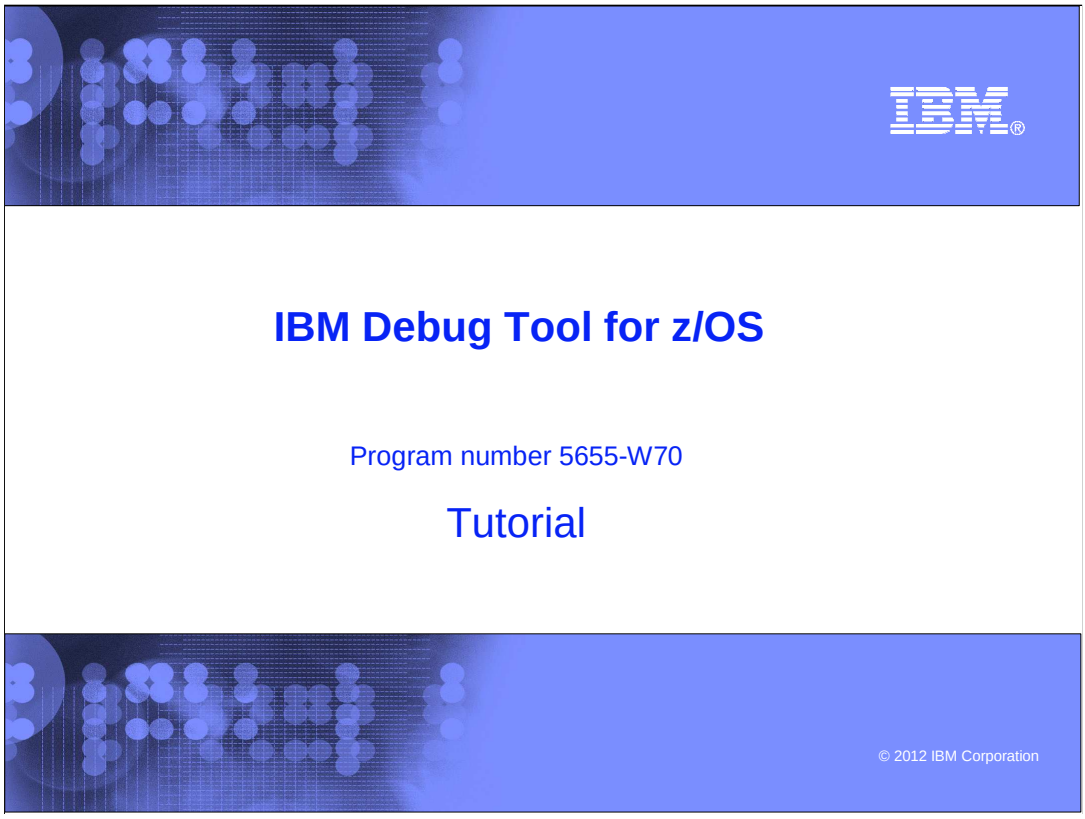
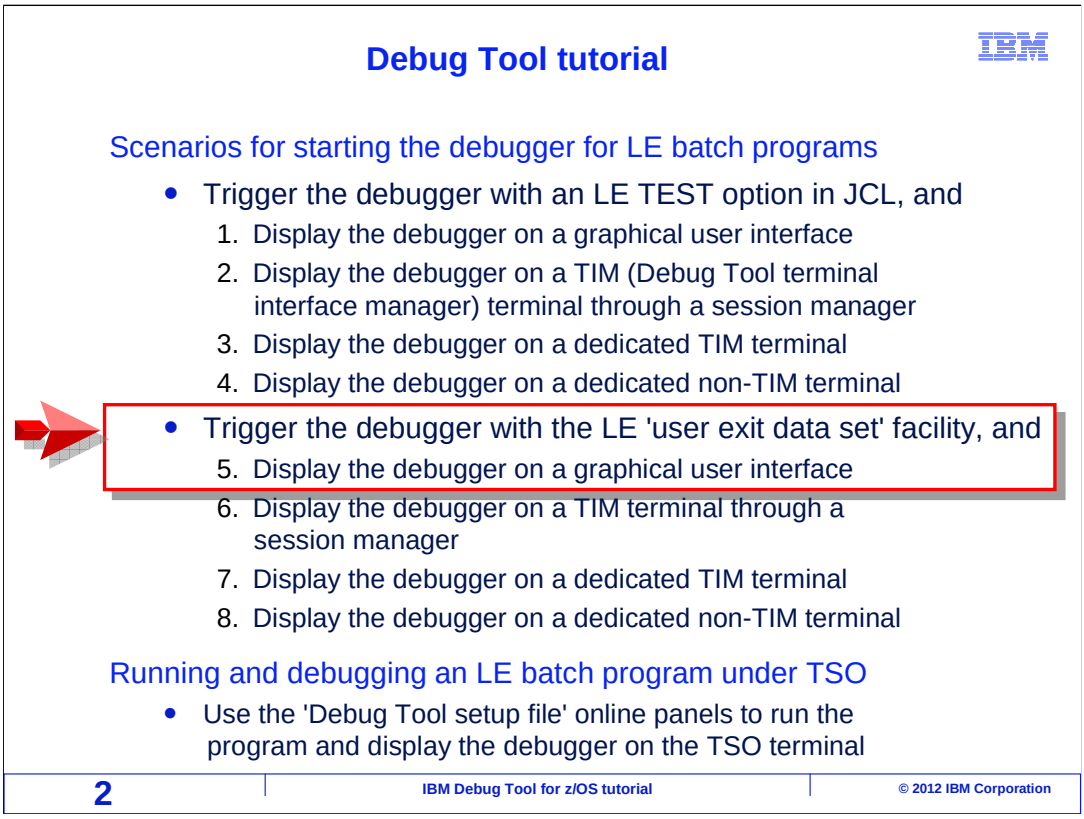


