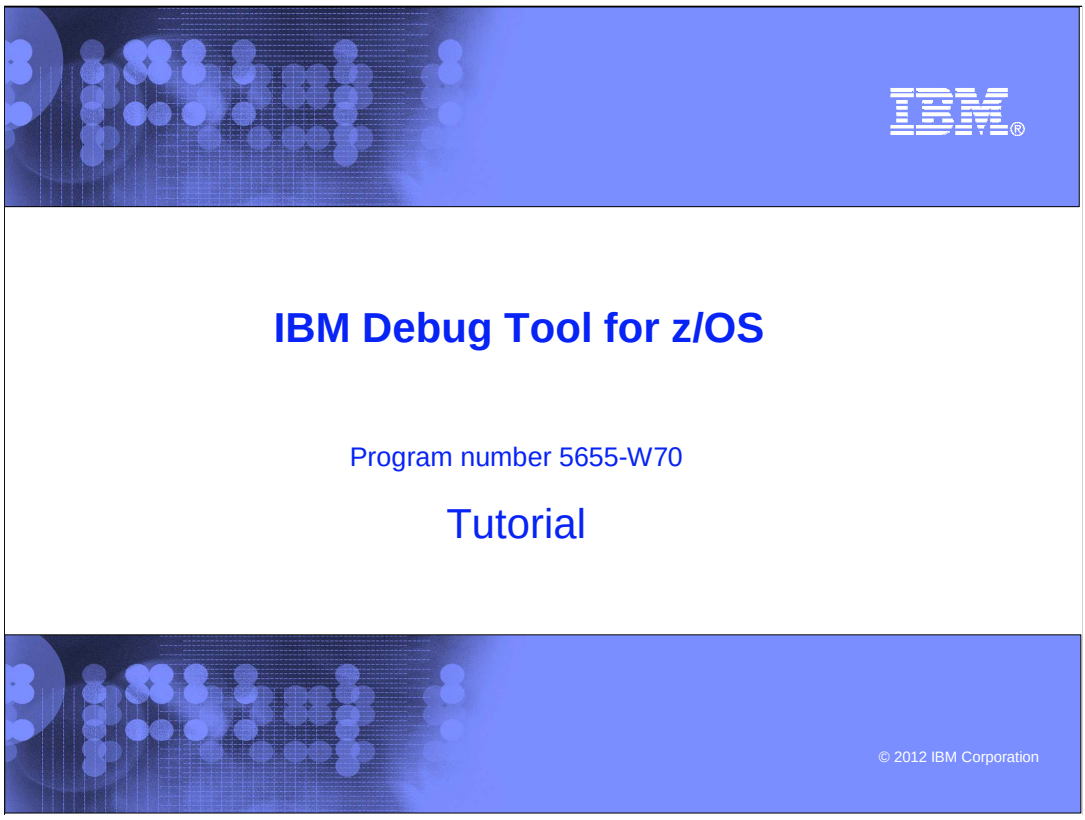
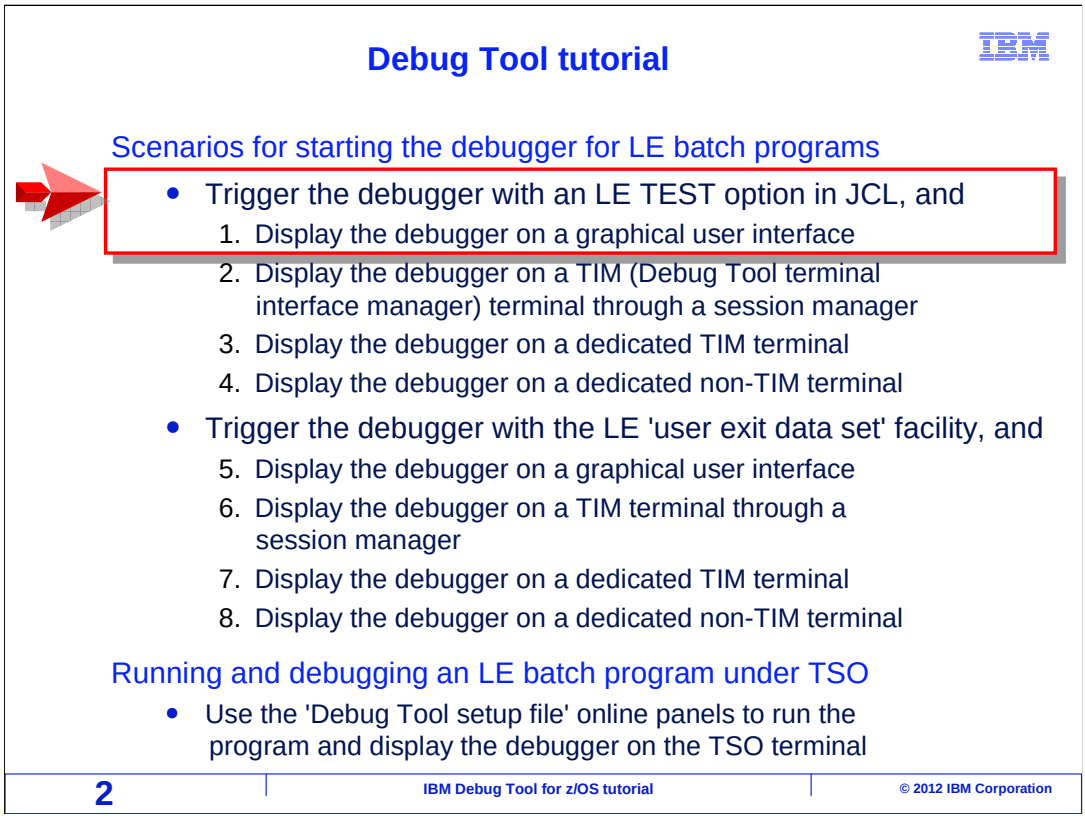


This is the tutorial for IBM Debug Tool for z/OS®, one of the IBM zSeries® problem determination tools.



In this section, you will see a scenario for starting Debug Tool for a batch application. In this scenario, a graphical interface is used, the application runs as a batch job, and a TEST option is coded in the JCL to trigger the debugger. You can skip this section if you do not plan to use Debug Tool this way on your system.

