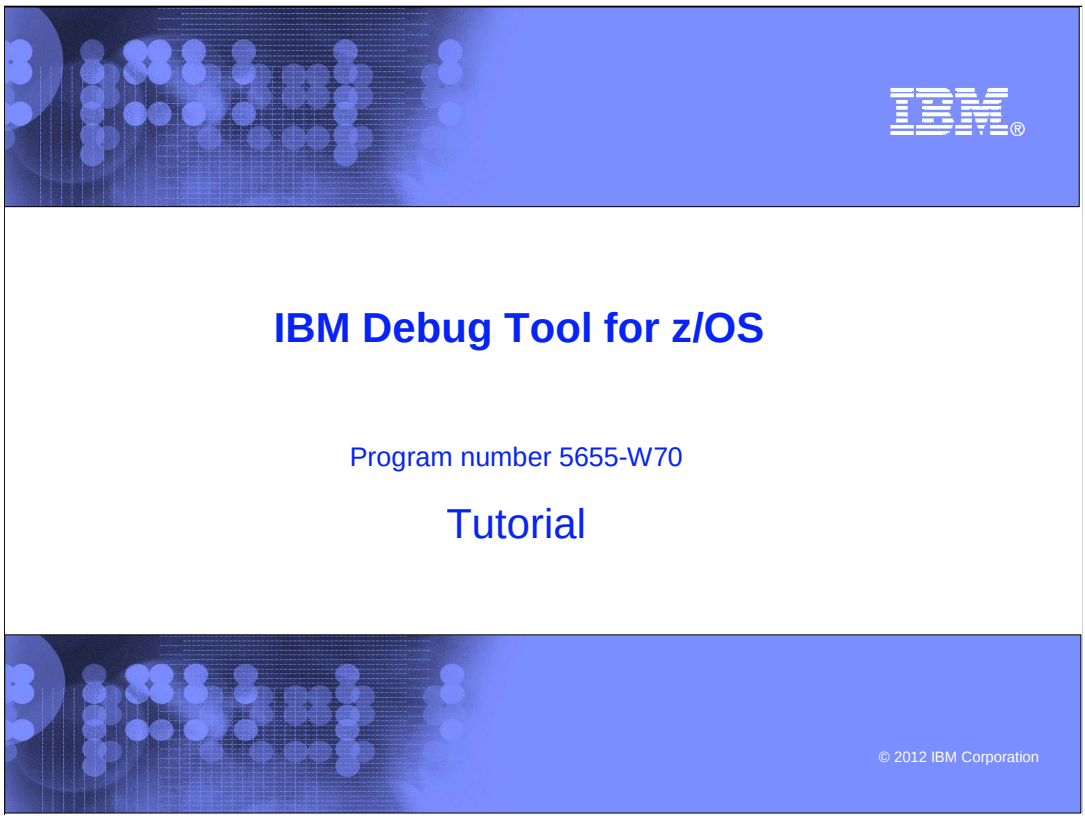
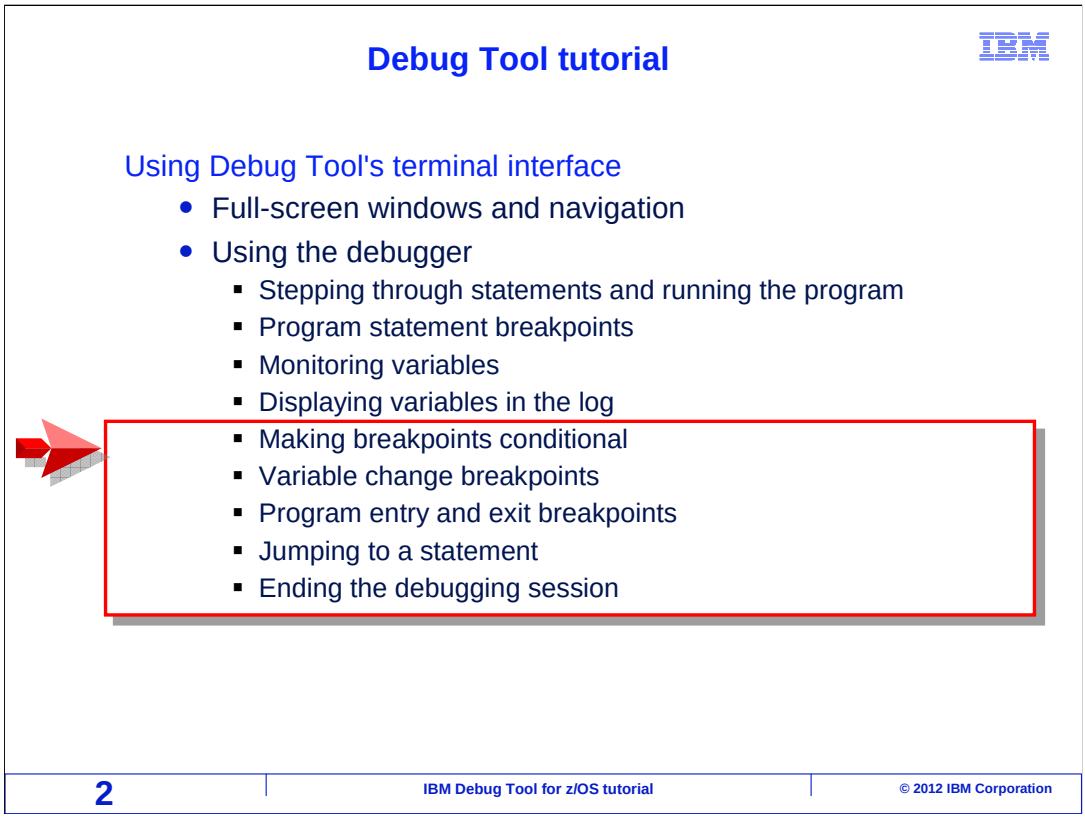


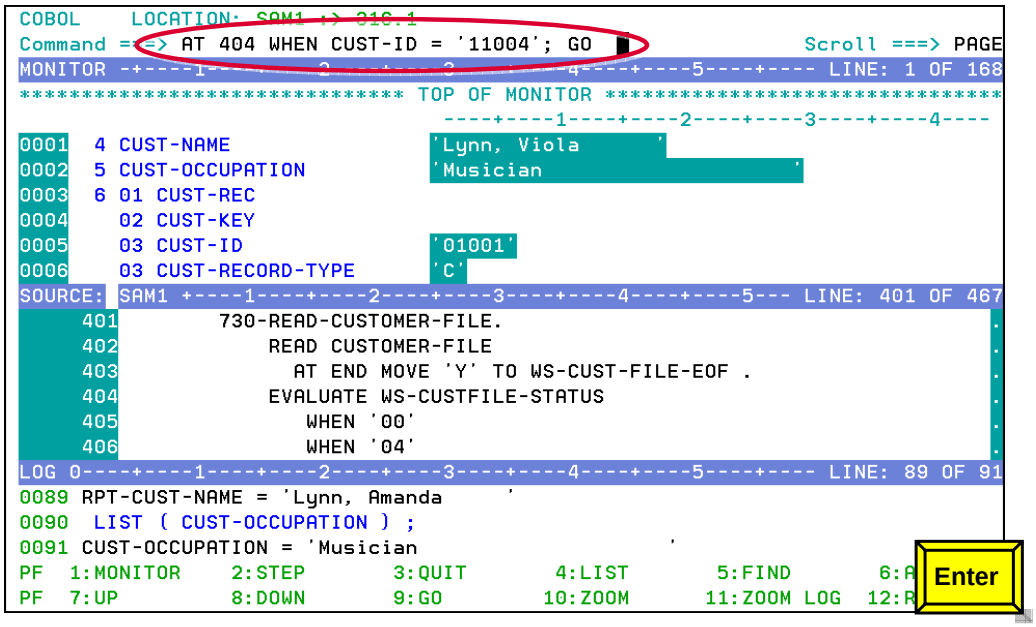
This is the tutorial for IBM Debug Tool for z/OS®, one of the IBM zSeries® problem determination tools.



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.

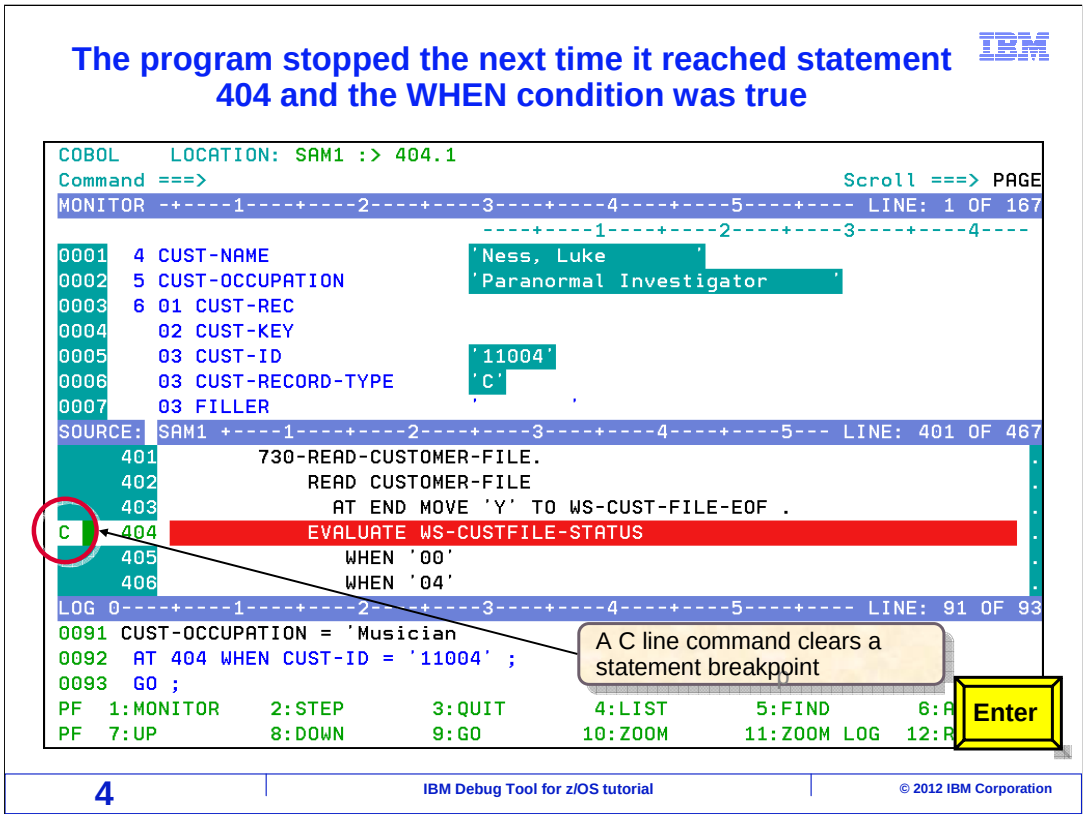
Specify a WHEN clause to set a conditional breakpoint



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.

