



This is the tutorial for IBM Debug Tool for z/OS[®], one of the IBM zSeries[®] problem determination tools.

Using Debug Tool's terminal interface

- Full-screen windows and navigation
- Using the debugger
 - Stepping through statements and running the program
 - Program statement breakpoints
 - Monitoring variables
 - Displaying variables in the log
 - Making breakpoints conditional
 - Variable change breakpoints
 - Program entry and exit breakpoints
 - Jumping to a statement
 - Ending the debugging session



This is the third of three sections that describes how to use the debugger.

This section will cover making breakpoints conditional, variable change breakpoints, program entry and exit breakpoints, jumping to a statement, and how to end the debugging session.

Specify a WHEN clause to set a conditional breakpoint



```
COBOL LOCATION: SAM1 -> 010.1
Command => AT 404 WHEN CUST-ID = '11004'; GO
MONITOR -----1-----2-----3-----4-----5----- LINE: 1 OF 168
***** TOP OF MONITOR *****
-----1-----2-----3-----4-----
0001 4 CUST-NAME          'Lynn, Viola'
0002 5 CUST-OCCUPATION   'Musician'
0003 6 01 CUST-REC
0004 02 CUST-KEY
0005 03 CUST-ID         '01001'
0006 03 CUST-RECORD-TYPE 'C'
SOURCE: SAM1 -----1-----2-----3-----4-----5----- LINE: 401 OF 467
401 730-READ-CUSTOMER-FILE.
402 READ CUSTOMER-FILE
403 AT END MOVE 'Y' TO WS-CUST-FILE-EOF .
404 EVALUATE WS-CUSTFILE-STATUS
405 WHEN '00'
406 WHEN '04'
LOG 0-----1-----2-----3-----4-----5----- LINE: 89 OF 91
0089 RPT-CUST-NAME = 'Lynn, Amanda'
0090 LIST ( CUST-OCCUPATION ) ;
0091 CUST-OCCUPATION = 'Musician'
PF 1:MONITOR 2:STEP 3:QUIT 4:LIST 5:FIND 6:R
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:R
```

Enter

3 IBM Debug Tool for z/OS tutorial © 2012 IBM Corporation

Earlier, you saw that an "AT" command can be used to set a breakpoint. For example, "AT 404" will set a breakpoint at statement 404, and the program would then stop every time it reaches that statement.

A breakpoint can be made conditional by specifying the "WHEN" option. In this example, the command: "AT 404 WHEN CUST-ID = '11004'" is entered. That will set a breakpoint at statement 404, but it will only stop if the condition is true. Notice that in this program, statement 404 comes after a statement that reads a record from a file. Setting a conditional breakpoint like this is a good way to allow the program to run until a specific data value is encountered.

A semi-colon, and "go" command is also specified to run the program, and Enter is pressed.

The program stopped the next time it reached statement 404 and the WHEN condition was true



```
COBOL LOCATION: SAM1 :> 404.1
Command ==> Scroll ==> PAGE
MONITOR +-----1-----2-----3-----4-----5----- LINE: 1 OF 167
-----1-----2-----3-----4-----
0001  4 CUST-NAME           'Ness, Luke'
0002  5 CUST-OCCUPATION    'Paranormal Investigator'
0003  6 01 CUST-REC
0004  02 CUST-KEY
0005  03 CUST-ID          '11004'
0006  03 CUST-RECORD-TYPE 'C'
0007  03 FILLER
SOURCE: SAM1 +-----1-----2-----3-----4-----5--- LINE: 401 OF 467
401  730-READ-CUSTOMER-FILE.
402  READ CUSTOMER-FILE
403  AT END MOVE 'Y' TO WS-CUST-FILE-EOF .
C 404  EVALUATE WS-CUSTFILE-STATUS
405  WHEN '00'
406  WHEN '04'
LOG 0-----1-----2-----3-----4-----5----- LINE: 91 OF 93
0091 CUST-OCCUPATION = 'Musician
0092 AT 404 WHEN CUST-ID = '11004' ;
0093 GO ;
PF 1:MONITOR  2:STEP  3:QUIT  4:LIST  5:FIND  6:PAUSE
PF 7:UP       8:DOWN  9:GO    10:ZOOM 11:ZOOM LOG 12:RECALL
Enter
```

A C line command clears a statement breakpoint

4

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

The program ran until it reached statement 404 and the condition was true. In this example, it reached the statement several times when the condition was not yet true, but did not stop until it was.

A "C" line command is typed on the statement, and Enter is pressed.

Result of the C line command The breakpoint was cleared



```
COBOL      LOCATION: SAM1  :> 404.1
Command ==>
Scroll ==> PAGE
MONITOR  +-----1-----2-----3-----4-----5----- LINE: 1 OF 167
          +-----1-----2-----3-----4-----
0001  4  CUST-NAME           'Ness, Luke'
0002  5  CUST-OCCUPATION     'Paranormal Investigator'
0003  6  01 CUST-REC
0004  02 CUST-KEY
0005  03 CUST-ID            '11004'
0006  03 CUST-RECORD-TYPE   'C'
0007  03 FILLER
SOURCE:   SAM1  +-----1-----2-----3-----4-----5----- LINE: 401 OF 467
401      730-READ-CUSTOMER-FILE.
402      READ CUSTOMER-FILE
403      AT END MOVE 'Y' TO WS-CUST-FILE-EOF .
404      EVALUATE WS-CUSTFILE-STATUS
405      WHEN '00'
406      WHEN '04'
LOG 0-----1-----2-----3-----4-----5----- LINE: 92 OF 94
0092  AT 404 WHEN CUST-ID = '11004' ;
0093  GO ;
0094  CLEAR AT 404 ;
PF  1: MONITOR   2: STEP      3: QUIT      4: LIST      5: FIND      6: AT/CLEAR
PF  7: UP        8: DOWN      9: GO       10: ZOOM     11: ZOOM LOG  12: RETRIEVE
```

5

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

The "C" line command cleared the statement breakpoint.

