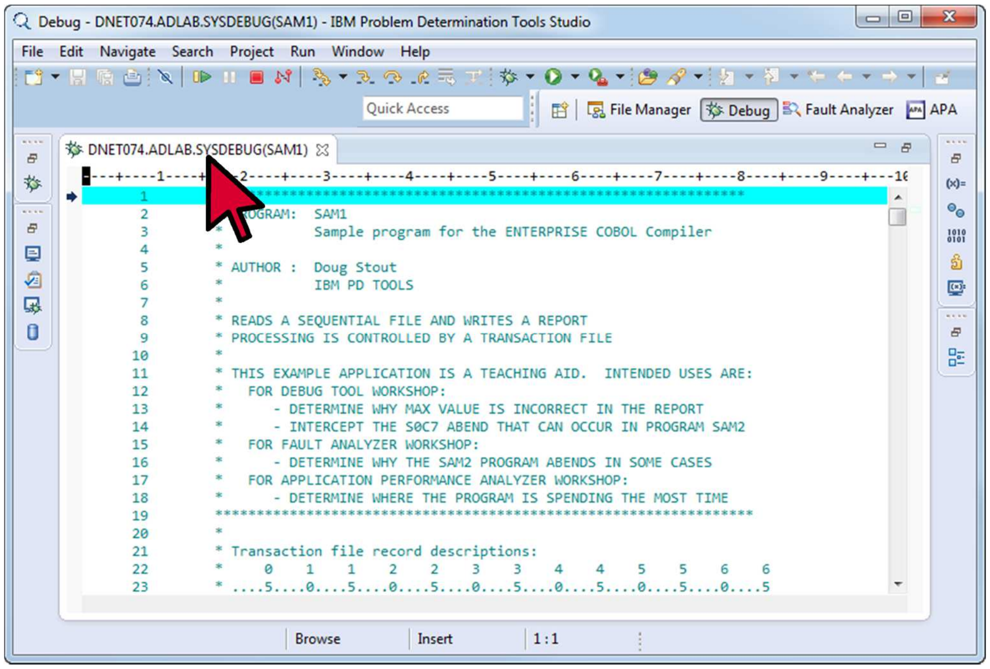
a. You will work with several of the views during this exercise. As an introduction, here is a brief description of the views that are used the most:
   i. The Source view displays and lets you work with the program that is being debugged. The current statement is highlighted. In the above example it is the blue line.
   ii. The Debug view shows the current call chain, from the main program to the current program.
   iii. The Monitor view is used to monitor variables. If you add a variable to the monitor view, you can watch it change as you step through the program.
   iv. The Variables view can be used to auto-monitor variables, so that variables referenced by each statement are displayed automatically as you step through the program.
   v. The Breakpoints view displays and lets you work with breakpoints.
   vi. The Console view lets you to enter commands to the Debug Tool engine. If you are familiar with the Debug Tool 3270 terminal interface, many of the same commands can be entered here.

4. A view can be maximized. For example, maximize the source view:
   a. **Double-click the tab** of the source view.



```
-----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
      ****************************************************************
      * PROGRAM:   SAM1
      *            Sample program for the ENTERPRISE COBOL Compiler
      *
      * AUTHOR :   Doug Stout
```

   b. The source view is maximized to the full eclipse workbench window.
   c. **Double-click the tab** of the source view again.



Debug - DNET074.ADLAB.SYSDEBUG(SAM1) - IBM Problem Determination Tools Studio
File
Edit
Navigate
Search
Project
Run
Window
Help
Quick Access
File Manager
Debug
Fault Analyzer
APA
DNET074.ADLAB.SYSDEBUG(SAM1)
Browse
Insert
1 : 1

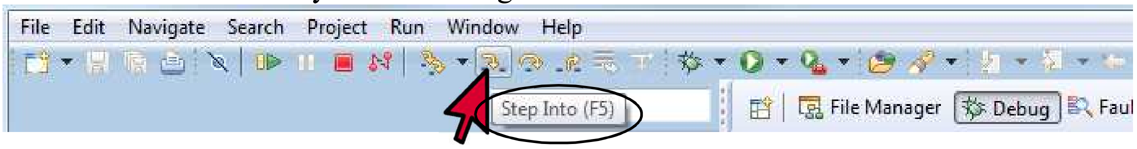
```
-----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----10
      ****************************************************************
      * PROGRAM:   SAM1
      *            Sample program for the ENTERPRISE COBOL Compiler
      *
      * AUTHOR :   Doug Stout
      *            IBM PD TOOLS
      *
      * READS A SEQUENTIAL FILE AND WRITES A REPORT
      * PROCESSING IS CONTROLLED BY A TRANSACTION FILE
      *
      * THIS EXAMPLE APPLICATION IS A TEACHING AID.  INTENDED USES ARE:
      *   FOR DEBUG TOOL WORKSHOP:
      *      - DETERMINE WHY MAX VALUE IS INCORRECT IN THE REPORT
      *      - INTERCEPT THE S0C7 ABEND THAT CAN OCCUR IN PROGRAM SAM2
      *   FOR FAULT ANALYZER WORKSHOP:
      *      - DETERMINE WHY THE SAM2 PROGRAM ABENDS IN SOME CASES
      *   FOR APPLICATION PERFORMANCE ANALYZER WORKSHOP:
      *      - DETERMINE WHERE THE PROGRAM IS SPENDING THE MOST TIME
      *****************************************************************
      *
      * Transaction file record descriptions:
      *     0    1    1    2    2    3    3    4    4    5    5    6    6
      * ....5....0....5....0....5....0....5....0....5....0....5....0....5
```

d. It is returned to its previous size. Most views can be maximized. It can sometimes be helpful to maximize a view to provide more room to work with its contents.

5. Notice that the tool bar near the top has several action buttons.

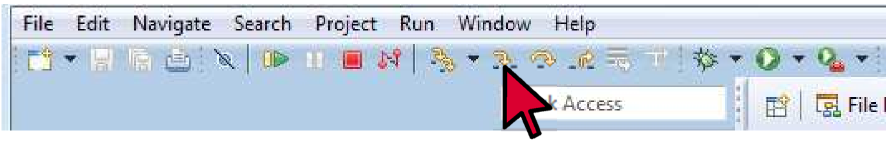
   a. You can hover over a button to determine what it does.

      - Hold your cursor, WITHOUT clicking, over the <u>Step Into</u> button (one of the yellow arrow icons).
      - Notice that a pop-up description is displayed, in this case: 'Step Into (F5)'. The (F5) indicates that pressing the F5 key will perform the same action as clicking the button. Use this hover feature when you are looking for a button.



6. You can step through statements in a program one at a time.

   a. Click the <u>Step Into</u> button.



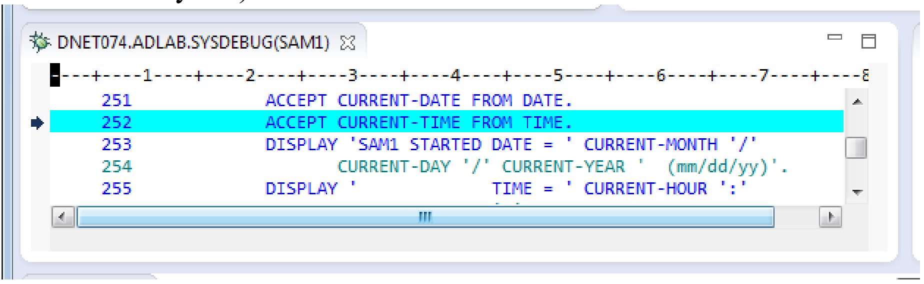
   b. Notice that the highlighted line in the source window moved.

      - Note: 'Step Into' is the button typically used to step through a program.

   c. **Press** the **<u>F5</u>** key.

      - Notice that F5 is the same as 'Step Into'. You can use which ever you prefer, either the button or the key.

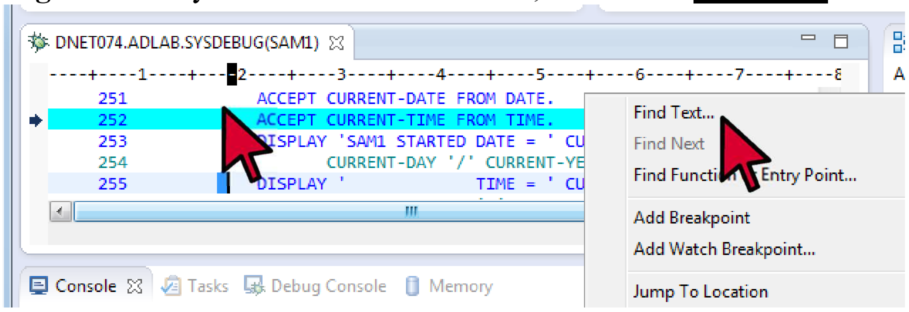
   d. **Continue to step** through the program until reaching statement 252. (It is OK if you step a few statements beyond).



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
 251          ACCEPT CURRENT-DATE FROM DATE.
 252          ACCEPT CURRENT-TIME FROM TIME.
 253          DISPLAY 'SAM1 STARTED DATE = ' CURRENT-MONTH '/'
 254                   CURRENT-DAY '/' CURRENT-YEAR '  (mm/dd/yy)'.
 255          DISPLAY '               TIME = ' CURRENT-HOUR ':'
```

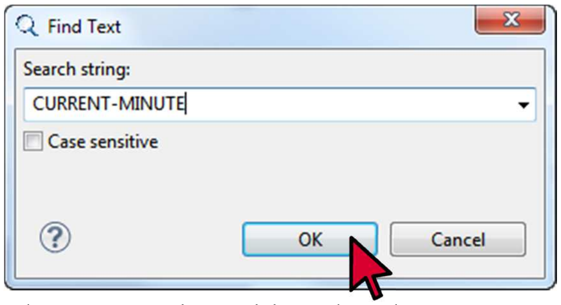
7. You can search for text in the source window.

   Right-click anywhere in the Source view, then select **<u>Find Text</u>**.

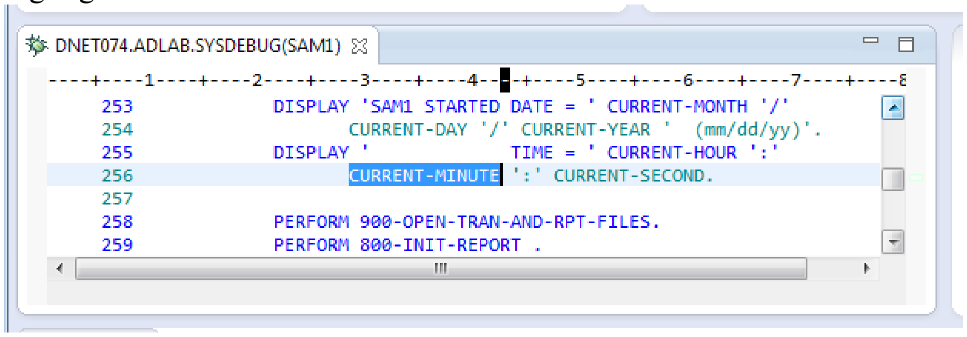


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
 251          ACCEPT CURRENT-DATE FROM DATE.
 252          ACCEPT CURRENT-TIME FROM TIME.
 253          DISPLAY 'SAM1 STARTED DATE = ' CU
 254                   CURRENT-DAY '/' CURRENT-YE
 255          DISPLAY '               TIME = ' CU
```

a. In the Find Text panel type **<u>CURRENT-MINUTE</u>** in the Search string field, then click **<u>OK</u>**.



