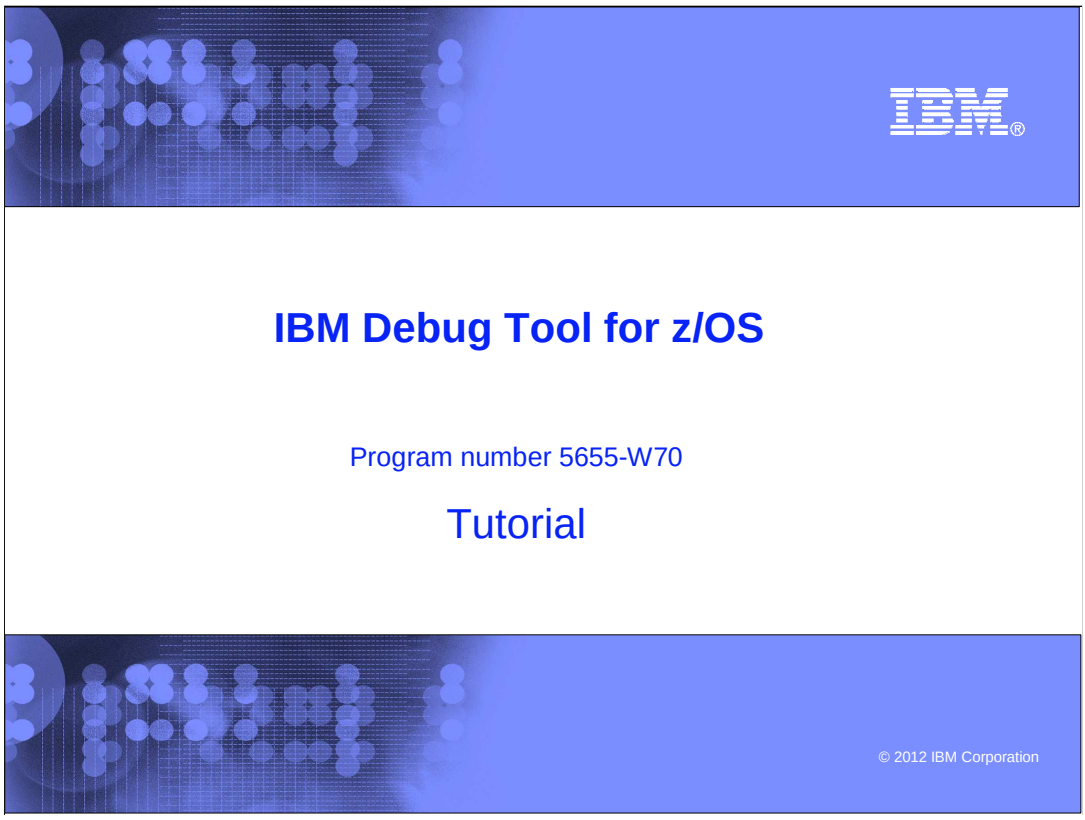
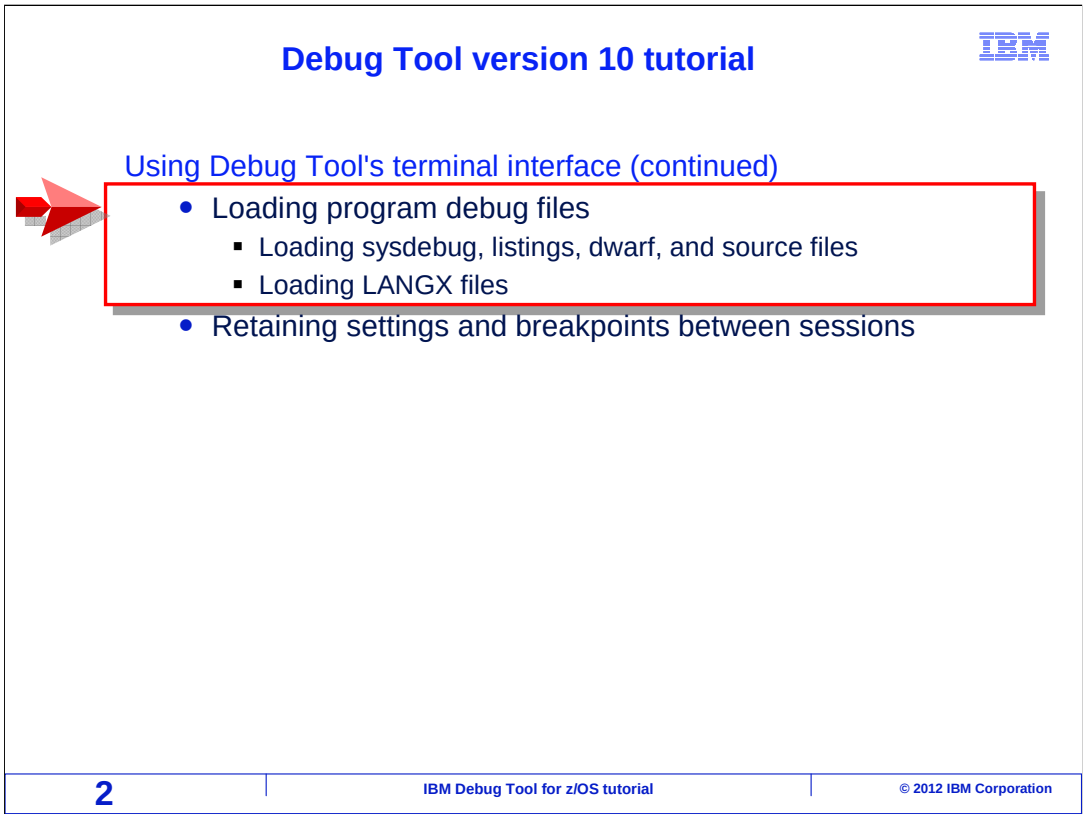


This is the tutorial for IBM Debug Tool for z/OS®, one of the IBM zSeries® problem determination tools.



In this section, you will see how the debugger uses debug files, and how you can specify debug files for your programs.

Providing program information for the debugger

- Debug Tool reads a debug file for a program to display information about that program
- The type of file needed depends on the compiler
  - A SYSDEBUG file
    - Used with Enterprise COBOL and Enterprise PL/I
    - The compiler creates it when a TEST(…,SEP) compiler option is specified
  - A LANGX file
    - Used with assembler, OS/VS COBOL, and COBOL II compiled with NOTEST
    - It is produced by a utility program (EQALANGX) that runs after a compile or assembly
  - A compiler listing
    - Used with older PL/I compilers, and COBOL II compiled with TEST
  - A dwarf or .mdbg file
    - Used with XL C/C++ when certain compiler options are specified
  - The program source file
    - Used with XL C/C++, and older versions of Enterprise PL/I

To show you program source statements and variables while you are working with programs, the debugger reads a debug file for each program. The type of file that it uses depends on the compiler used. Different compilers produce different kinds of debug files.

Sysdebug files are used with programs compiled with the Enterprise COBOL and Enterprise PL/I compilers.

LANGX files are used with assembler programs, and programs compiled with the OS/VS COBOL compiler, and can optionally be used with programs compiled with the VS COBOL II compiler.

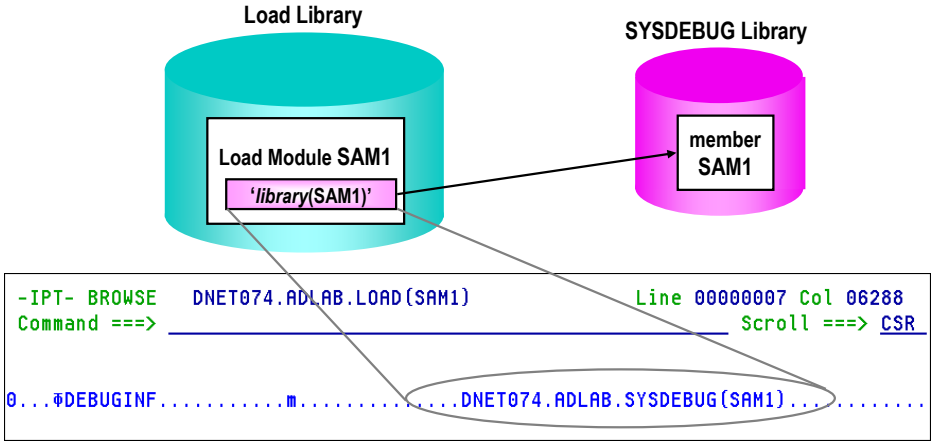
The debugger can read compiler listings with programs compiled with older PL/I compilers, and optionally with VS COBOL II programs.