Specify a WHEN option to make a breakpoint conditional

- If a WHEN option is coded, it is evaluated each time the breakpoint is encountered
 - The breakpoint triggers only if the condition is true
- Breakpoint syntax with WHEN:
 - AT breakpoint-specification WHEN condition
 - Examples:
 - AT 200 WHEN CUSTOMER-ID = '12345'
 - AT 300 WHEN ACCOUNT-BALANCE > 1000

6

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

You can specify a "when" option to make a breakpoint conditional. The "when" condition is checked each time the breakpoint is encountered, but the program pauses only if the condition is true.

The syntax of a when option is: "AT" then the breakpoint specification, such as a statement number, then "when", followed by a simple condition.

Using Debug Tool's terminal interface

- Full-screen windows and navigation
- Using the debugger
 - Stepping through statements and running the program
 - Program statement breakpoints
 - Monitoring variables
 - Displaying variables in the log
 - Making breakpoints conditional
 - Variable change breakpoints
 - Program entry and exit breakpoints
 - Jumping to a statement
 - Ending the debugging session



Next, you will see how to set breakpoints to pause the program based on the value of a variable.

A Find command with the MONITOR option can be used to find a variable in the monitor



```

LOCATION:  STEPL  7  SUB  1
==> F NUM-CUSTFILE-RECS MON
Enter
                                Scroll ==> PAGE
                                LINE: 40 OF 167
-----1-----2-----3-----4-----
0040 02 NUM-CUSTFILE-RECS +000000071
0041 02 NUM-CUSTOMER-RECS +000000020
0042 02 NUM-PRODUCT-RECS +000000051
0043 02 NUM-DETAIL-LINES +000000020
0044 02 NUM-PRINT-REQUESTS +000000001
0045 02 NUM-PRINT-COMPLETED +000000000
0046 02 NUM-TOTALS-REQUESTS +000000000
SOURCE: SAM1 +-----1-----2-----3-----4-----5----- LINE: 401 OF 467
401 730-READ-CUSTOMER-FILE.
402 READ CUSTOMER-FILE
403 AT END MOVE 'Y' TO WS-CUST-FILE-EOF .
404 EVALUATE WS-CUSTFILE-STATUS
405 WHEN '00'
406 WHEN '04'
LOG 0-----1-----2-----3-----4-----5----- LINE: 92 OF 94
0092 AT 404 WHEN CUST-ID = '11004' ;
0093 GO ;
0094 CLEAR AT 404 ;
PF 1:MONITOR  2:STEP  3:QUIT  4:LIST  5:FIND  6:AT/CLEAR
PF 7:UP  8:DOWN  9:GO  10:ZOOM  11:ZOOM LOG  12:RETRIEVE
    
```

A "find" command with a "MONitor" option can be used to locate a monitored variable. Here, the monitor window is positioned so that the NUM-CUSTFILE-RECS variable can be seen.

An AT CHANGE variable command sets a change breakpoint



The screenshot displays the IBM Debug Tool interface. At the top, a command prompt shows the command `AT CHANGE NUM-CUSTFILE-RECS` entered, with a yellow box labeled "Enter" pointing to the cursor. Below this, a table of program variables is shown:

Address	Variable Name	Value
0040	02 NUM-CUSTFILE-RECS	+000000071
0041	02 NUM-CUSTOMER-RECS	+000000020
0042	02 NUM-PRODUCT-RECS	+000000051
0043	02 NUM-DETAIL-LINES	+000000020
0044	02 NUM-PRINT-REQUESTS	+000000001
0045	02 NUM-PRINT-COMPLETED	+000000000
0046	02 NUM-TOTALS-REQUESTS	+000000000

Below the table, the source code for the program is displayed. Line 404 is highlighted in red and contains the command `EVALUATE WS-CUSTFILE-STATUS`. The program execution log at the bottom shows the following commands:

```
LOG 0  
0093 GO ;  
0094 CLEAR AT 404 ;  
0095 AT CHANGE NUM-CUSTFILE-RECS ;
```

The log also shows function key shortcuts: PF 1: MONITOR, 2: STEP, 3: QUIT, 4: LIST, 5: FIND, 6: A, 7: UP, 8: DOWN, 9: GO, 10: ZOOM, 11: ZOOM LOG, 12: R. A yellow box labeled "F9" is positioned over the "9: GO" command.

A command in the format: "AT CHANGE variable-name" sets a change breakpoint that will trigger when the named variable changes value. This differs from a statement breakpoint, in that it could happen anywhere in the program, not just on a specific statement.

In this example, the command "AT CHANGE NUM-CUSTFILE-RECS" is entered to set the breakpoint, and then the F9 key is pressed to run the program.