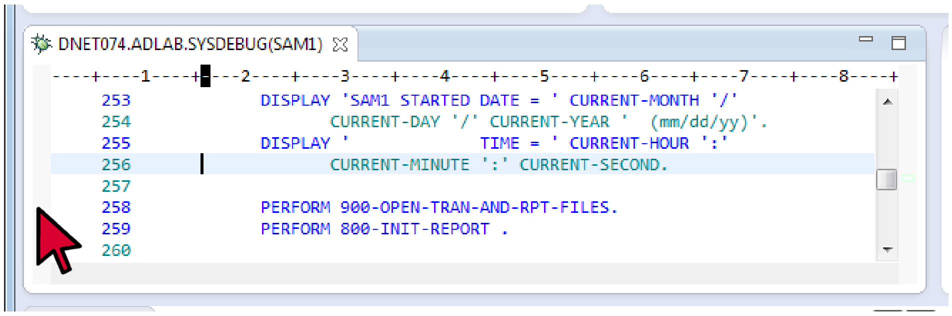
b. The program is positioned to the next occurrence of CURRENT-MINUTE with the text highlighted.



```
253              DISPLAY 'SAM1 STARTED DATE = ' CURRENT-MONTH '/'
254                     CURRENT-DAY '/' CURRENT-YEAR '  (mm/dd/yy)'.
255              DISPLAY '              TIME = ' CURRENT-HOUR ':'
256                     CURRENT-MINUTE ':' CURRENT-SECOND.
257
258              PERFORM 900-OPEN-TRAN-AND-RPT-FILES.
259              PERFORM 800-INIT-REPORT .
```

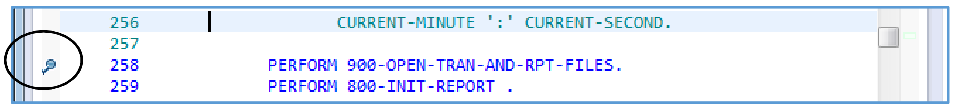
c. Tip: Pressing **<u>Ctrl-F</u>** will also open the Find Text dialog.

8. You can easily set a breakpoint at a statement and run the program until it reaches the breakpoint.
   a. Set a breakpoint on line 258:
      - **<u>Double-click</u>** in the blank area to the left of statement 258.



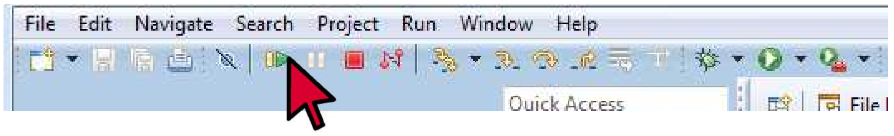
```
253              DISPLAY 'SAM1 STARTED DATE = ' CURRENT-MONTH '/'
254                     CURRENT-DAY '/' CURRENT-YEAR '  (mm/dd/yy)'.
255              DISPLAY '              TIME = ' CURRENT-HOUR ':'
256                     CURRENT-MINUTE ':' CURRENT-SECOND.
257
258              PERFORM 900-OPEN-TRAN-AND-RPT-FILES.
259              PERFORM 800-INIT-REPORT .
260
```

   b. A breakpoint icon is displayed.

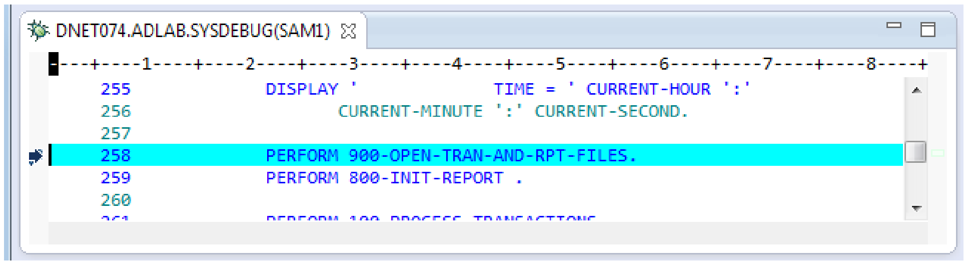


```
256                     CURRENT-MINUTE ':' CURRENT-SECOND.
257
258              PERFORM 900-OPEN-TRAN-AND-RPT-FILES.
259              PERFORM 800-INIT-REPORT .
```

   c. **Click** the **<u>RESUME</u>** button (green triangle) in the Tool Bar.



d. The program ran, continuing from where it already was, until it reached statement 258. Because of the breakpoint, execution is paused at 258. Statement 258 has not yet executed.

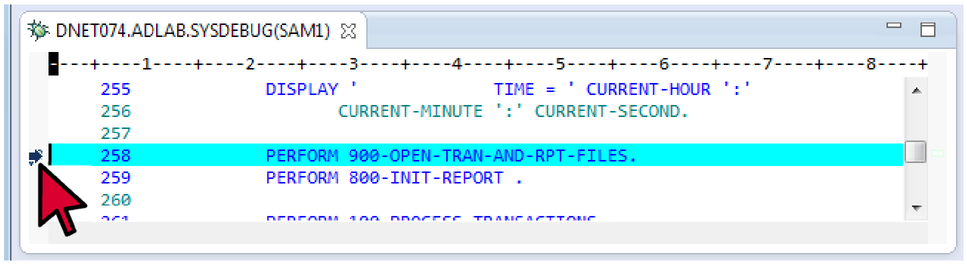


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
  255           DISPLAY '            TIME = ' CURRENT-HOUR ':'
  256                    CURRENT-MINUTE ':' CURRENT-SECOND.
  257
  258           PERFORM 900-OPEN-TRAN-AND-RPT-FILES.
  259           PERFORM 800-INIT-REPORT .
  260
  261           PERFORM 100-PROCESS-TRANSACTIONS
```

- Tip: If you prefer, you can use the <u>F8</u> key instead of the <u>Resume</u> button (but don't do it right now).
- Note: 'Resume' runs the program, continuing from its current location. The program will continue to run until it encounters a Breakpoint, encounters a condition (such as an abend), or completes.

e. The breakpoint can be removed:

- **<u>Double-click</u>** the breakpoint icon on statement 258.



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
  255           DISPLAY '            TIME = ' CURRENT-HOUR ':'
  256                    CURRENT-MINUTE ':' CURRENT-SECOND.
  257
  258           PERFORM 900-OPEN-TRAN-AND-RPT-FILES.
  259           PERFORM 800-INIT-REPORT .
  260
  261           PERFORM 100-PROCESS-TRANSACTIONS
```

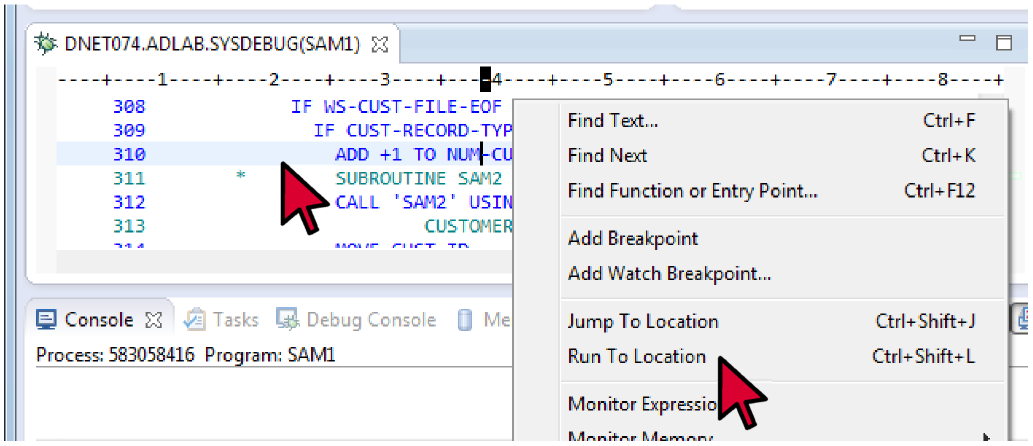
- Notice that the breakpoint is removed.

9. In the previous steps, you set a breakpoint and then ran the program until it reached it. However, there is another way to run to a particular statement and you do not even have to set a breakpoint first: the 'Run To Location' function.

a. Scroll the Source until line 310 is displayed.

b. **<u>Click</u>** on **line 310** to select it.

c. **<u>Right-click</u>** line 310, then select **<u>Run To Location</u>**.

