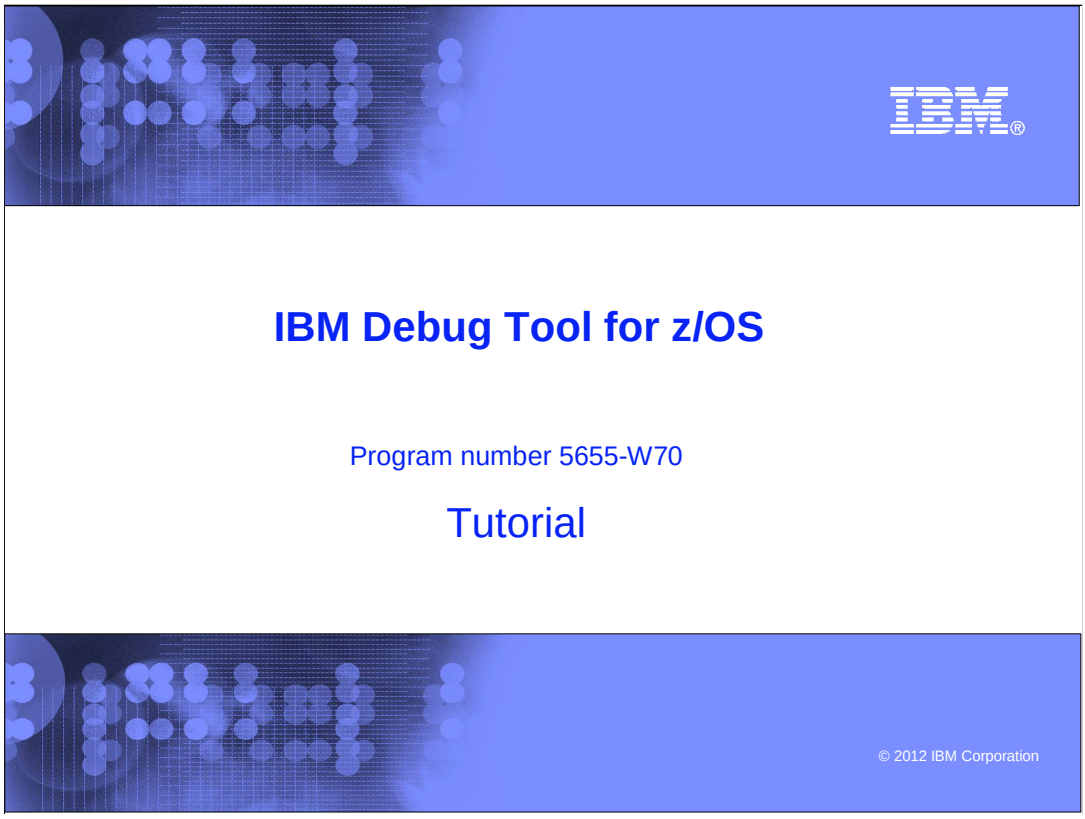
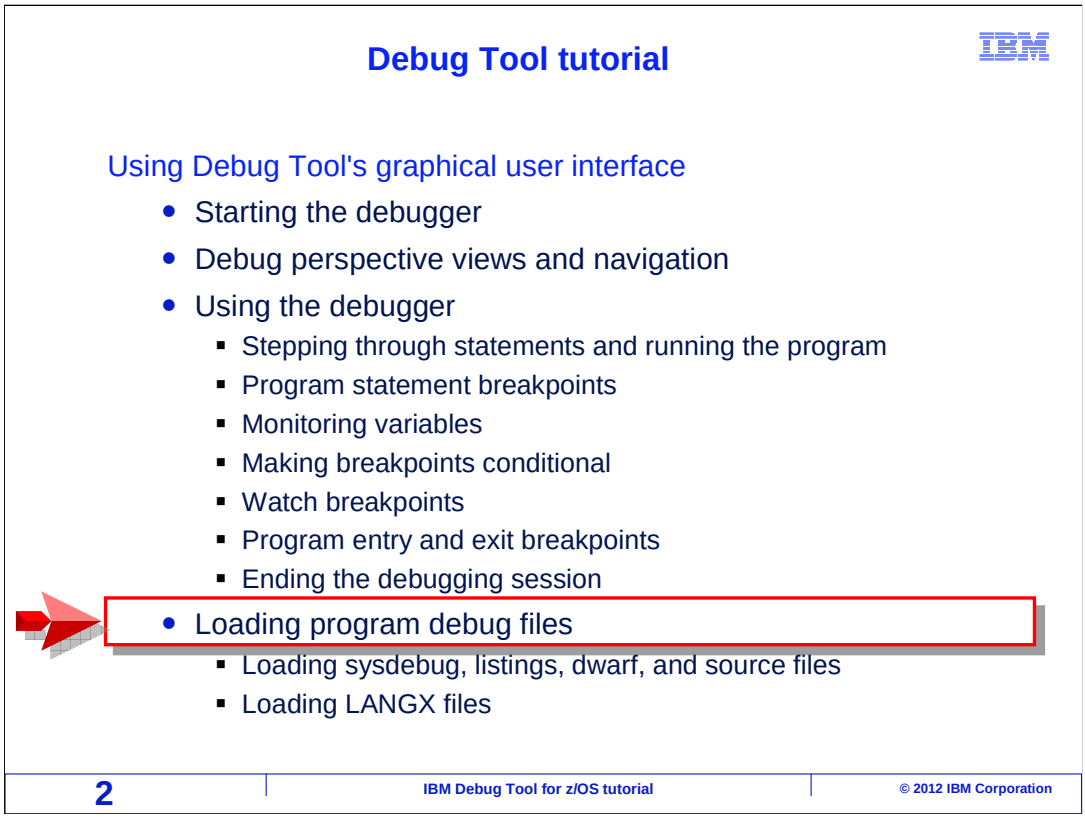