Stopped at the change breakpoint for variable NUM-CUSTFILE-RECS



```

COBOL      LOCATION: SAM1  :> 408.1
Command ==>
Scroll ==> PAGE
MONITOR  +-----1-----2-----3-----4-----5----- LINE: 40 OF 166
-----1-----2-----3-----4-----
0040      02 NUM-CUSTFILE-RECS      +000000072
0041      02 NUM-CUSTOMER-RECS     +000000020
0042      02 NUM-PRODUCT-RECS     +000000051
0043      02 NUM-DETAIL-LINES     +000000020
0044      02 NUM-PRINT-REQUESTS   +000000001
0045      02 NUM-PRINT-COMPLETED +000000000
0046      02 NUM-TOTALS-REQUESTS  +000000000
SOURCE:   SAM1 +-----1-----2-----3-----4-----5----- LINE: 406 OF 467
406      WHEN '04'
407      ADD +1 TO NUM-CUSTFILE-RECS
408      CONTINUE
409      WHEN '10'
410      MOVE 'Y' TO WS-CUST-FILE-EOF
411      WHEN OTHER
LOG 0-----1-----2-----3-----4-----5----- LINE: 94 OF 96
0094     CLEAR AT 404 ;
0095     AT CHANGE NUM-CUSTFILE-RECS ;
0096     GO ;
PF 1:MONITOR  2:STEP  3:QUIT  4:LIST  5:FIND  6:AT/CLEAR
PF 7:UP      8:DOWN  9:GO   10:ZOOM 11:ZOOM LOG 12:RETRIEVE
    
```

A change breakpoint stops *after* the statement that changes the contents of the variable

The change breakpoint caused the program to pause when the variable changed value. The program is paused after the statement that caused the change. By default, a change breakpoint will stop when the named variable changes from any value to any other value.

A CLEAR AT CHANGE *variable* command clears a change breakpoint



Enter

```
CLE AT CHA NUM-CUSTFILE-RECS
```

Scroll ==> PAGE
LINE: 40 OF 166

0040	02	NUM-CUSTFILE-RECS	+000000072
0041	02	NUM-CUSTOMER-RECS	+000000020
0042	02	NUM-PRODUCT-RECS	+000000051
0043	02	NUM-DETAIL-LINES	+000000020
0044	02	NUM-PRINT-REQUESTS	+000000001
0045	02	NUM-PRINT-COMPLETED	+000000000
0046	02	NUM-TOTALS-REQUESTS	+000000000

SOURCE: SAM1 +-----1-----2-----3-----4-----5----- LINE: 406 OF 467

```
406          WHEN '04'  
407          ADD +1 TO NUM-CUSTFILE-RECS  
408          CONTINUE  
409          WHEN '10'  
410          MOVE 'Y' TO WS-CUST-FILE-EOF  
411          WHEN OTHER
```

LOG 0-----1-----2-----3-----4-----5----- LINE: 95 OF 97

```
0095 AT CHANGE NUM-CUSTFILE-RECS ;  
0096 GO ;  
0097 CLEAR AT CHANGE NUM-CUSTFILE-RECS ;  
PF 1:MONITOR  2:STEP  3:QUIT  4:LIST  5:FIND  6:AT/CLEAR  
PF 7:UP       8:DOWN  9:GO   10:ZOOM 11:ZOOM LOG 12:RETRIEVE
```

11 | IBM Debug Tool for z/OS tutorial | © 2012 IBM Corporation

You can use a "clear" command to clear a change breakpoint. The command: "clear at change num-custfile-recs" clears the change breakpoint for the named variable. You can set and clear change breakpoints for different variables.

- **AT CHANGE *variable-name***
 - Sets a change breakpoint
 - The program will stop when the variable's value changes with subsequent GO or RUNTO commands
 - Example:
 - AT CHANGE CUST-ID
- **AT CHANGE *variable-name* WHEN *condition***
 - Sets a conditional change breakpoint
 - The program will stop when the variable's value changes with GO or RUNTO commands, *and* the condition is true
 - Examples:
 - AT CHANGE CUST-ID WHEN CUST-ID = '12345'
 - AT CHANGE CUST-ID WHEN ACCT-BAL > 1000
- **CLEAR AT CHANGE *variable-name***
 - Clears a change breakpoint

Use the command "AT CHANGE *variable-name*" to set a change breakpoint. The breakpoint will trigger when the variable's value changes, regardless of where that happens in the program.

You can make a change breakpoint conditional by specifying a "when" option. With a "when" option, the program will pause when the variable's value changes, but only if the condition is true.

To remove a change breakpoint, use the command syntax "CLEAR AT CHANGE *variable-name*".

Performance tip



- For the best performance, use statement breakpoints instead of change breakpoints
 - A statement breakpoint will trigger only when the statement is reached
 - A change breakpoint makes the debugger check the value of the variable after every statement executes
- Use change breakpoints if you do not know which statement will change the variable
- But if you do, set a statement breakpoint after the statement that causes the change
 - For example:
 - AT 805 WHEN CUST-ID = '12345'
 - will provide much better performance than:
 - AT CHANGE CUST-ID WHEN CUST-ID = '12345'

13

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

There is a performance consideration when you use change breakpoints. A change breakpoint differs from a statement breakpoint, in that the debugger must check the value of the variable after every statement in the program runs. If you are debugging an especially large program, or a program that runs for a long time, you may want to consider using statement breakpoints instead of change breakpoints if you can.

If you do not know where in the program a target variable will be changed, then you may need to use a change breakpoint. But if you do know, you may be able to set a conditional statement breakpoint after the statement or statements that cause the change. That way, the debugger will only check the value of the variable at one specific place in the program, instead of after every statement, which is much more efficient.