This is the tutorial for IBM Debug Tool for z/OS®, one of the IBM zSeries® problem determination tools.



In this section, you will see a scenario for starting Debug Tool for a batch application. In this scenario the application runs as a batch job, and the Debug Tool 'user exit data set' utility is used to set a trigger that will start the debugger when the job runs. The debugger displays on graphical user interface software running on your workstation. You can skip this section if you do not plan to use Debug Tool this way on your system.

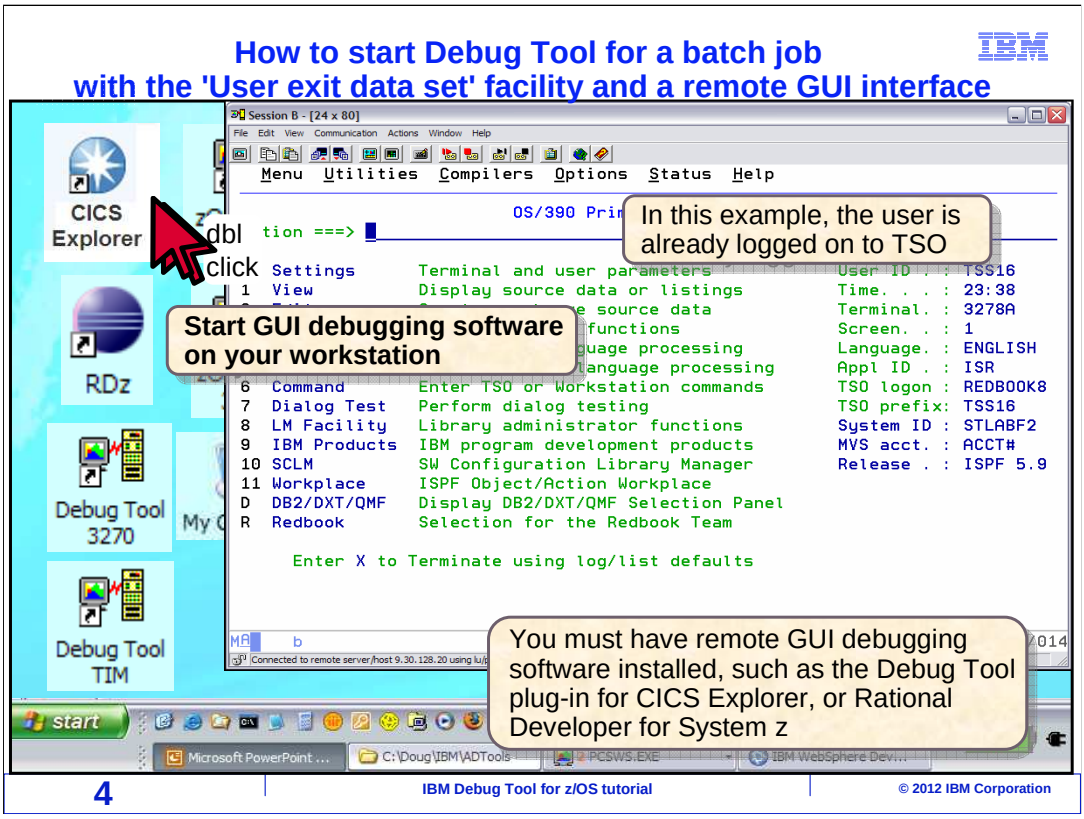
Debug in batch using the 'user exit data set' facility and a GUI interface

- Description
  - The 'User exit data set' online panels in TSO are used to set a trigger for the debugger.
    - A special library may be required in STEPLIB or JOBLIB to enable the Debug Tool LE user exit
  - 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

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, using the 'Debug Tool user exit data set' utility panels in TSO.