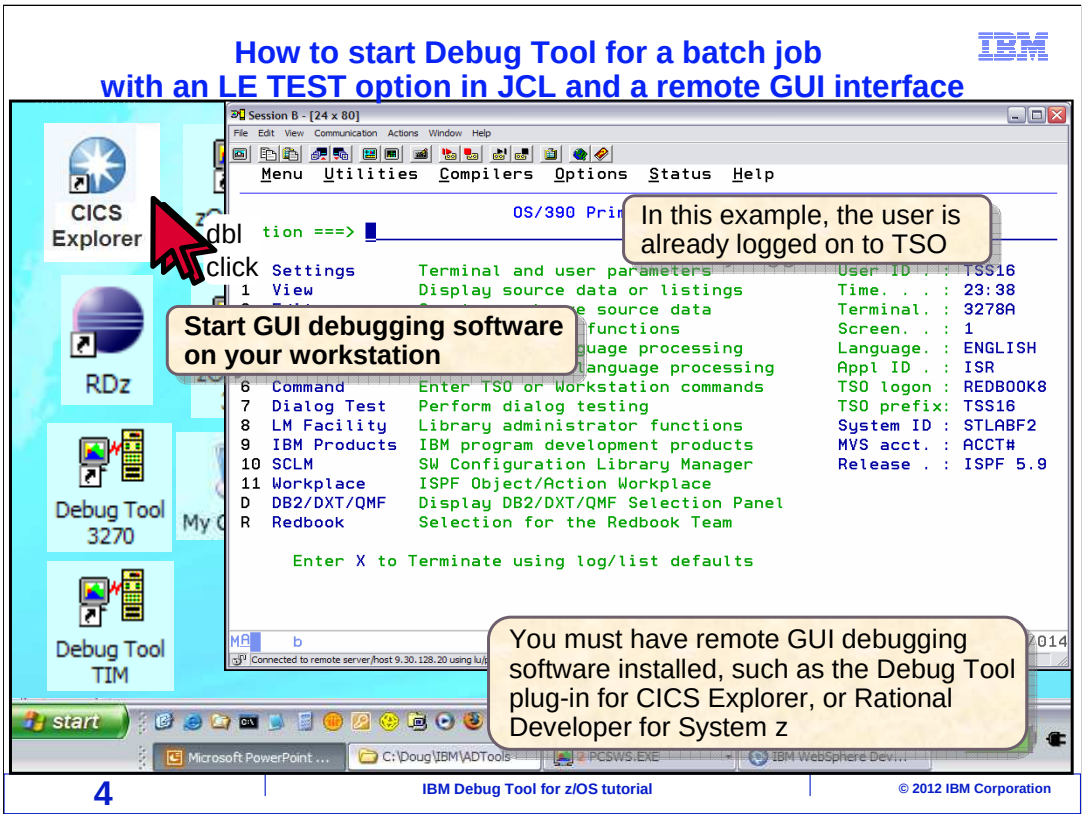
Debug in batch using a TEST option in JCL and a GUI interface

- Description
  - A TEST(…) option is coded in the application's run-time JCL to trigger the debugger
  - The debugger displays on GUI debugging software on your workstation
- This method can be used:
  - To debug Language Environment (LE) programs running in batch jobs
    - including programs that access IMS™, DB2®, or other types of databases
    - and non-LE programs that run in the call chain under an LE program
  - If compatible GUI debugging software is installed on your workstation
    - Such as Rational® Developer for System z®, or CICS Explorer® with the Debug Tool plug-in
- When not to use this method:
  - If GUI debugging workstation software is not available, use a terminal interface instead
  - On older versions of z/OS this method may not work for IMS programs. In that case, use the 'Debug Tool user exit data set' facility instead.

This is a commonly used, simple method to start the debugger and display it on GUI debugging software on your workstation. You set a trigger, to start the debugger when the application runs, by coding a Language Environment (LE) TEST option in the program's run-time JCL.

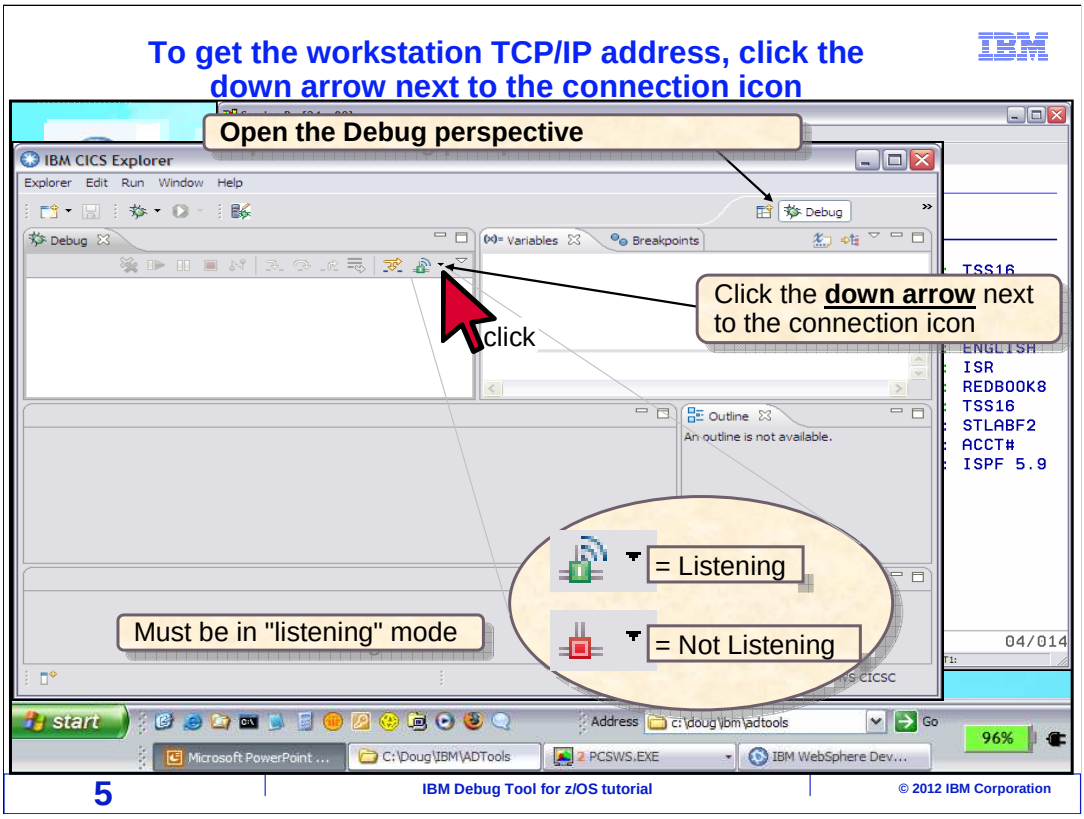
This method can be used to debug LE-conforming programs running in a batch job, including programs that access DB2, IMS, and other types of databases. Even non-LE subroutines can be debugged using this method, if there is at least one LE program higher in the call chain. Software such as Rational Developer for System z or CICS Explorer with the Debug Tool plug-in must be installed on your workstation.