This is the tutorial for IBM Debug Tool for z/OS®, one of the IBM zSeries® problem determination tools.



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

The debugger reads a debug file for each program, and uses it to show you program source statements and variables. The type of file it uses depends on the compiler. 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.

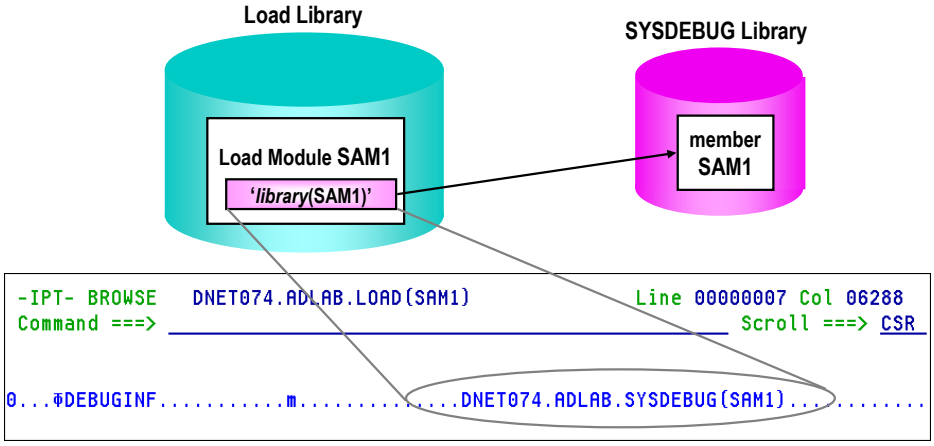
The debugger can read compiler listings with programs compiled with older PL/I compilers, and optionally with VS COBOL II programs.

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.

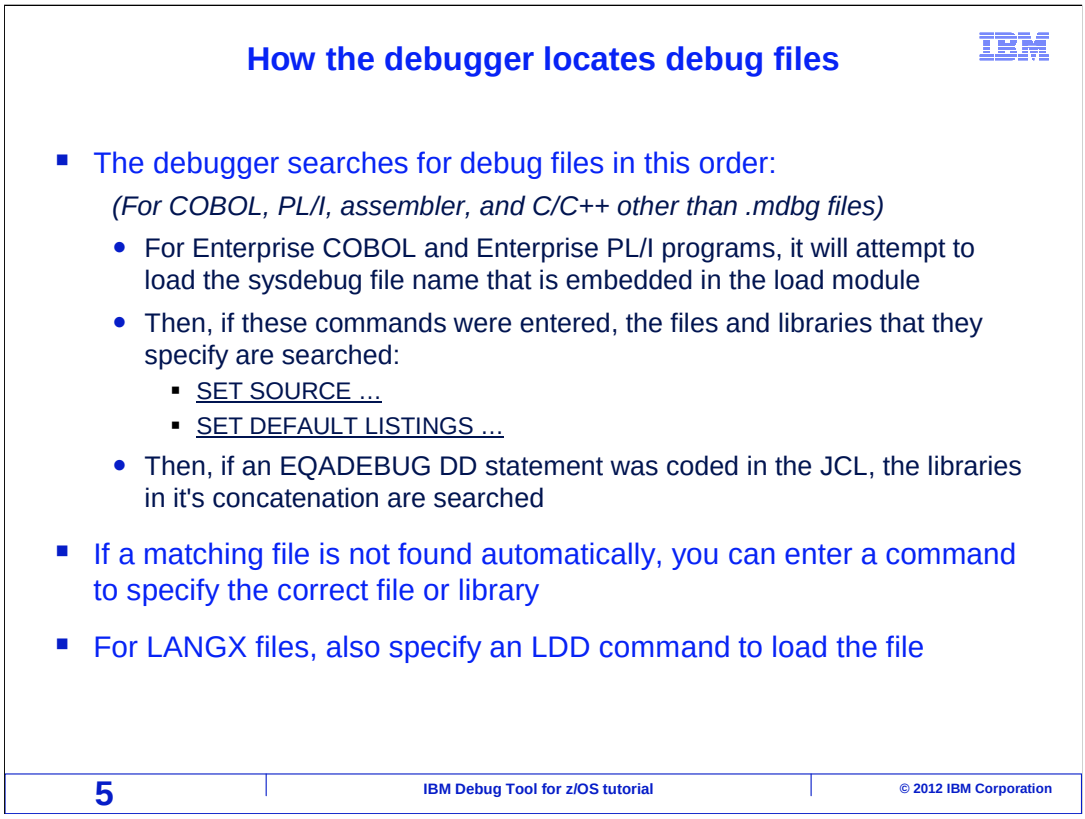
The debugger can automatically locate the file for some compilers

- Enterprise COBOL and Enterprise PL/I compilers 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. 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. This is a helpful feature, because the debugger can automatically find the file when the debugger starts or when a new program is entered.