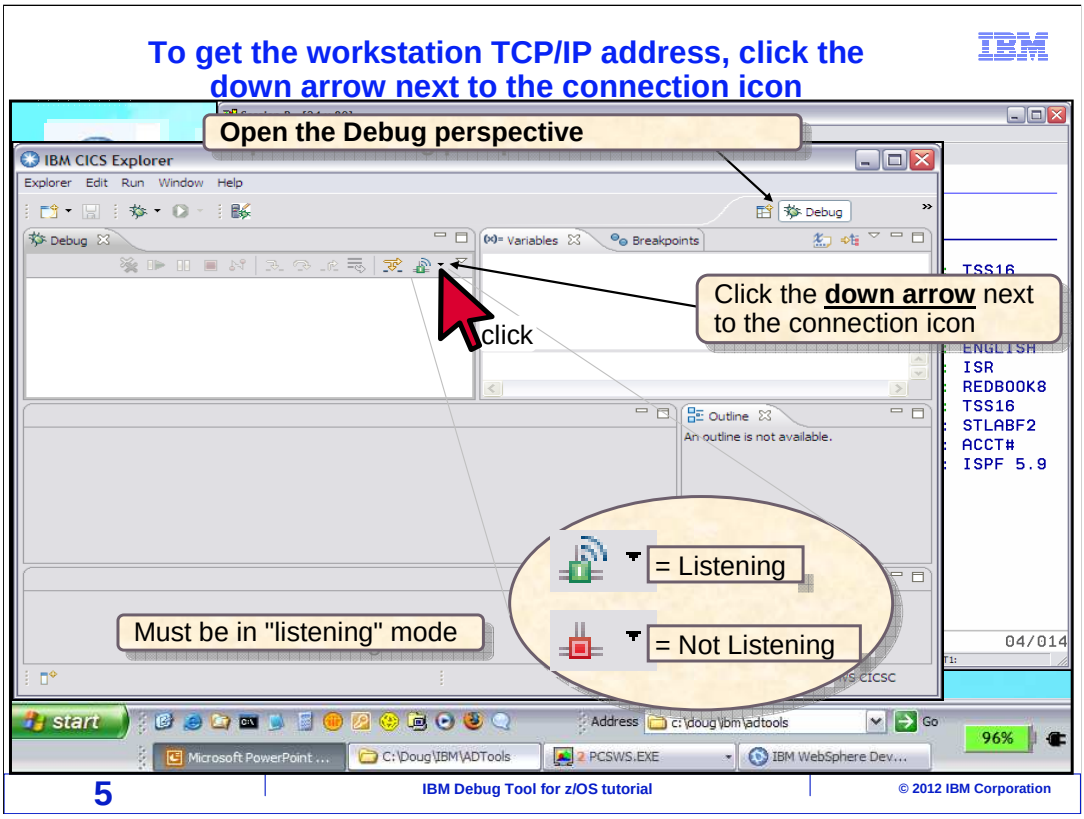
This method can be used to debug Language Environment (LE) conforming programs running in a batch job, including programs that access DB2, IMS, and other types of databases. Even non-LE subroutines can be debugged using this method, if there is at least one LE program higher in the call chain. Software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

You cannot use this method if compatible GUI debugging software is not available.



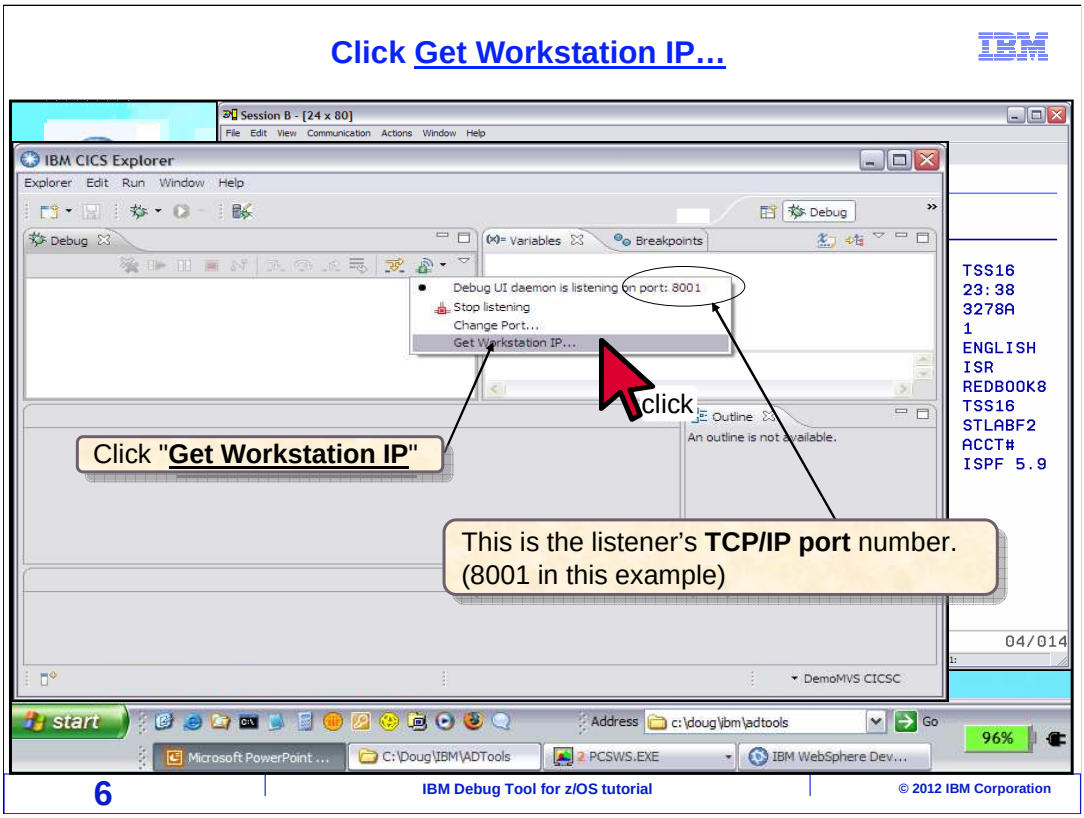
In this example, the user is already logged on to TSO. The first step is to start the remote GUI debugging software on your workstation. This could be software such as Rational Developer for System z or CICS Explorer with the Debug Tool plug-in.