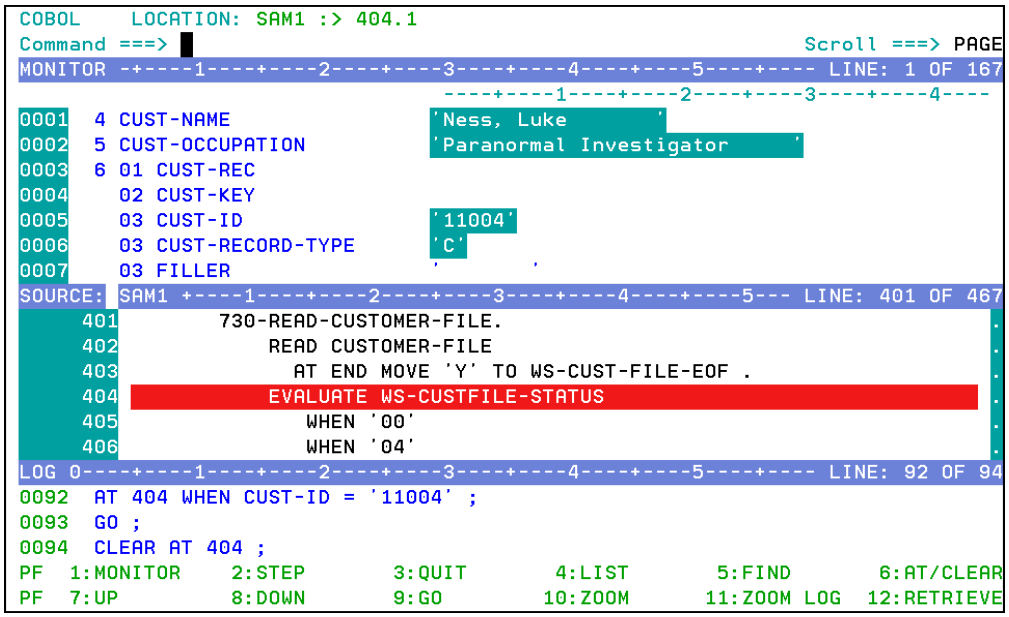
A semi-colon, and "go" command is also specified to the run the program, and Enter is pressed.



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



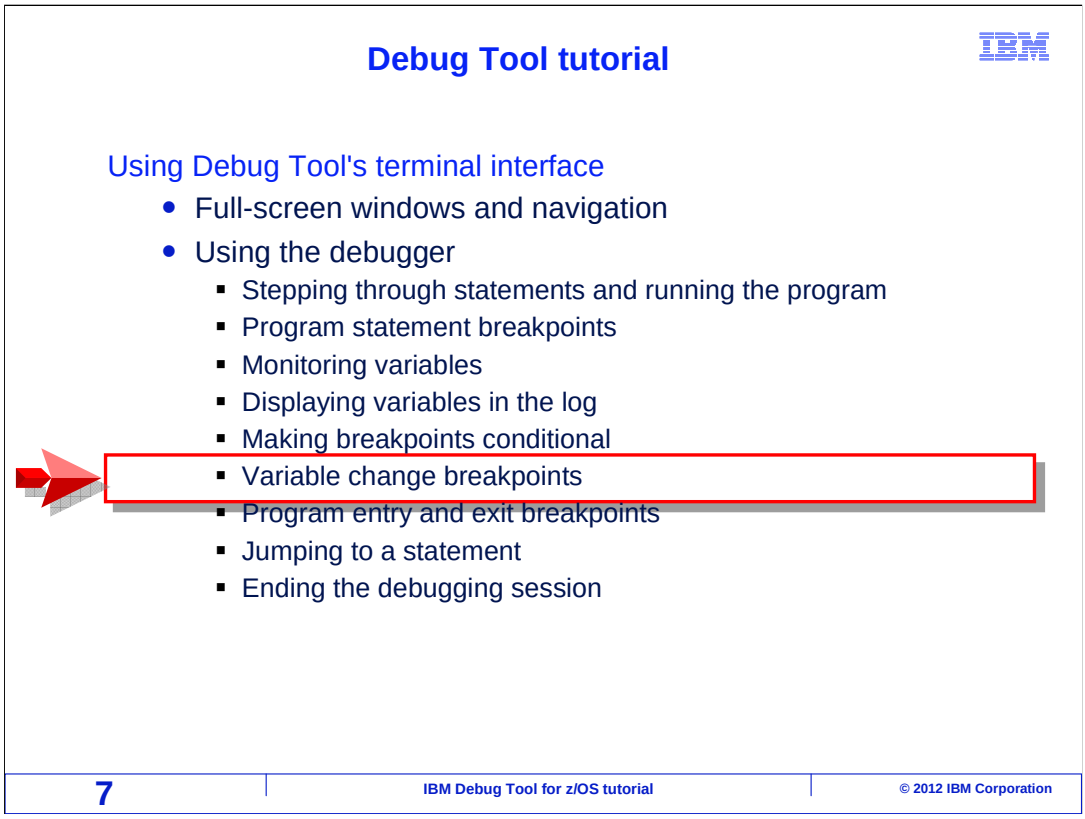
The "C" line command cleared the statement breakpoint.

Specify a WHEN option to make a breakpoint conditional

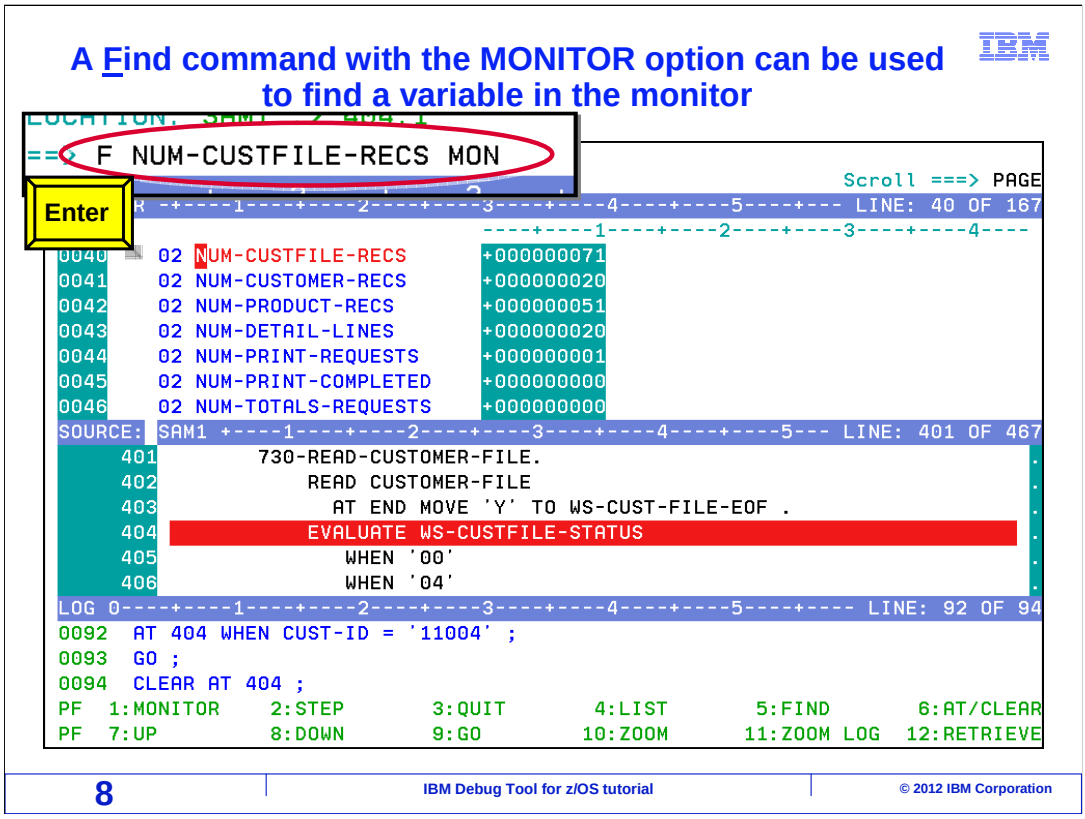
- If a WHEN option is coded, it is evaluated each time the breakpoint is encountered
  - The breakpoint triggers only if the condition is true
- Breakpoint syntax with WHEN:
  - **AT** *breakpoint-specification* **WHEN** *condition*
  - Examples:
    - AT 200 WHEN CUSTOMER-ID = '12345'
    - AT 300 WHEN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

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.