Using Debug Tool's terminal interface

- Full-screen windows and navigation
- Using the debugger
 - Stepping through statements and running the program
 - Program statement breakpoints
 - Monitoring variables
 - Displaying variables in the log
 - Making breakpoints conditional
 - Variable change breakpoints
 - Program entry and exit breakpoints
 - Jumping to a statement
 - Ending the debugging session



So far, you have seen how to set breakpoints that trigger at specific statements, or when a variable's value changes. Next, you will see how to set other types of breakpoints that trigger when a specific program or subprogram is entered or exited.

Run to a subroutine call in the program



```
COBOL LOCATION: SAM1 :> 408.1
Command ==> Scroll ==> PAGE
MONITOR +-----1-----2-----3-----4-----5----- LINE: 40 OF 166
-----1-----2-----3-----4-----
0040 02 NUM-CUSTFILE-RECS +000000072
0041 02 NUM-CUSTOMER-RECS +000000020
0042 02 NUM-PRODUCT-RECS +000000051
0043 02 NUM-DETAIL-LINES +000000020
0044 02 NUM-PRINT-REQUESTS +000000001
0045 02 NUM-PRINT-COMPLETED +000000000
0046 02 NUM-TOTALS-REQUESTS +000000000
SOURCE: SAM1 +-----1-----2-----3-----4-----5----- LINE: 310 OF 467
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
LOG 0-----1-----2-----3-----4-----5----- LINE: 95 OF 97
0095 AT CHANGE NUM-CUSTFILE-RECS ;
0096 GO ;
0097 CLEAR AT CHANGE NUM-CUSTFILE-RECS ;
PF 1:MONITOR 2:STEP 3:QUIT 4:LIST 5:FIND 6:PAUSE 7:ENTER
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:PAGE
```

15

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation



But first, here is a simple example of how you follow the logic of an application from one program to another. In this example, a program named SAM1 is running. You can tell because the program name is displayed in the header, which is the very top line on the screen, and it is also displayed in the title line just above the source window.

An "R" line command is entered to run the program until it reaches statement 312.

You can STEP into a subprogram or function



```
COBOL LOCATION: SAM1 :> 312.1
Command ==> Scroll ==> PAGE
MONITOR +-----1-----2-----3-----4-----5----- LINE: 40 OF 182
-----1-----2-----3-----4-----
0040 02 NUM-CUSTFILE-RECS +000000072
0041 02 NUM-CUSTOMER-RECS +000000021
0042 02 NUM-PRODUCT-RECS +000000051
0043 02 NUM-DETAIL-LINES +000000020
0044 02 NUM-PRINT-REQUESTS +000000001
0045 02 NUM-PRINT-COMPLETED +000000000
0046 02 NUM-TOTALS-REQUESTS +000000000
SOURCE: SAM1 +-----1-----2-----3-----4-----5----- LINE: 310 OF 467
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
LOG 0-----1-----2-----3-----4-----5----- LINE: 96 OF 98
0096 GO ;
0097 CLEAR AT CHANGE NUM-CUSTFILE-RECS ;
0098 RUNTO 312 ;
PF 1:MONITOR 2:STEP 3:QUIT 4:LIST 5:FIND 6:PAGE
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:PAGE
```

F2

16 IBM Debug Tool for z/OS tutorial © 2012 IBM Corporation

The program ran to 312, which is a "CALL" statement that will pass control to another program named SAM2. It has not yet executed that statement.

To follow the logic into the subprogram, all you have to do is step into the "CALL" statement. Sitting on the "CALL", the F2 key is pressed to step.

After stepping into a subprogram



```
COBOL LOCATION: SAM2 ENTRY
Command ==>
Scroll ==> PAGE
MONITOR +---1---+---2---+---3---+---4---+---5---+--- LINE: 35 OF 40
-----1-----2-----3-----4-----
0035 02 BALANCE-COUNT +0000020.00
0036 02 BALANCE-TOT +0007908.55
0037 02 BALANCE-MIN +0000011.11
0038 02 BALANCE-MAX +0000067.68
0039 02 BALANCE-RANGE +0000056.57
0040 02 BALANCE-AVG +0000395.42
***** BOTTOM OF MONITOR *****
SOURCE: SAM2 +---1---+---2---+---3---+---4---+---5---+--- LINE: 28 OF 118
28 PROGRAM-ID. SAM2.
29 ENVIRONMENT DIVISION.
30 INPUT-OUTPUT SECTION.
31 *****
32 DATA DIVISION.
33
LOG 0 +---1---+---2---+---3---+---4---+---5---+--- LINE: 07 OF 99
0097 CLEAR AT CHANGE NUM-CUSTFILE-RECS ;
0098 RUNTO 312 ;
0099 STEP ;
PF 1: MONITOR 2: STEP 3: QUIT 4: LIST 5: FIND
PF 7: UP 8: DOWN 9: GO 10: ZOOM 11: ZOOM LOG 12: R
F2
```

17

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

That stepped into the subprogram, SAM2.

There are two cases when you are sitting on a "CALL" statement and step. Either the subprogram has been compiled for use with the debugger, or not. For example, in the case of an Enterprise COBOL subprogram, if it is compiled with the "TEST" compiler option, then it has been compiled for the debugger. If it has been compiled with "NOTEST", then it has not been compiled for use with the debugger.