However, if the debug file has been moved or renamed, the debugger will not be able to find it automatically. There are several ways for you to specify the new name of the file.



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.

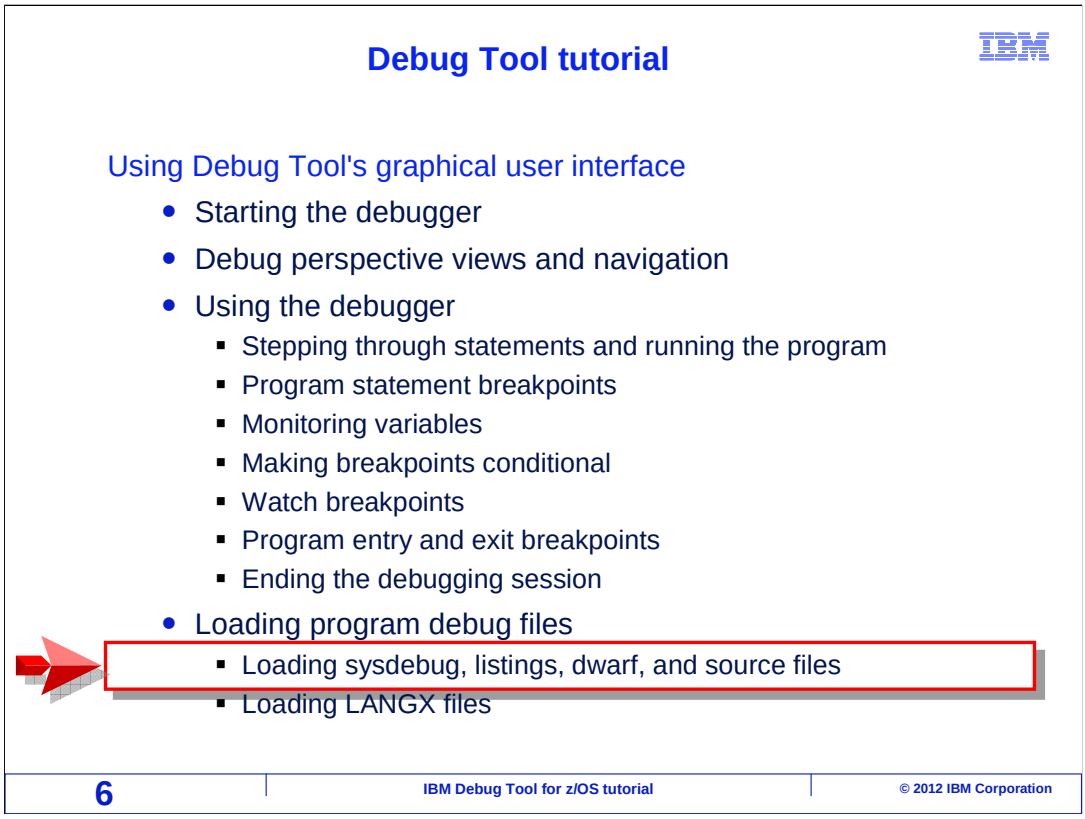
For Enterprise COBOL and Enterprise PL/I programs, it attempts to load the sysdebug file based on the name embedded in the load module. If the file is found, the debugger performs a check to validate that the timestamp in the sysdebug file matches the timestamp of the module. If it matches, the file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

If a match is not found, the debugger looks other places. The user can specify a "SET SOURCE …" debug engine 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 load the file.