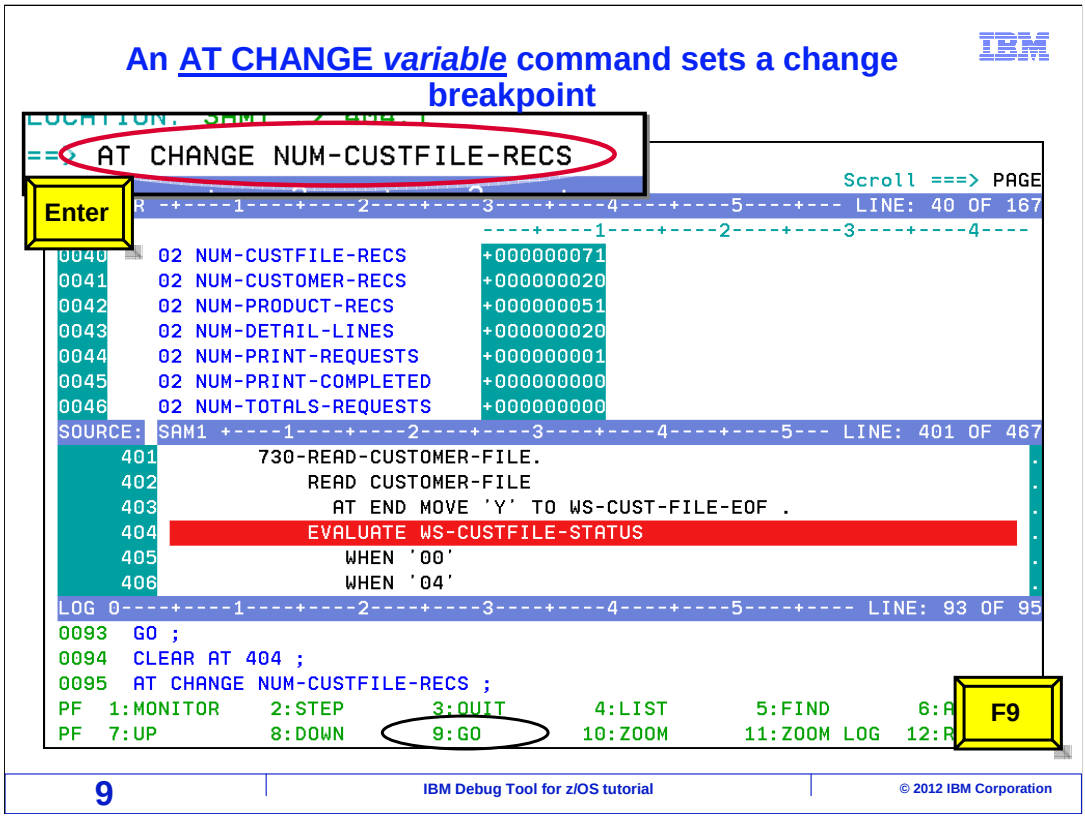
The syntax of a when option is: "AT" then the breakpoint specification, such as a statement number, then "when", followed by a simple condition.



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



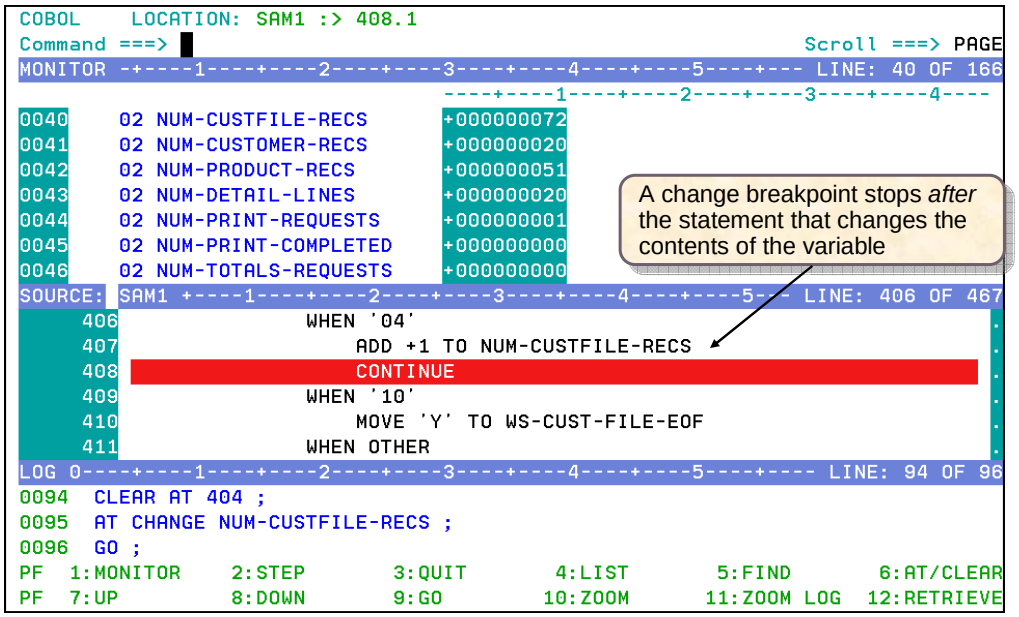
A "find" command with a "MONitor" option can be used to locate a monitored variable. Here, the monitor window is positioned so that the NUM-CUSTFILE-RECS variable can be seen.



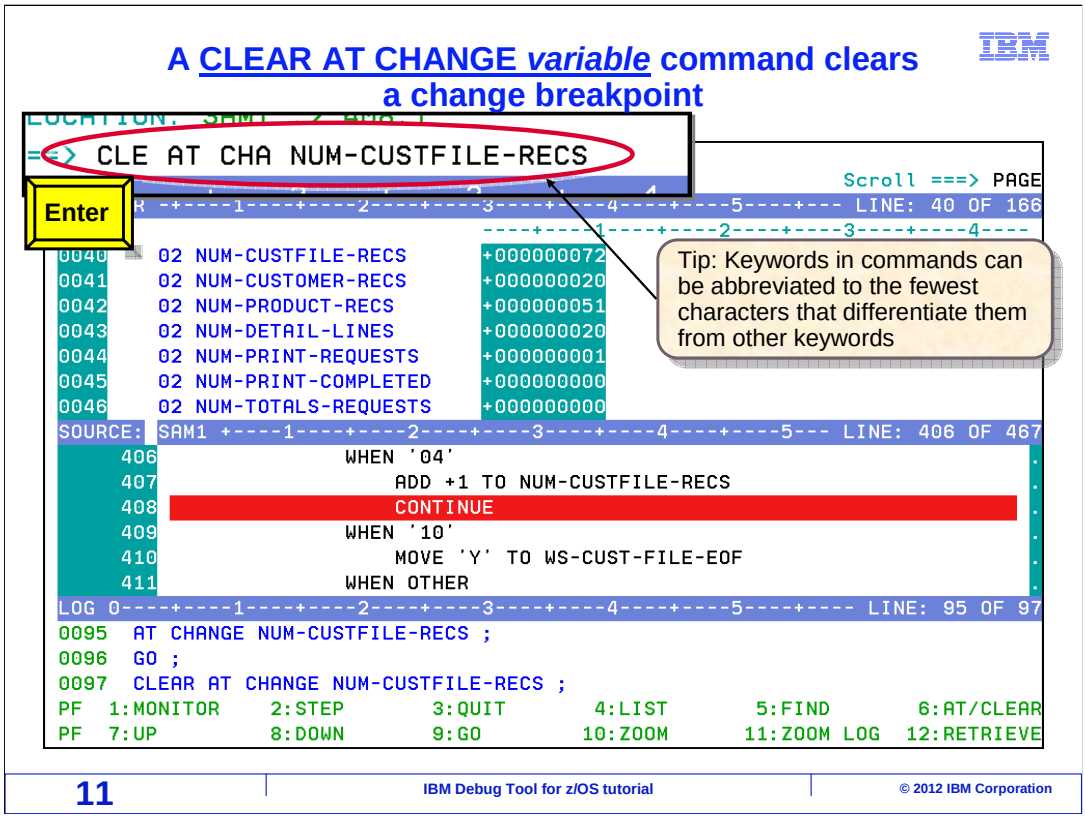
A command in the format: "AT CHANGE variable-name" sets a change breakpoint that will trigger when the named variable changes value. This differs from a statement breakpoint, in that it could happen anywhere in the program, not just on a specific statement.

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



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.



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.

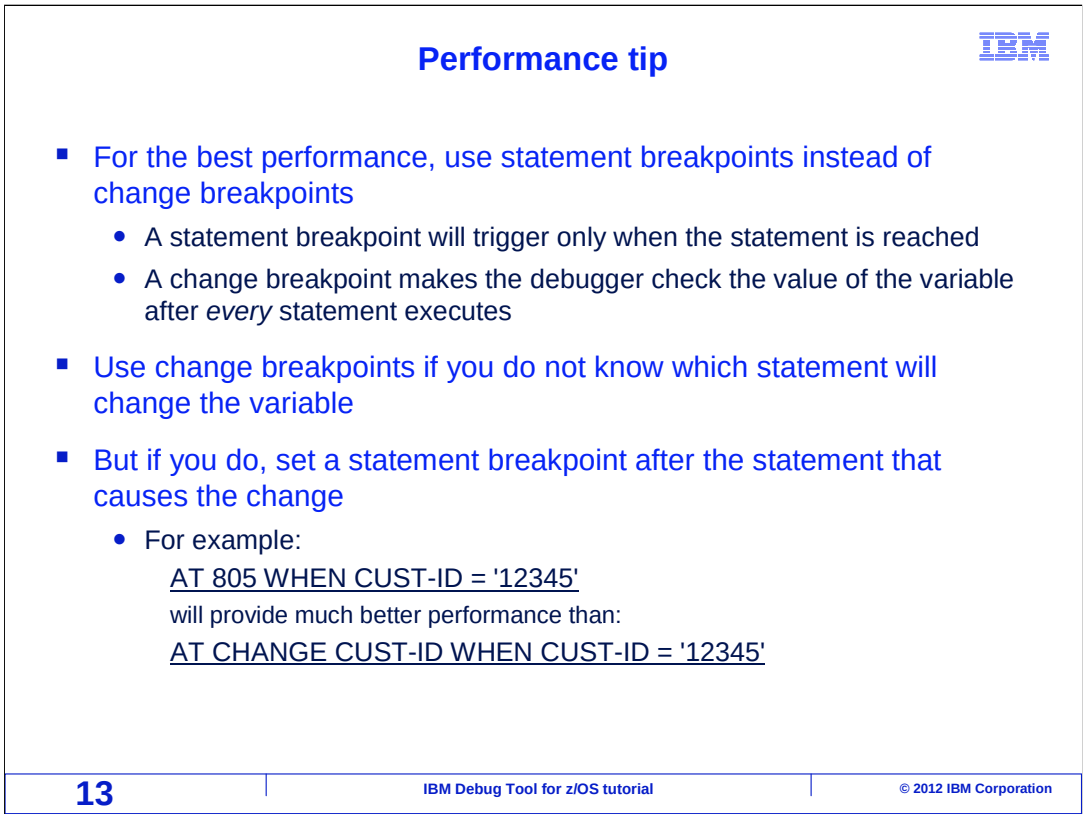
Change breakpoints

- **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.