This is generally the best method when a GUI is used. However, you cannot use this method if compatible GUI debugging software is not installed. Also, if you are running on an older version of z/OS (1.7 or earlier), a TEST option may not trigger the debugger for IMS batch programs. In that case, the 'user exit data set' method could be used instead.



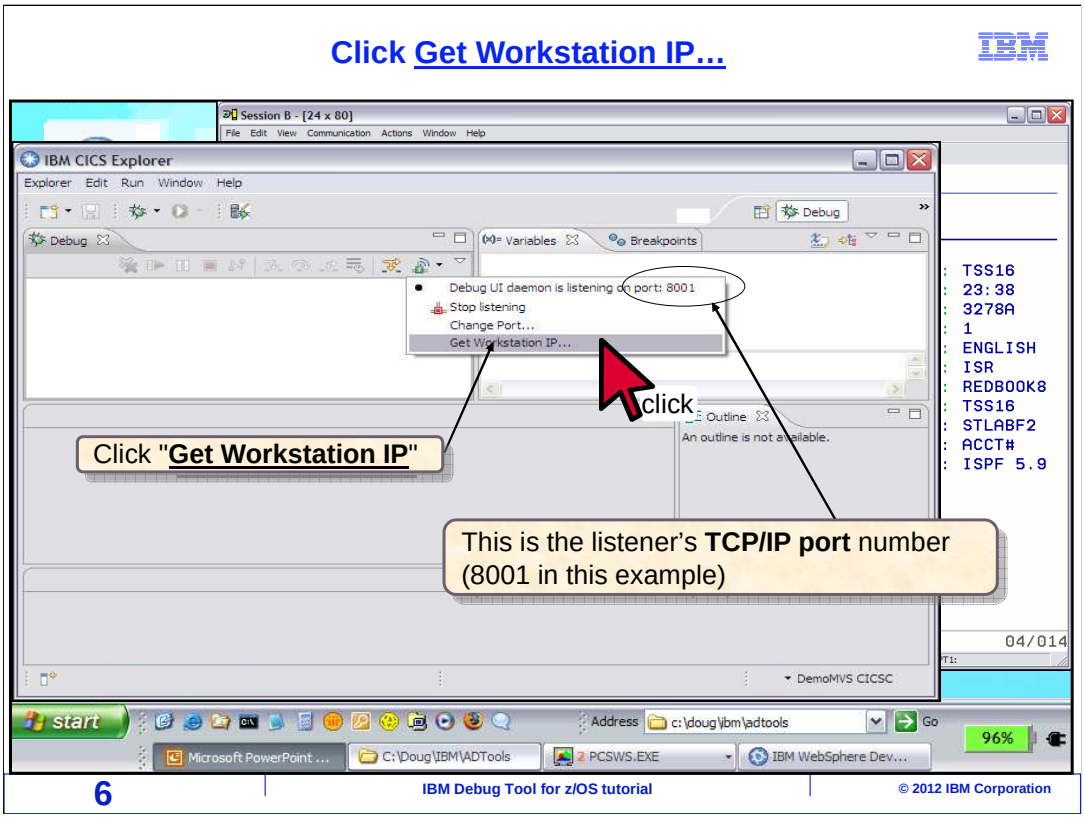
In this example, the user is already logged on to TSO. The first step is to start the remote GUI debugging software on your workstation. This could be software such as Rational Developer for System z or CICS Explorer with the Debug Tool plug-in.

Here, CICS Explorer is started.

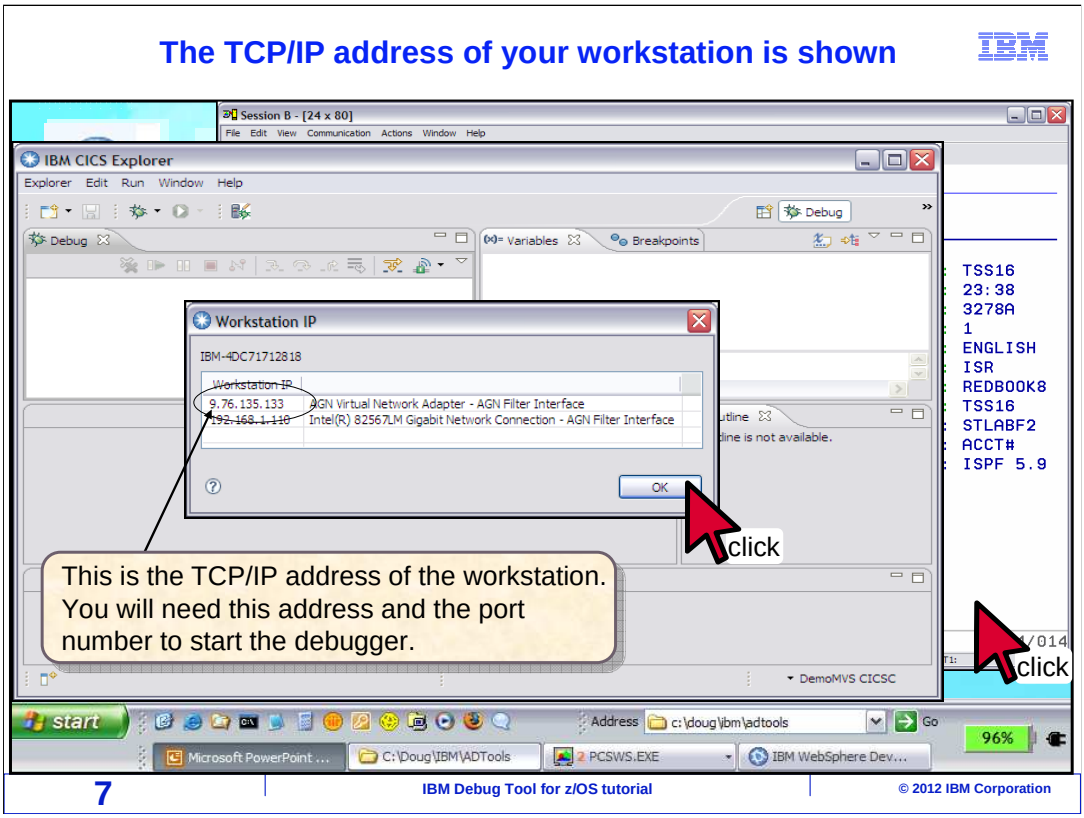


That started CICS Explorer. If the "Debug" view is not displayed, switch to the Debug perspective. It must be in “listening” mode. There is an icon on the toolbar that shows the listening status. If it is not listening, the listener icon is red, and you can click the icon to switch it on. The icon is green when the listener is on.