Here, CICS Explorer is started.

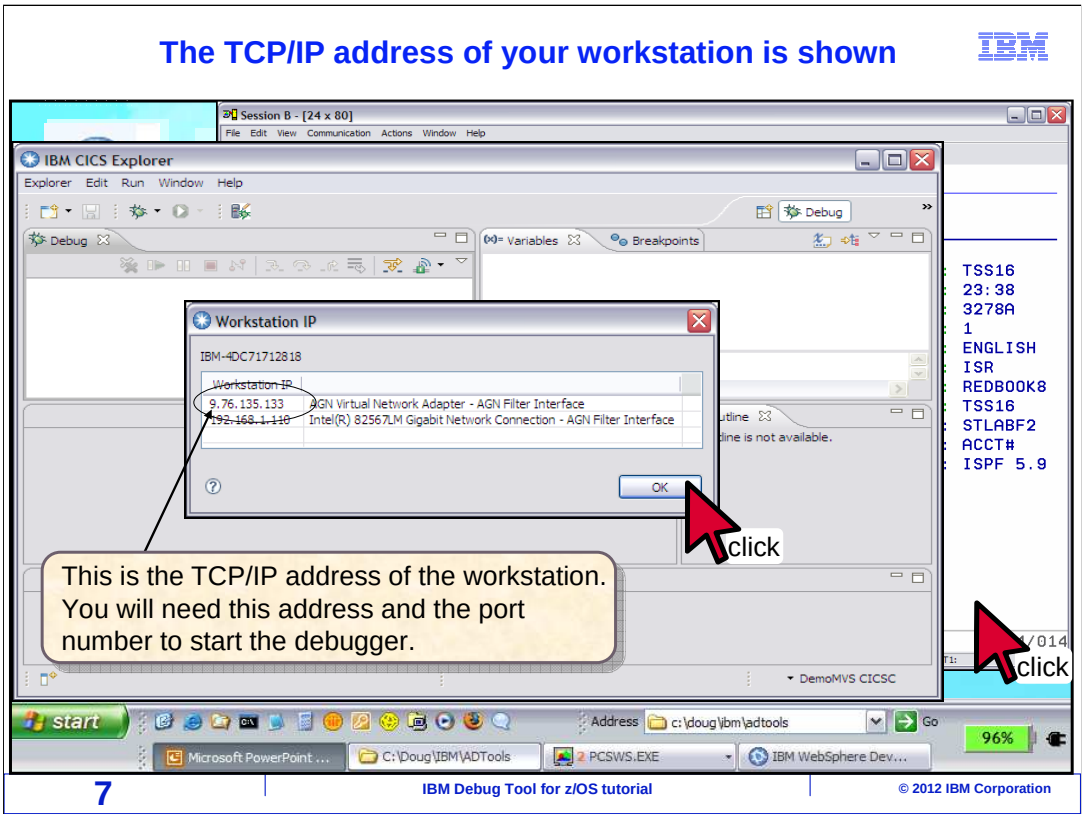


That started CICS Explorer. If the "Debug" view is not displayed, switch to the Debug perspective. It must be in “listening” mode. There is an icon on the toolbar that shows the listening status. If it is not listening, the listener icon is red, and you can click the icon to switch it on. The icon is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