A dwarf or .mdbg file can be used with programs compiled with the XL C/C++ compiler, and the actual program source file itself can optionally be used with XL C/C++ programs and programs compiled with older versions of Enterprise PL/I.

Compile processes should be updated so that the appropriate file is generated automatically when you compile or assemble a program.

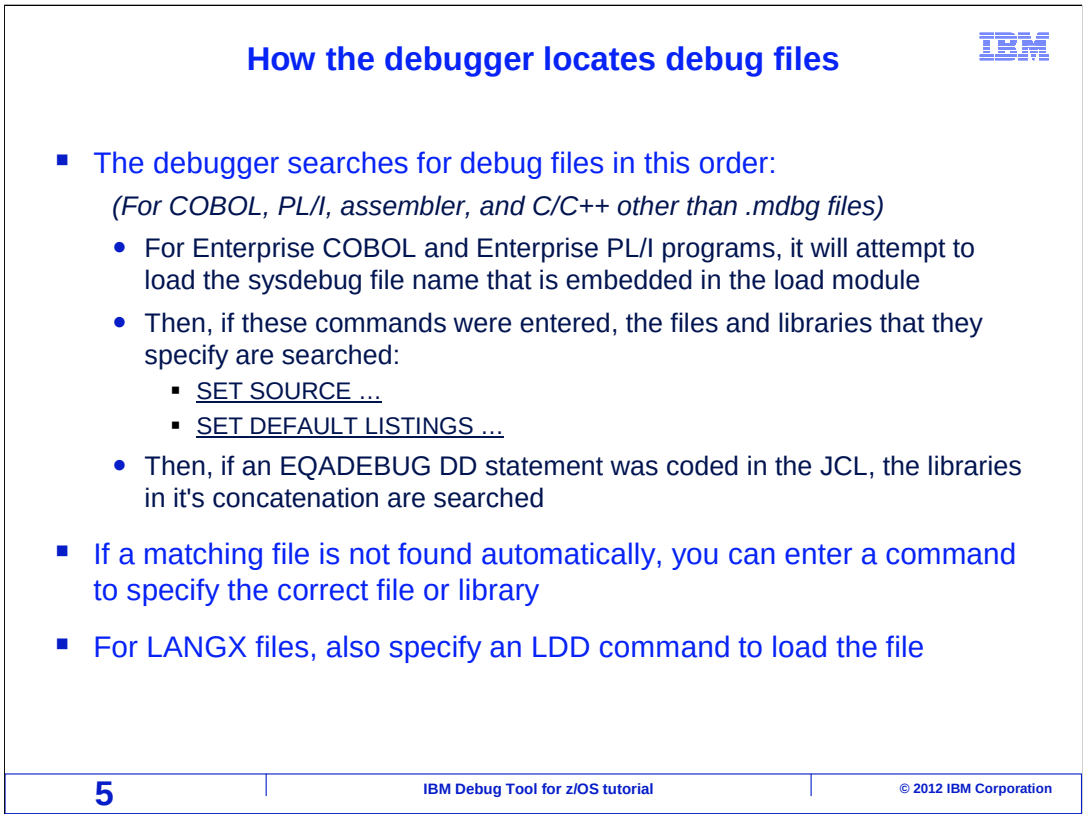
The debugger can automatically locate the file for some compilers

- Enterprise COBOL and Enterprise PL/I compilers can embed the name of the SYSDEBUG file in the load module
  - The debugger can automatically load it if it exists



Enterprise COBOL and Enterprise PL/I compilers can embed the name of the sysdebug file directly in the load module. This is a helpful feature, because the debugger can automatically find and load the file when the debugger starts or when a new program is entered.

If you browse a load module generated with one of these compilers, you will be able to see the name of the sysdebug file in the module.



When the debugger starts, or when it detects that a new program has been entered, it can attempt to automatically load the file containing the needed source information.

For Enterprise COBOL and Enterprise PL/I programs, it will attempt to load the sysdebug file based on the name embedded in the load module. If the file is found, the debugger performs a check to validate that the timestamp in the sysdebug file matches the timestamp of the module. If it matches, the file is used.

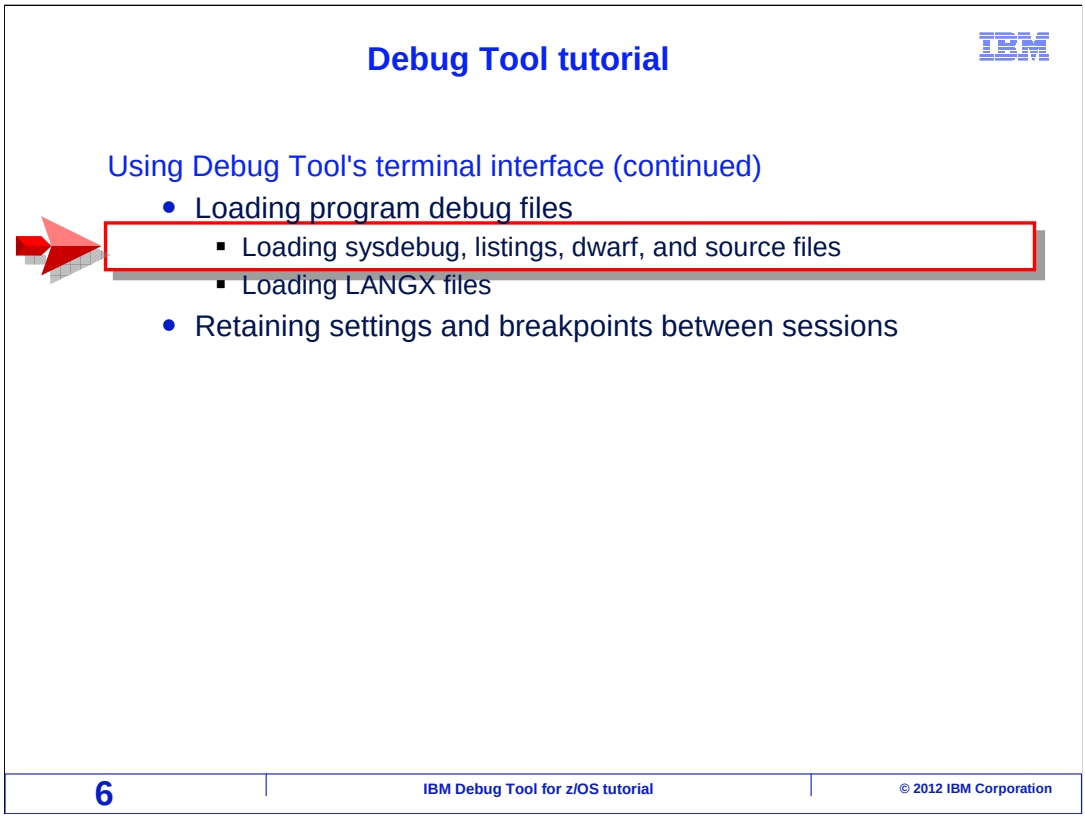
If a match is not found, the debugger will look other places. The user can specify a "SET SOURCE …" setting to specify the name of the debug file for the program. If a "SET SOURCE" setting has been specified for the program, the debugger opens the specified file and validates the timestamp.

If a match is still not found, the debugger checks for a "SET DEFAULT LISTINGS" setting. This setting provides a list of libraries to be searched. Each library in the list is searched for a member with a matching name and timestamp.