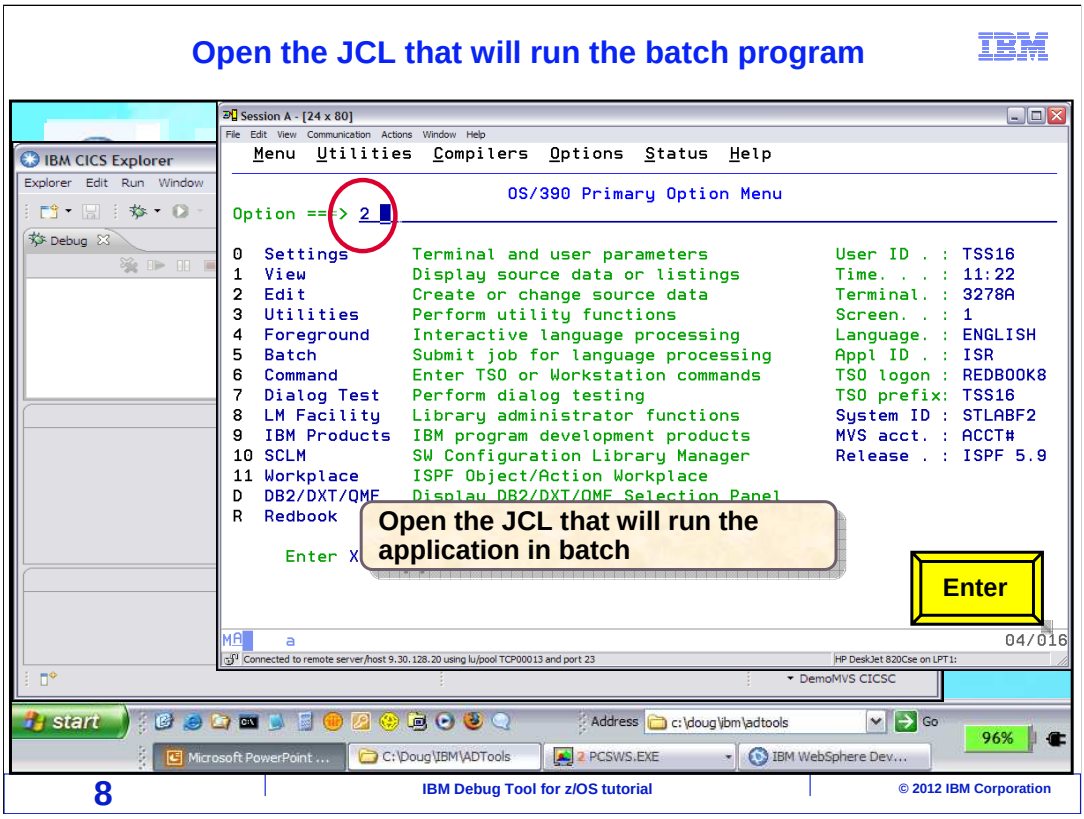
You will need to determine your workstation's IP address. Click the small black down arrow next to the listening icon.



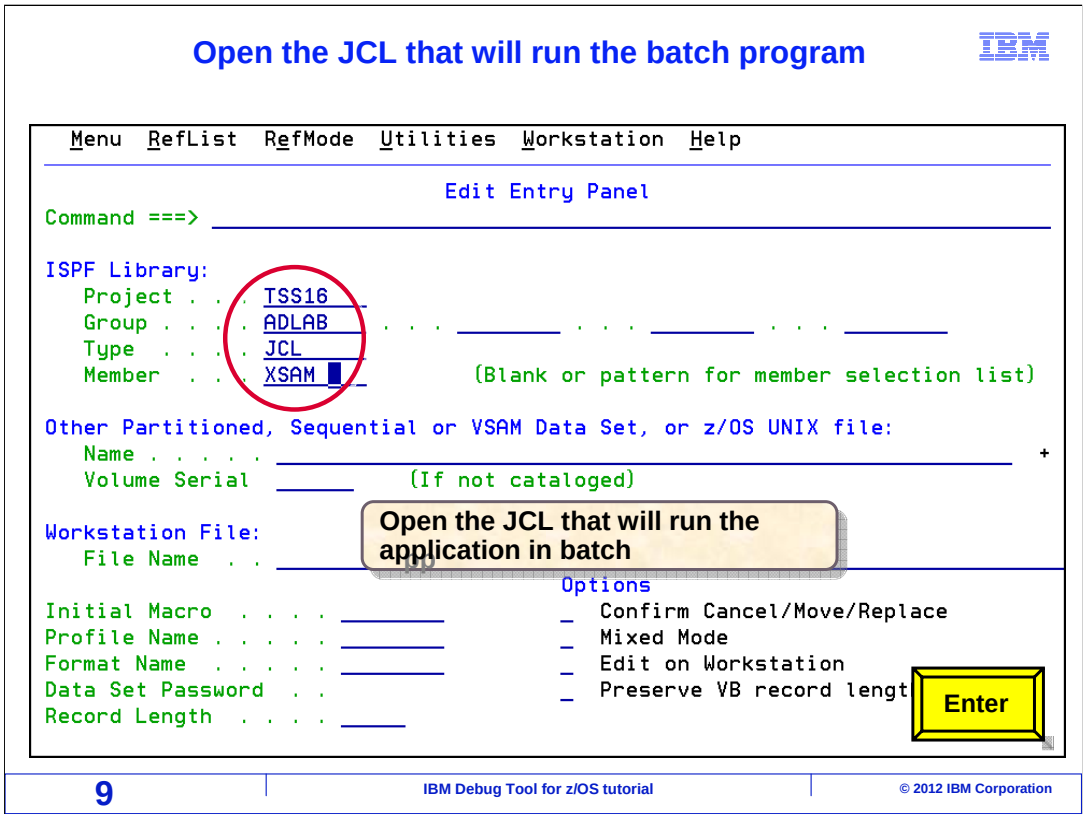
That displays a menu. Make a note of the listener's IP port. In this example, it is 8001. Next, click "Get Workstation IP".