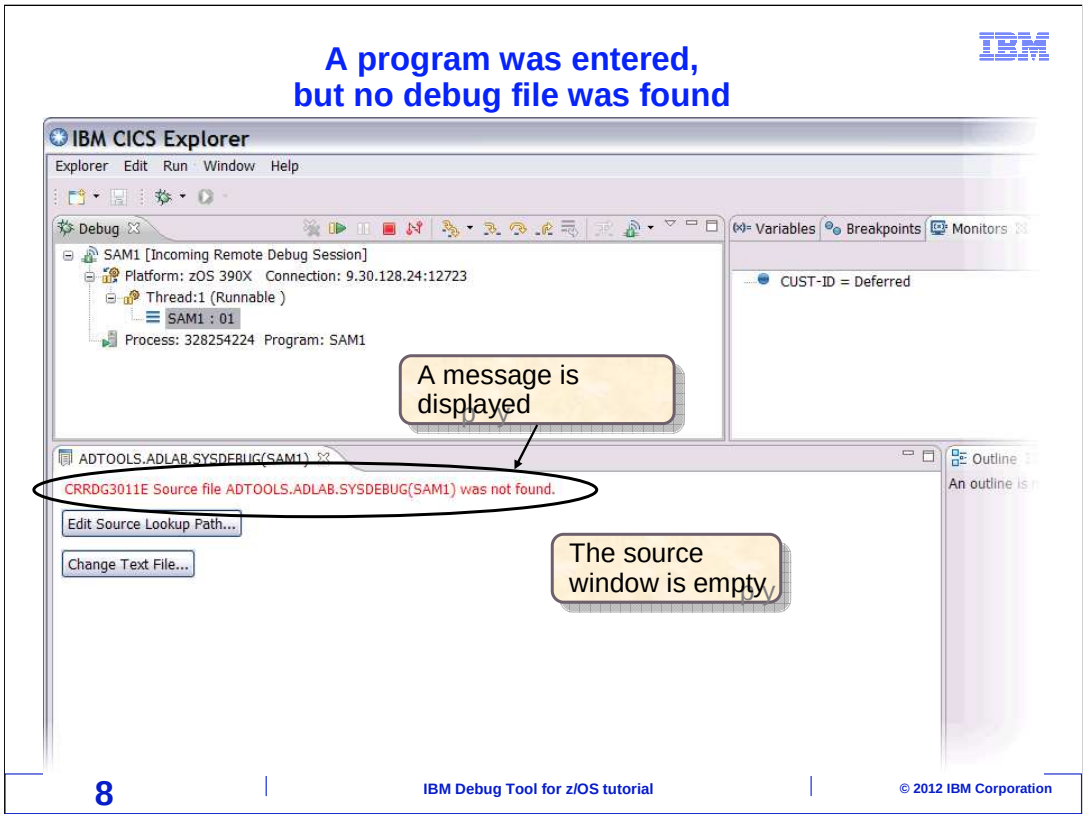


Next, you will see how to load a sysdebug file, a compiler listing, a dwarf file, or a program source file.

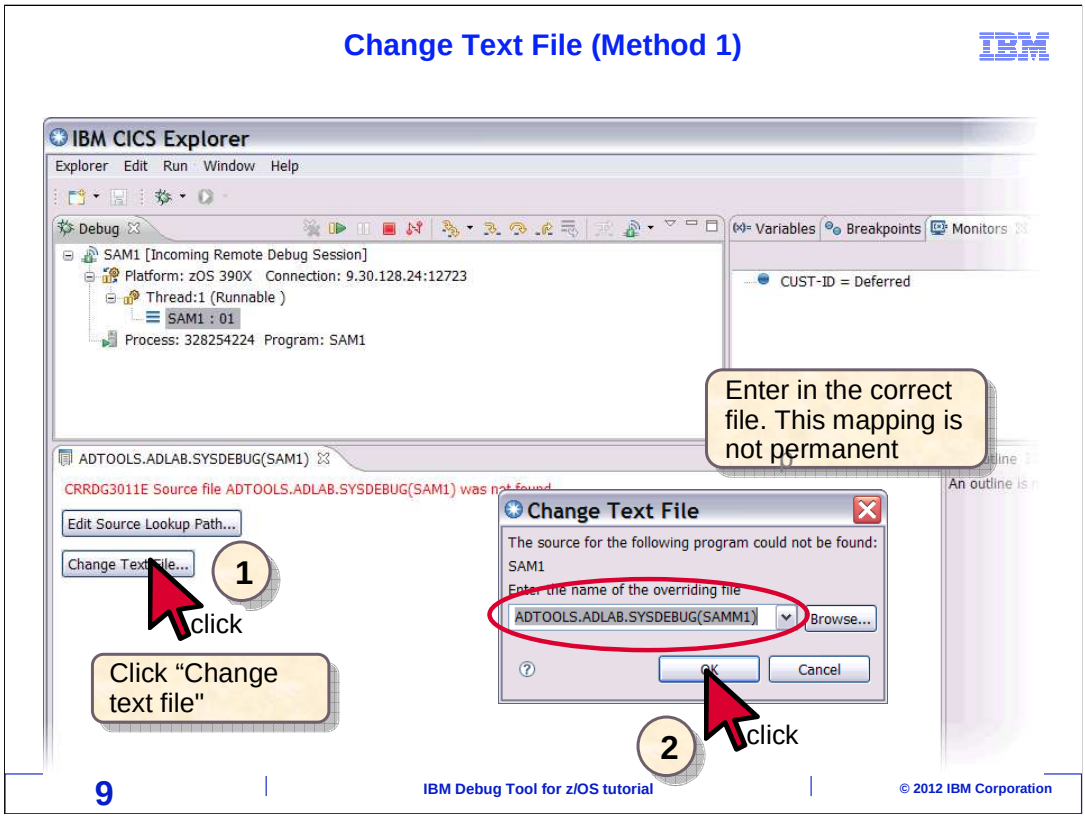
Example of loading sysdebug, listings, dwarf, and source files

- Use these examples 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)

The following examples can be used for programs that load sysdebug files, listings, dwarf files, and source files. These include programs compiled with Enterprise COBOL, Enterprise PLI, XL C/C++, and others.