Finally, the debugger checks to see if an EQADEBUG DD statement exists, which is another way to specify a list of libraries to be searched. These libraries are all checked for a matching member name and timestamp.

Only after exhausting all of these possibilities will the debugger display a message indicating that source information could not be found.

If that happens, you can then enter commands to specify a file or libraries to search for the correct file. In many cases, the debugger will then find and load the file automatically. For LANGX files, an "LDD" command must also be specified to explicitly load the file.

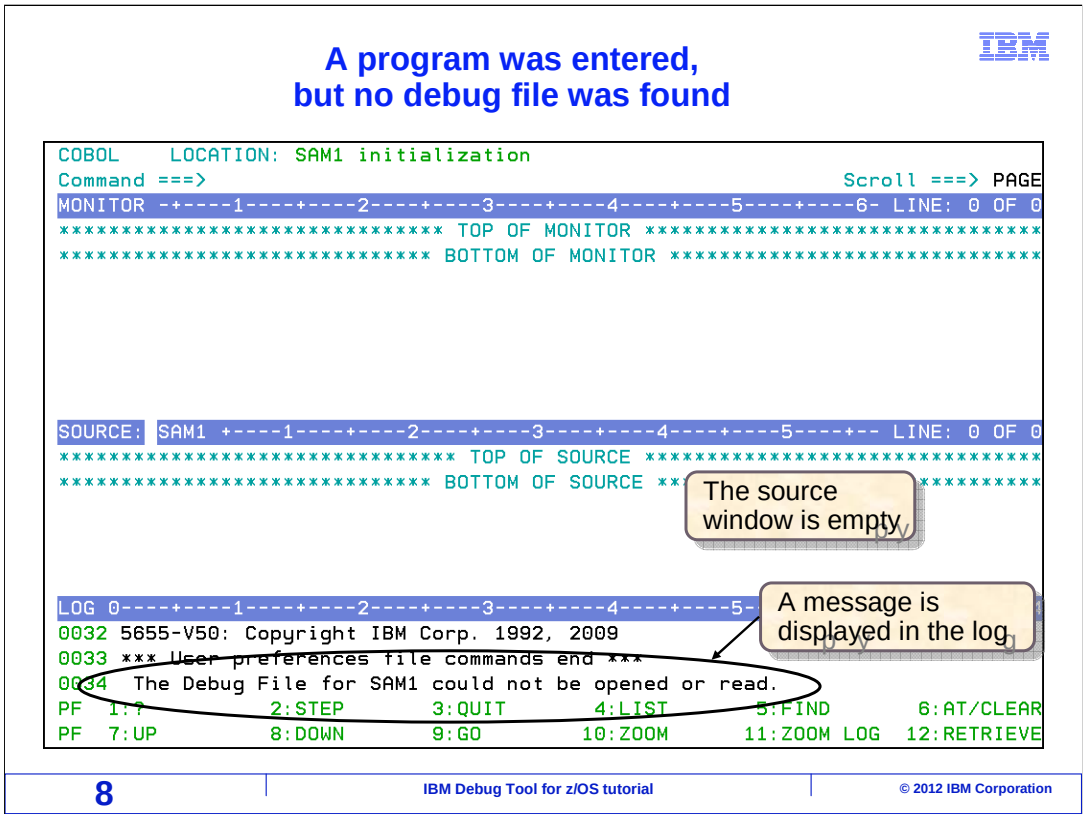


Next, you will see how to load a sysdebug file, a compiler listing, a dwarf file, or a program source file.

Example of loading sysdebug, listings, dwarf, and source files

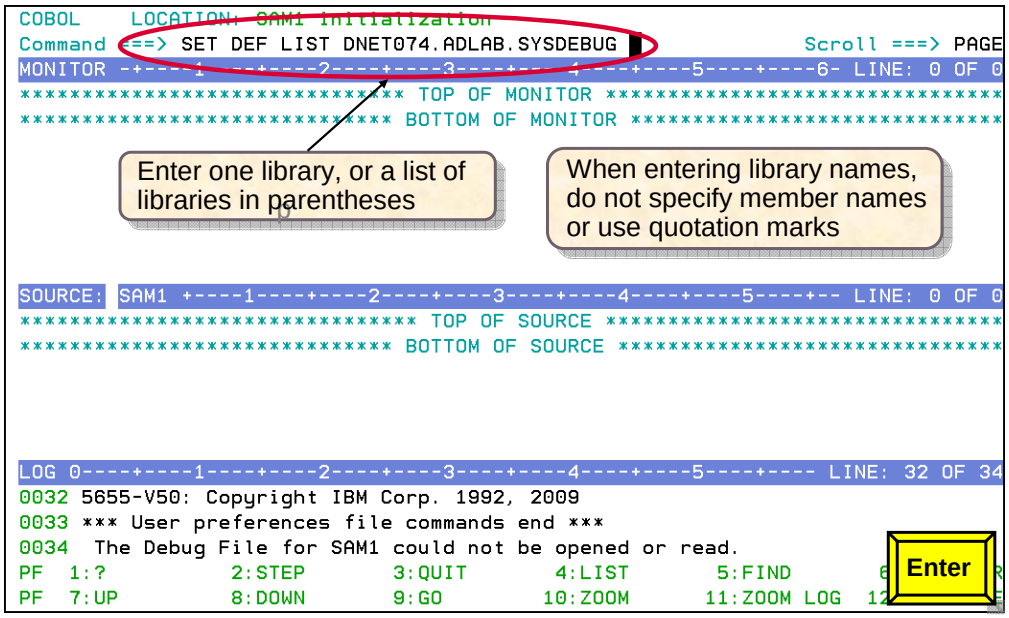
- Use this example for programs compiled with:
  - Enterprise COBOL
  - Enterprise PL/I
  - COBOL II (compiled with a TEST compiler option)
  - PL/I for MVS and VM
  - OS PL/I
  - XL C/C++ (when not using a .mdbg file)

This example shows the process for manually loading a debug file when you are working with a program compiled with Enterprise COBOL, Enterprise PL/I, COBOL II (when a TEST compiler option is used), PL/I for MVS and VM, OS PL/I, and XL C/C++ (when not using a .mdbg file). The process and commands will be the same, even though the debugger will use some different types of debug files for different compilers.



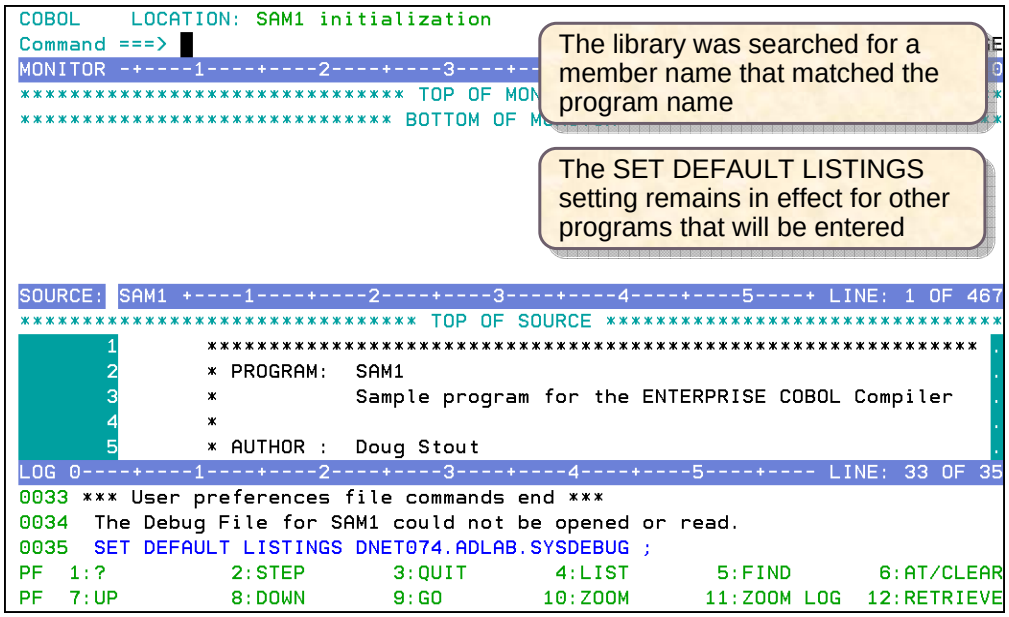
In this example, the debugger started, but did not find a debug file for the program. Notice that the source window is empty, and a message is displayed in the log describing the problem.