That displays your workstation's IP address. Make a note of it. At this point, you have prepared the remote GUI debugging software, and it ready to receive a debugging session. Click OK to close the pop-up window. Now go back to TSO by clicking your TSO terminal session.

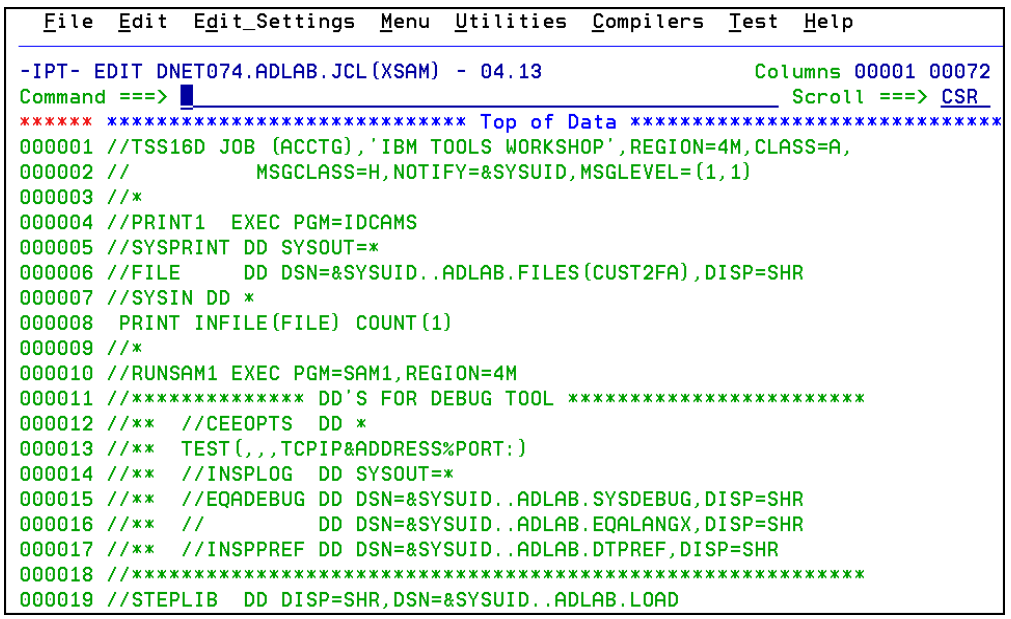


In TSO, open the JCL that you use to run the application you want to debug. In this example, the ISPF editor is selected.

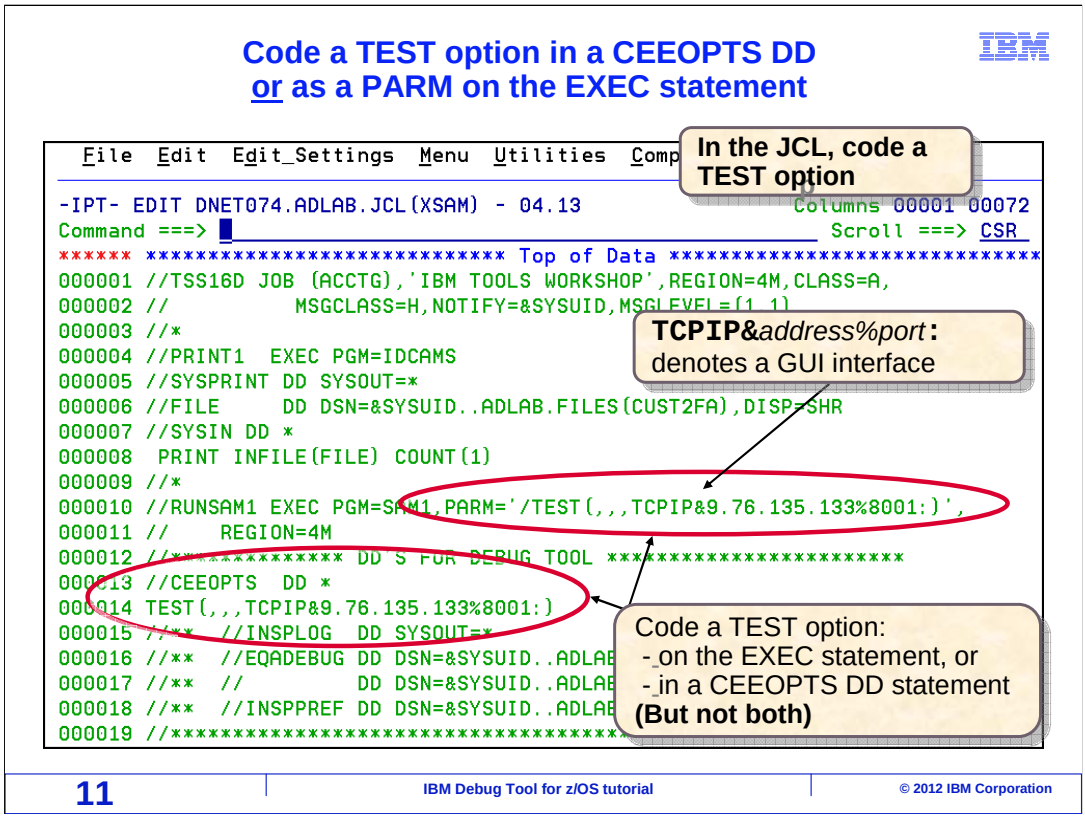


And open the JCL in the editor.