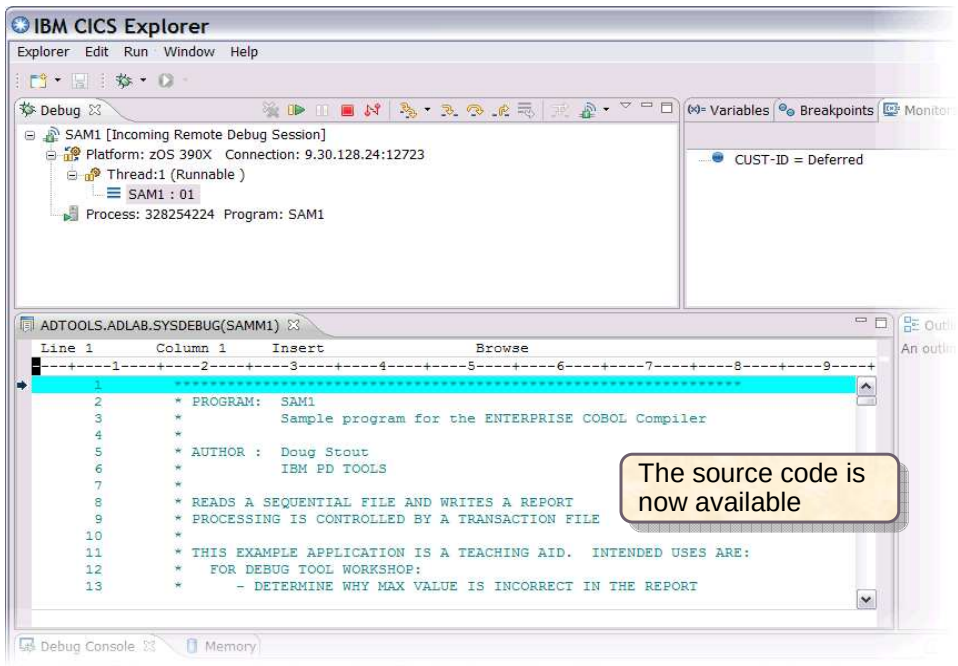


In this example, the debugger started, but it did not find a debug file for the program. The source window is empty, and a message indicates that the source file was not found. Two buttons are displayed: “edit source lookup path” which can be selected to specify a list of libraries, and “Change Text File” which lets you specify the debug file name.



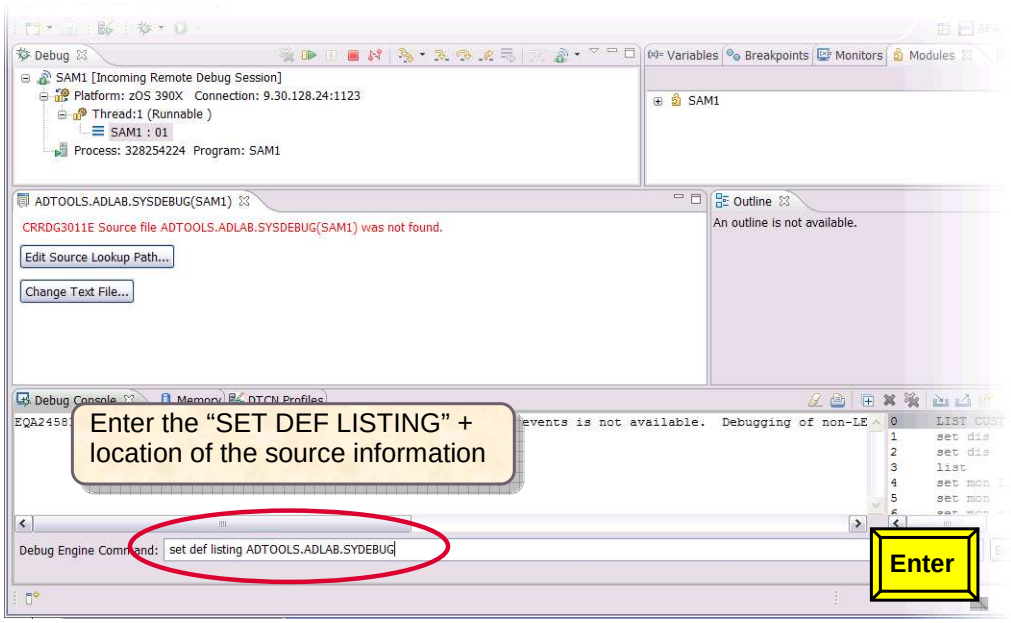
The “Change Text File” button is clicked. In the pop-up, enter the name of the correct debug file for the program, and click “OK”.