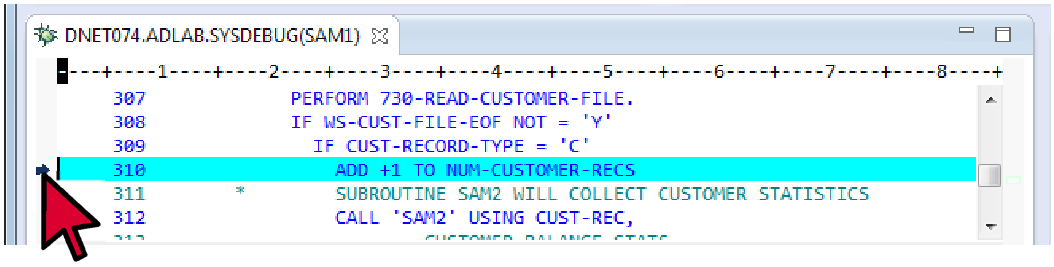


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
  308               IF WS-CUST-FILE-EOF
  309                  IF CUST-RECORD-TYP
  310                     ADD +1 TO NUM-CU
  311          *          SUBROUTINE SAM2
  312                     CALL 'SAM2' USIN
  313                             CUSTOMER
```

```
Process: 583058416  Program: SAM1
```

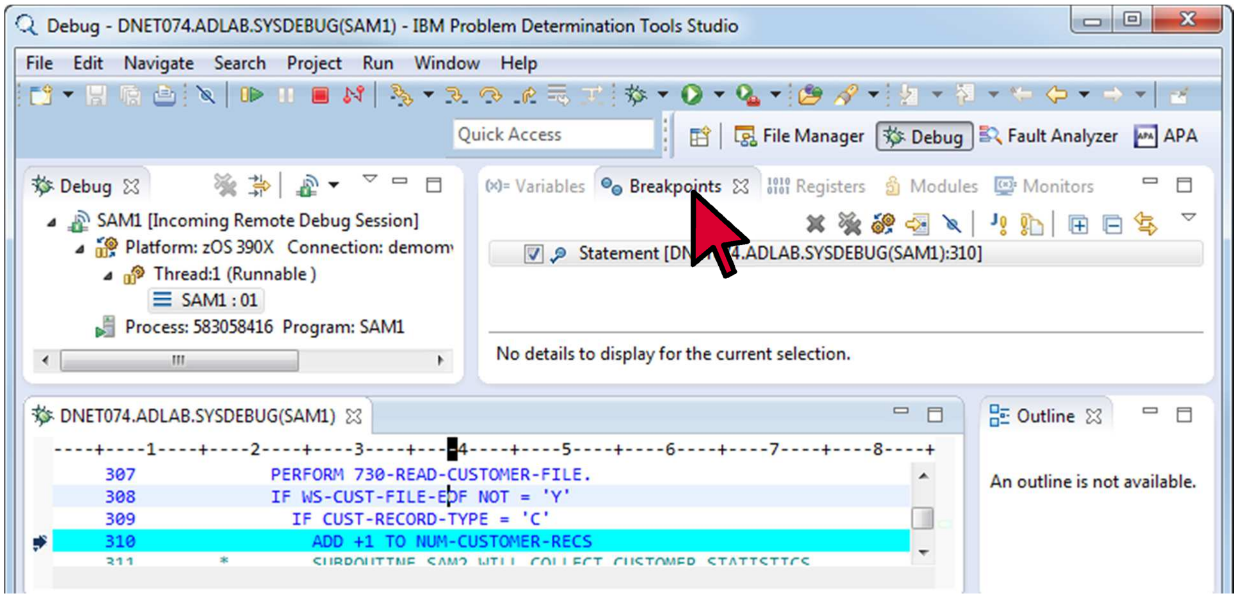
d. The program ran, continuing from where it was, until it reached statement 310. Execution is paused at 310. Statement 310 has not yet executed.
   - Note: 'Run To Location' runs the program, continuing from its current location. The program will continue to run until it reaches the Run-To statement, or encounters a Breakpoint, or encounters a condition (such as an abend), or completes.

10. You can also work with breakpoints from the breakpoints view.
   a. Note: You can use the breakpoints view to display, create, modify and remove breakpoints. Breakpoints can also be temporarily enabled and disabled.
   b. **Double-click** next to **line 310** to set a breakpoint there.



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
 307            PERFORM 730-READ-CUSTOMER-FILE.
 308            IF WS-CUST-FILE-EOF NOT = 'Y'
 309              IF CUST-RECORD-TYPE = 'C'
 310                ADD +1 TO NUM-CUSTOMER-RECS
 311        *       SUBROUTINE SAM2 WILL COLLECT CUSTOMER STATISTICS
 312                CALL 'SAM2' USING CUST-REC,
```

   c. Click the tab of the Breakpoints view .



SAM1 [Incoming Remote Debug Session]
Platform: zOS 390X Connection: demom
Thread:1 (Runnable)
SAM1 : 01
Process: 583058416 Program: SAM1

Statement [DNET074.ADLAB.SYSDEBUG(SAM1):310]

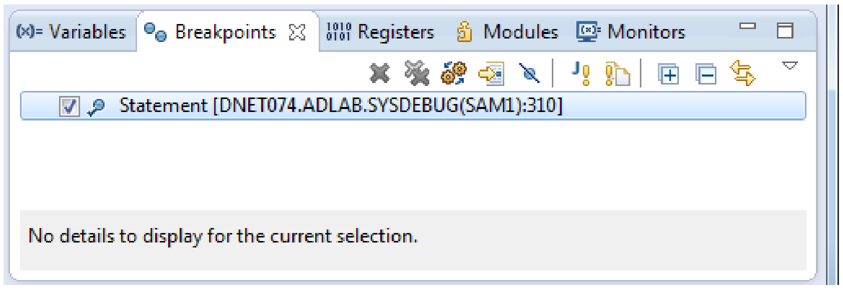
No details to display for the current selection.

DNET074.ADLAB.SYSDEBUG(SAM1)

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
 307            PERFORM 730-READ-CUSTOMER-FILE.
 308            IF WS-CUST-FILE-EOF NOT = 'Y'
 309              IF CUST-RECORD-TYPE = 'C'
 310                ADD +1 TO NUM-CUSTOMER-RECS
```

An outline is not available.

   d. Notice that it displays a list of breakpoints.

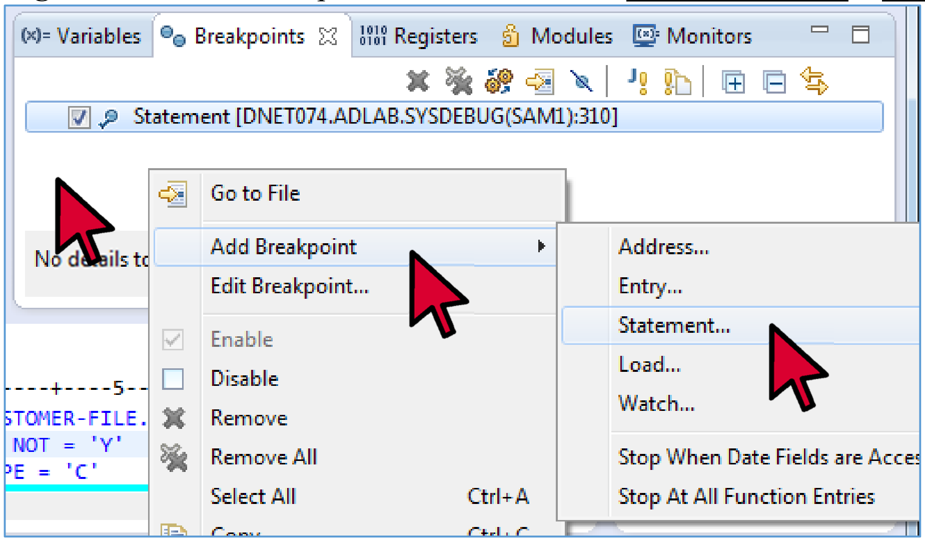


Statement [DNET074.ADLAB.SYSDEBUG(SAM1):310]

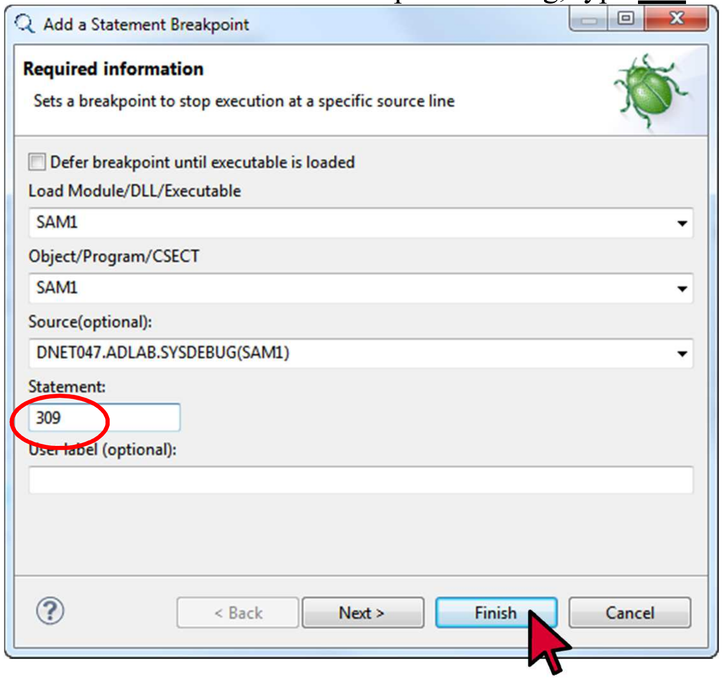
No details to display for the current selection.

11. Breakpoints can also be created in the Breakpoint view.

a. **Right-click** in the Breakpoints view, then select **Add Breakpoint** > **Statement**.



b. In the 'Add a Statement Breakpoint' dialog, type **309** in the Statement field, then **click Finish**.

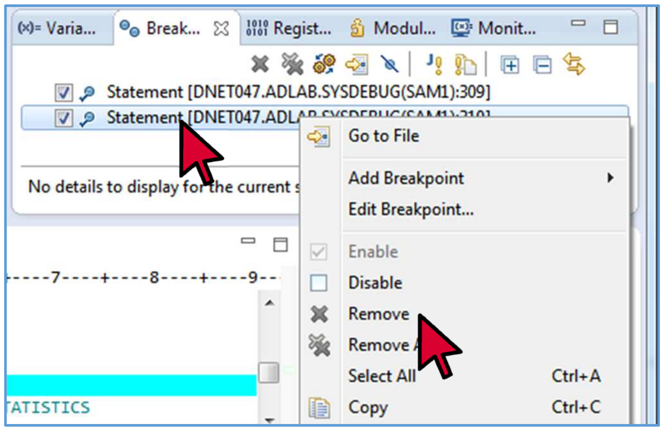


Required information
Sets a breakpoint to stop execution at a specific source line

Defer breakpoint until executable is loaded
Load Module/DLL/Executable: SAM1
Object/Program/CSECT: SAM1
Source(optional): DNET047.ADLAB.SYSDEBUG(SAM1)
Statement: 309
User label (optional):

12. Notice that a breakpoint was added at statement 309.

13. You can also remove a breakpoint from the Breakpoints view.
    a. Right-click the breakpoint for line 310, then select **Remove**.



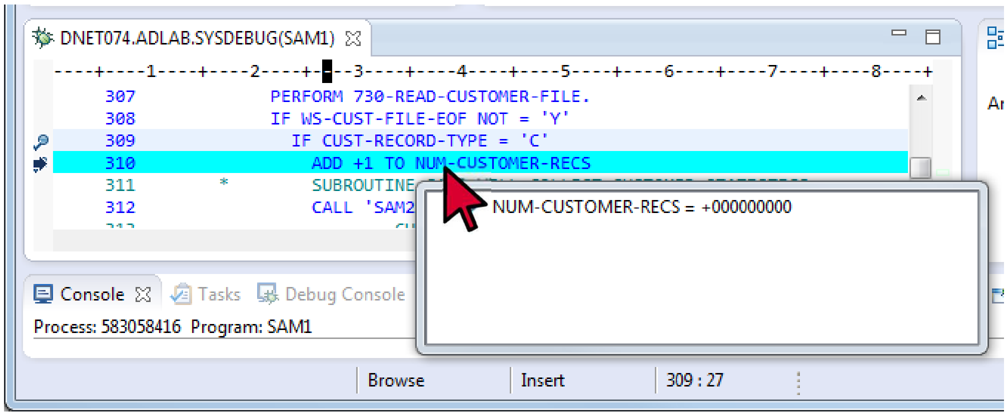
Statement [DNET047.ADLAB.SYSDEBUG(SAM1):309]
Statement [DNET047.ADLAB.SYSDEBUG(SAM1):310]

No details to display for the current selection.

    b. Notice that the breakpoint is removed.
        - Tip: You can also select and then click the 'Remove breakpoint' icon in the Breakpoint view tool bar to remove a breakpoint.

14. Note: there are many ways to display and work with variables in the debugger. In the next steps, you will become familiar with the hover feature, the Variables view and the Monitors views.

15. The hover feature provides an easy way to display the value of a variable.
    a. **Scroll** the source window (if necessary) **to display line 310**. Notice that statement 310 references the NUM-CUSTOMER-RECS variable.
    b. Hold your cursor, WITHOUT clicking, over the NUM-CUSTOMER-RECS variable on line 310.
    c. Notice that the value of the variable is displayed in a pop-up window.