In the example shown, the subprogram is compiled for use with the debugger, so the debugger stepped into it. If this program had not been compiled for the debugger, then the "STEP" command would run the subprogram, but the debugger would not pause in it. The debugger would pause at the next statement after the "STEP" statement in the higher level program.

In this example, the F2 key is pressed several times to step through statements in the subprogram.

You can STEP out of a subprogram or function



```
COBOL LOCATION: SAM2 :> 82.1
Command ==>
MONITOR +-----1-----2-----3-----4-----5-----6 LINE: 1 OF 24
-----1-----2-----3-----4-----
0001 4 CUST-NAME           'Ness, Luke'
0002 5 CUST-OCCUPATION    'Paranormal Investigator'
0003 6 01 CUST-REC
0004 02 CUST-KEY
0005 03 CUST-ID           '11004'
0006 03 CUST-RECORD-TYPE 'C'
0007 03 FILLER
SOURCE: SAM2 +-----1-----2-----3-----4-----5----- LINE: 78 OF 118
78          PERFORM 500-INIT-STATISTICS.
79          PERFORM 100-CALC-BALANCE-STATISTICS.
80          MOVE 'N' TO WS-FIRST-TIME-SW
81          MOVE 'PROGRAM ENDED' TO WS-PROGRAM-STATUS.
82          GOBACK.
83
LOG 0-----1-----2-----3-----4-----5----- LINE: 103 OF 105
0103 STEP ;
0104 STEP ;
0105 STEP ;
PF 1:MONITOR 2:STEP 3:QUIT 4:LIST 5:FIND 6:PAUSE
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:PAGE
```



After stepping many times, the last statement in the subprogram is reached. F2 is pressed to step again.

At the exit of the subprogram



```
COBOL LOCATION: SAM2 EXIT
Command ==>
MONITOR +-----1-----2-----3-----4-----5-----6 LINE: 1 OF 22
-----1-----2-----3-----4-----
0001  4 CUST-NAME           'Ness, Luke'
0002  5 CUST-OCCUPATION    'Paranormal Investigator'
0003  6 01 CUST-REC
0004  02 CUST-KEY
0005  03 CUST-ID          '11004'
0006  03 CUST-RECORD-TYPE 'C'
0007  03 FILLER
SOURCE: SAM2 +-----1-----2-----3-----4-----5----- LINE: 26 OF 118
26 *****
27 IDENTIFICATION DIVISION.
28 PROGRAM-ID. SAM2.
29 ENVIRONMENT DIVISION.
30 INPUT-OUTPUT SECTION.
31 *****
LOG 0 +-----1-----2-----3-----4-----5----- LINE: 104 OF 106
0104 STEP ;
0105 STEP ;
0106 STEP ;
PF 1: MONITOR  2: STEP  3: QUIT  4: LIST  5: FIND  6: PAGE
PF 7: UP       8: DOWN  9: GO    10: ZOOM 11: ZOOM LOG 12: PAGE
F2
```

Notice that the header indicates that the subprogram, SAM2, is exiting. F2 is pressed to step again.

Returned to the calling program



```
COBOL      LOCATION: SAM1 :> 314.1
Command ==>
Scroll ==> PAGE
MONITOR  +-----1-----2-----3-----4-----5----- LINE: 1 OF 165
          +-----1-----2-----3-----4-----
0001  4 CUST-NAME           'Ness, Luke'
0002  5 CUST-OCCUPATION     'Paranormal Investigator'
0003  6 01 CUST-REC
0004  02 CUST-KEY
0005  03 CUST-ID           '11004'
0006  03 CUST-RECORD-TYPE  'C'
0007  03 FILLER
SOURCE:   SAM1 +-----1-----2-----3-----4-----5--- LINE: 310 OF 467
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
LOG 0-----1-----2-----3-----4-----5----- LINE: 105 OF 107
0105  STEP ;
0106  STEP ;
0107  STEP ;
PF  1: MONITOR  2: STEP  3: QUIT  4: LIST  5: FIND  6: AT/CLEAR
PF  7: UP       8: DOWN  9: GO    10: ZOOM 11: ZOOM LOG 12: RETRIEVE
```

20

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

After stepping, the subprogram returned control back to the higher level program. The debugger paused at the next logical statement, which is the statement after the CALL.

Now you have seen one way to follow program logic into a subprogram that has been compiled for use with the debugger. You can step directly into it, and step out of it back to the calling program.

An AT ENTRY name command sets an entry breakpoint for a program or routine



```
COBOL LOCATION: SAM1 .> 314.1
Command == AT ENTRY SAM3; GO
MONITOR +-----1-----2-----3-----4-----5----- LINE: 1 OF 165
-----1-----2-----3-----4-----
0001 4 CUST-NAME           'Ness, Luke'
0002 5 CUST-OCCUPATION    'Paranormal Investigator'
0003 6 01 CUST-REC
0004 02 CUST-KEY
0005 03 CUST-ID          '11004'
0006 03 CUST-RECORD-TYPE 'C'
0007 03 FILLER
SOURCE: SAM1 +-----1-----2-----3-----4-----5--- LINE: 310 OF 467
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
LOG 0 +-----1-----2-----3-----4-----5----- LINE: 105 OF 107
0105 STEP ;
0106 STEP ;
0107 STEP ;
PF 1: MONITOR 2: STEP 3: QUIT 4: LIST 5: FIND 6: PAUSE
PF 7: UP 8: DOWN 9: GO 10: ZOOM 11: ZOOM LOG 12: PAGE
```



But in a complex application, it can be cumbersome to have to step into any subprogram that you want to debug. So there is a simpler way to pause when a specific subprogram is entered – an "entry" breakpoint.