The debug file was loaded from the indicated location

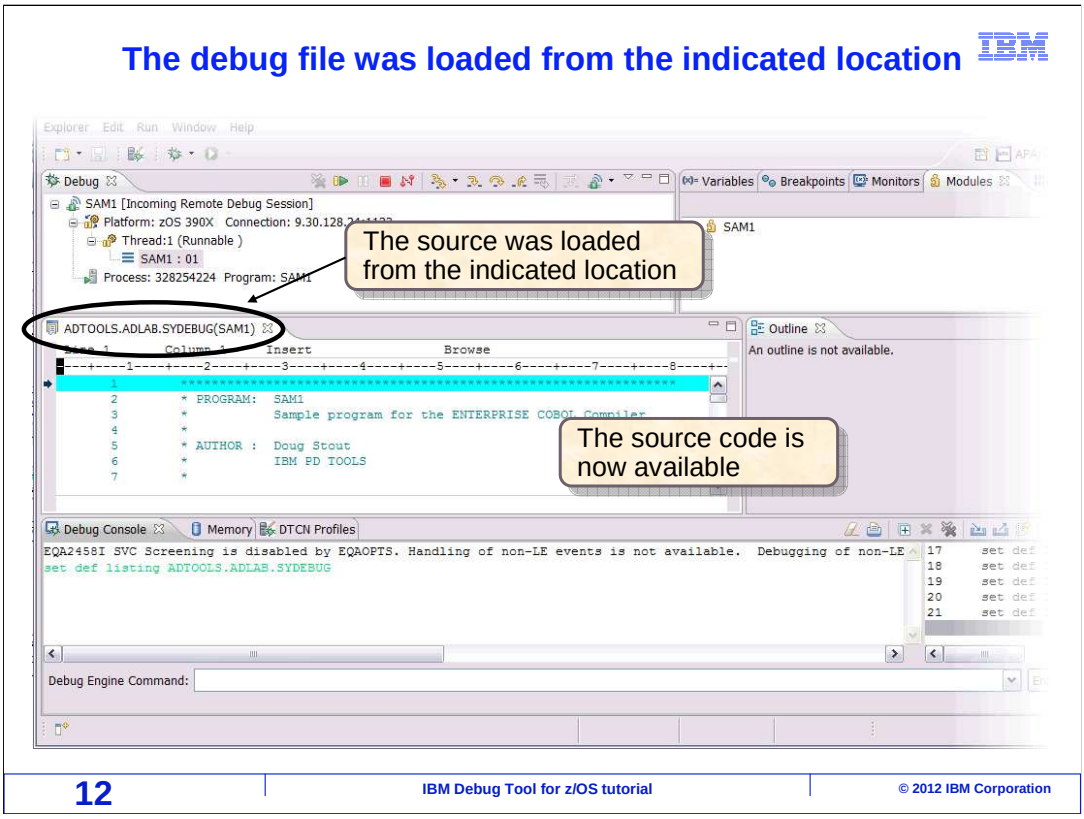


The debugger loads and validates the time stamp of the debug file. If it is correct, the program source is displayed. You can use this method to load the debug file for each program you want to debug.

SET DEF LISTING (Method 2)

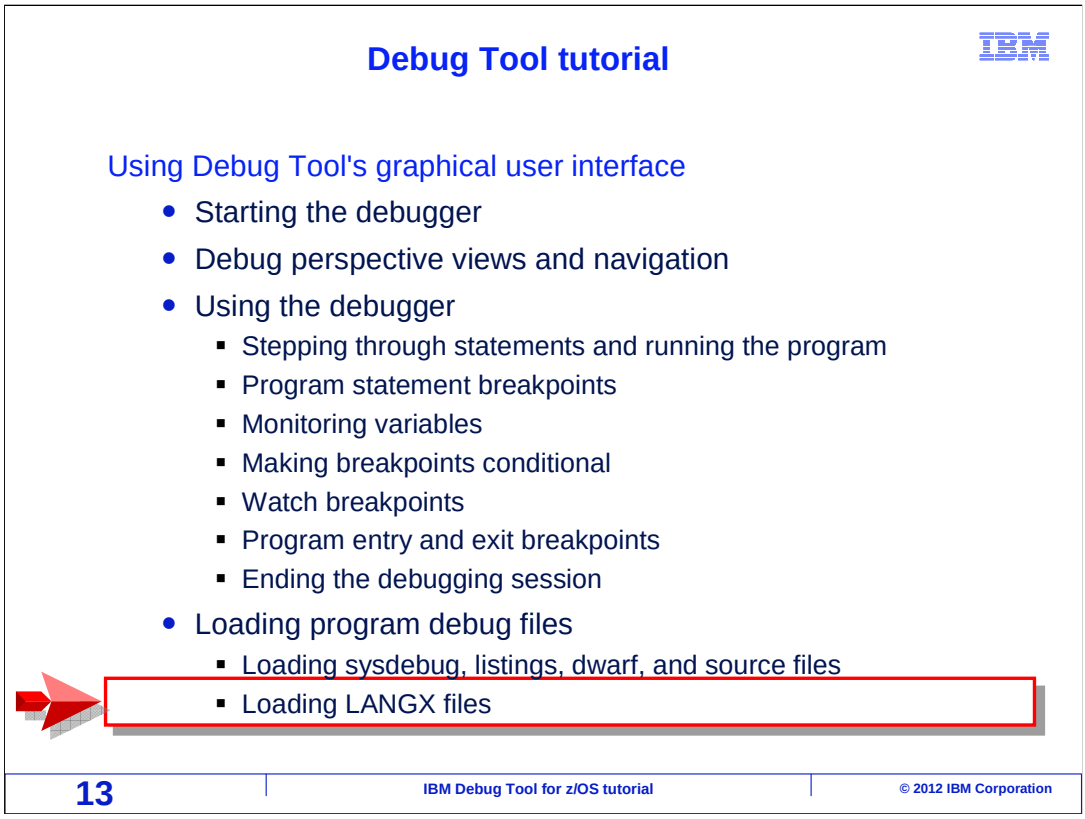


Another way to specify the locations of debug files is with a "SET DEFAULT LISTINGS …" command specifying one or more libraries. In this example, a sysdebug library is specified. Enter only the name of the library or PDS, not the member name. The debugger will automatically look for a matching member name.