This is the JCL before adding a TEST option



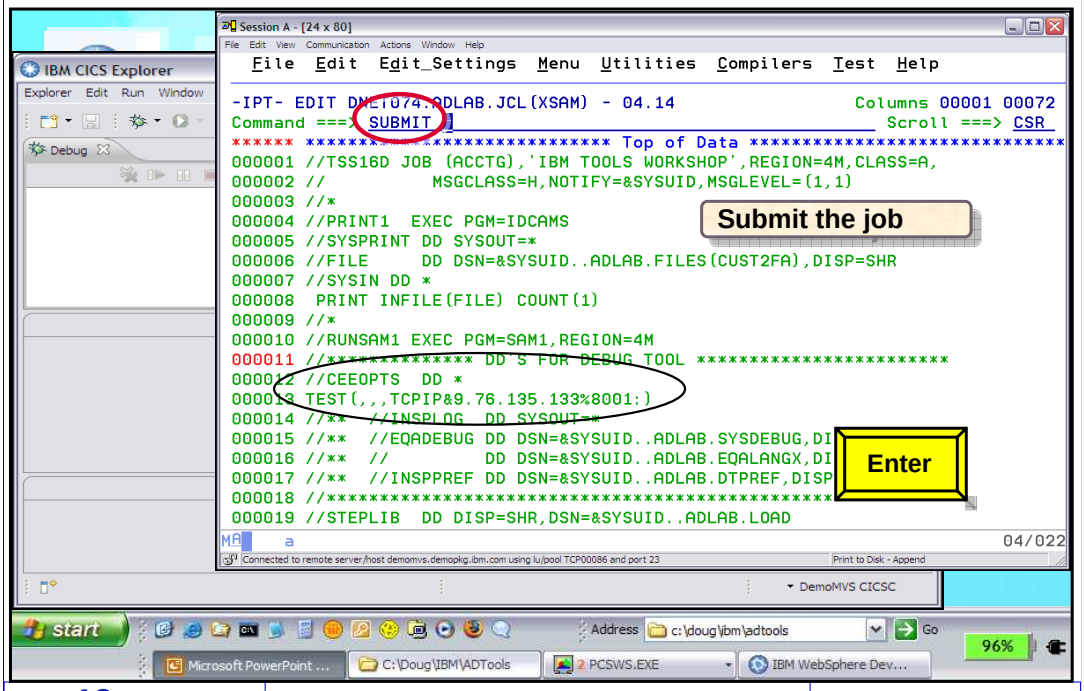
This is JCL that is used to run the application program in this example. It is shown here before any changes have been made.



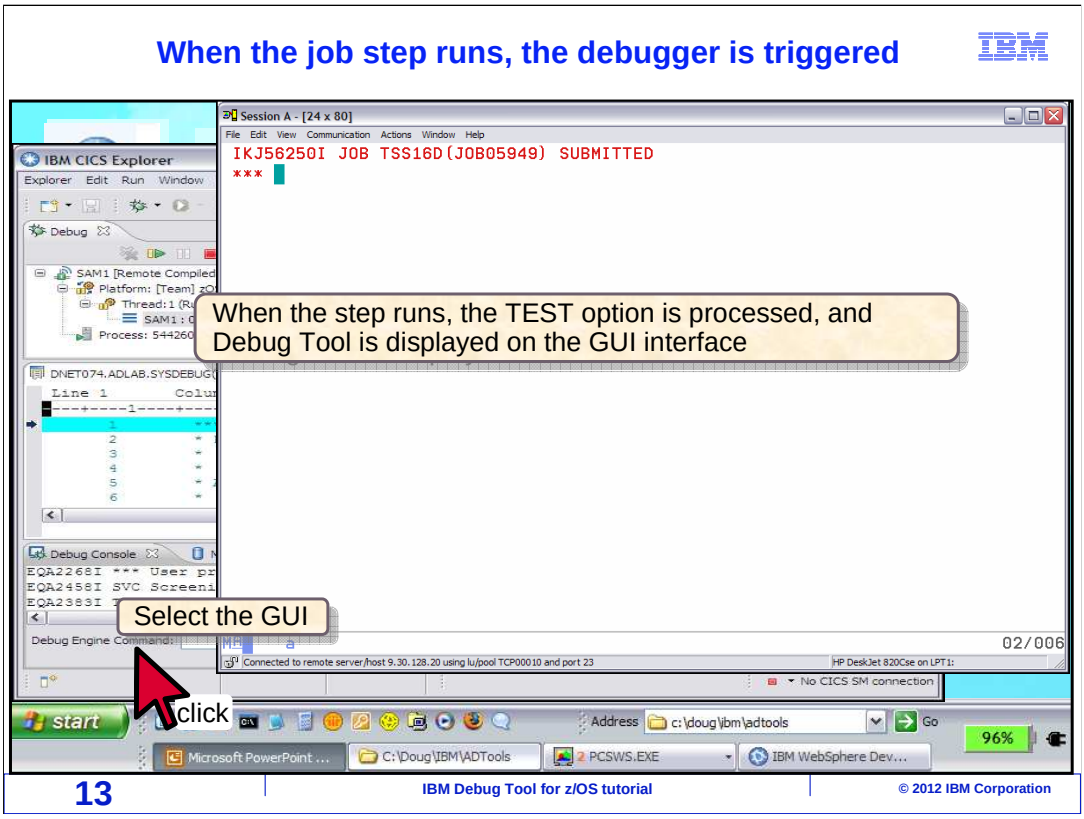
To set a trigger for the debugger, code a Language Environment TEST option in the JCL. There are two ways to do it. Depending on how the JCL is coded, you may be able to code a TEST option directly on the EXEC statement that runs the program. Or you can add a special DD statement called CEEOPTS, and code a TEST option as input data in the CEEOPTS DD. Although both methods are shown in this example, only code one or the other.

Notice how the TEST option is coded. After the third comma, it has "TCPIP&tcp address%listener port:". The keyword “TCPIP” specifies that a remote GUI interface will be used. The IP address is coded after the ampersand. This is the IP address of your workstation, where the remote GUI debugging software is listening. The listener port number is coded after the percent sign and before a colon.

