



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

### Using Debug Tool's graphical user interface

- Starting the debugger
- Debug perspective views and navigation
- Using the debugger
  - Stepping through statements and running the program
  - Program statement breakpoints
  - Monitoring variables
  - Making breakpoints conditional
  - Watch breakpoints
  - Program entry and exit breakpoints
  - Ending the debugging session
- Loading program debug files
  - Loading sysdebug, listings, dwarf, and source files
  - Loading LANGX files



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

3

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

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.

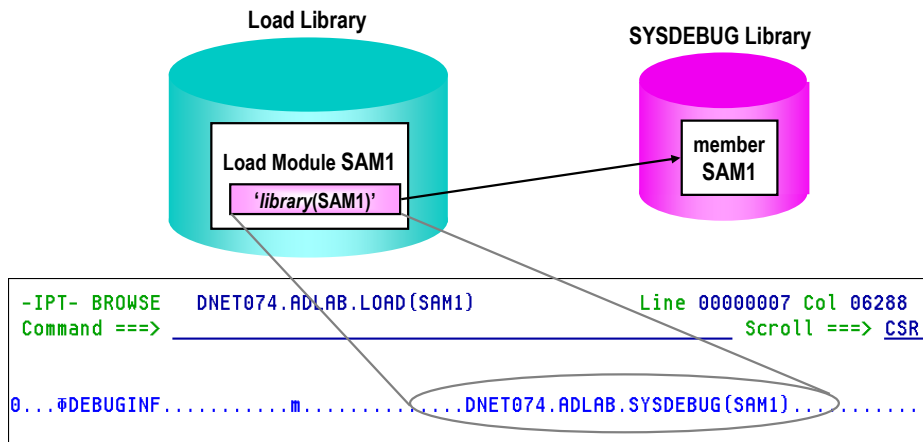
A dwarf or .mdbg file can be used with programs compiled with the XL C/C++ compiler, and the actual program source file itself can optionally be used with XL C/C++ programs and programs compiled with older versions of Enterprise PL/I.

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



4

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

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.

## How the debugger locates debug files



- The debugger searches for debug files in this order:  
(For COBOL, PL/I, assembler, and C/C++ other than .mdbg files)
  - For Enterprise COBOL and Enterprise PL/I programs, it will attempt to load the sysdebug file name that is embedded in the load module
  - Then, if these commands were entered, the files and libraries that they specify are searched:
    - SET SOURCE ...
    - SET DEFAULT LISTINGS ...
  - Then, if an EQADEBUG DD statement was coded in the JCL, the libraries in it's concatenation are searched
- If a matching file is not found automatically, you can enter a command to specify the correct file or library
- For LANGX files, also specify an LDD command to load 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.

### Using Debug Tool's graphical user interface

- Starting the debugger
- Debug perspective views and navigation
- Using the debugger
  - Stepping through statements and running the program
  - Program statement breakpoints
  - Monitoring variables
  - Making breakpoints conditional
  - Watch breakpoints
  - Program entry and exit breakpoints
  - Ending the debugging session
- Loading program debug files
  - Loading sysdebug, listings, dwarf, and source files
  - Loading LANGX files