When the SET DEFAULT LISTINGS command is entered, the debugger searches the library and loads a matching debug file. Notice that program source code is displayed in the source window now.

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



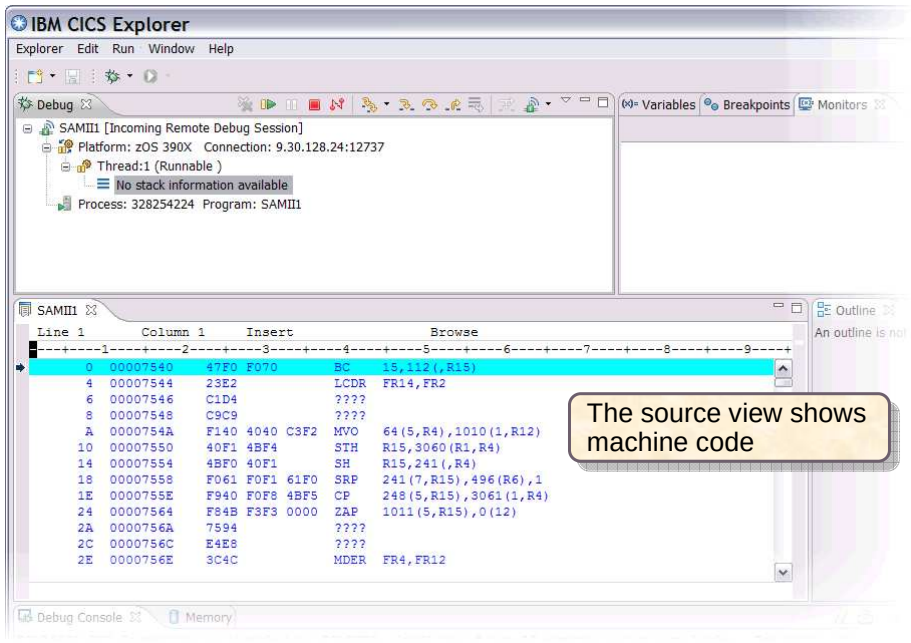
Next, you will see how to load a LANGX file.

Example of loading LANGX debug files

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

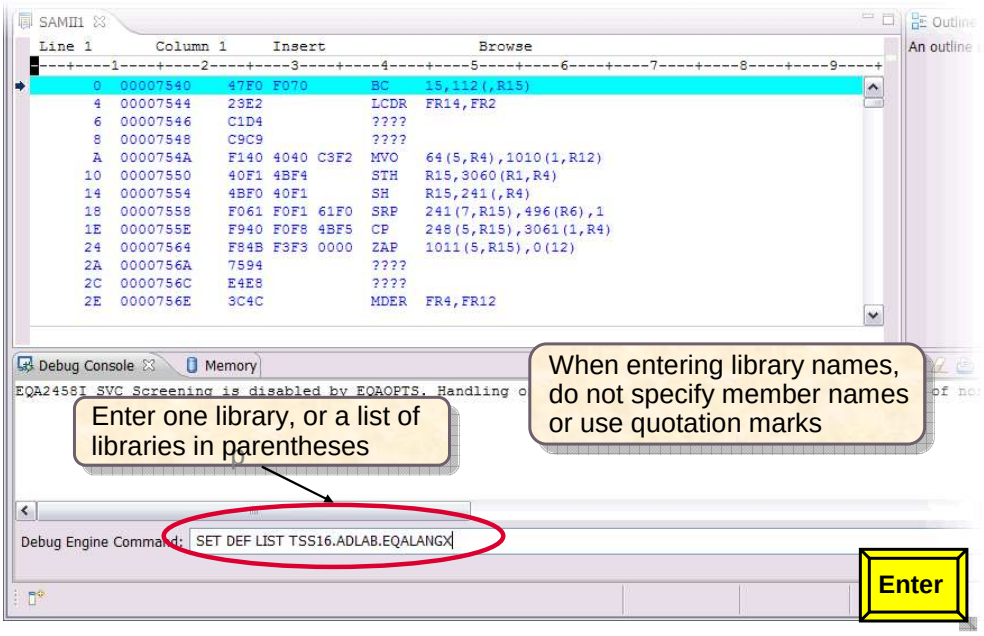
LANGX files are used with programs compiled with Assembler, VS COBOL II, and OS/VS COBOL.