```
----+----1----+----2----+-█--3----+----4----+----5----+----6----+----7----+----8----+
307             PERFORM 730-READ-CUSTOMER-FILE.
308             IF WS-CUST-FILE-EOF NOT = 'Y'
309               IF CUST-RECORD-TYPE = 'C'
310                 ADD +1 TO NUM-CUSTOMER-RECS
311        *        SUBROUTINE
312                 CALL 'SAM2
```

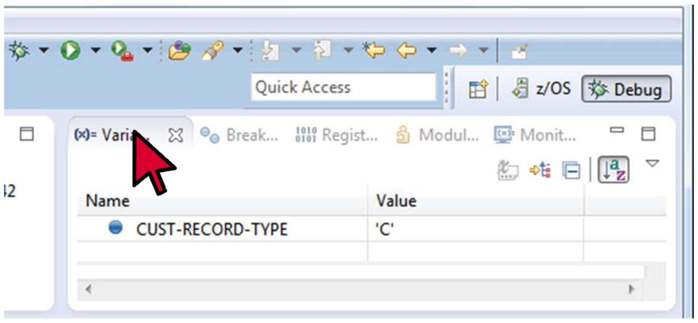
NUM-CUSTOMER-RECS = +000000000

Process: 583058416 Program: SAM1

        - Tip: hovering in the Source window is a quick and easy way to display variable values.

16. The Variables view provides additional ways to display variable values.

a. **Click** the tab of the **Variables** view to display it (in case it is not already on top).

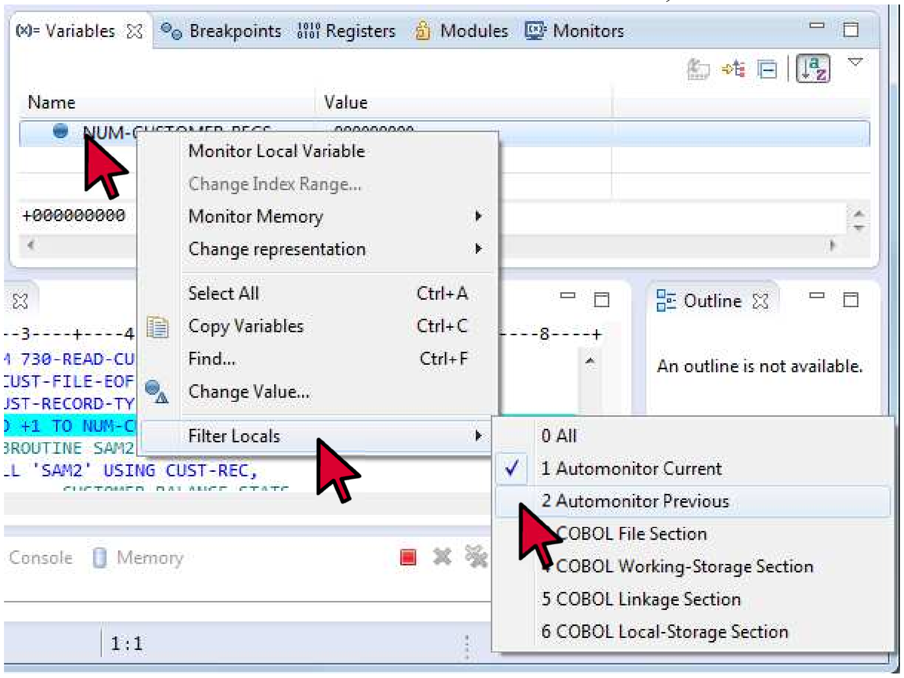


Name	Value
CUST-RECORD-TYPE	'C'

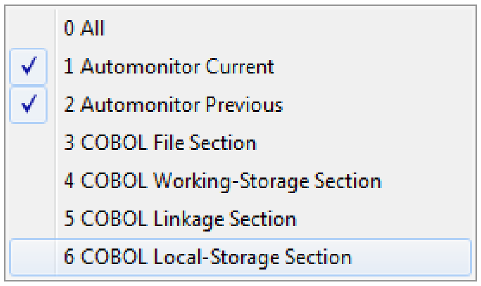
b. Note: the variables view can be used to automatically monitor the values of referenced variables as you step through a program.

c. Turn on the Automonitor filters:

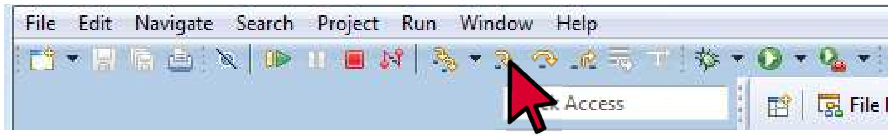
- **Right-click** any variable in the Variables view, then select **Filter Locals**.
- Notice that there are selections for 'Automonitor Current' and Automonitor Previous'. If either of them is not selected with a check mark, click it to select it.



- If necessary, repeat until 'Automonitor Current and 'Automonitor Previous' are *both* selected.



d. **Click** the **Step Into** button.



- Notice the current statement in the source window. The variables referenced by the current statement are automatically displayed in the Variables view.

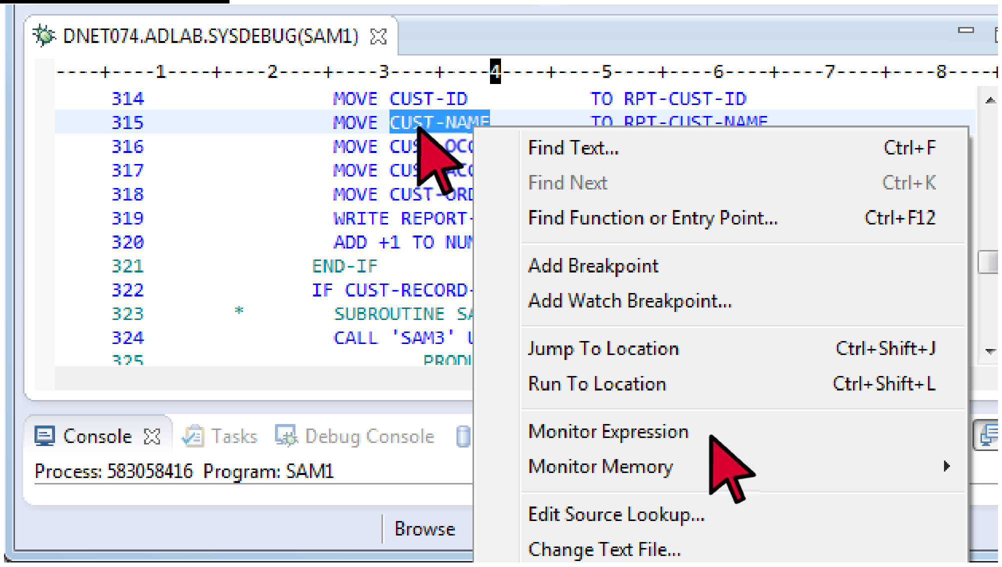
e. **Click** the **Step Into** button several times again.

- Another statement is reached. Notice that variables referenced by the new current statement are displayed automatically in the Variables view.
- The 'Automonitor Current' filter displays variables for the current statement.
- The Automonitor Previous' filter displays variables for the previously displayed statement, so that you can automatically see changes made to variables as you step.

f. Tip: You can display an item in the Variables view in hexadecimal display by right-clicking it, then selecting Change representation > Hexadecimal.

17. Notice that as you stepped, the application stepped into a subprogram called SAM2.
    a. Continue to step (using the Step Into button, or the F5 key) until the application returns back to the calling program SAM1.
    b. Note: It will take approximately 20 or 30 steps until it reaches the GOBACK statement on line 82 and returns.

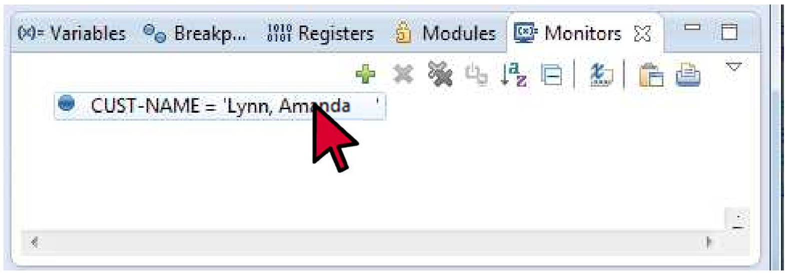
18. The Monitors view provides the ability to add variables permanently to the display. Different from the Variables view, an item added to the Monitors view remains visible until it is removed.
    a. Ensure that you have stepped back to program SAM1.
    b. Scroll the source window until line 315 is displayed.
    c. On line 315, **double-click** the variable **CUST-NAME** to select it, then **right-click** it, then select **Monitor Expression**.



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
314                MOVE CUST-ID            TO RPT-CUST-ID
315                MOVE CUST-NAME          TO RPT-CUST-NAME
316                MOVE CUS
317                MOVE CUS
318                MOVE CUST-ORD
319                WRITE REPORT-
320                ADD +1 TO NUM
321              END-IF
322              IF CUST-RECORD-
323      *         SUBROUTINE SA
324                CALL 'SAM3' U
325                        PRODU
```

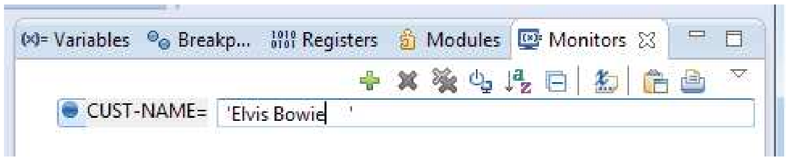
   d. Click the tab of the Monitors view.
   e. Notice that CUST-NAME is added to the Monitors view and its current value is displayed.