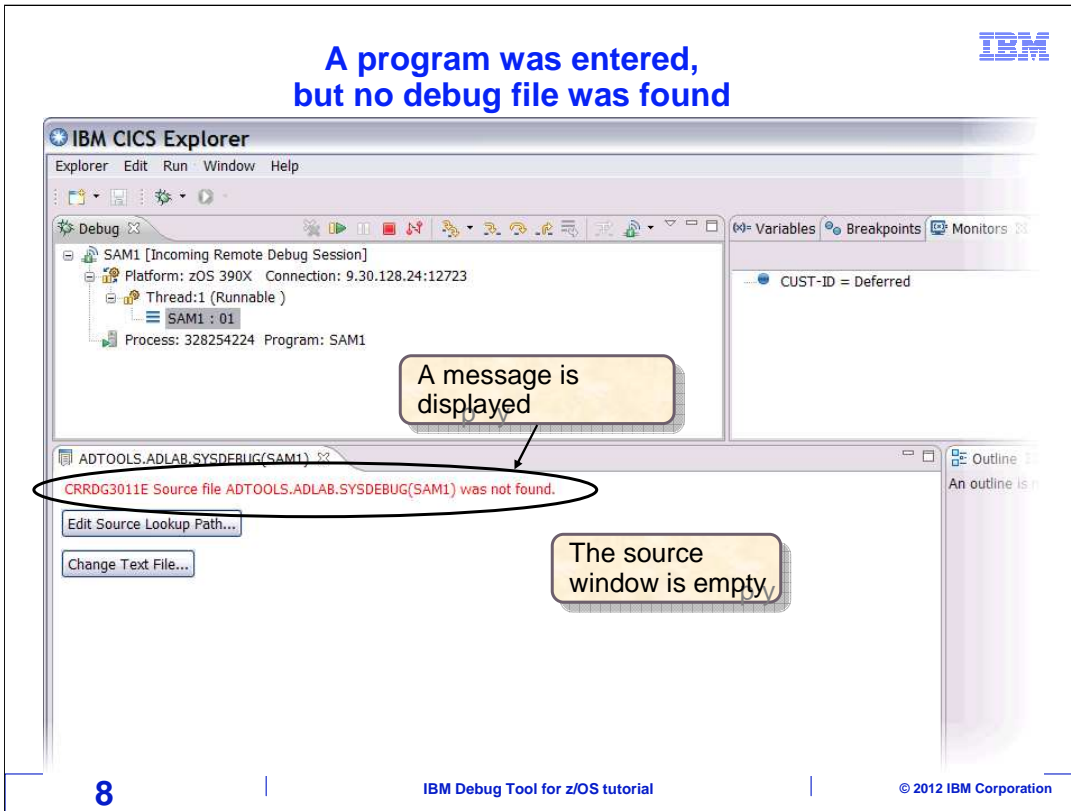
Next, you will see how to load a sysdebug file, a compiler listing, a dwarf file, or a program source file.

## 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.

## A program was entered, but no debug file was found



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.



## Change Text File (Method 1)



Enter in the correct file. This mapping is not permanent

Click "Change text file"

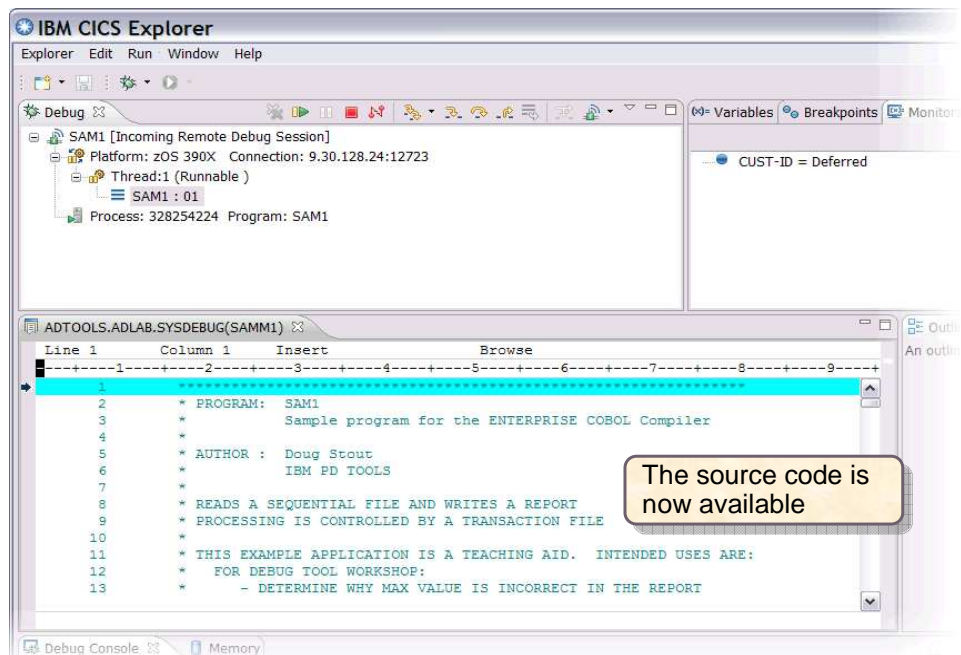
1 click

2 click

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

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



10

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

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.