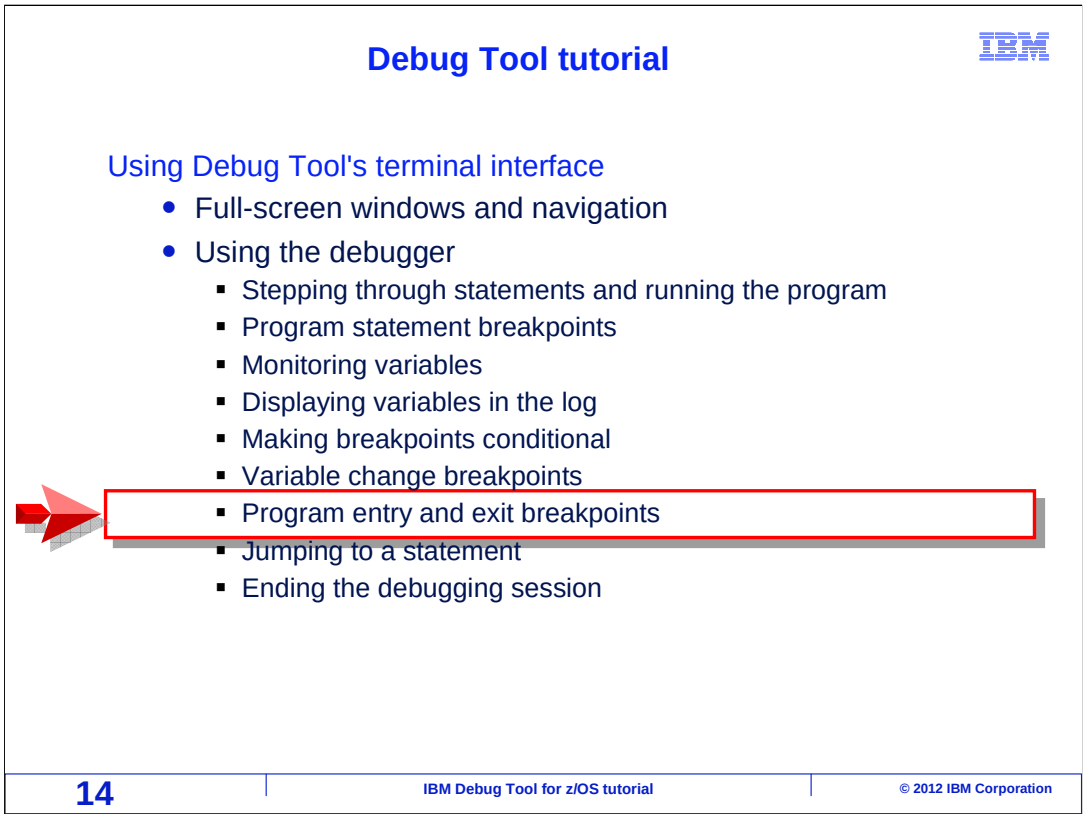
You can make a change breakpoint conditional by specifying a "when" option. With a "when" option, the program will pause when the variable's value changes, but only if the condition is true.

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



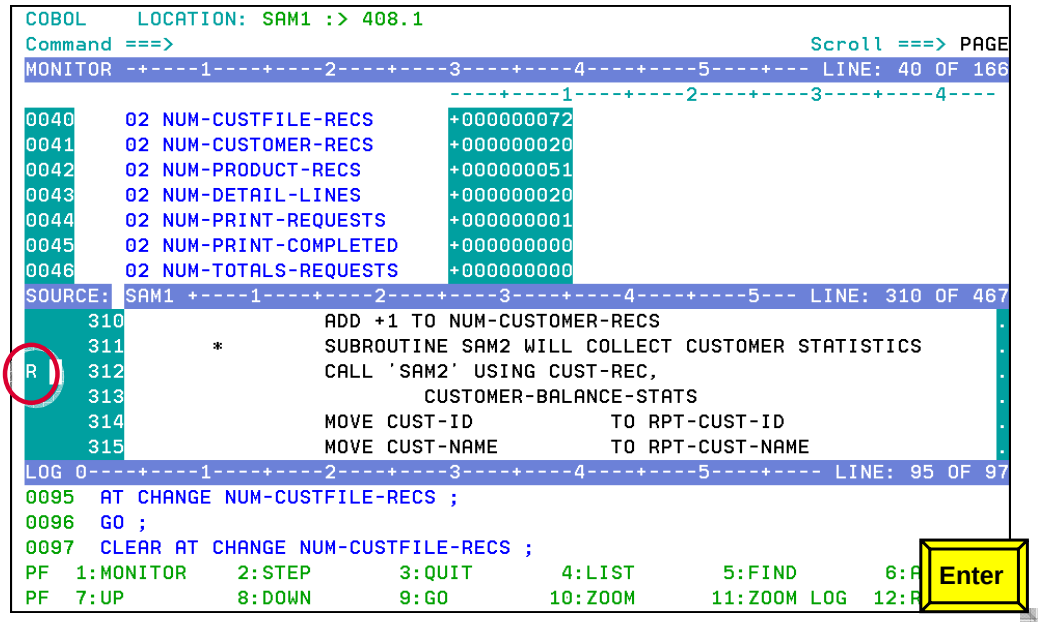
There is a performance consideration when you use change breakpoints. A change breakpoint differs from a statement breakpoint, in that the debugger must check the value of the variable after every statement in the program runs. If you are debugging an especially large program, or a program that runs 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.



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.

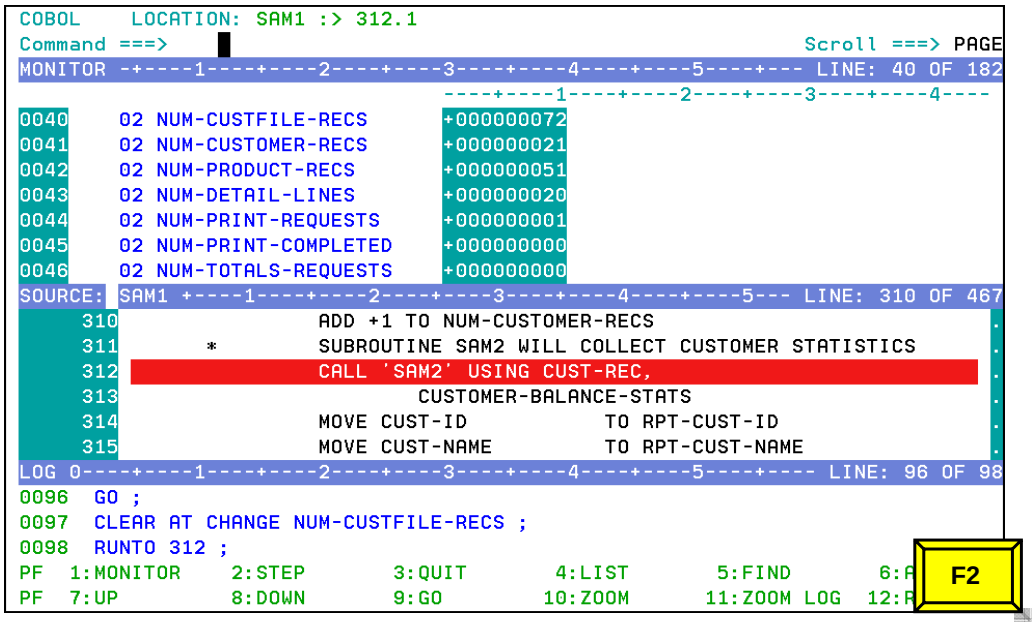
Run to a subroutine call in the program



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.