Use a SET DEFAULT LISTINGS … command to specify debug file libraries



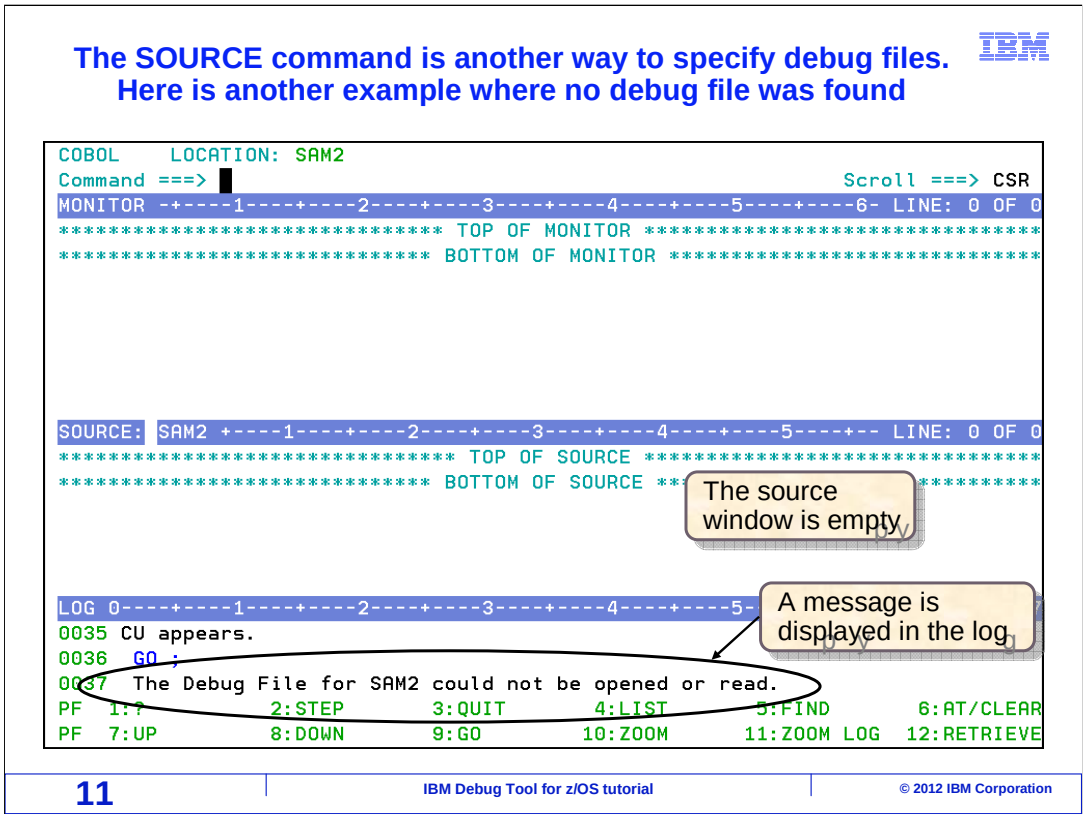
A "SET DEFAULT LISTINGS …" command is entered. In this example, a sysdebug library is specified. Enter only the name of the library or PDS, not the member name. The debugger will automatically look for a matching member name.

The debug file was loaded from the library



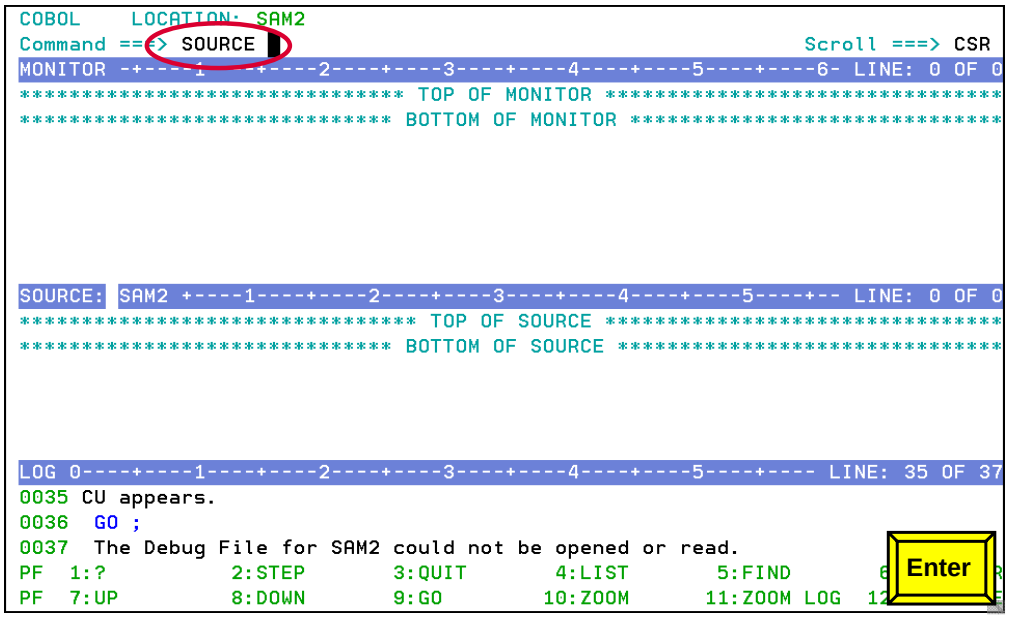
When the SET DEFAULT LISTINGS command was entered, the debugger searched the library and loaded the debug file. Notice that program source is displayed in the source window now.

The "SET DEFAULT LISTINGS …" setting is a good way to specify the location of debug files. A list of libraries can be specified, if needed. The setting remains in effect, and when each subprogram is entered, the same libraries will be searched again to find the right file. So this setting automates the search.



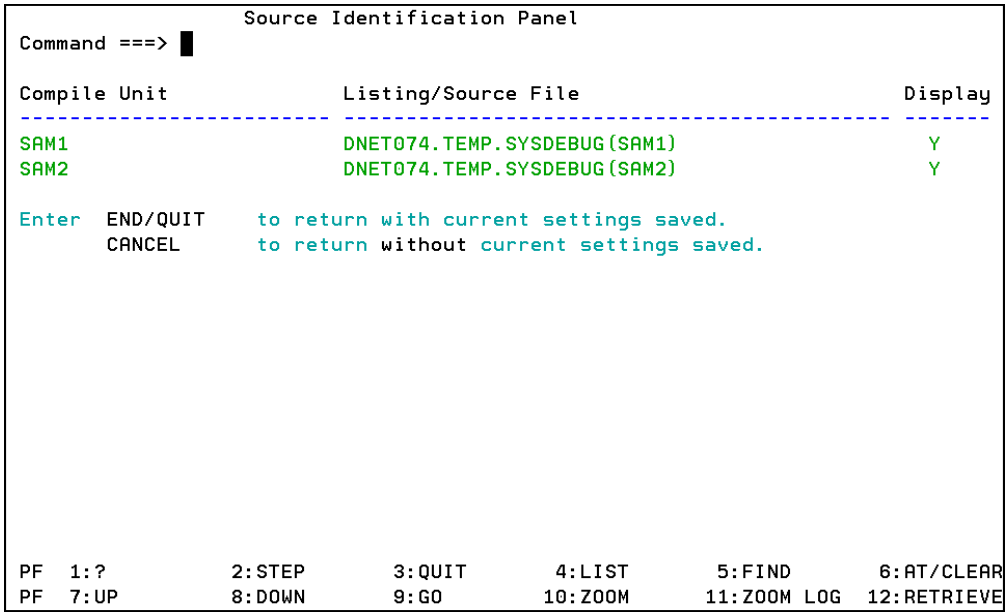
Another way to specify the location of a debug file is with the "SOURCE" command. Here is another example where the debugger started, but did not find a debug file for the program. As in the last example, the source window is empty, and a message is displayed in the log.