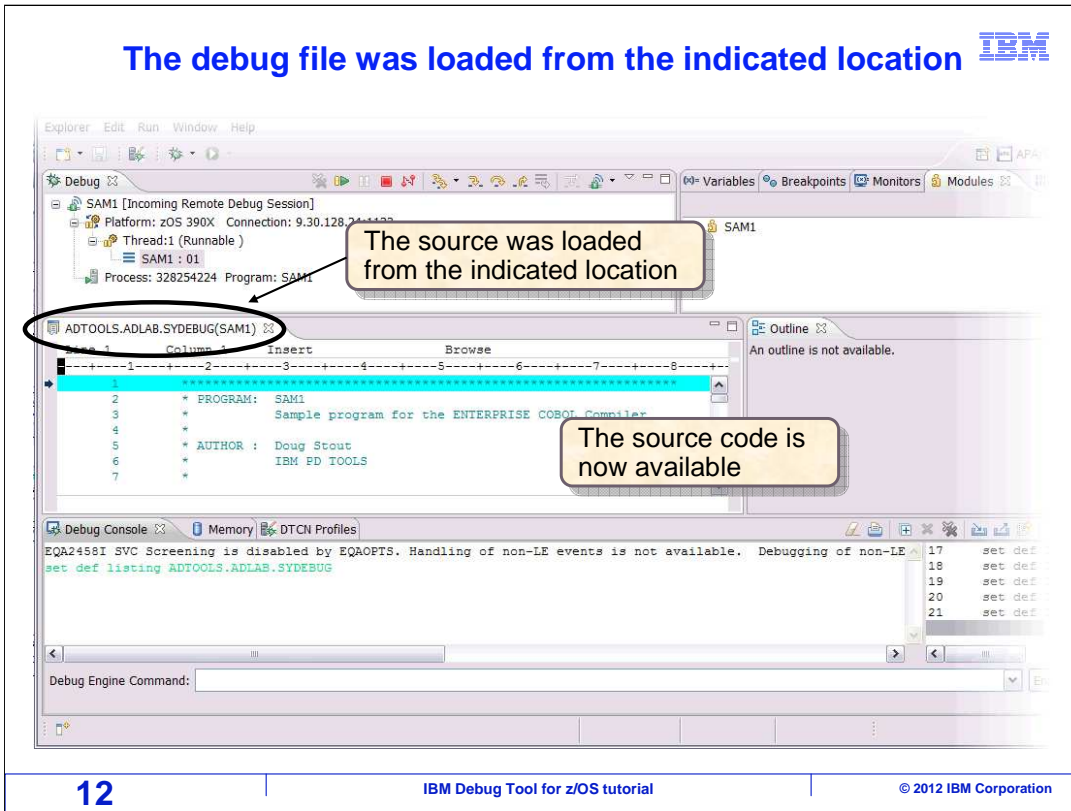
## SET DEF LISTING (Method 2)



The screenshot displays the IBM Debug Tool for z/OS interface. At the top, the title bar reads "SET DEF LISTING (Method 2)" and the IBM logo is in the upper right. The main window is divided into several panes. The top-left pane shows the "Debug" tree with "SAM1 [Incoming Remote Debug Session]" selected, including details for Platform (zOS 390X), Thread (1), and Process (328254224). The top-right pane shows "Variables" with "SAM1". The middle-left pane shows the source file "ADTOOLS.ADLAB.SYSDEBUG(SAM1)" with a red error message: "CRRDG3011E Source file ADTOOLS.ADLAB.SYSDEBUG(SAM1) was not found." Below this are buttons for "Edit Source Lookup Path..." and "Change Text File...". The middle-right pane shows "Outline" with the message "An outline is not available." The bottom-left pane is the "Debug Console" showing the command "set def listing ADTOOLS.ADLAB.SYDEBU" entered in the "Debug Engine Command" field, which is circled in red. A yellow "Enter" button is positioned to the right of the command field. A callout box with a black border and white background points to the command field, containing the text "Enter the 'SET DEF LISTING' + location of the source information". The bottom of the screenshot features a footer with the number "11" on the left, "IBM Debug Tool for z/OS tutorial" in the center, and "© 2012 IBM Corporation" on the right.