A program was entered, but no Langx file was found

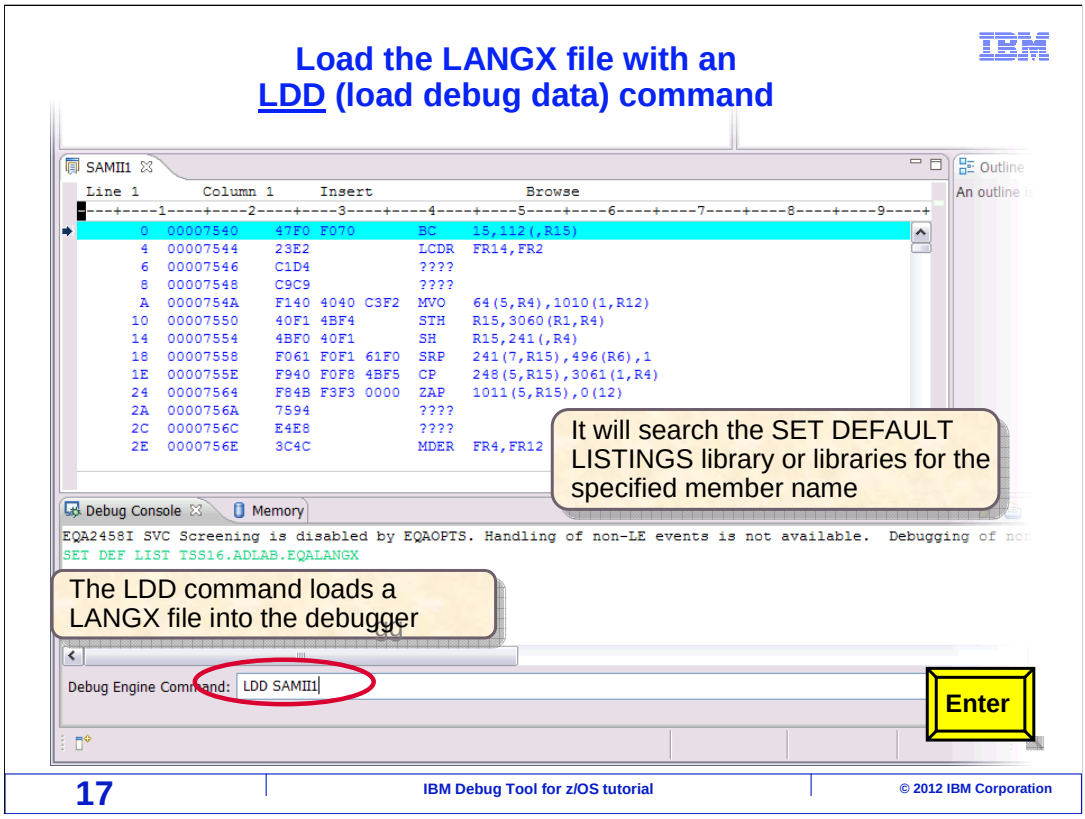


In this example an Assembler program is being debugged, the debug session launched, but the source view is not displaying program source code. Instead, it may display disassembled machine code.

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

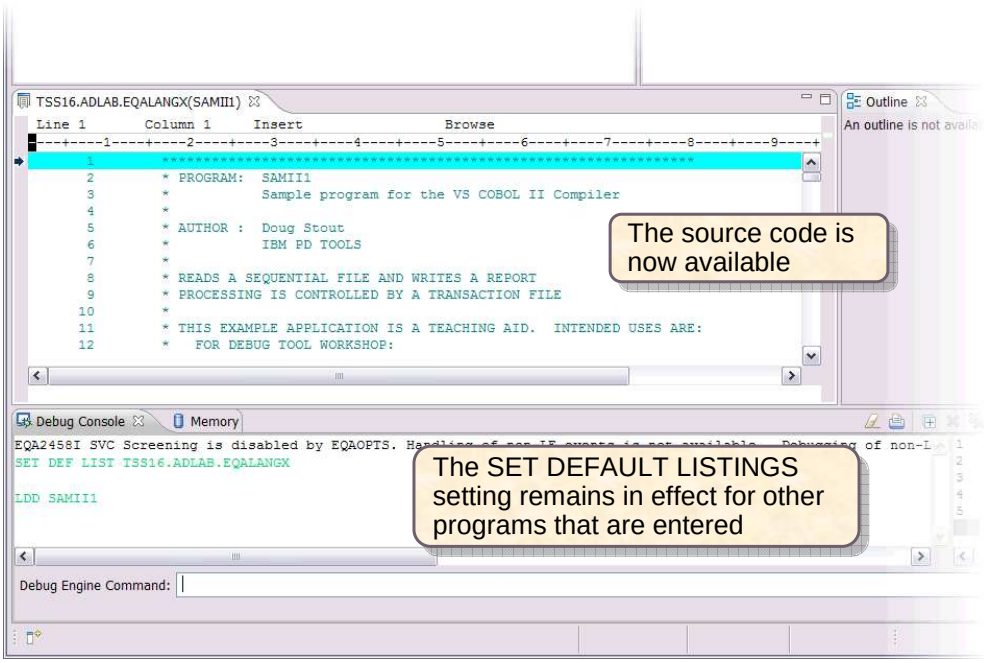


A SET DEFAULT LISTINGS command is entered in the debug console. In this example, a library containing LANGX files is specified. Enter only the name of the PDS or library, not the member name.



Now the search path has been defined. To load the LANGX file into the debugger enter an LDD (load debug data) command with the member name.

The Langx file was loaded from the library



The LDD command searched for the specified member in the library search concatenation, and loaded it. The program source code is now displayed.

That is the end of this section, specifying program files in the GUI debugger.

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_DTv12s20GUILoadingProgramDebugFiles.ppt

This module is also available in PDF format at: ../DTv12s20GUILoadingProgramDebugFiles.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.