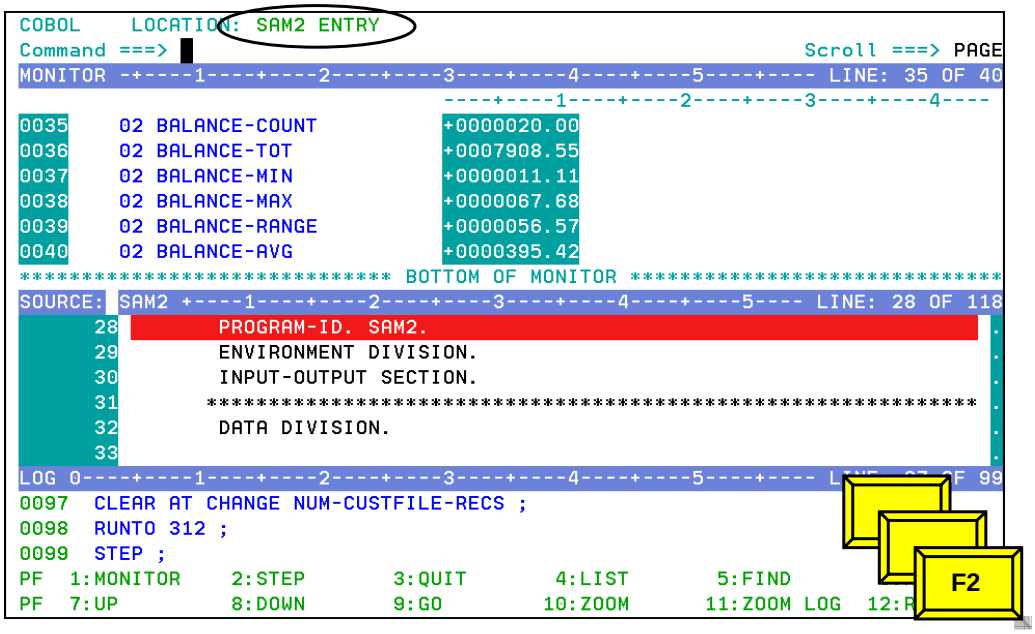
You can STEP into a subprogram or function



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



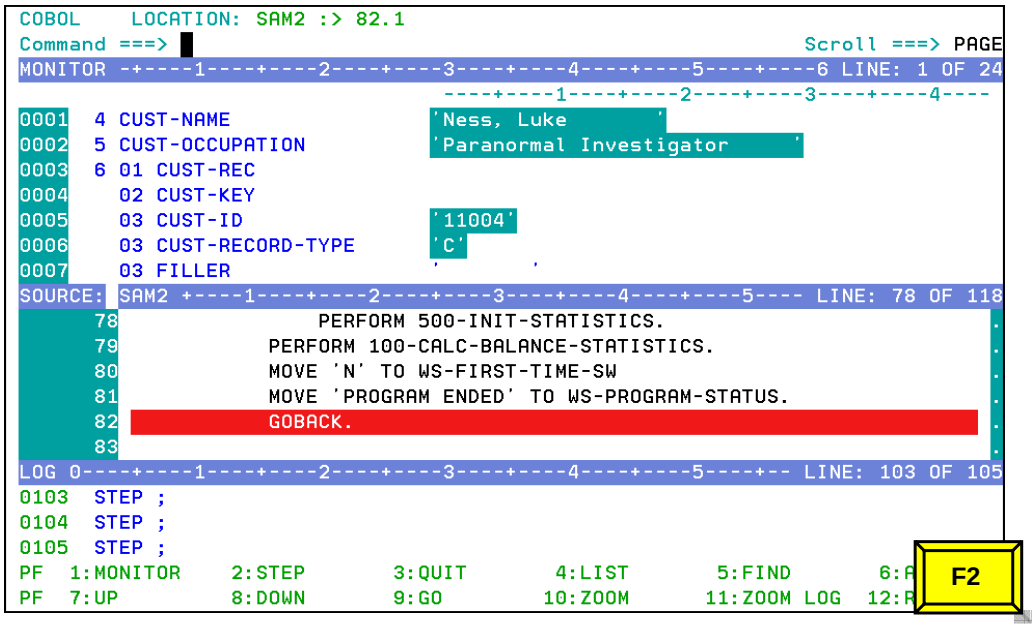
That stepped into the subprogram, SAM2.

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.

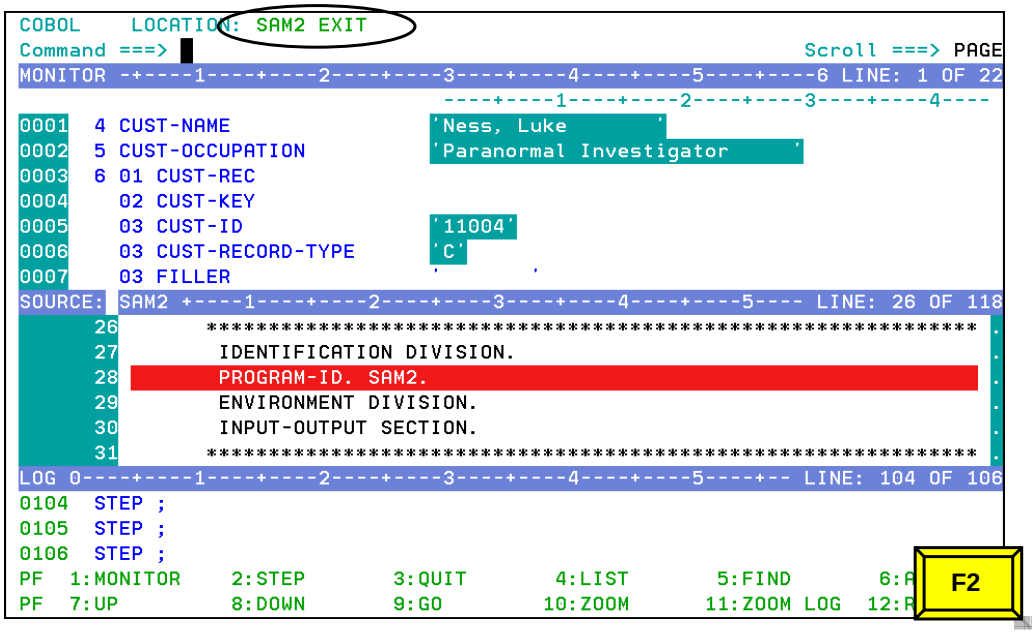
In this example, the F2 key is pressed several times to step through statements in the subprogram.

You can STEP out of a subprogram or function



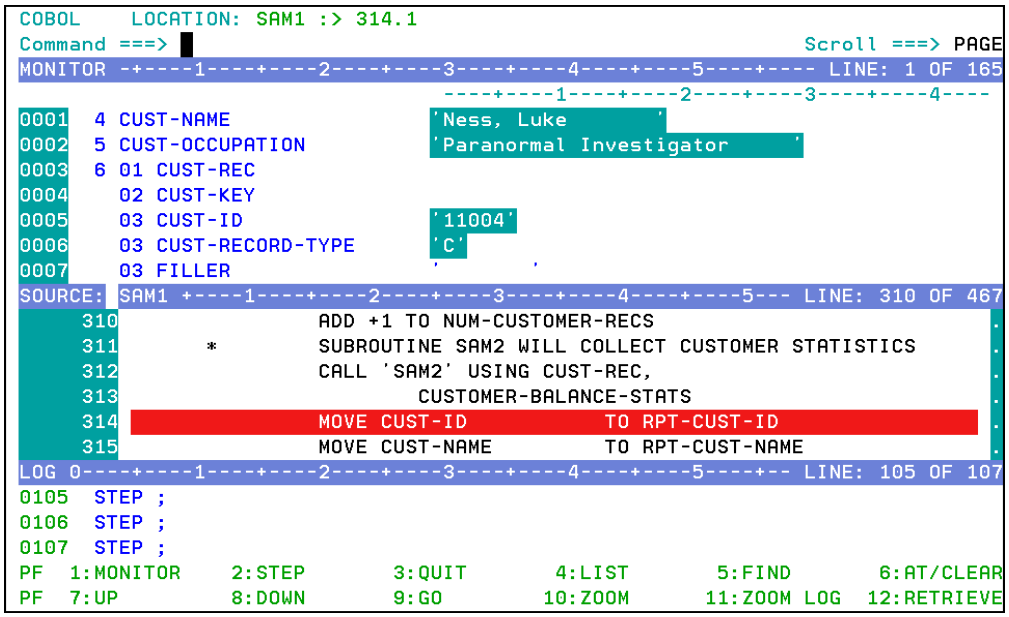
After stepping many times, the last statement in the subprogram is reached. F2 is pressed to step again.

At the exit of the subprogram



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

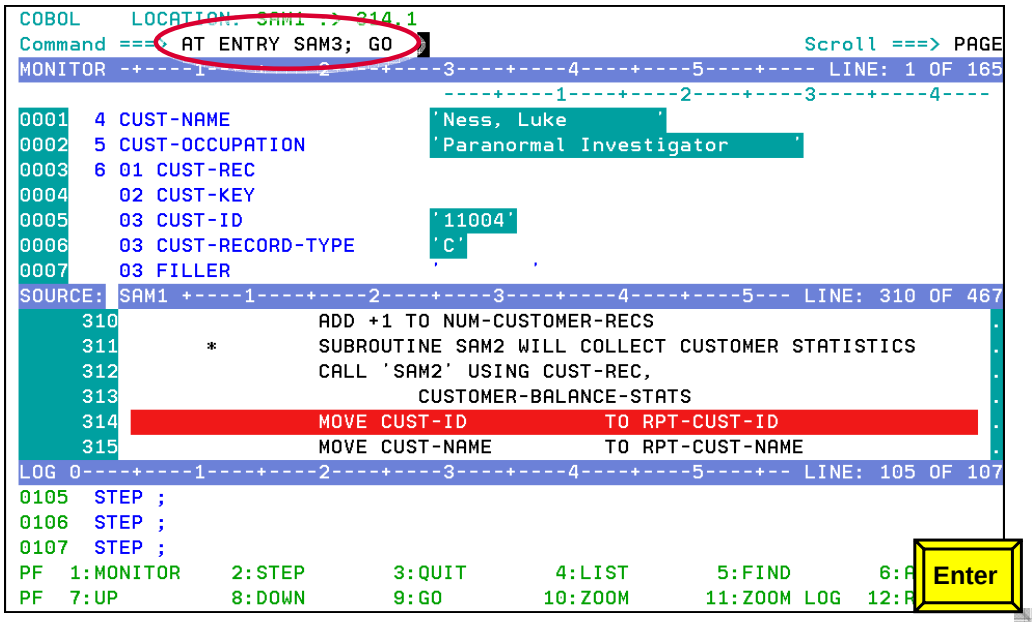
Returned to the calling program



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.