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.

## The debug file was loaded from the indicated location



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.

### Using Debug Tool's graphical user interface

- Starting the debugger
- Debug perspective views and navigation
- Using the debugger
  - Stepping through statements and running the program
  - Program statement breakpoints
  - Monitoring variables
  - Making breakpoints conditional
  - Watch breakpoints
  - Program entry and exit breakpoints
  - Ending the debugging session
- Loading program debug files
  - Loading sysdebug, listings, dwarf, and source files
  - Loading LANGX files



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

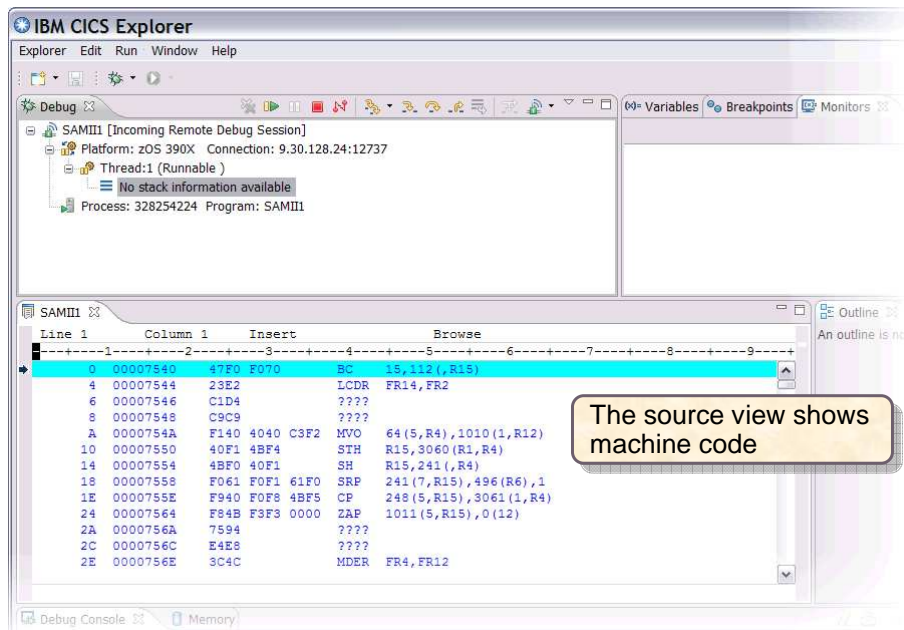
## Example of loading LANGX debug files



- 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



15

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

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



The screenshot shows the SAMIII debug console interface. The main window displays assembly code with columns for Line 1, Column 1, Insert, and Browse. The code includes instructions like BC, Lcdr, SH, MVO, SRP, CP, and ZAP. Below the code is the Debug Console window, which contains the command: `SET DEF LIST TSS16.ADLAB.EQALANGX`. A red circle highlights the command text, and a yellow box with the text "Enter" is positioned to the right of the command line. Two callout boxes provide instructions: "Enter one library, or a list of libraries in parentheses" and "When entering library names, do not specify member names or use quotation marks".