19. You can change the value of a variable.
   a. Double-click the area containing the data.



CUST-NAME = 'Lynn, Amanda '

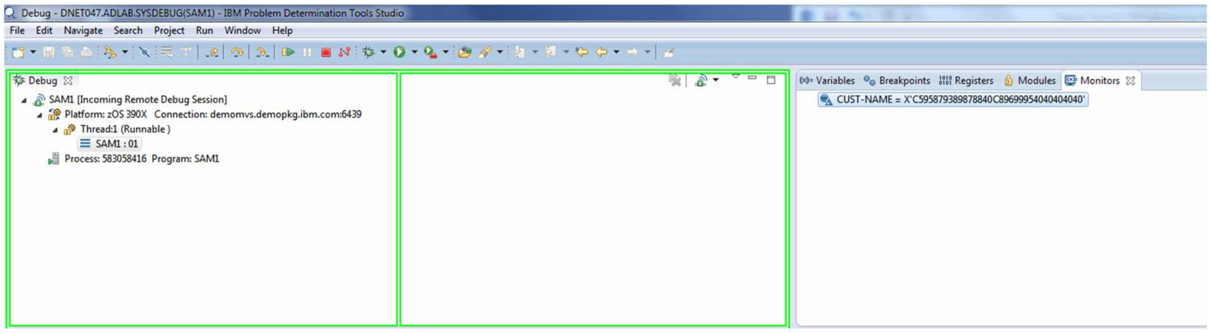
   b. Overtype the data. Make up a name and enter it into the CUST-NAME variable, then press **Enter**.
      - Hint: since it is a character field, the single quotation marks are required.



CUST-NAME= 'Elvis Bowie '

   c. The value of the variable is changed.

20. You may prefer to have both the Monitors view and Variables view visible at the same time. This is possible because the locations and sizes of views can be rearranged.
   a. Click and hold the tab of the Monitors view. Drag it to one side or the other until a green outline of the view appears where you want it. Release the view at a desired location.

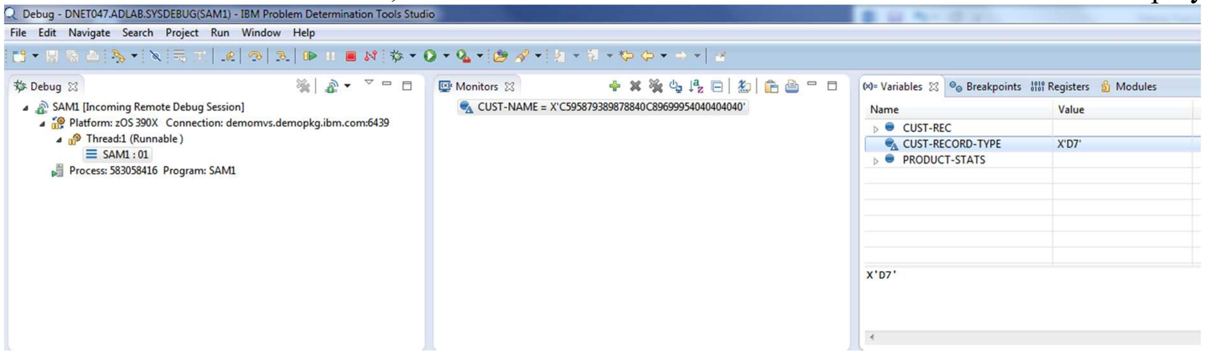


Debug - DNET047.ADLAB.SYSDEBUG(SAM1) - IBM Problem Determination Tools Studio

SAM1 [Incoming Remote Debug Session]
Platform: zOS 390X Connection: demomvs.demopkg.ibm.com:6439
Thread:1 (Runnable)
SAM1 : 01
Process: 583058416 Program: SAM1

CUST-NAME = X'C595879389878840C89699954040404040'

   b. The Monitor view was moved, and the Variables view and Monitors view can both be displayed:



Debug - DNET047.ADLAB.SYSDEBUG(SAM1) - IBM Problem Determination Tools Studio

SAM1 [Incoming Remote Debug Session]
Platform: zOS 390X Connection: demomvs.demopkg.ibm.com:6439
Thread:1 (Runnable)
SAM1 : 01
Process: 583058416 Program: SAM1

CUST-NAME = X'C595879389878840C89699954040404040'

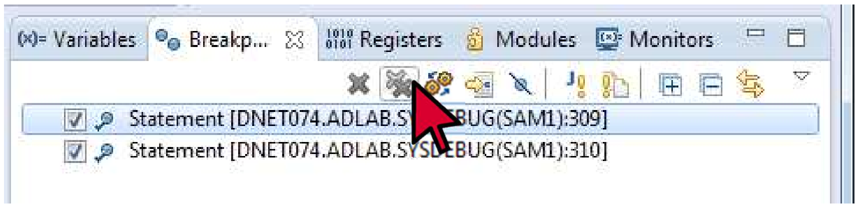
Name	Value
CUST-REC	
CUST-RECORD-TYPE	X'D7'
PRODUCT-STATS	

X'D7'

c. Hint: If the views accidentally get rearranged in an undesirable way, remember that selecting Window > Reset perspective will return all views to their default locations.

21. You can make breakpoints conditional.

a. Remove all existing breakpoints:
   - In the Breakpoints view **Click** the **Remove all Breakpoints** icon.



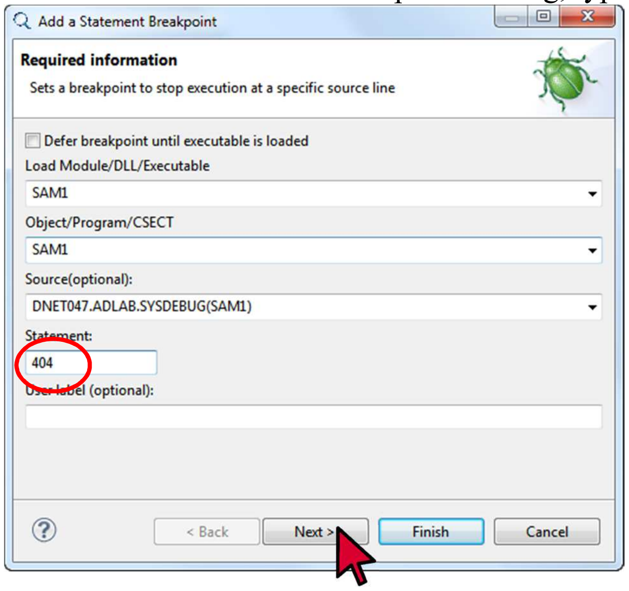
Statement [DNET074.ADLAB.SYSDEBUG(SAM1):309]
Statement [DNET074.ADLAB.SYSDEBUG(SAM1):310]

   - A confirmation dialogue is displayed.
     - Click **YES** .
   - Notice that all breakpoints are removed.

b. Add variable CUST-ID to the Monitor view:
   - Scroll to line 314.
   - **Double-click** variable **CUST-ID** to highlight it, then **right-click** it, then select **Monitor Expression**.

c. Next, add a conditional breakpoint:
   - **Right-click** in the Breakpoints view, then select **Add Breakpoint** > **Statement**.

d. In the 'Add a Statement Breakpoint' dialog, type **404** in the statement field, then click **Next**.



Required information

Sets a breakpoint to stop execution at a specific source line

☐ Defer breakpoint until executable is loaded

Load Module/DLL/Executable: SAM1

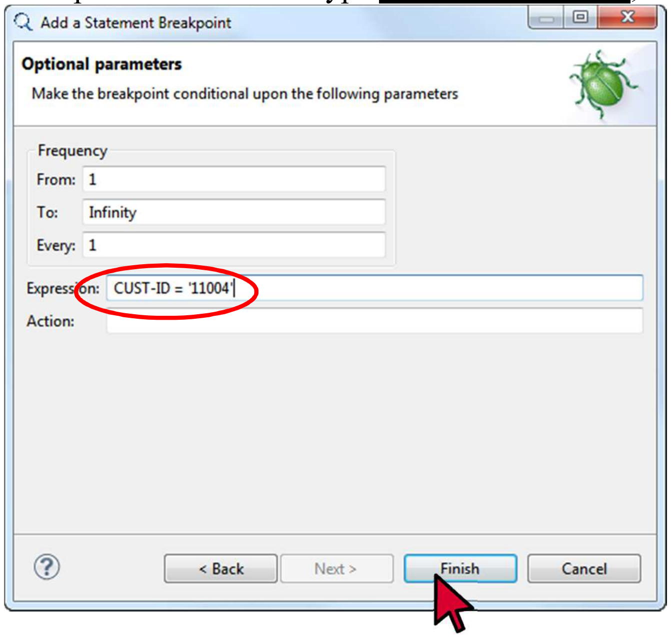
Object/Program/CSECT: SAM1

Source(optional): DNET047.ADLAB.SYSDEBUG(SAM1)

Statement: 404

User label (optional):

e. In the Expression field area type **CUST-ID='11004'**, then click **Finish**.


Add a Statement Breakpoint

Optional parameters

Make the breakpoint conditional upon the following parameters

Frequency

From: 1

To: Infinity

Every: 1

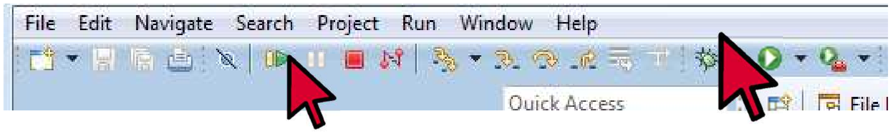
Expression: CUST-ID = '11004'

Action:

- Note: This breakpoint will check the condition each time the program reaches statement 404. If the condition is true, then the program will be paused at 404.

f. The new breakpoint appears in the Breakpoints view.

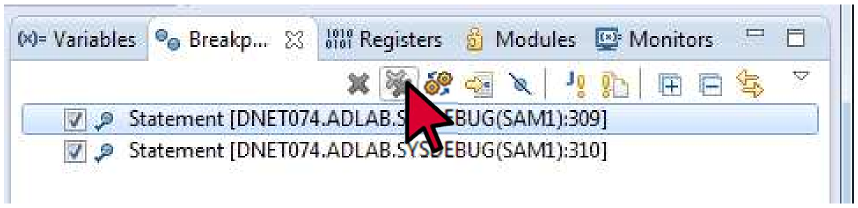
g. **Click** the **RESUME** button in the tool bar.



h. The program is paused at statement 404. Notice in the Monitors view that the value of CUST-ID is '11004'. The breakpoint triggered only when 404 was reached and the condition was true.