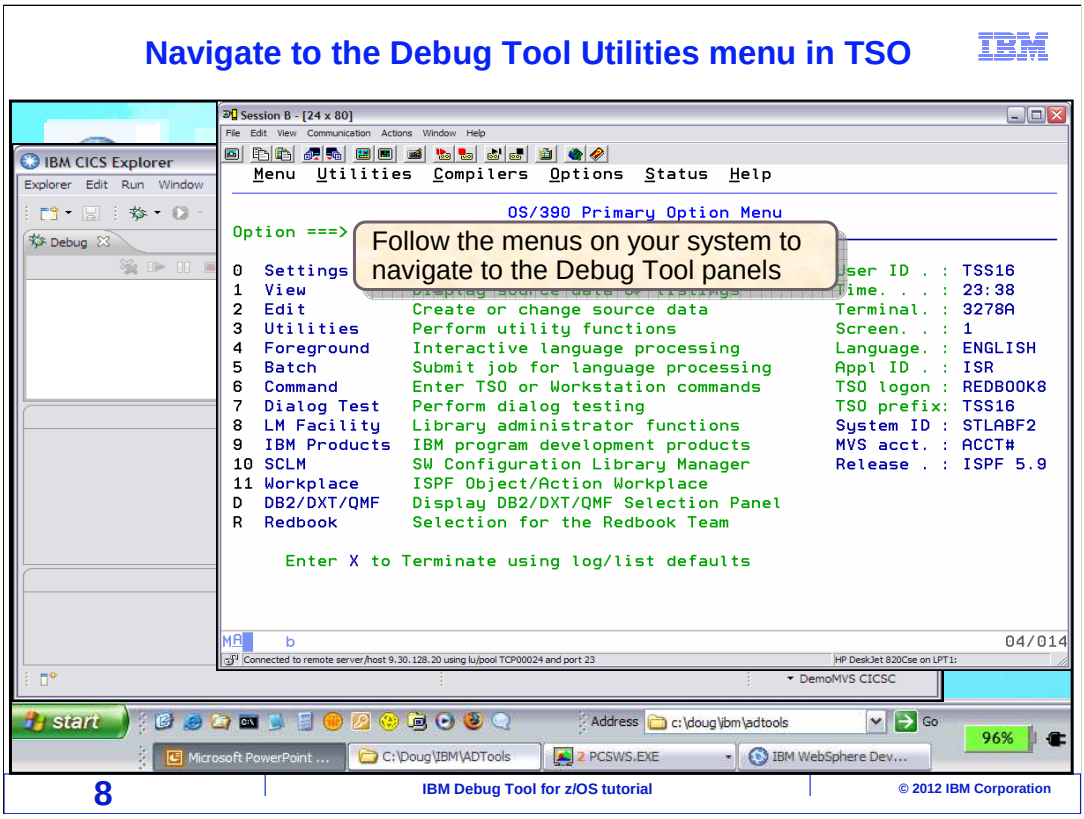
You will need to determine your workstation's IP address. Click the small black down arrow next to the listening icon.



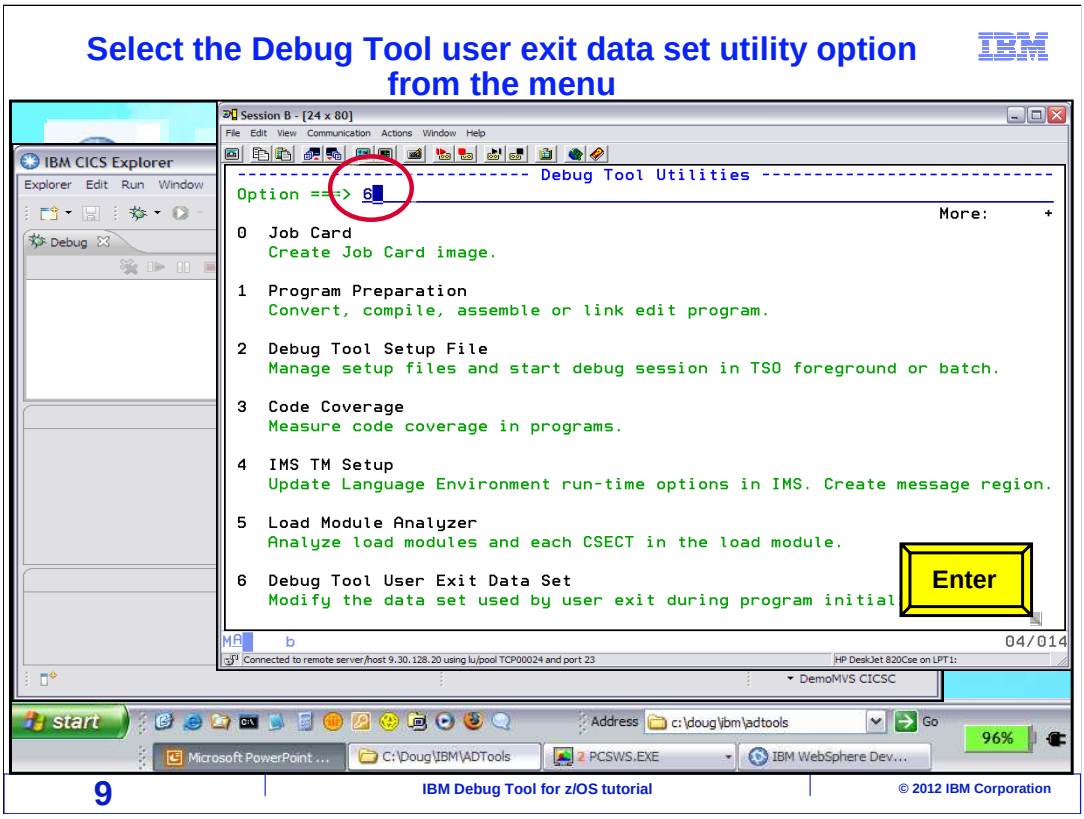
That displays a menu. Make a note of the listener's IP port. In this example, it is 8001. Next, click “Get Workstation IP”.