Submit the JCL to run the batch job

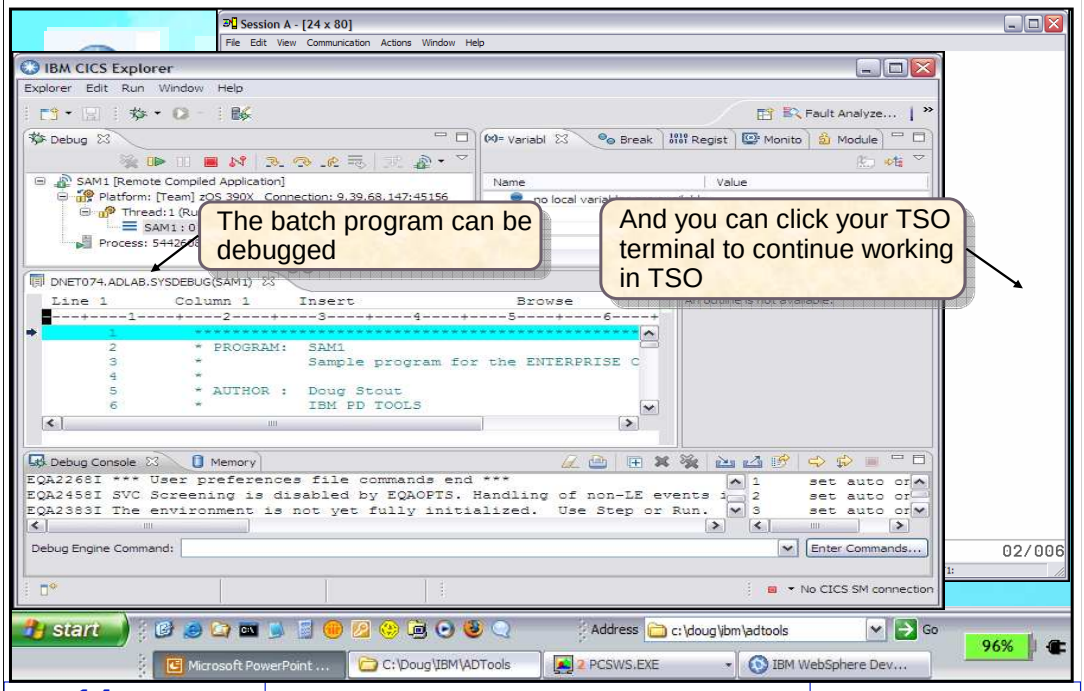


The job is ready to run. The JCL is submitted to batch.



When a step runs that has a TEST option specified, Language Environment receives the TEST option and starts Debug Tool. The debugger is displayed by the GUI debugging software. Here, the remote debugging window is selected by clicking it.

The debugger automatically displays



The batch job is running on the host z/OS system, and Debug Tool is communicating over the network with the GUI software. Notice that you can control the program from the GUI debugger, but you can also click your TSO terminal window to continue working in TSO at the same time. Be aware that the TSO session is no longer needed. You could log off from TSO and continue to debug, because Debug Tool is controlling the application that is running in its own batch address space.

Choosing between the CEEOPTS DD or EXEC parm

- There are two ways to code a TEST option in JCL to start Debug Tool:
  - in a CEEOPTS DD, or
  - on the EXEC statement
- Comparing the two methods:
  - Both are easy, but CEEOPTS can be easier because:
    - it is simpler to cut and paste into JCL without making a syntax mistake
    - It can be used with a JCL PROC without changing the PROC
    - It will trigger the debugger when the first LE program runs, even if it is a subprogram
      - but a TEST option on the EXEC statement will *only* trigger if the *main* program is an LE program

There are two ways to code a TEST option in JCL to start Debug Tool. On the EXEC statement, or with a CEEOPTS DD statement. For most people, the CEEOPTS DD method is a little easier. Here is why.

When you use the CEEOPTS method, it can be easier to “cut and paste” a TEST option into JCL from other JCL where you have coded it before. Also, if a cataloged JCL PROC is used, you may not be able to code a parm on the EXEC statement, because it is embedded in the PROC. With a CEEOPTS DD, it is easy to code it in your JCL to make it take effect in any step you want in your PROC. You will see an example of that in a minute.

Finally, when you use CEEOPTS, Debug Tool will trigger when the first LE program is called, even if it is a subprogram. That is an advantage over the other method, because when you code a TEST option on the EXEC statement, Debug Tool will only trigger if the main program is an LE program.

A CEEOPTS DD with a TEST(…) option can be used to trigger the debugger

- When LE initializes, it looks for a CEEOPTS DD statement
  - If CEEOPTS is present in the JCL step, LE reads it to retrieve run-time options
  - If a TEST(…) option is present, LE starts the debugger
- Considerations
  - LE version 1.7 or later is required (shipped with z/OS V1.7)
  - LE will *not* read a CEEOPTS DD :
    - In older versions of z/OS:
      - with IMS applications (batch or IMS/TM)
      - when the LE Library Routine Retention facility (LRR) is used

When you run an LE program, which includes programs compiled with compilers such as Enterprise COBOL and Enterprise PL/I, Language Environment initializes when the program starts. When LE initializes, as part of its normal processing, it looks for a CEEOPTS DD statement. If it finds one, it reads it to retrieve run-time options. When you pass a TEST option to LE, it will start Debug Tool.

But there are a couple of considerations and restrictions. First, you must be at LE version 1.7 or later. Since LE is shipped with z/OS, typically that means that the z/OS operating must be at version 1.7 or later.

LE will not read a CEEOPTS DD statement with IMS applications on older versions of z/OS. So, if you want to debug an IMS batch application and you are running on an older system, you cannot use a TEST option in JCL to do it. In that case, use the 'Debug Tool user exit data set facility' to trigger the debugger.

Options for coding a CEEOPTS DD

- Example:

```
//STEP5  EXEC PGM=MYPROG
//CEEOPTS DD *
TEST(,,,TCPIP&address%port:)
```

  - where *address* = the IP address of your workstation, and
  - *port* = the listening port number configured in the GUI interface
  - TCPIP directs the debugger to use a GUI interface. It will connect to the interface on the workstation at the IP address.

- You can use a file instead of in-stream data:

```
//CEEOPTS DD DSN=MY.CEEOPTS.FILE,DISP=SHR
```

- When executing a JCL PROC, use the *stepname*.CEEOPTS syntax to name the step you want to debug:

```
//RUNPROC EXEC PROC99
//ASTEP.CEEOPTS DD *
TEST(,,,TCPIP&address%port:)
```


Here are examples of coding CEEOPTS DD statements in your JCL. In the first example, CEEOPTS is coded as one of the DD statements in a step. The TEST option can be coded as in-stream data as shown.

The second example shows that you can have your TEST option coded in a file if you find that more convenient.

The third example shows how you can code a CEEOPTS DD statement when the JCL executes a PROC. In this case, notice that the EXEC statement runs a PROC named PROC99. The “ASTEP.CEEOPTS” DD statement will add the CEEOPTS DD to a proc step named ASTEP. When it is done this way, you do not have to make any changes to the PROC itself.