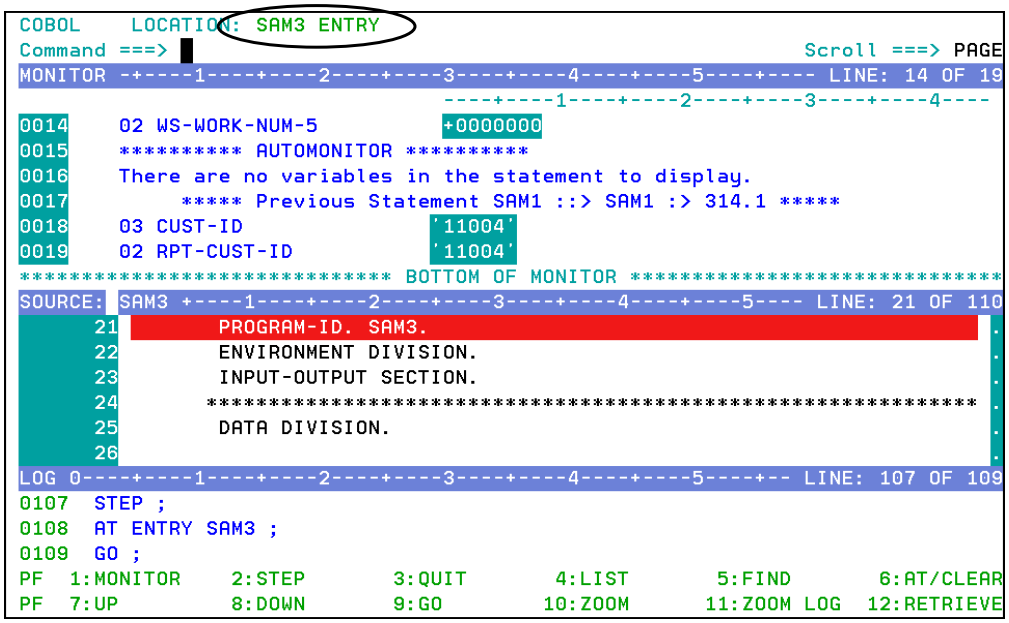
An AT ENTRY *name* command sets an entry breakpoint for a program or routine



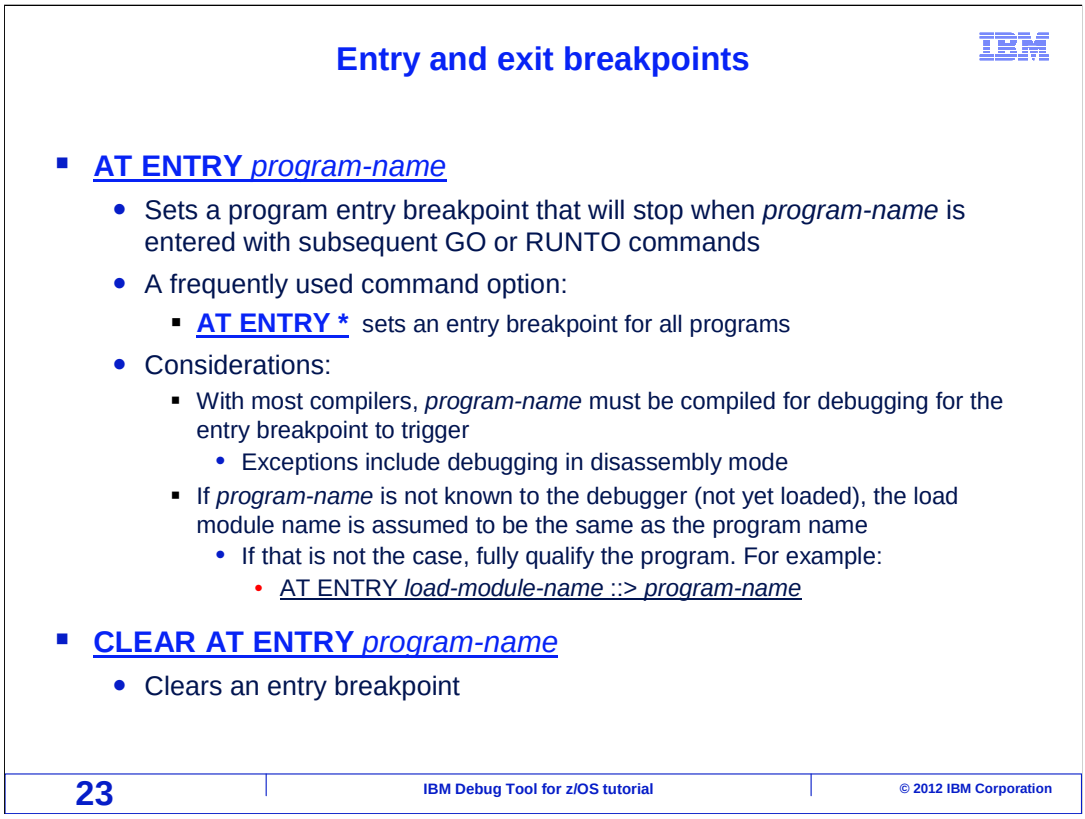
But in a complex application, it can be cumbersome to have to step into any subprogram that you want to debug. So there is a simpler way to pause when a specific subprogram in entered – an "entry" breakpoint.

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



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.



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.

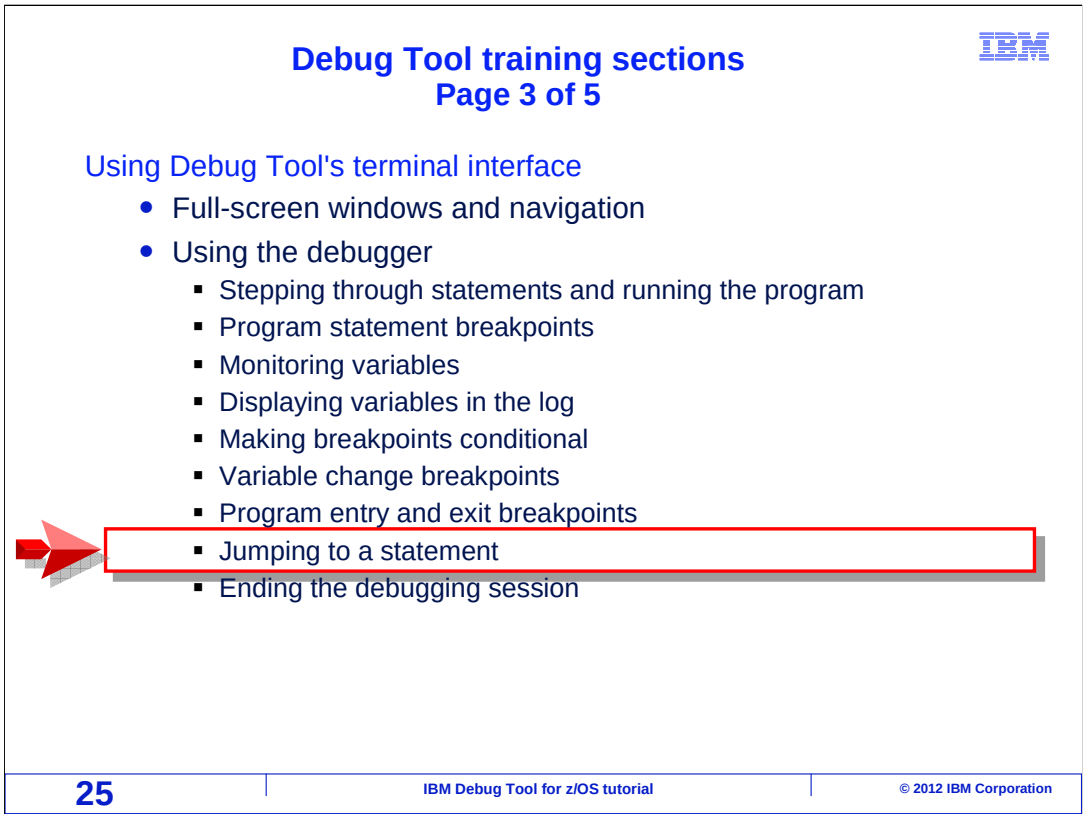
Entry and exit breakpoints

- **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.

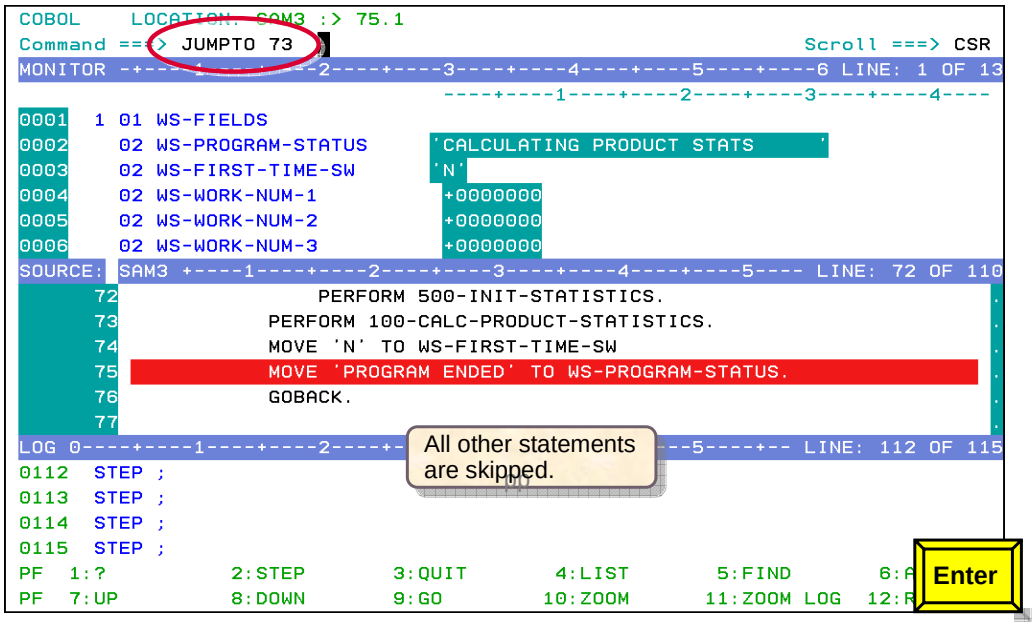
The command "AT EXIT program-name" will set a breakpoint that will pause when the named program is exited. Consider setting an exit breakpoint on the main program, as this gives you a last chance to examine program variables before the application ends.