22. Note: So far in the exercise, you have only used statement breakpoints. Next, you will add a watch breakpoint. A watch breakpoint pauses the program when a named variable changes, instead of at a specific statement.

a. In the Breakpoints view, remove all breakpoints:
   - Click the Remove all Breakpoints icon.



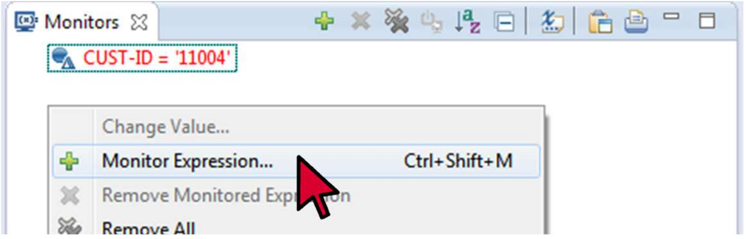
Statement [DNET074.ADLAB.SYSDEBUG(SAM1):309]

Statement [DNET074.ADLAB.SYSDEBUG(SAM1):310]

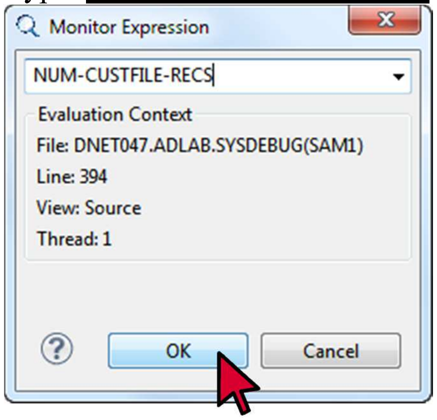
   - A confirmation dialogue is displayed.
     - Click **YES** .
   - Notice that all breakpoints are removed.

b. Add variable NUM-CUSTFILE-RECS to the Monitor view:

- **Right-click** in the Monitors view, then select **Monitor Expression**.

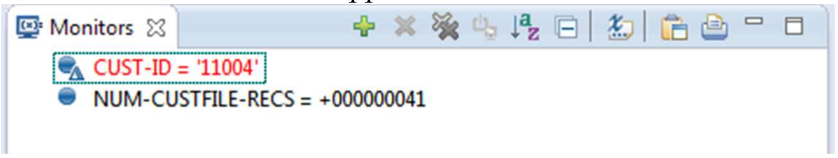

Monitors
CUST-ID = '11004'
Change Value...
Monitor Expression... Ctrl+Shift+M
Remove Monitored Expression
Remove All

- Type **NUM-CUSTFILE-RECS** then click **OK**.



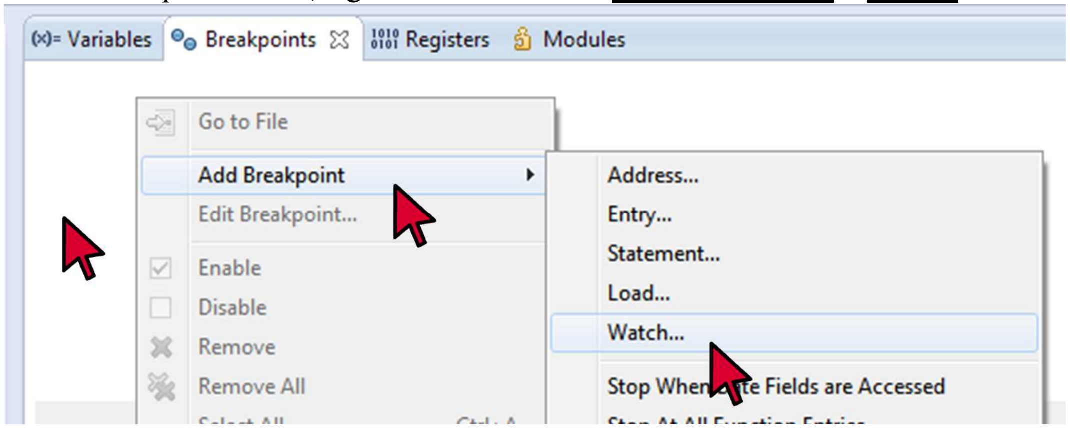
Evaluation Context
File: DNET047.ADLAB.SYSDEBUG(SAM1)
Line: 394
View: Source
Thread: 1

- NUM-CUSTFILE-RECS appears in the Monitor view. Notice its current value.

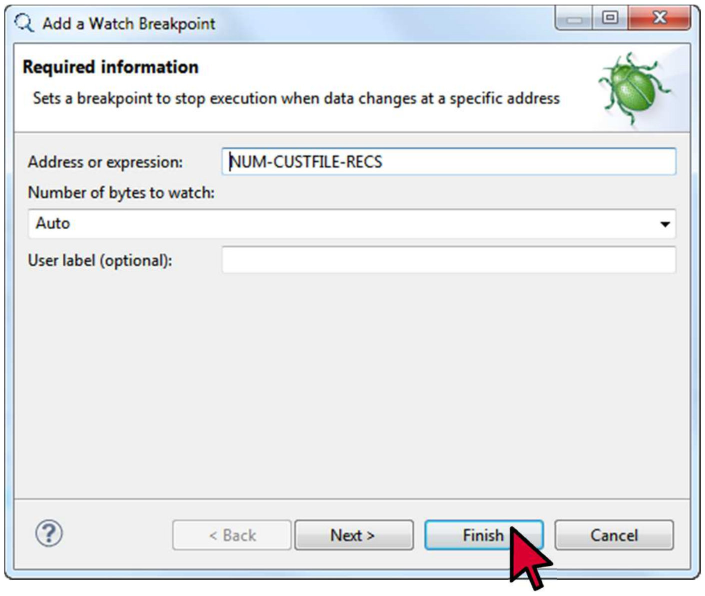


CUST-ID = '11004'
NUM-CUSTFILE-RECS = +000000041

c. In the Breakpoints view, right-click then select **Add Breakpoint** > **Watch**.



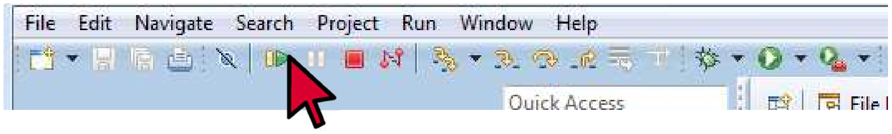
d. In the 'Add a Watch Breakpoint' dialog, type **NUM-CUSTFILE-RECS** in the Address or expression field, then click **Finish**.


Add a Watch Breakpoint
Required information
Sets a breakpoint to stop execution when data changes at a specific address
< Back
Next >
Finish
Cancel

Address or expression: NUM-CUSTFILE-RECS
Number of bytes to watch: Auto
User label (optional):

e. The breakpoint was added.

f. **Click** the **RESUME** button in the Tool Bar.



g. The watch breakpoint paused the program *after* the statement that caused it to change executed. Notice in the Monitors view that the value of NUM-CUSTFILE-RECS has changed.

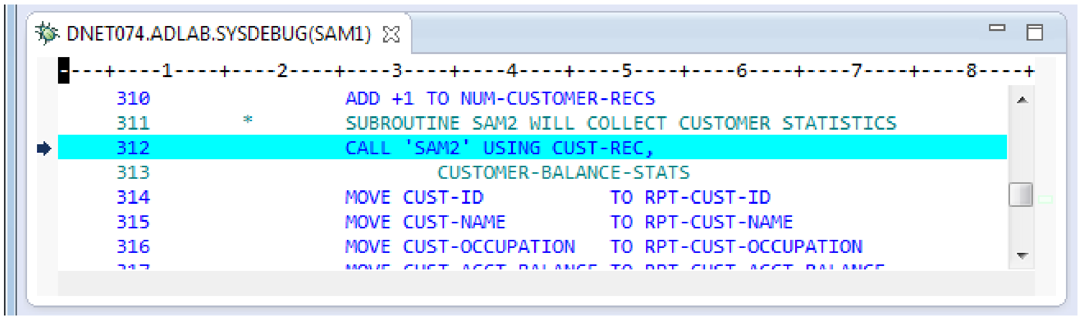
h. Tip: You can make a watch breakpoint conditional if you only want the program to stop when the variable is changed to a certain value.

23. Note: You can use the debugger to easily follow program logic into and out of subprograms. One simple method is to simply step into a called program.

    a. Run to statement 312:

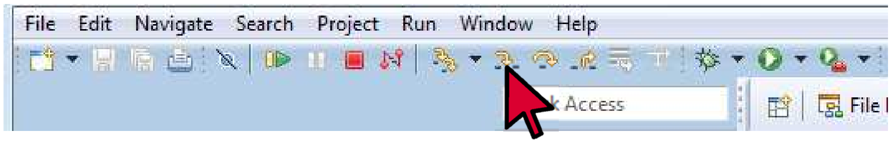
        - **Click** statement **312** in the source view to select it.
        - Then **right-click** statement **312** in the source view, then select **Run To Location**.

b. Notice that line 312 is a CALL statement that will execute a subprogram named SAM2.


DNET074.ADLAB.SYSDEBUG(SAM1)

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
    310              ADD +1 TO NUM-CUSTOMER-RECS
    311         *    SUBROUTINE SAM2 WILL COLLECT CUSTOMER STATISTICS
    312              CALL 'SAM2' USING CUST-REC,
    313                      CUSTOMER-BALANCE-STATS
    314              MOVE CUST-ID             TO RPT-CUST-ID
    315              MOVE CUST-NAME           TO RPT-CUST-NAME
    316              MOVE CUST-OCCUPATION     TO RPT-CUST-OCCUPATION
```

c. Click the <u>Step Into</u> button.

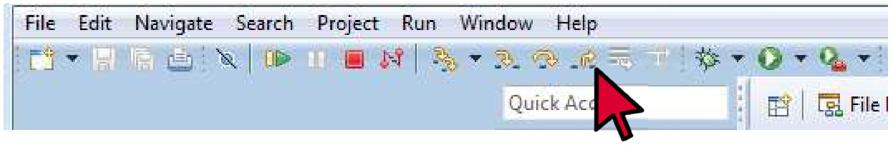


d. Notice that the debugger stepped into the subprogram, SAM2. Using 'Step into' at a CALL statement is one method you can use to follow program logic into a subprogram.

e. Click <u>Step Into</u> a few more times to get a few lines deeper into SAM2.

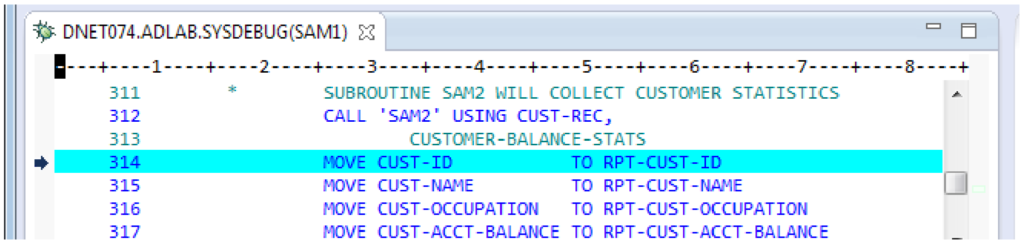
f. Note: An easy way to run to the completion of the subprogram is 'Step Return'.

g. **Click** the **<u>Step Return</u>** button in the tool bar.



h. Notice that the application ran until it returned to the calling program. The source window is positioned back in the main program, on the statement *after* the CALL to SAM2.

- Note: 'Step Return' runs the current subprogram to completion, and pauses the debugger after control has been returned to the calling program.


DNET074.ADLAB.SYSDEBUG(SAM1)

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
    311         *    SUBROUTINE SAM2 WILL COLLECT CUSTOMER STATISTICS
    312              CALL 'SAM2' USING CUST-REC,
    313                      CUSTOMER-BALANCE-STATS
    314              MOVE CUST-ID             TO RPT-CUST-ID
    315              MOVE CUST-NAME           TO RPT-CUST-NAME
    316              MOVE CUST-OCCUPATION     TO RPT-CUST-OCCUPATION
    317              MOVE CUST-ACCT-BALANCE   TO RPT-CUST-ACCT-BALANCE
```