The command "AT ENTRY SAM3" sets an entry breakpoint that will trigger when subprogram SAM3 is entered. SAM3 could be a program called by the current program, or it could be further down in the call chain. A semi-colon and a "GO" command is also typed into the command line, and Enter is pressed.

After stopping at an AT ENTRY breakpoint



```
COBOL LOCATION: SAM3 ENTRY
Command ==>
MONITOR +-----1-----2-----3-----4-----5----- LINE: 14 OF 19
-----+-----1-----2-----3-----4-----
0014 02 WS-WORK-NUM-5 +0000000
0015 ***** AUTOMONITOR *****
0016 There are no variables in the statement to display.
0017 ***** Previous Statement SAM1 ::> SAM1 :> 314.1 *****
0018 03 CUST-ID '11004'
0019 02 RPT-CUST-ID '11004'
***** BOTTOM OF MONITOR *****
SOURCE: SAM3 +-----1-----2-----3-----4-----5----- LINE: 21 OF 110
21 PROGRAM-ID, SAM3.
22 ENVIRONMENT DIVISION.
23 INPUT-OUTPUT SECTION.
24 *****
25 DATA DIVISION.
26
LOG 0-----1-----2-----3-----4-----5----- LINE: 107 OF 109
0107 STEP ;
0108 AT ENTRY SAM3 ;
0109 GO ;
PF 1: MONITOR 2: STEP 3: QUIT 4: LIST 5: FIND 6: AT/CLEAR
PF 7: UP 8: DOWN 9: GO 10: ZOOM 11: ZOOM LOG 12: RETRIEVE
```

The breakpoint triggered when the SAM3 program was entered. It is paused at the entry of the subprogram. An entry breakpoint is an easy way to run the application until it reaches to a specific program.

- **AT ENTRY *program-name***
 - Sets a program entry breakpoint that will stop when *program-name* is entered with subsequent GO or RUNTO commands
 - A frequently used command option:
 - **AT ENTRY *** sets an entry breakpoint for all programs
 - Considerations:
 - With most compilers, *program-name* must be compiled for debugging for the entry breakpoint to trigger
 - Exceptions include debugging in disassembly mode
 - If *program-name* is not known to the debugger (not yet loaded), the load module name is assumed to be the same as the program name
 - If that is not the case, fully qualify the program. For example:
 - **AT ENTRY *load-module-name* ::> *program-name***
- **CLEAR AT ENTRY *program-name***
 - Clears an entry breakpoint

Use an "AT ENTRY *program-name*" breakpoint to pause when the named program is entered. You can also use the command "AT ENTRY *" to stop when any subprogram is entered, regardless of the name.

With most compilers, the subprogram must be compiled for debugging for the entry breakpoint to trigger. An exception is when you are debugging in disassembly mode.

If the named program is not known to the debugger (meaning that it has not yet been called or loaded), then the load module name is assumed to be the same as the program name. If the program and load module have different names, then you must fully qualify the name, and the syntax is shown in the example.

Use a "CLEAR AT ENTRY *program-name*" command to clear an entry breakpoint.

- **AT EXIT *program-name***
 - Sets a program exit breakpoint
 - The program will stop when exiting *program-name* with GO or RUNTO commands
 - Considerations:
 - Same as for entry breakpoints
 - Tip: set an exit breakpoint on the main program for a last chance to examine variables before the application ends

- **CLEAR AT EXIT *program-name***
 - Clears an exit breakpoint

Exit breakpoints are similar to entry breakpoints, but they trigger when a program is being exited, instead of when it is entered.

The command "AT EXIT *program-name*" will set a breakpoint that will pause when the named program is exited. Consider setting an exit breakpoint on the main program, as this gives you a last chance to examine program variables before the application ends.

Use a "CLEAR AT EXIT *program-name*" command to clear an exit breakpoint.

Debug Tool training sections

Page 3 of 5



Using Debug Tool's terminal interface

- Full-screen windows and navigation
- Using the debugger
 - Stepping through statements and running the program
 - Program statement breakpoints
 - Monitoring variables
 - Displaying variables in the log
 - Making breakpoints conditional
 - Variable change breakpoints
 - Program entry and exit breakpoints
 - Jumping to a statement
 - Ending the debugging session



25

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

Next, you will see how to alter the flow of a program by jumping to a statement.

JUMPTO changes the program flow by passing control directly to another statement

```

COBOL LOCATION: SAM3 ;> 75.1
Command ==> JUMPTO 73 Scroll ==> CSR
MONITOR -----2-----3-----4-----5-----6 LINE: 1 OF 13
-----1-----2-----3-----4-----
0001 1 01 WS-FIELDS
0002 02 WS-PROGRAM-STATUS 'CALCULATING PRODUCT STATS'
0003 02 WS-FIRST-TIME-SW 'N'
0004 02 WS-WORK-NUM-1 +0000000
0005 02 WS-WORK-NUM-2 +0000000
0006 02 WS-WORK-NUM-3 +0000000
SOURCE: SAM3 +-----1-----2-----3-----4-----5----- LINE: 72 OF 110
72 PERFORM 500-INIT-STATISTICS.
73 PERFORM 100-CALC-PRODUCT-STATISTICS.
74 MOVE 'N' TO WS-FIRST-TIME-SW
75 MOVE 'PROGRAM ENDED' TO WS-PROGRAM-STATUS.
76 GOBACK.
77
LOG 0-----1-----2-----3-----4-----5----- LINE: 112 OF 115
0112 STEP ;
0113 STEP ;
0114 STEP ;
0115 STEP ;
PF 1: ? 2: STEP 3: QUIT 4: LIST 5: FIND 6: A
PF 7: UP 8: DOWN 9: GO 10: ZOOM 11: ZOOM LOG 12: R
    
```

All other statements
are skipped.