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

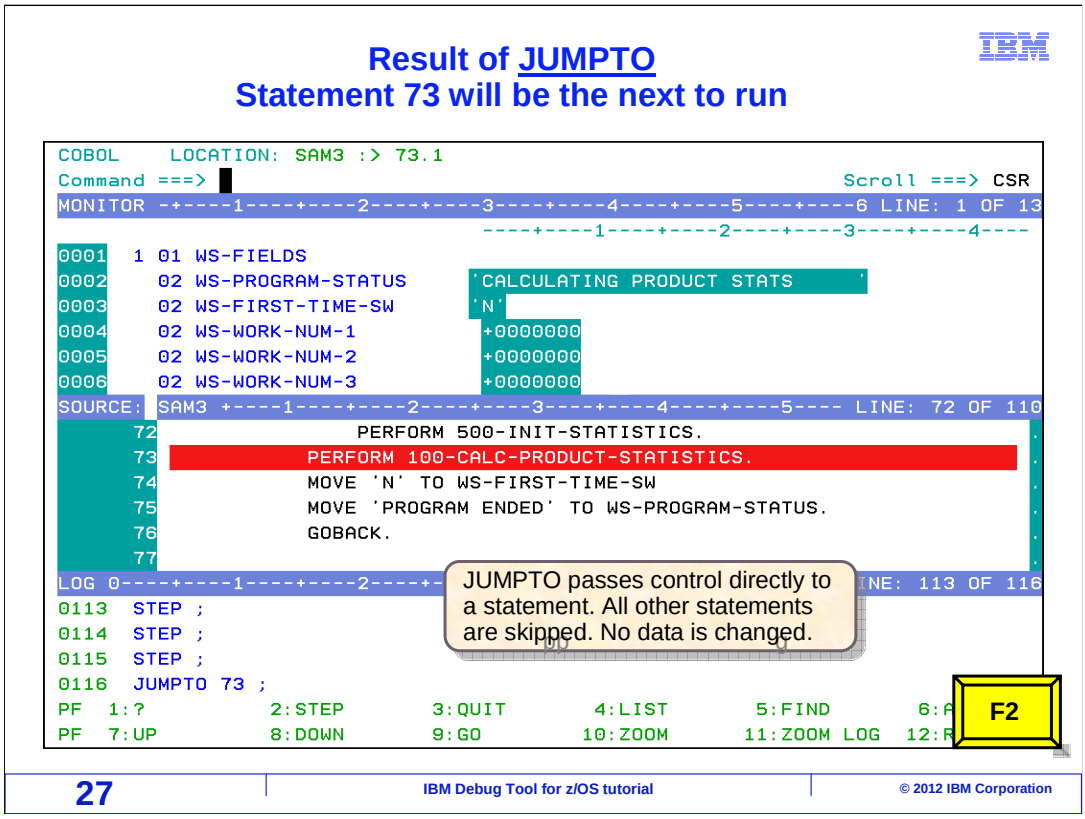


Next, you will see how to alter the flow of a program by jumping to a statement.

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

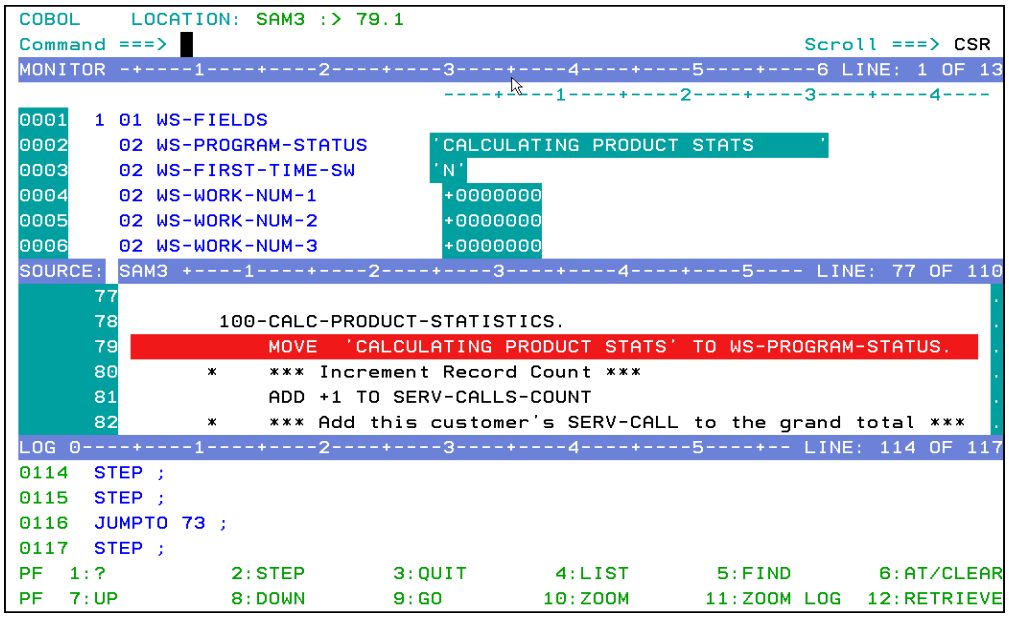


In this example, statement seventy-five is the current statement, and should be the next to run. A "jumpto 73" command is entered to pass control directly to statement seventy-three.

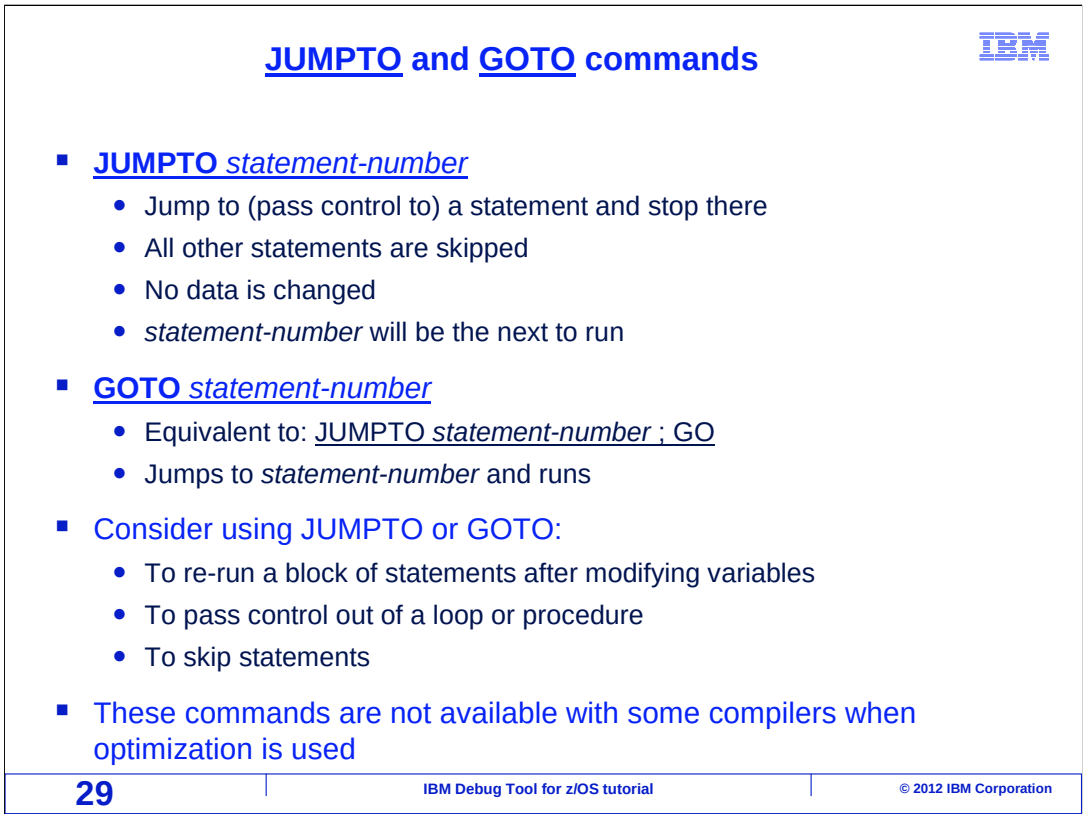


After entering the jumpto command, no statements ran. But now seventy-three is the current statement, and will be the next to run. A "jumpto" command does not change any variable values. The "step" function key is pressed.