A SOURCE command shows a list of programs and their debug files



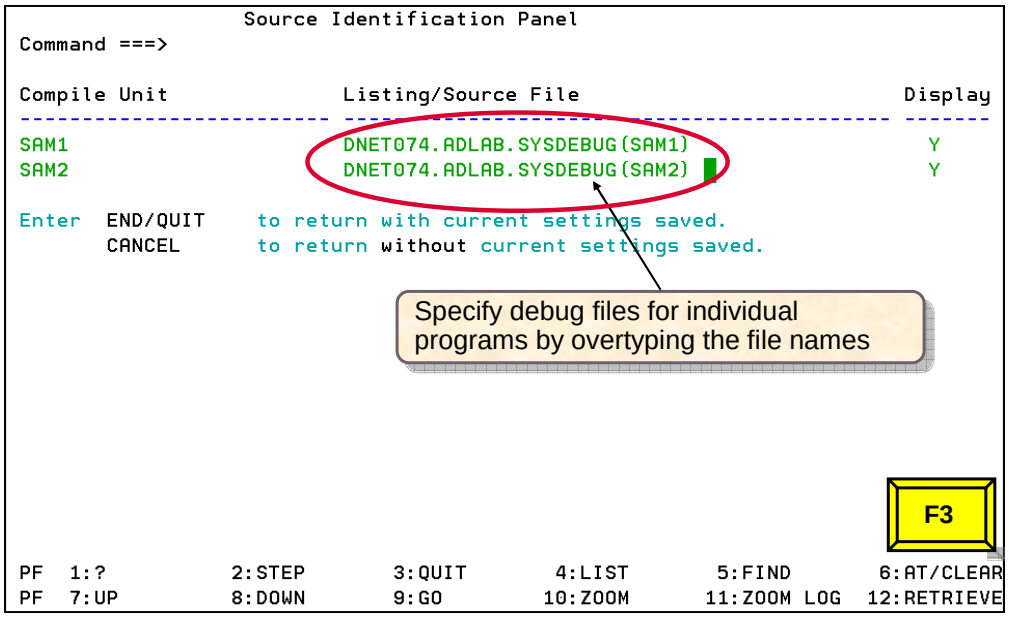
"SOURCE" is typed on the command line, and Enter is pressed.

The source identification panel displays programs and their debug files



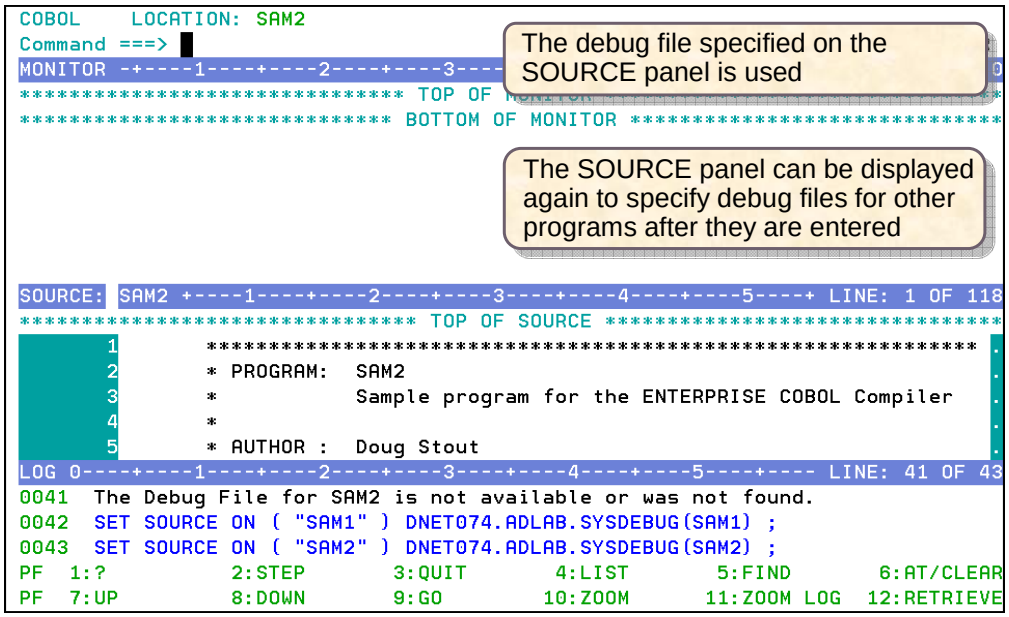
The source identification panel is displayed. It shows a list of programs that have appeared, and their corresponding debug files. In this example, the debug files were renamed after the programs were compiled, which is why the debugger could not find them automatically.

The debug file can be specified for each program



You can specify the name of a debug file next to each program, to control the side file name on a program by program basis. Here the correct file names are entered. Press F3 to return to the debugger screen.

The debug file was loaded from the library



The debugger loaded the debug files specified. Notice that program source is displayed in the source window now. The SOURCE command can be used to specify debug file names individually for each program. As other subprograms are entered, the SOURCE command can be entered again to specify the debug files for the additional programs.

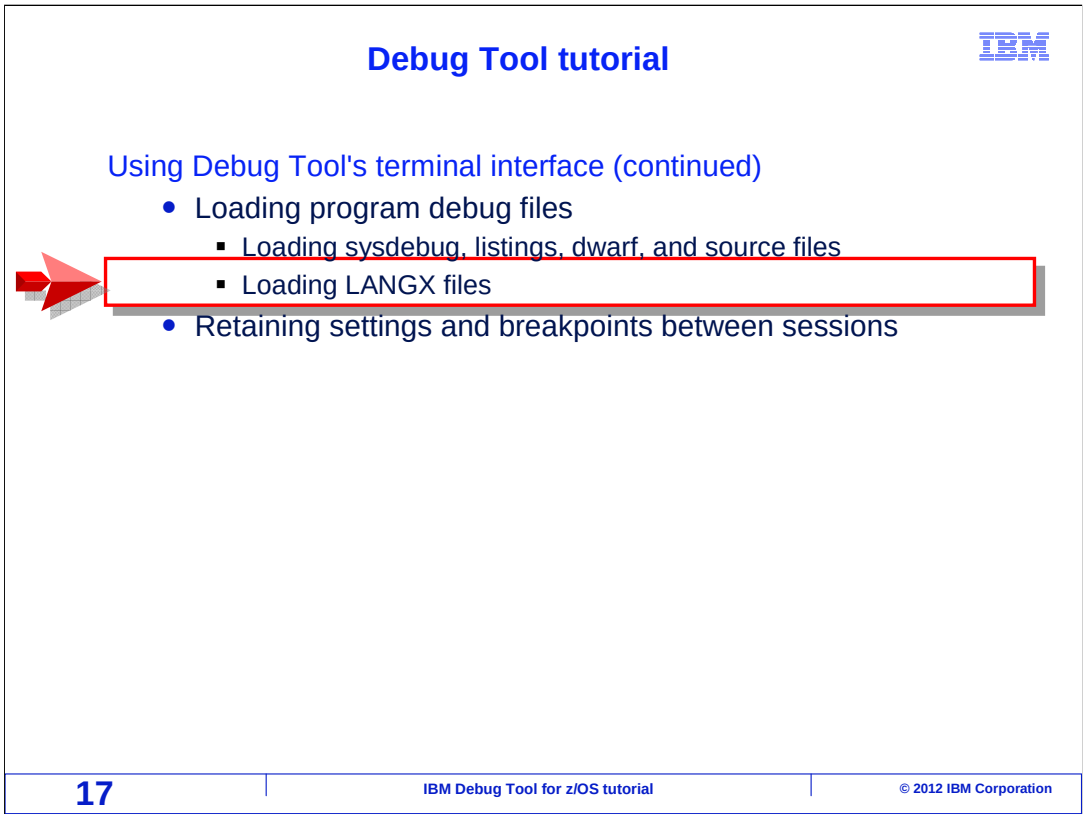
Loading program debug files

- **SET DEFAULT LISTINGS** *library-name* , or
- **SET DEF LIST** *library-name* , or
- **SET DEF LIST (*library-1*, *library-2*, *library-3*, …)**
  - Defines a library or libraries whose members are searched for program debug files
    - Tip: to enter a long library concatenation, use the POPUP command first to make room to enter a multi-line command
  - You can concatenate sysdebug files, langx files, compiler listings, and expanded source files all in the same list

- **SOURCE** or **LISTING**
  - Displays a panel that shows programs and their corresponding debug files
  - The names of individual debug files can be specified on the panel

A "SET DEFAULT LISTINGS …" command is used to specify a library, or a list of libraries that the debugger will search for debug files. In the same list, you can specify a combination of the different types of libraries as needed, including sysdebug, LANGX, compiler listings, and expanded source files.