Line 1	Column 1	Insert	Browse
0	00007540	47F0 F070	BC 15,112(,R15)
4	00007544	23E2	LCDR FR14,FR2
6	00007546	C1D4	????
8	00007548	C9C9	????
A	0000754A	F140 4040 C3F2	MVO 64(5,R4),1010(1,R12)
10	00007550	40F1 4BF4	STH R15,3060(R1,R4)
14	00007554	4BF0 40F1	SH R15,241(,R4)
18	00007558	F061 F0F1 61F0	SRP 241(7,R15),496(R6),1
1E	0000755E	F940 F0F8 4BF5	CP 248(5,R15),3061(1,R4)
24	00007564	F84B F3F3 0000	ZAP 1011(5,R15),0(12)
2A	0000756A	7594	????
2C	0000756C	E4E8	????
2E	0000756E	3C4C	MDER FR4,FR12

Debug Engine Command: `SET DEF LIST TSS16.ADLAB.EQALANGX`

Enter

16

IBM Debug Tool for z/OS tutorial

© 2012 IBM Corporation

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.



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

The screenshot shows the IBM Debug Tool for z/OS interface. At the top, a disassembly window displays assembly code for member SAMIII. The code includes instructions like MVO, STH, SH, SRP, CP, and ZAP with their respective registers and addresses. Below the disassembly window, the Debug Console shows the command 'LDD SAMIII' entered in the 'Debug Engine Command' field. A yellow callout box points to the command line, stating 'The LDD command loads a LANGX file into the debugger'. Another yellow callout box points to the disassembly window, stating 'It will search the SET DEFAULT LISTINGS library or libraries for the specified member name'. A yellow 'Enter' button is located to the right of the command line.

Line 1	Column 1	Insert	Browse
0	00007540	47F0 F070	BC 15,112(,R15)
4	00007544	23E2	LCDR FR14,FR2
6	00007546	C1D4	????
8	00007548	C9C9	????
A	0000754A	F140 4040 C3F2	MVO 64(5,R4),1010(1,R12)
10	00007550	40F1 4BF4	STH R15,3060(R1,R4)
14	00007554	4BF0 40F1	SH R15,241(,R4)
18	00007558	F061 F0F1 61F0	SRP 241(7,R15),496(R6),1
1E	0000755E	F940 F0F8 4BF5	CP 248(5,R15),3061(1,R4)
24	00007564	F84B F3F3 0000	ZAP 1011(5,R15),0(12)
2A	0000756A	7594	????
2C	0000756C	E4E8	????
2E	0000756E	3C4C	MDER FR4,FR12

Debug Engine Command: LDD SAMIII Enter

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 screenshot displays the IBM Debug Tool interface. The top window, titled 'TSS16.ADLAB.EQALANGX(SAMIII)', shows the source code of a program. The code is as follows:

```
Line 1      Column 1      Insert      Browse
-----1-----2-----3-----4-----5-----6-----7-----8-----9-----
1  *****
2  * PROGRAM:  SAMIII
3  *          Sample program for the VS COBOL II Compiler
4  *
5  * AUTHOR :  Doug Stout
6  *          IBM PD TOOLS
7  *
8  * READS A SEQUENTIAL FILE AND WRITES A REPORT
9  * PROCESSING IS CONTROLLED BY A TRANSACTION FILE
10 *
11 * THIS EXAMPLE APPLICATION IS A TEACHING AID.  INTENDED USES ARE:
12 * FOR DEBUG TOOL WORKSHOP:
```

A yellow callout box points to the source code with the text: "The source code is now available".

The bottom window, titled 'Debug Console', shows the following output:

```
EQAL2458I SVC Screening is disabled by EQAOPTS. Handling of non-IE events is not available. Debugging of non-I
SET DEF LIST TSS16.ADLAB.EQALANGX
LDD SAMIII
```

A yellow callout box points to the 'SET DEF LIST' command with the text: "The SET DEFAULT LISTINGS setting remains in effect for other programs that are entered".

At the bottom of the screenshot, there is a footer with the number '18', the text 'IBM Debug Tool for z/OS tutorial', and the copyright notice '© 2012 IBM Corporation'.

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](mailto:iea@us.ibm.com?subject=Feedback_about_DTv12s20GUILoadingProgramDebugFiles.ppt)

This module is also available in PDF format at: [../DTv12s20GUILoadingProgramDebugFiles.pdf](.../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](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.