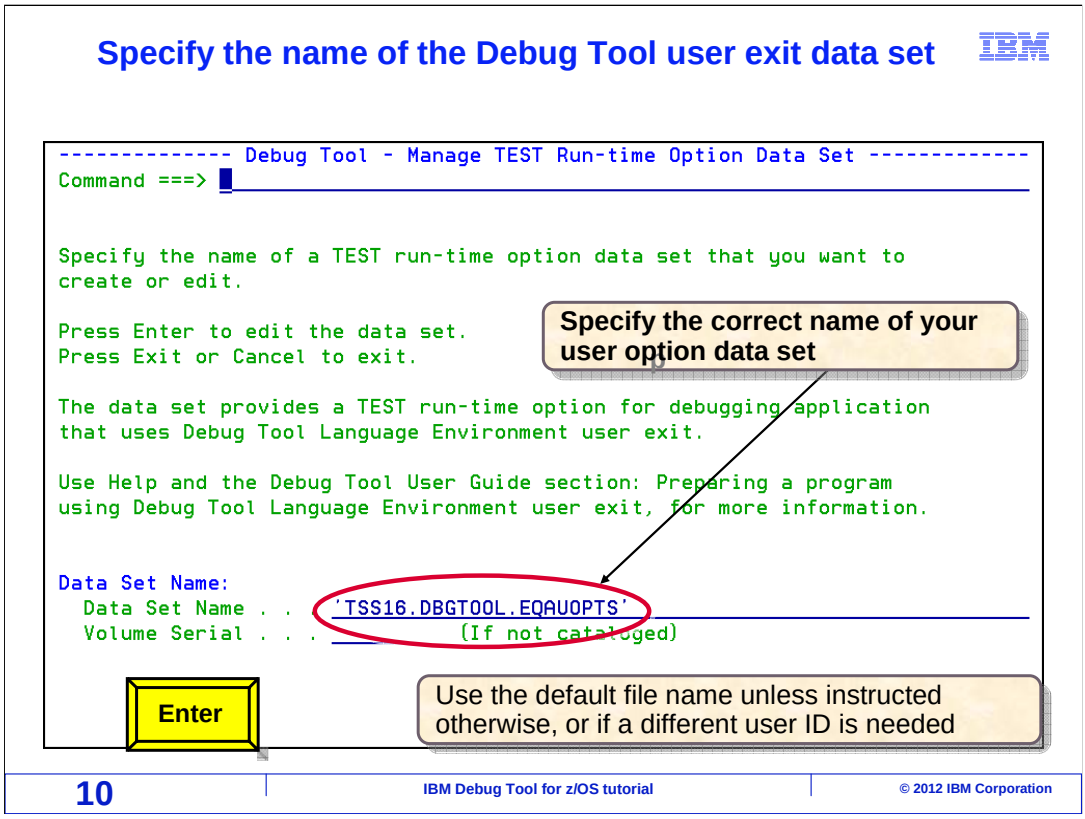
That displays your workstation's IP address. Make a note of it. At this point, you have prepared the remote GUI debugging software, and it ready to receive a debugging session. Click OK to close the pop-up window. Now go back to TSO by clicking your TSO terminal session.



In TSO, navigate to the Debug Tool ISPF panels. The person who installed Debug Tool on your system may have defined a menu option for you to get there.



The 'Debug Tool Utilities' menu is displayed. Select option 6 for the 'Debug Tool user exit data set' panels.

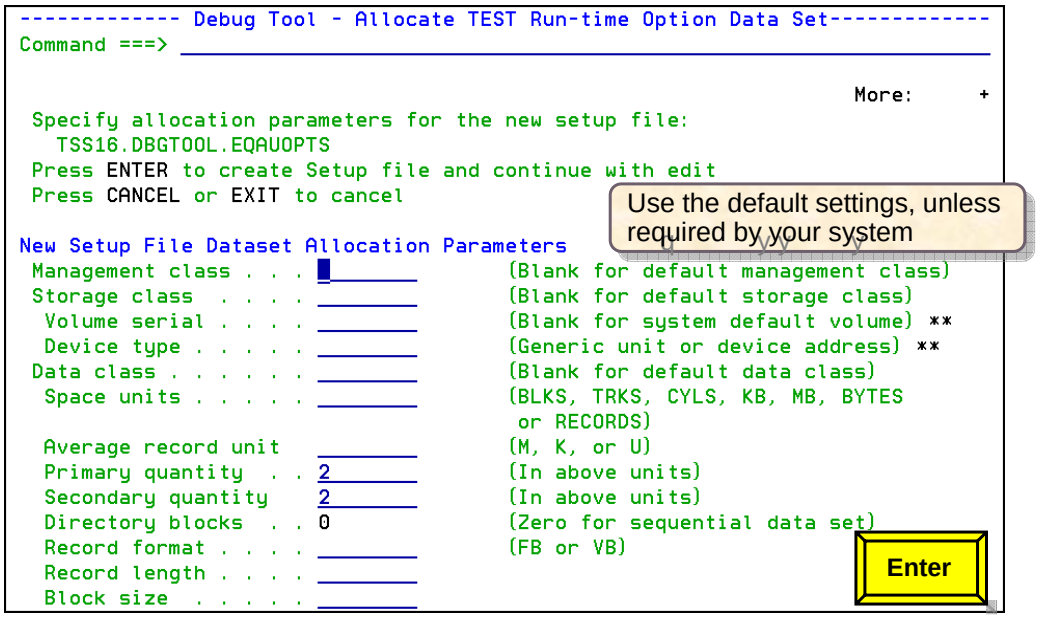


In this utility, you store your debugging profile in a file. The file name must follow a specific convention. By default, it is yourID.DBGTOOL.EQAUOPTS.