You can enter a "SOURCE" command to display the "Source identification panel", where you can manually specify the name of the debug file to be loaded for each program individually.



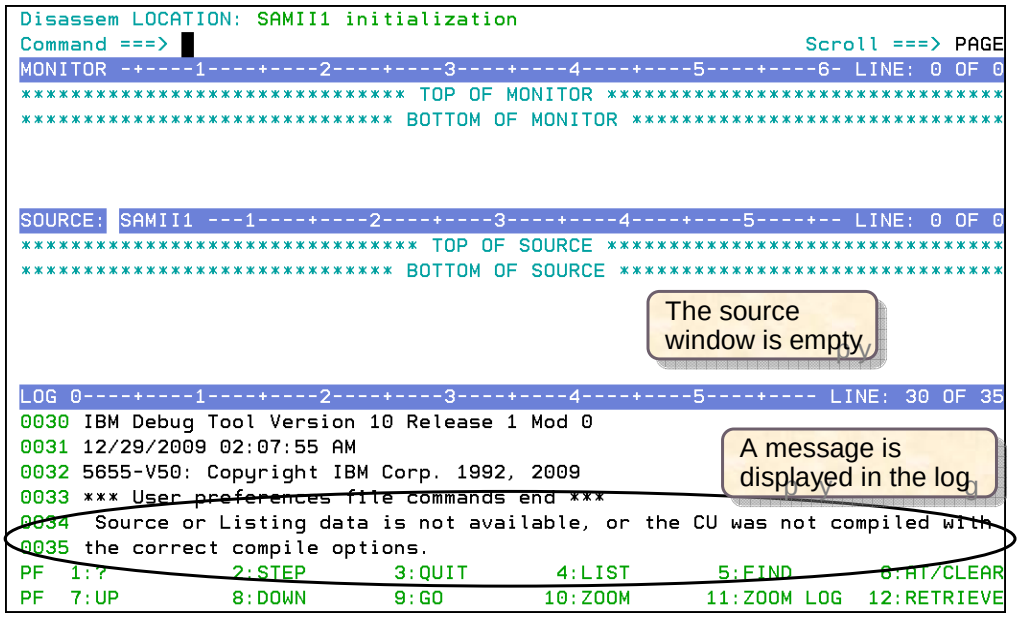
Next, you will see how to load a LANGX file.

Example of loading LANGX debug files

- Use this example to load LANGX debug files
- LANGX files are used with programs compiled with:
  - Assembler
  - VS COBOL II (compiled with the NOTEST compiler option)
  - OS/VS COBOL

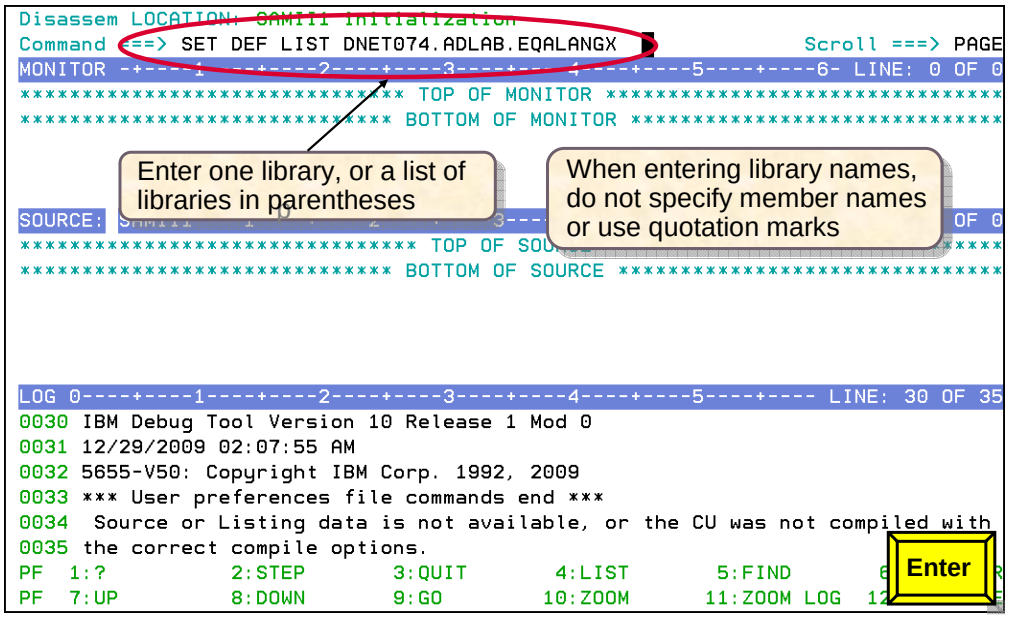
This example shows the process for manually loading a LANGX file. These are used with assembler programs, and programs compiled with the OS/VS COBOL compiler, or optionally with VS COBOL II when compiled with the NOTEST compiler option.

A program was entered, but no LANGX file was found



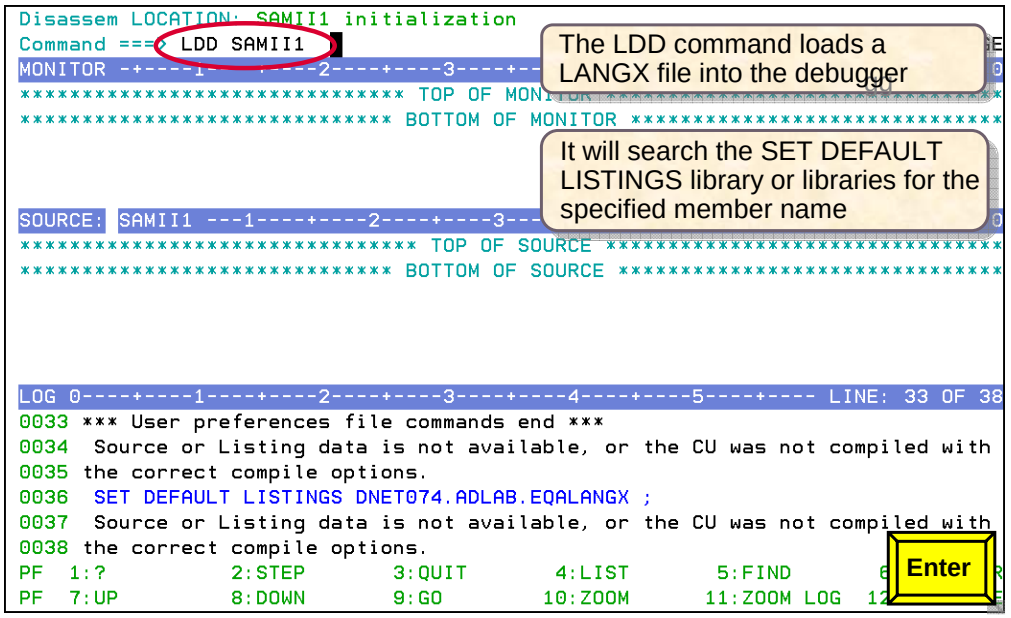
In this example, the debugger started, but did not find a debug file for the program. Notice that the source window is empty, and a message is displayed in the log describing the problem.