After a step



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

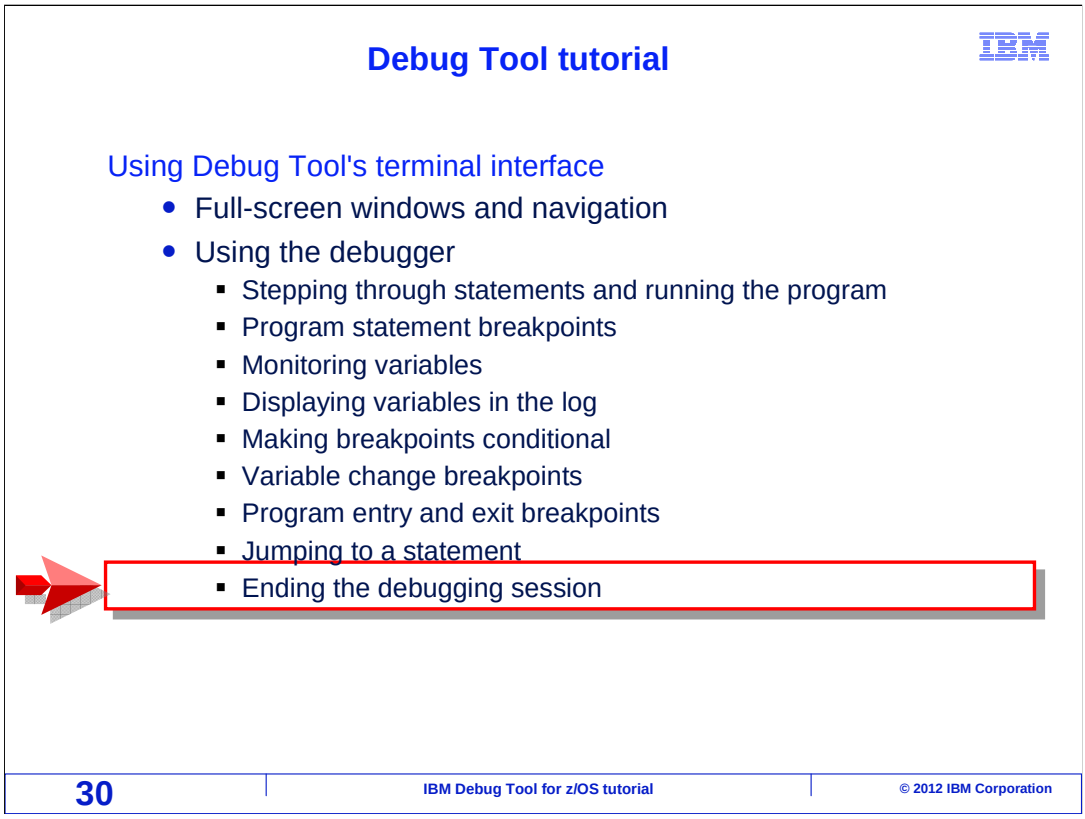


Use a "jumpto" command to alter the flow of a program. "Jumpto" passes control directly to the statement number you specify, and stops there. All other statements are skipped, and no data is changed..

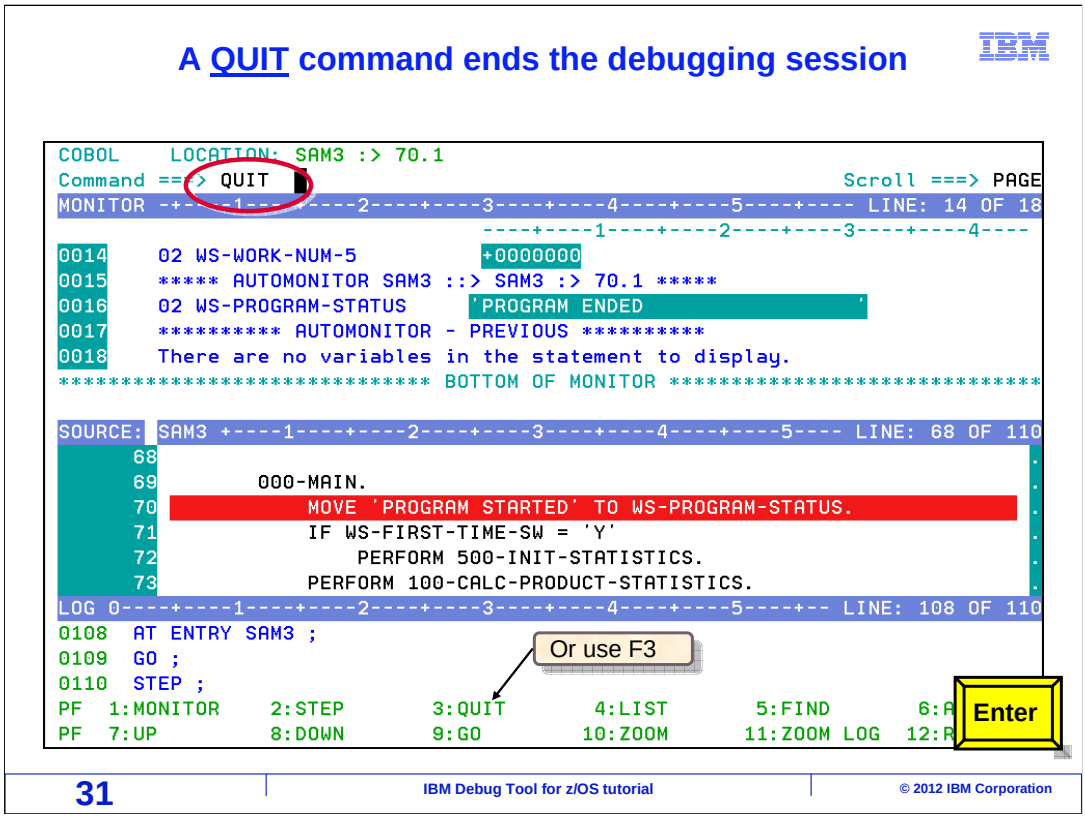
Another, similar command is "goto". It is the same as a jumpto command followed by a go. The difference is that "jumpto" will wait for you to continue running the program after the jump, whereas "goto" will not wait. The program continues running immediately. With a "goto", the jump is done, but the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



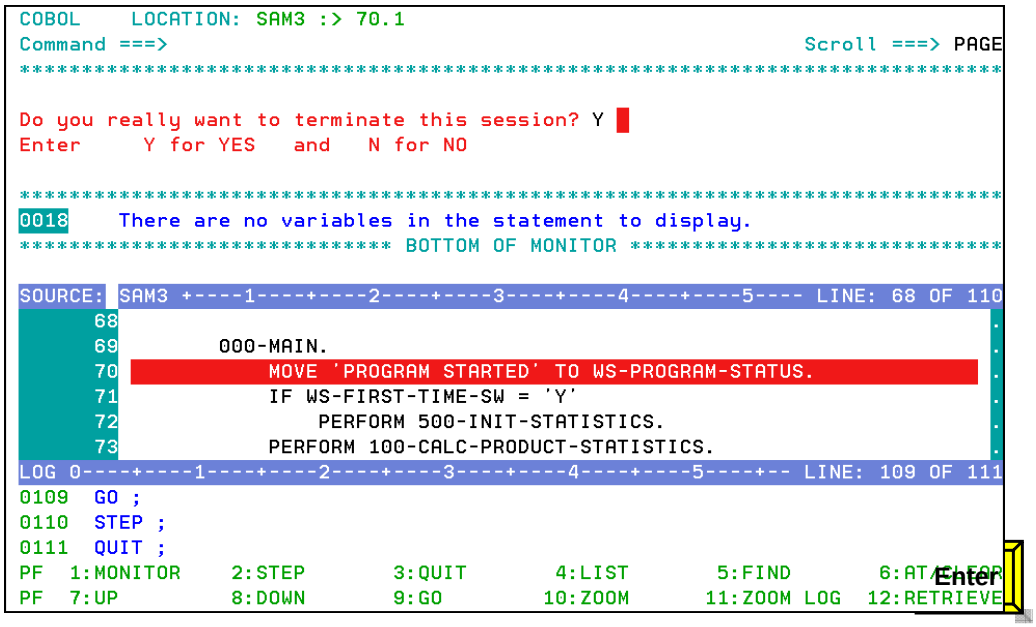
Next, you will see options available to end a debugging session.



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.

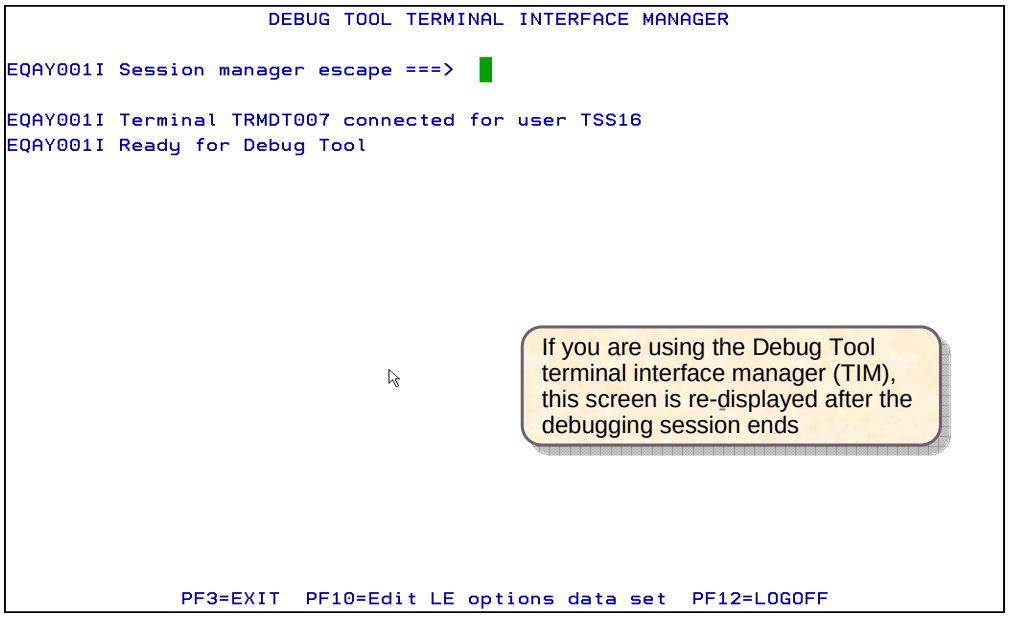
However, you can end a debugging session at any time by typing "QUIT" on the command line, and pressing Enter.

Enter Y at the prompt to terminate the session



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



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

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

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" 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.