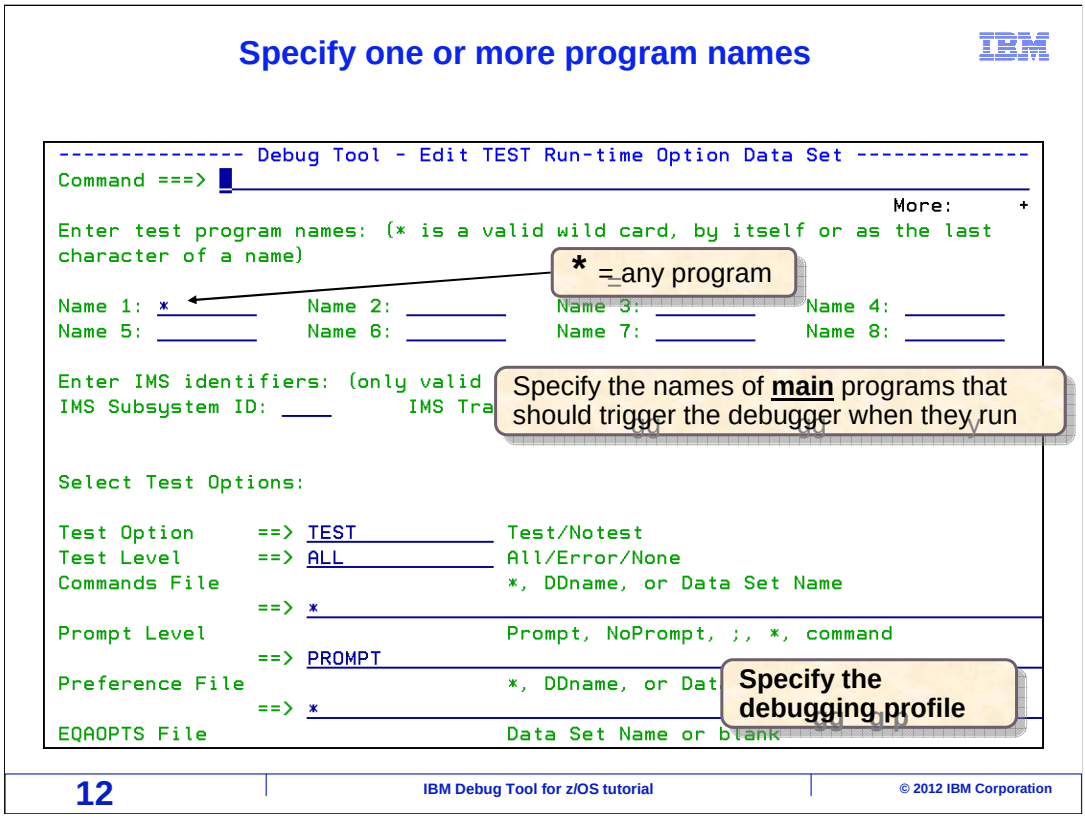
If you have not used this panel before, it will automatically fill in a file name in the 'data set name' field. Use the default name unless you have been instructed otherwise by your system programmer. The naming convention may have been customized for your system, and the convention must be followed for this feature to work. Later, when you run your batch job, a Debug Tool system exit will look for a file according to the convention.

Press Enter. If the file does not already exist, it will be created automatically.

If the Debug Tool user exit data set does not already exist, this allocation panel is displayed



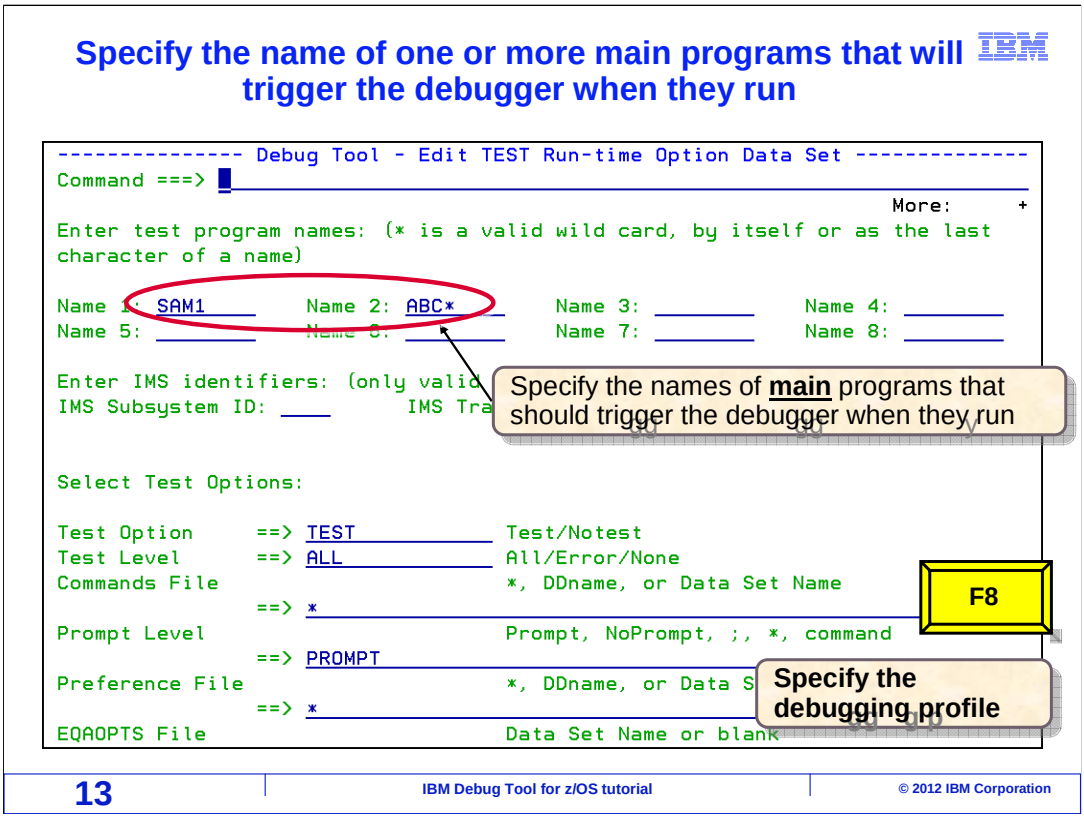
In this example, the file did not exist, so the file allocation panel is displayed where you can specify attributes for the new file. On many systems, you do not need to change anything and you can take the defaults. Press Enter to continue.