Use a SET DEFAULT LISTINGS … command to specify LANGX file libraries



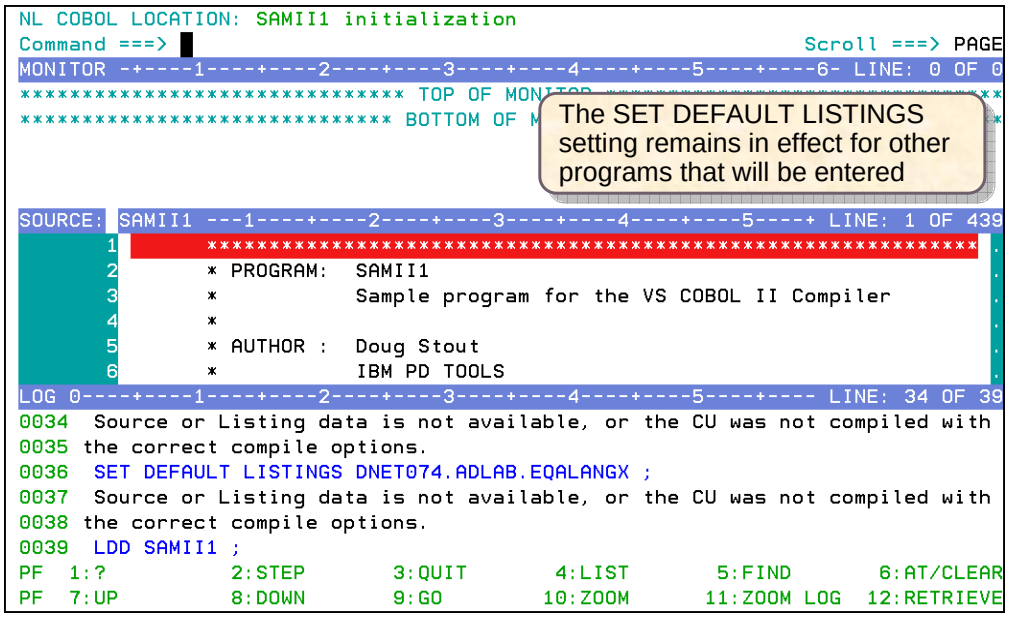
A SET DEFAULT LISTINGS command is entered. In this example, a library containing LANGX files is specified. Enter only the name of the PDS, not the member name.

Load the LANGX file with an LDD (load debug data) command



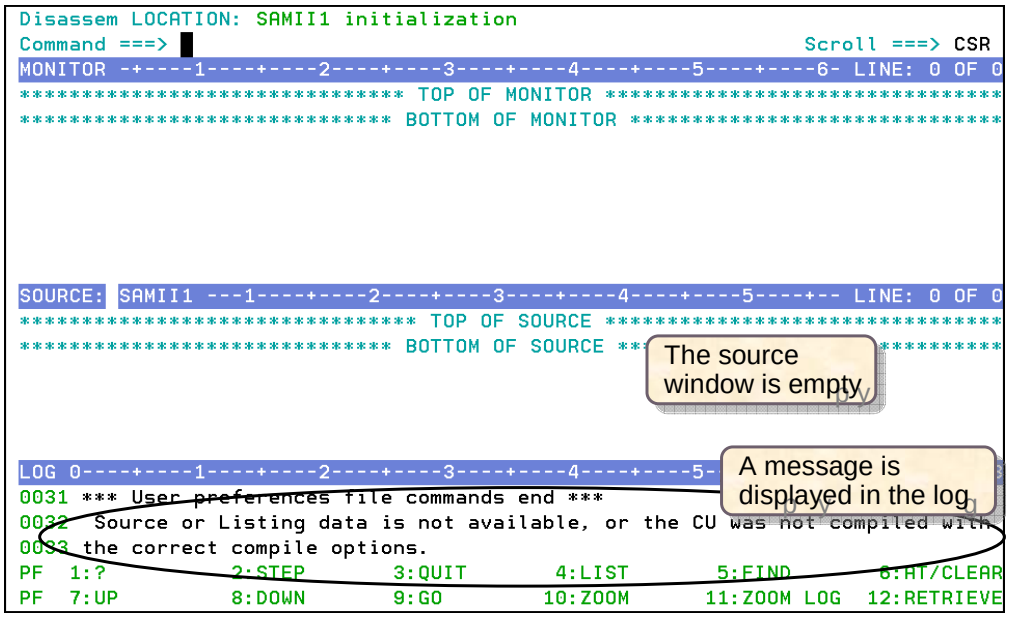
With LANGX files, an additional command, LDD (for load debug data) is needed to load the program. The LDD command is typed on the command line, specifying the member name to be loaded, and Enter is pressed.

The LANGX file was loaded from the library



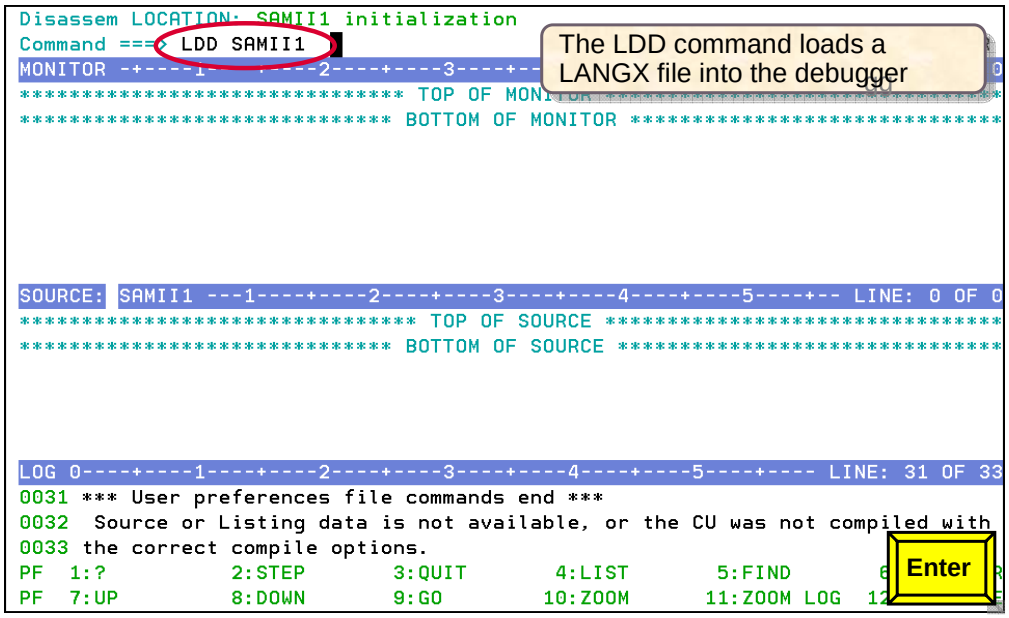
Because of the LDD command, the member was found in the library search concatenation, and was loaded. The program source code is now displayed in the source window.