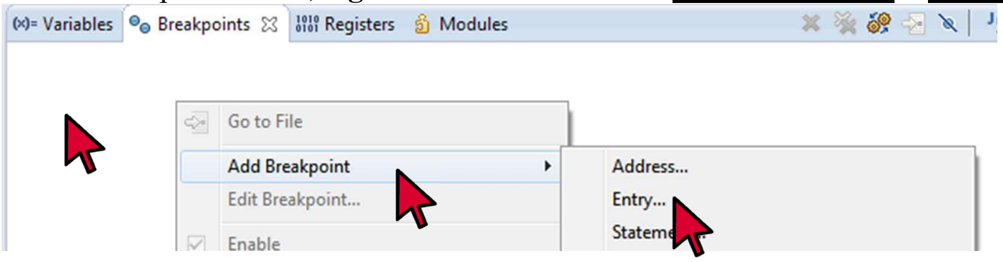
24. Instead of stepping into subprograms, a quicker way to run until reaching a specific subprogram is to use an 'entry' breakpoint.

a. Note: The SAM1 application calls SAM2 several times. Next, you will create an Entry breakpoint that will pause the next time program SAM2 is entered.

b. First, remove all existing breakpoints:

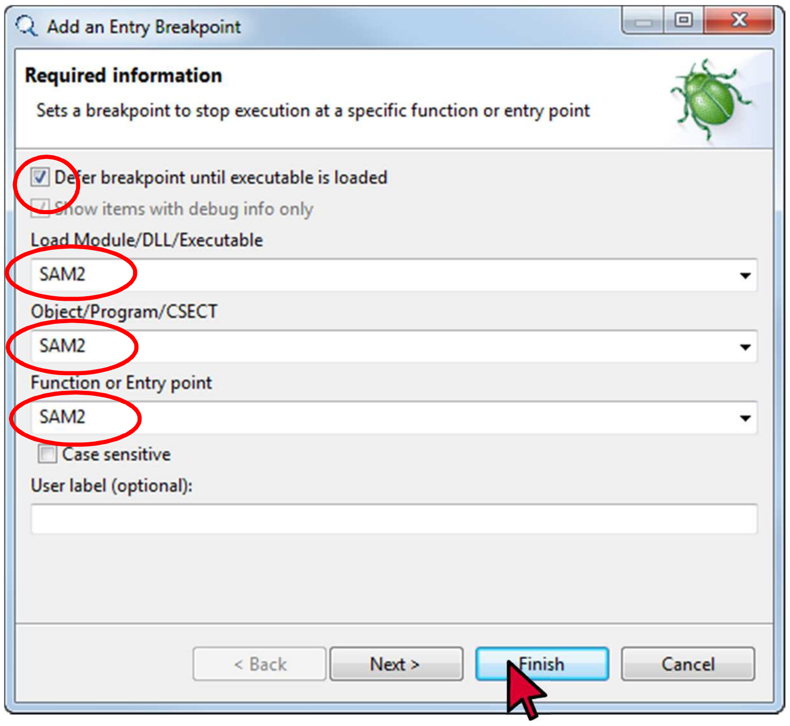
- **Click** the **Remove All Breakpoints** button in the breakpoints view, then click **Yes** at the prompt.

c. In the Breakpoints view, **right-click** and then select **Add Breakpoint** > **Entry**.



d. In the 'Add an Entry Breakpoint' dialog:

- **Click** to select the option: **Defer breakpoint until executable is loaded**.
- Type **SAM2** into each of the three fields (Load Module/DLL/Executable, Object/Program/CSECT, and Function/Entry Point). These will fully qualify the Entry breakpoint.
- Click **Finish**.

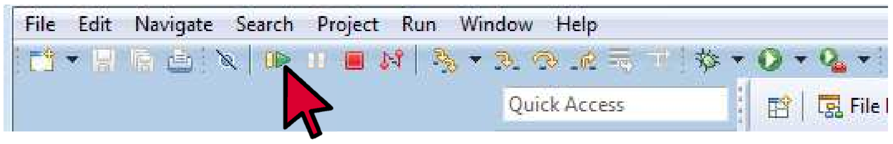


Sets a breakpoint to stop execution at a specific function or entry point

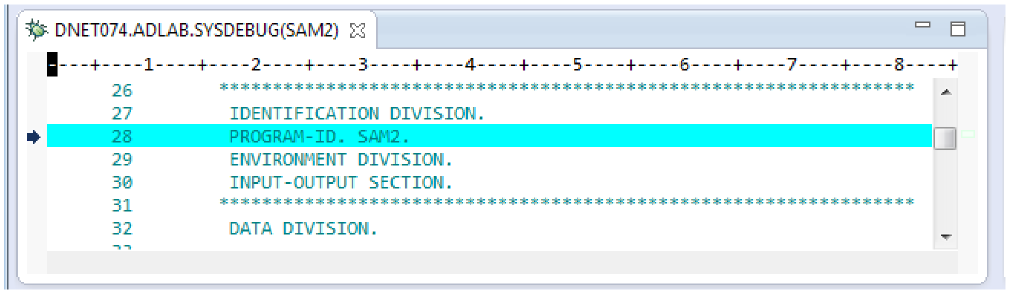
[✓] Defer breakpoint until executable is loaded
[✓] Show items with debug info only
Load Module/DLL/Executable: SAM2
Object/Program/CSECT: SAM2
Function or Entry point: SAM2
[ ] Case sensitive
User label (optional):

e. The Entry Breakpoint was added, and appears in the Breakpoints view.

f. **Click** the **Resume** button in the tool bar.

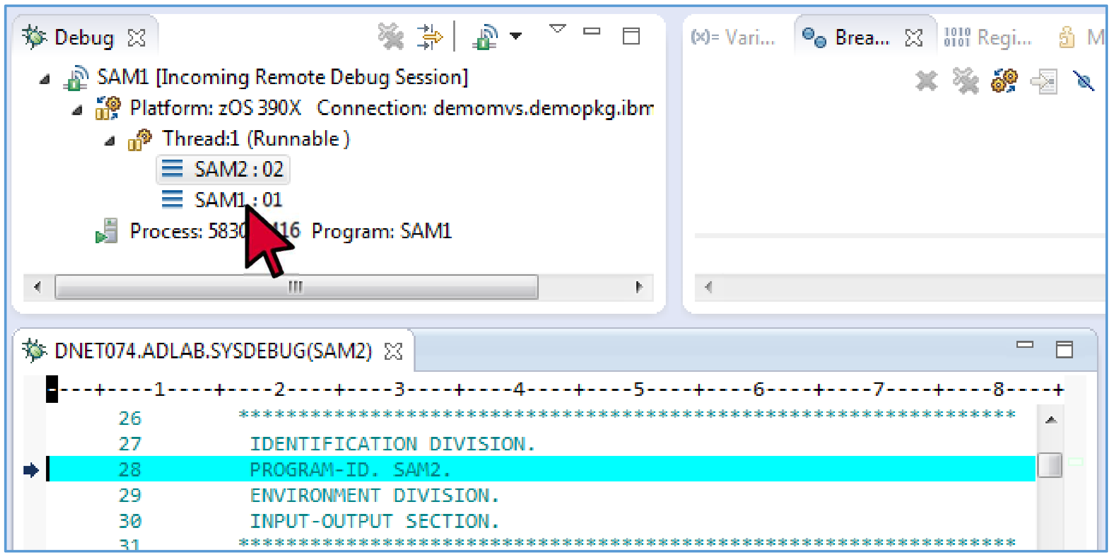


g. The application is paused at the beginning of SAM2. An Entry breakpoint provides an easy way to run the application until it reaches a specific program of interest.



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
       ******************************************************************
        IDENTIFICATION DIVISION.
        PROGRAM-ID.  SAM2.
        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
       ******************************************************************
        DATA DIVISION.
```

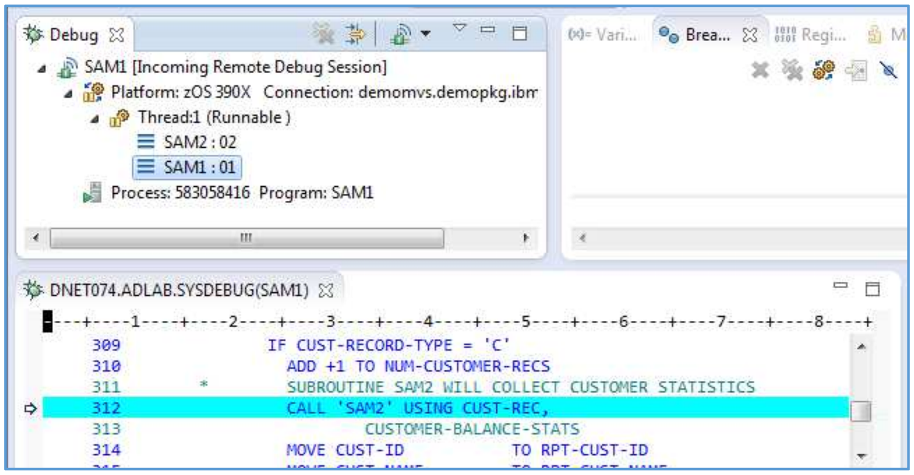
25. The Debug view can be used to display the source for different programs in the call chain (thread).
    a. Note: The application is currently running in the SAM2 program, and the source for SAM2 is displayed in the source window.
    b. Notice that the Debug view shows the current call chain. SAM1 is the main program, and SAM2 is a subprogram.
    c. Click **SAM1** in the Debug view.