Enter

In this example, statement seventy-five is the current statement, and should be the next to run. A "jumpto 73" command is entered to pass control directly to statement seventy-three.

Result of JUMPTO Statement 73 will be the next to run

COBOL LOCATION: SAM3 :> 73.1
 Command ==> █ Scroll ==> CSR
 MONITOR +-----1-----2-----3-----4-----5-----6 LINE: 1 OF 13
 +-----1-----2-----3-----4-----

```

0001 1 01 WS-FIELDS
0002 02 WS-PROGRAM-STATUS 'CALCULATING PRODUCT STATS'
0003 02 WS-FIRST-TIME-SW 'N'
0004 02 WS-WORK-NUM-1 +0000000
0005 02 WS-WORK-NUM-2 +0000000
0006 02 WS-WORK-NUM-3 +0000000
SOURCE: SAM3 +-----1-----2-----3-----4-----5-----6 LINE: 72 OF 110
72 PERFORM 500-INIT-STATISTICS.
73 PERFORM 100-CALC-PRODUCT-STATISTICS.
74 MOVE 'N' TO WS-FIRST-TIME-SW
75 MOVE 'PROGRAM ENDED' TO WS-PROGRAM-STATUS.
76 GOBACK.
77
LOG 0-----1-----2-----3-----4-----5-----6 LINE: 113 OF 116
0113 STEP ;
0114 STEP ;
0115 STEP ;
0116 JUMPTO 73 ;
PF 1: ? 2: STEP 3: QUIT 4: LIST 5: FIND 6: A
PF 7: UP 8: DOWN 9: GO 10: ZOOM 11: ZOOM LOG 12: R
  
```

JUMPTO passes control directly to a statement. All other statements are skipped. No data is changed.

F2

27

IBM Debug Tool for z/OS tutorial © 2012 IBM Corporation

After entering the jumpto command, no statements ran. But now seventy-three is the current statement, and will be the next to run. A "jumpto" command does not change any variable values. The "step" function key is pressed.

After a step



```
COBOL      LOCATION: SAM3  :> 79.1
Command ==>
MONITOR  +-----1-----2-----3-----4-----5-----6 LINE: 1 OF 13
          +-----1-----2-----3-----4-----
0001  1 01 WS-FIELDS
0002  02 WS-PROGRAM-STATUS      'CALCULATING PRODUCT STATS'
0003  02 WS-FIRST-TIME-SW      'N'
0004  02 WS-WORK-NUM-1          +0000000
0005  02 WS-WORK-NUM-2          +0000000
0006  02 WS-WORK-NUM-3          +0000000
SOURCE:  SAM3  +-----1-----2-----3-----4-----5----- LINE: 77 OF 110
77
78      100-CALC-PRODUCT-STATISTICS.
79      MOVE 'CALCULATING PRODUCT STATS' TO WS-PROGRAM-STATUS.
80      * *** Increment Record Count ***
81      ADD +1 TO SERV-CALLS-COUNT
82      * *** Add this customer's SERV-CALL to the grand total ***
LOG 0- +-----1-----2-----3-----4-----5----- LINE: 114 OF 117
0114  STEP ;
0115  STEP ;
0116  JUMPTO 73 ;
0117  STEP ;
PF 1: ?      2: STEP      3: QUIT      4: LIST      5: FIND      6: AT/CLEAR
PF 7: UP      8: DOWN      9: GO       10: ZOOM     11: ZOOM LOG  12: RETRIEVE
```

28

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

The statement that was "jumped" to ran, and the program will continue from this point.

- JUMPTO *statement-number*
 - Jump to (pass control to) a statement and stop there
 - All other statements are skipped
 - No data is changed
 - *statement-number* will be the next to run
- GOTO *statement-number*
 - Equivalent to: JUMPTO *statement-number*; GO
 - Jumps to *statement-number* and runs
- Consider using JUMPTO or GOTO:
 - To re-run a block of statements after modifying variables
 - To pass control out of a loop or procedure
 - To skip statements
- These commands are not available with some compilers when optimization is used

Use a "jumpto" command to alter the flow of a program. "Jumpto" passes control directly to the statement number you specify, and stops there. All other statements are skipped, and no data is changed..

Another, similar command is "goto". It is the same as a jumpto command followed by a go. The difference is that "jumpto" will wait for you to continue running the program after the jump, whereas "goto" will not wait. The program continues running immediately. With a "goto", the jump is done, but the next thing the debugger will display is the next breakpoint that is encountered, wherever that is.

Consider using a "jumpto" or "goto" to back up and re-run a block of statements after you have modified variable values. That let's you try some "what-if" scenarios with different values through the same area of code. It can also be used to pass control out of a loop or procedure, or to skip statements. But be careful. You can jump to a statement that does not make logical sense, which can result in logic errors or even abends.

Be aware that these commands are not available with certain compilers when the compiler's optimization options are turned on.