The SOURCE command is another way to specify debug files. Here is another example where the Langx file was not found



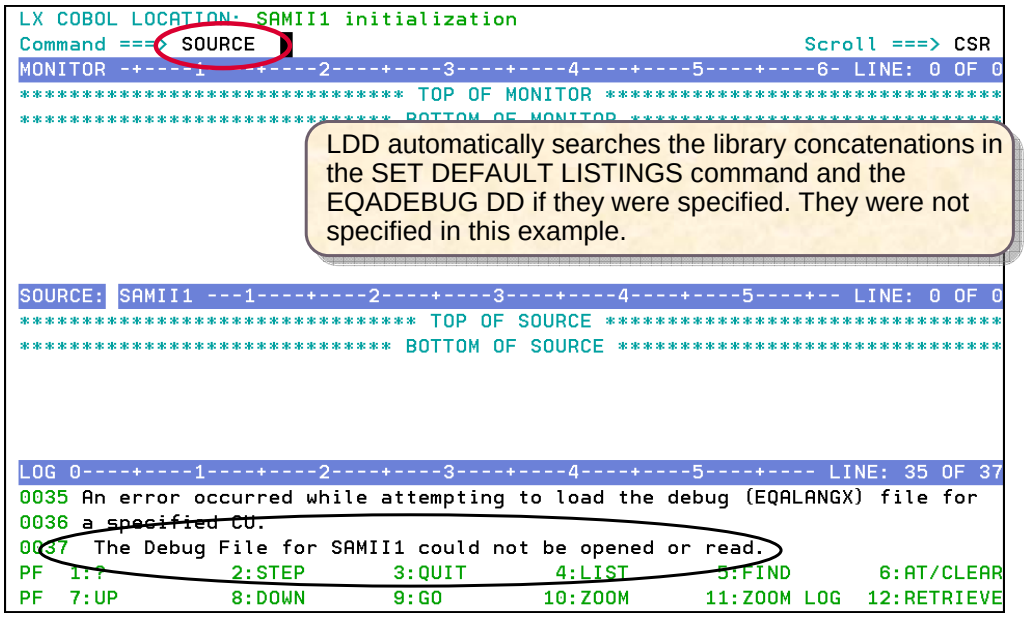
Another way to specify the location of a debug file is with the "SOURCE" command. Here is another example where the debugger started, but did not find a debug file for the program. Like in the last example, the source window is empty, and a message is displayed in the log.

Request the LANGX file to be loaded with an LDD (load debug data) command



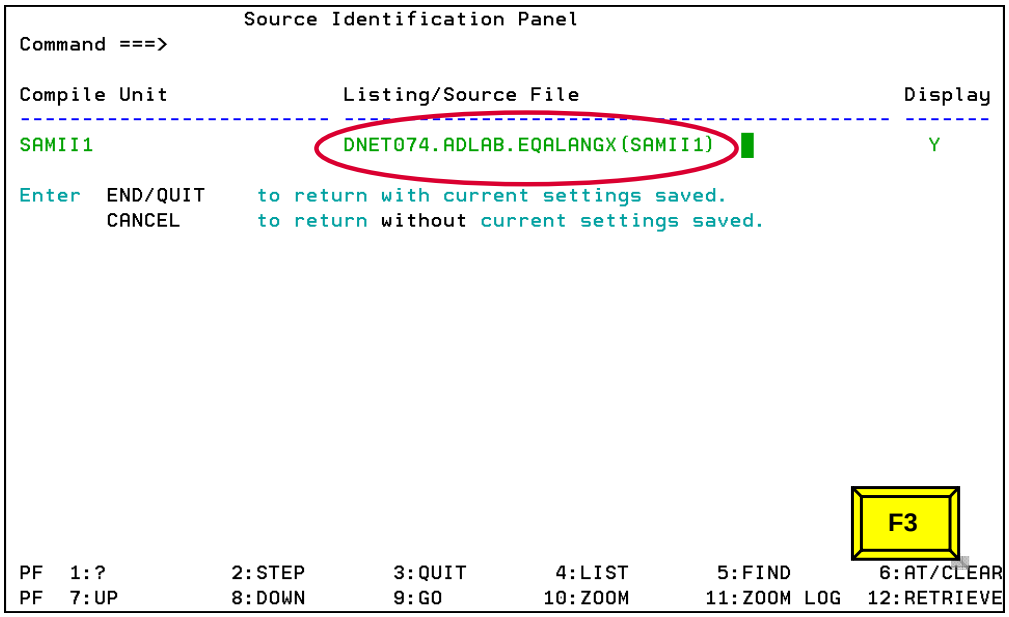
Use an LDD (load debug data) command to load the Langx file. The LDD command is typed on the command line, specifying the member name to be loaded, and Enter is pressed.

LDD did not find the Langx file.
Use a SOURCE command to specify individual LANGX files



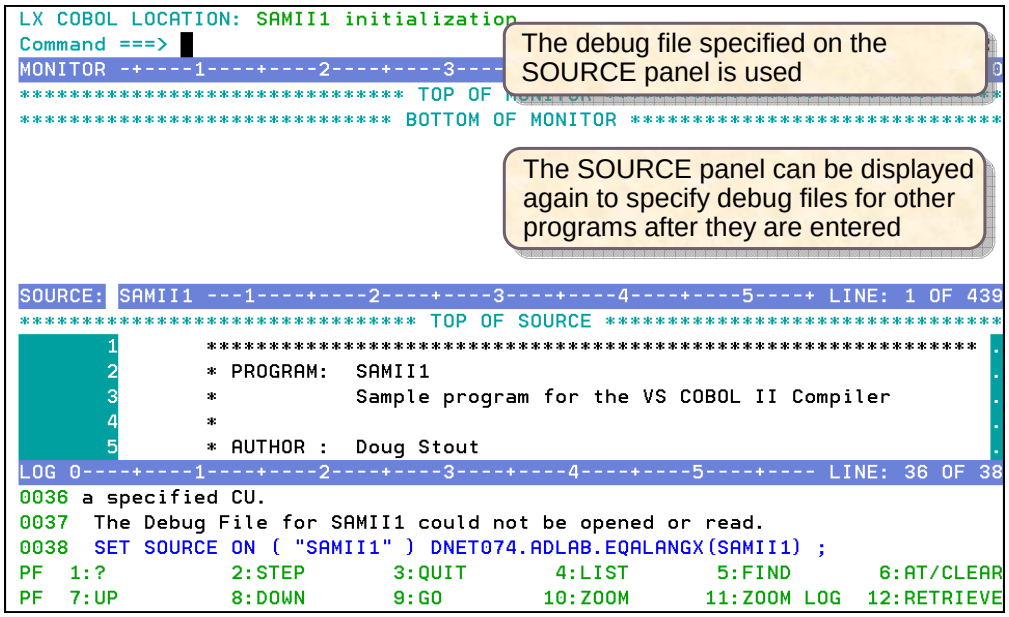
"SOURCE" is typed on the command line, and Enter is pressed.

The source identification panel displays programs and their debug files



The source identification panel is displayed. It shows a list of programs that have appeared, and their corresponding debug files. You can specify the name of a Langx file next to each program. Press F3 to return to the debugger screen.

The debug file was loaded from the library



The debugger loaded the Langx file. Notice that program source is displayed in the source window. The SOURCE command can be used to specify debug file names individually for each program. As other subprograms are entered, the SOURCE command can be entered again to specify the debug files for the additional programs.

Loading program debug files

- **SET DEFAULT LISTINGS** *library-name* , or
- **SET DEF LIST** *library-name* , or
- **SET DEF LIST (*library-1*, *library-2*, *library-3*, …)**
  - Defines a library or libraries whose members are searched for program debug files
    - Tip: to enter a long library concatenation, use the POPUP command first to have room to enter a multi-line command
  - You can concatenate sysdebug files, langx files, compiler listings, and expanded source files all in the same list
- **LDD** *member-name*
  - Loads the LANGX file in *member-name* from the library or libraries specified by the SET DEF LISTINGS … setting or an EQADEBUG DD
- **SOURCE** or **LISTING**
  - Displays a panel that shows programs and their corresponding debug files
  - The names of individual debug files can be specified on the panel

A "SET DEFAULT LISTINGS …" command is used to specify a library, or a list of libraries that the debugger will search for debug files.

Use an "LDD member-name" command to explicitly load the LANGX file. The libraries specified by the "SET DEFAULT LISTINGS …" command and the EQADEBUG DD statement are searched for the member.

You can enter a "SOURCE" command to display the "Source identification panel", where you can manually specify the name of the debug file to be loaded for each program individually.

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_DTv12s15LoadingProgramDebugFiles.ppt

This module is also available in PDF format at: ../DTv12s15LoadingProgramDebugFiles.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.