```
SAM1 [Incoming Remote Debug Session]
  Platform: zOS 390X  Connection: demomvs.demopkg.ibm
    Thread:1 (Runnable )
      SAM2 : 02
      SAM1 : 01
  Process: 5830  16  Program: SAM1
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
       ******************************************************************
        IDENTIFICATION DIVISION.
        PROGRAM-ID.  SAM2.
        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
       ******************************************************************
```

d. The source view displays SAM1, even though SAM2 is still the active program.



```
SAM1 [Incoming Remote Debug Session]
  Platform: zOS 390X  Connection: demomvs.demopkg.ibm
    Thread:1 (Runnable )
      SAM2 : 02
      SAM1 : 01
  Process: 583058416  Program: SAM1
```

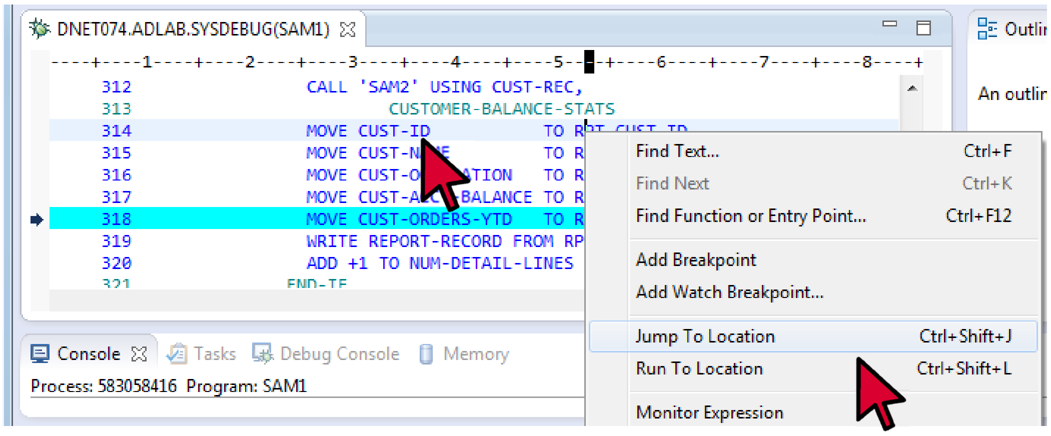
DNET074.ADLAB.SYSDEBUG(SAM1)

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
309               IF CUST-RECORD-TYPE = 'C'
310                 ADD +1 TO NUM-CUSTOMER-RECS
311        *        SUBROUTINE SAM2 WILL COLLECT CUSTOMER STATISTICS
312                 CALL 'SAM2' USING CUST-REC,
313                         CUSTOMER-BALANCE-STATS
314                 MOVE CUST-ID            TO RPT-CUST-ID
```

e. **Click SAM2** in the Debug view again to re-display SAM2 in the source view.
   - Note: The source view will re-display the active program automatically if you perform an action such as Step or Resume.

26. To prevent stopping again the next time SAM2 is called, remove the Entry breakpoint for program SAM2:
    a. In the breakpoints view, click (to select) the **Entry** breakpoint for SAM2.
    b. Then click the **Remove Selected Breakpoints** button.
    c. Notice that the entry breakpoint for SAM2 is deleted.
27. For the next step, SAM1 needs to be the active program.
    a. Click the **Step Into** button.
    b. If SAM2 is the active program (displayed in the Source window after a step), then click the **Step return** button now so the application will run until it returns to SAM1.
28. You can use 'Jump To Location' to alter program flow.
    a. Note: 'Jump To Location' allows you to reposition the execution pointer (the blue line) to another statement in the program logic. You can do this if you want to skip over certain statements, or if you want to go back and re-execute a section of logic. Use this feature carefully – if you jump to a location that is not logical for your program, your program may attempt to do illogical things!
    b. First, run to statement 318:
       - Click statement **318** in the source view to select it.
       - Then right-click statement **318** in the source view, then select **Run To Location**.
    c. The program is paused at statement 318.
    d. Next, jump backward in the program to statement 314:
       - **Click** statement **314** in the source view to select it.

- Then **right-click** statement **<u>314</u>** in the source view, then select **<u>Jump To Location</u>**.



```
----+----1----+----2----+----3----+----4----+----5---+----6----+----7----+----8----+
 312            CALL 'SAM2' USING CUST-REC,
 313                    CUSTOMER-BALANCE-STATS
 314            MOVE CUST-ID             TO RPT-CUST-ID
 315            MOVE CUST-N  E           TO R
 316            MOVE CUST-O    ATION     TO R
 317            MOVE CUST-A    BALANCE   TO R
 318            MOVE CUST-ORDERS-YTD     TO R
 319            WRITE REPORT-RECORD FROM RP
 320            ADD +1 TO NUM-DETAIL-LINES
 321          END-IF
```

Process: 583058416 Program: SAM1

e. Notice that the execution pointer (the long blue line) is repositioned to line 314.

f. The next Step or Resume will continue execution from line 314.

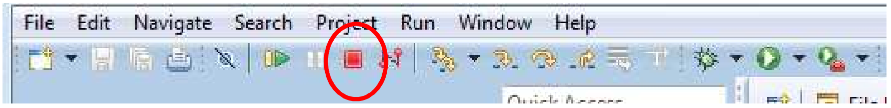
29. Ending the debug session.

a. Note: There are four ways to terminate a debugging session (don’t do any of these yet):

i. If the application runs to its normal completion, the debugger ends automatically when the application terminates itself.

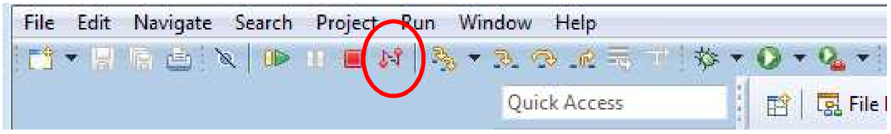
ii. The ‘Terminate’ button tells Debug Tool to immediately terminate the application with a zero return code, and also ends the current debugging session.

- The Terminate button:



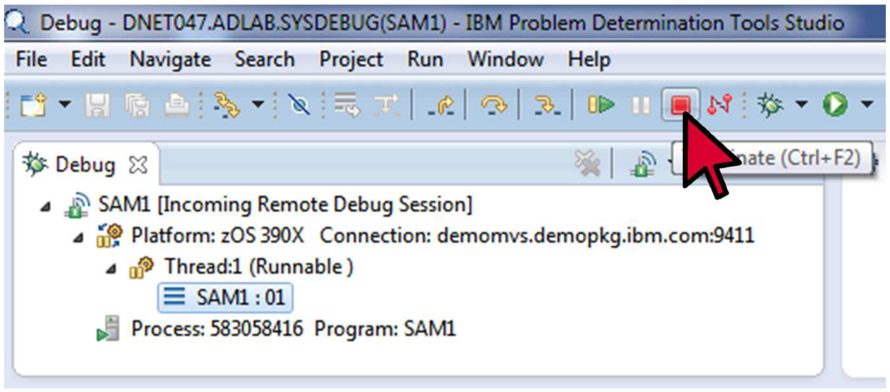
iii. The ‘Disconnect’ button tells Debug Tool to end the debugging session, but allows the application to continue running on its own.

- The Disconnect button:



iv. The ‘Terminate and Abend’ action (available from the Debug view context menu > Options) tells Debug Tool to abend the application, and also ends the current debugging session. This can be a good option if you want to collect a dump or produce a report from Fault Analyzer, or if you want to initiate a rollback of updates to DB2 or IMS databases.

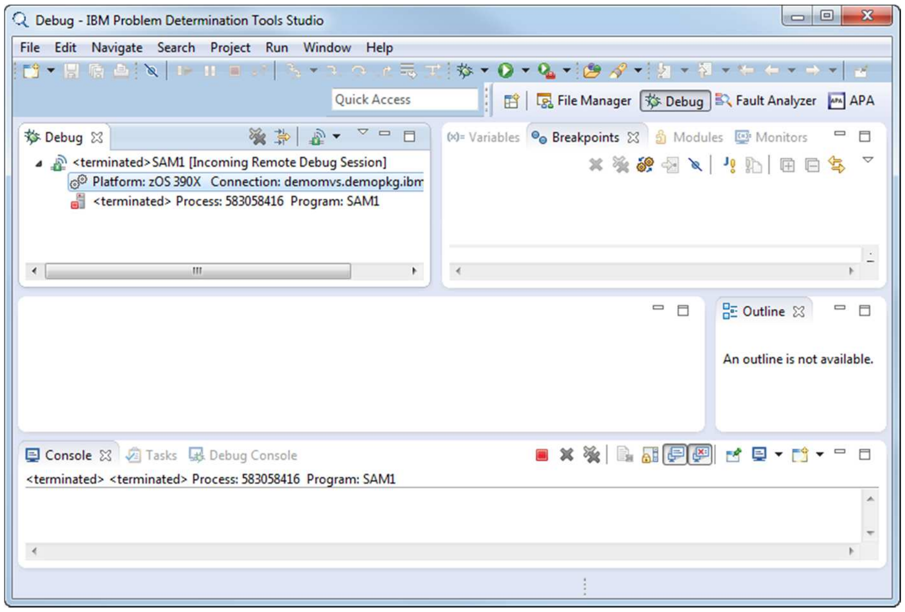
b. To terminate the debugging session and the application, click the **Terminate** button now:


Debug - DNET047.ADLAB.SYSDEBUG(SAM1) - IBM Problem Determination Tools Studio

SAM1 [Incoming Remote Debug Session]
Platform: zOS 390X Connection: demomvs.demopkg.ibm.com:9411
Thread:1 (Runnable)
SAM1 : 01
Process: 583058416 Program: SAM1

c. The application was terminated immediately, without running any more statements, with a zero return code.

d. The Debug Tool session ended.


Debug - IBM Problem Determination Tools Studio

<terminated>SAM1 [Incoming Remote Debug Session]
Platform: zOS 390X Connection: demomvs.demopkg.ibm
<terminated> Process: 583058416 Program: SAM1

An outline is not available.

<terminated> <terminated> Process: 583058416 Program: SAM1

e. The Debug Tool listener is still active, and it is ready to debug the next application.

Summary

Congratulations, you have completed the exercises and have familiarized yourself with the Debug Tool eclipse interface.

In this lab:

- You started the debugger with a batch application
- You learned how to set breakpoints, run to places of interest in a program, work with variables and other important debugging skills

If you have any questions about Debug Tool, please contact the instructor.