A TEST(...) option on the EXEC statement can be used to trigger the debugger

- When LE initializes, it examines the PARM string on the EXEC statement for LE run-time options
  - A slash character (/) in the parm string signals that LE options are present
  - If an LE TEST(…) option is present, LE starts the debugger
- For COBOL programs, code LE options after the last slash

  COBOL

```
//STEP5  EXEC PGM=COBPROG,
//  PARM='/TEST(,,,TCPIP&address%port:)'
```

- For non-COBOL LE programs, code LE options before the first slash

  PL/I, C/C++, LE Assembler

```
//STEP5  EXEC PGM=MYPROG,
//  PARM=' TEST(,,,TCPIP&address%port:)/'
```


There are two ways to code a TEST option in JCL. Either use a CEEOPTS DD statement as you have already seen, or code a TEST option on the EXEC statement.

Here is how to code a TEST option on the EXEC statement. When LE initializes, it examines the PARM string on the EXEC statement for LE run-time options and looks for a slash. For COBOL programs, everything after the last slash is taken as LE options. For non-COBOL LE programs, everything before the first slash is taken as LE options. In either case, you must code a slash at the appropriate location in your PARM string.

Here is an important restriction. LE will examine the PARM string from the EXEC statement only if the main program is an LE program. That means that you cannot use this method to start Debug Tool if the main program is a non-LE program.

The slash (/) character is a separator between program parameters and LE options

- You can code program parameters together with a TEST option
- These examples pass the string 'ABC,1234' to the program
  - LE options and the slash (/) are stripped off, and only the remaining string is passed to the application program
  - **For a COBOL program, code LE options after the last slash:**

```
//STEP5  EXEC PGM=COBPROG,
// PARM='ABC,1234/TEST(,,,TCPIP&address%port:)'
```

  - **For a PL/I, C/C++, or LE assembler program, code LE options before the first slash:**

```
PL/I, C/C++, LE Assembler:
//STEP5  EXEC PGM=PLIPROG,
// PARM='TEST(,,,TCPIP&address%port:)/ABC,1234'
```


When you code a TEST option on the EXEC statement, you can still code parameters for your program. Here are examples that pass the data string “ABC,1234” to the application program as a parameter.

Remember that for COBOL programs, all parameters coded after the last slash are taken as LE options. So to pass parameters to the program, code them all before the last slash.

For non-COBOL LE programs, all parameters coded before the first slash are taken as LE options. So to pass parameters to your program, code them all after the first slash.

In either case, LE strips its options and the slash out of the parm string before control is passed to the application program. So the LE options are not received by your program.

EXEC parm considerations

- z/OS limits the JCL parameter string to 100 bytes
  - Adding the TEST option can make the total length too long
- If the parm string becomes too long, you can:
  - Use a different method to trigger Debug Tool, or
  - Truncate the user portion of the parm string, and then add the missing data manually with the debugger after the program starts
- If the parm string becomes too long to fit on a line, it can be continued to the next line using this syntax
  - Example:

```
//RUNSAM1 EXEC PGM=SAM1,
//  PARM=('THIS,IS AN,EXAMPLE,OF,A,VERY,LO',
//   'NG,PARAMETER/TEST(,,,TCPIP&address%port:)'),
//        REGION=4M
```


z/OS limits JCL parameters to a total of 100 bytes. It is possible to run into a situation where adding a TEST option will make the total length exceed 100 bytes.

If the parameter string becomes too long, you have a couple of options. First, you can use a different method to start Debug Tool. Either use a CEEOPTS DD statement, or use the 'Debug Tool user exit data set' facility. Another option is to truncate the user portion of the string, and then add the missing data manually using the debugger after the program starts.

If the string becomes too long to fit on one line, it can be continued to the next line. An example is shown of the syntax used to code a long parameter string that spans multiple lines.

Coding a TEST option with a batch DB2 application

- Do not code a TEST option on the EXEC statement in DB2 batch jobs if the JCL executes program IKJEFTxx as in this example
- A TEST option will not trigger the debugger with program IKJEFTxx, because it is not an LE program
  - **Instead, use a CEEOPTS DD or the 'Debug Tool user exit data set' facility**
  - or optionally, the TEST option can be coded in the DB2 RUN parms:

```
//*         RUN DB2 PROGRAM
//STEP6 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSNA)
 RUN  PROGRAM(PHONEP01) PLAN(PHONEP01) -
      PARMS('/TEST(,,,TCPIP&address%port:)')  -
      LIB('DNET603.ADLAB.LOAD')
 END
//*
```

COBOL example (slash before TEST)

Do not code a TEST option on the EXEC statement when running a DB2 batch application that executes program IKJEFT01. The debugger will not be triggered, since that program is not an LE program. Instead, consider using a CEEOPTS DD. Or optionally, a TEST option can be coded in the run parms in SYSTSIN as in the example.

Coding a TEST option with a batch IMS application

- Do not code a TEST option on the EXEC statement for IMS batch jobs if the JCL executes program DFSRRCxx as in this example
- A TEST option will not trigger the debugger with program DFSRRCxx, because it is not an LE program
  - **Instead, use a CEEOPTS DD or the 'Debug Tool user exit data set' facility**
  - Example of batch IMS JCL:

```
//*
//*        RUN IMS PROGRAM
//STEP6 EXEC PGM=DFSRRC00,
//      PARM=('DLI,IMSPROG,TDIMSP,200,,,,,,,,,N')
//DFSRESLB DD DSN=IMS.RESLIB,DISP=SHR
//IMS      DD DSN=FMN.PSBLIB,DISP=SHR
//CEEOPTS DD *
PARMS('/TEST(,,,TCPIP&address%port:)
   .
   .
```


Also, do not code a TEST option on the EXEC statement when running an IMS batch application that executes program DFSRRC00. The debugger will not be triggered, since that program is not an LE program. Instead, consider using a CEEOPTS DD or the 'Debug Tool user exit data set' facility.

That is the end of this section, which described running a program in batch, triggering the debugger by coding a TEST option in JCL, and displaying the debugger on a remote graphical user 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_DTv12s04DebuggingBatchGUITest.ppt

This module is also available in PDF format at: ../DTv12s04DebuggingBatchGUITest.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, CICS, CICS Explorer, DB2, IMS, Rational, System z, 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.