Using Debug Tool's terminal interface

- Full-screen windows and navigation
- Using the debugger
 - Stepping through statements and running the program
 - Program statement breakpoints
 - Monitoring variables
 - Displaying variables in the log
 - Making breakpoints conditional
 - Variable change breakpoints
 - Program entry and exit breakpoints
 - Jumping to a statement
 - Ending the debugging session



Next, you will see options available to end a debugging session.

A QUIT command ends the debugging session



```
COBOL LOCATION: SAM3 :> 70.1
Command ==> QUIT
MONITOR -----1-----2-----3-----4-----5----- LINE: 14 OF 18
-----1-----2-----3-----4-----
0014 02 WS-WORK-NUM-5 +0000000
0015 ***** AUTOMONITOR SAM3 ::> SAM3 :> 70.1 *****
0016 02 WS-PROGRAM-STATUS 'PROGRAM ENDED'
0017 ***** AUTOMONITOR - PREVIOUS *****
0018 There are no variables in the statement to display.
***** BOTTOM OF MONITOR *****

SOURCE: SAM3 +---1---+---2---+---3---+---4---+---5--- LINE: 68 OF 110
68
69 000-MAIN.
70 MOVE 'PROGRAM STARTED' TO WS-PROGRAM-STATUS.
71 IF WS-FIRST-TIME-SW = 'Y'
72 PERFORM 500-INIT-STATISTICS.
73 PERFORM 100-CALC-PRODUCT-STATISTICS.

LOG 0-----1-----2-----3-----4-----5----- LINE: 108 OF 110
0108 AT ENTRY SAM3 ;
0109 GO ;
0110 STEP ;
PF 1:MONITOR 2:STEP 3:QUIT 4:LIST 5:FIND 6:A
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:R
```

Or use F3

Enter

If you are debugging a program, and you issue a GO command, it will pause when the next breakpoint is triggered. However, if you issue a GO command but the program does not reach or trigger any of your breakpoints, it will run to termination. That is one way to end your debugging session - just run the program until it is finished. The application will end, and the debugging session will be cleared from your terminal.

However, you can end a debugging session at any time by typing "QUIT" on the command line, and pressing Enter.

Enter Y at the prompt to terminate the session



```
COBOL LOCATION: SAM3 :> 70.1
Command ==> Scroll ==> PAGE
*****
Do you really want to terminate this session? Y
Enter Y for YES and N for NO
*****
0018 There are no variables in the statement to display.
***** BOTTOM OF MONITOR *****
SOURCE: SAM3 +---1---+---2---+---3---+---4---+---5--- LINE: 68 OF 110
68
69 000-MAIN.
70 MOVE 'PROGRAM STARTED' TO WS-PROGRAM-STATUS.
71 IF WS-FIRST-TIME-SW = 'Y'
72 PERFORM 500-INIT-STATISTICS.
73 PERFORM 100-CALC-PRODUCT-STATISTICS.
LOG 0 +---1---+---2---+---3---+---4---+---5--- LINE: 109 OF 111
0109 GO ;
0110 STEP ;
0111 QUIT ;
PF 1:MONITOR 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CL/EP
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:RETRIEVE
```

32

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

You receive the prompt: “Do you really want to terminate this session?”. If you enter Y for Yes and press Enter, that will immediately terminate the program at its current location with a zero return code.

The debugging session ended



```
DEBUG TOOL TERMINAL INTERFACE MANAGER

EQAY001I Session manager escape ==> █

EQAY001I Terminal TRMDT007 connected for user TSS16
EQAY001I Ready for Debug Tool

PF3=EXIT PF10=Edit LE options data set PF12=LOGOFF
```

If you are using the Debug Tool terminal interface manager (TIM), this screen is re-displayed after the debugging session ends

The application terminated, and the debugging session ended. The program was stopped, and no more statements ran.

Quit the debugging session



- **QUIT** or the default **F3** key
 - Displays the "Do you really want to terminate..." prompt
 - If Y is entered, the program is terminated at the current location with a zero return code
- **QQ**
 - Same as QUIT, but without the prompt
- Frequently used command options:
 - **QUIT ABEND**
 - Terminates the program at the current location and forces an abend
 - Consider using this option to:
 - Capture the abend with IBM Fault Analyzer for z/OS (if installed)
 - avoid running subsequent steps in a multi-step job
 - roll back database updates
 - **QUIT DEBUG**
 - Disconnects the debugger and allows the program to run normally

34

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

To terminate an application immediately, use a "QUIT" command or the F3 key. If you want to avoid the prompt, enter a "QQ" command instead. And there are a couple of other options.

Use a "QUIT ABEND" command to terminate the program at its current location with an abend. You might do that if you want the system to collect a dump. If you have IBM's Fault Analyzer product, perhaps you plan to use it to see a detailed analysis.

A "QUIT DEBUG" command, however, does something very different. It allows your application to continue running without the debugger. The debugging engine is disconnected from the application, and the application is released to run on its own.

At this point in the tutorials, you have seen the basic commands and techniques you need to know to debug a program. That is the end of this section, using Debug Tool's terminal interface.

Feedback



Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_DTv12s14UsingTheDebuggerPart3.ppt

This module is also available in PDF format at: [../DTv12s14UsingTheDebuggerPart3.pdf](http://DTv12s14UsingTheDebuggerPart3.pdf)

35

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, z/OS, and zSeries are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at <http://www.ibm.com/legal/copytrade.shtml>

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION 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 ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.

© Copyright International Business Machines Corporation 2012. All rights reserved.