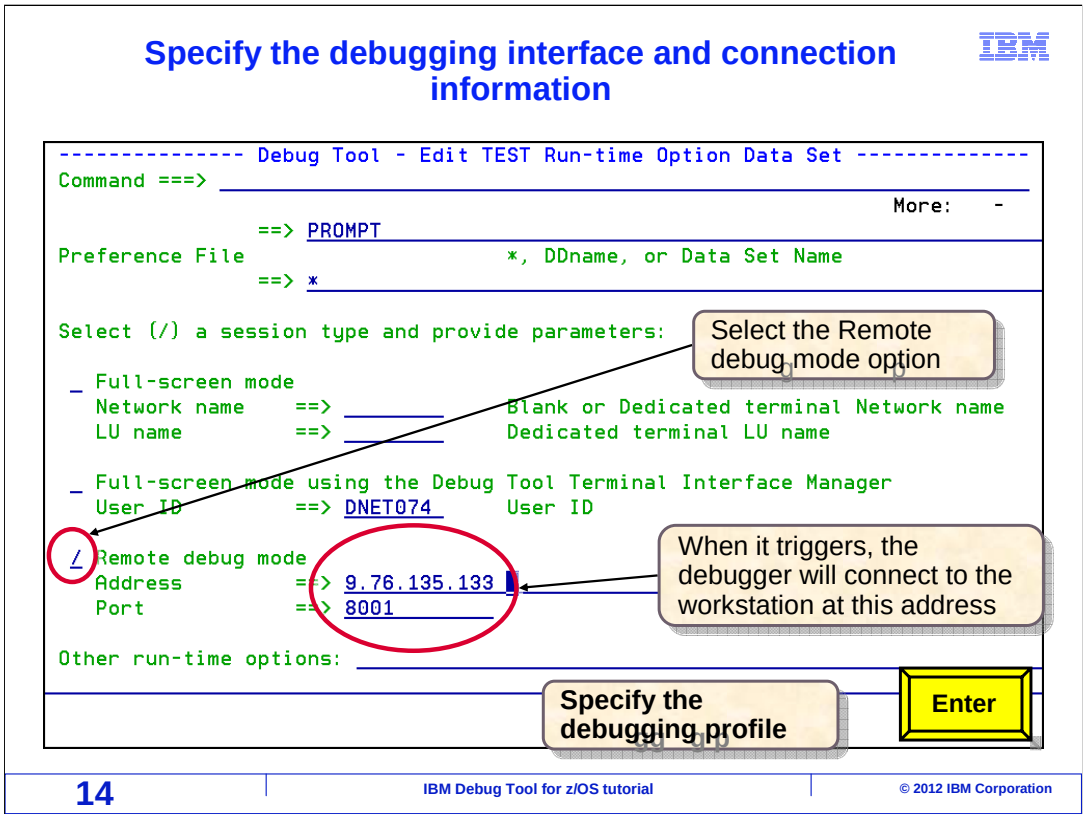
Now the Debug Tool user option data set has been created, and the panel titled: 'Edit test run-time option data set' is displayed. Here you specify the name or names of the main programs that should trigger the debugger when they run.

Only the highest level LE program, typically the main program, will trigger the debugger. There is no reason to specify the names of lower level programs or subprograms. You do not need to specify all of the subprograms that you want to debug. In fact, entering subprogram names on this panel will have no effect at all. After the debugger has triggered, you will be able to use debugging commands to tell the debugger to stop at any subprogram you want to see.

Notice that you can specify an asterisk in the program name, which specifies that the debugger will trigger for the first LE program that runs in the call chain, regardless of its name. This can be useful if you do not know the name of the highest level LE program, or if you want to use the same debugging profile with different programs.

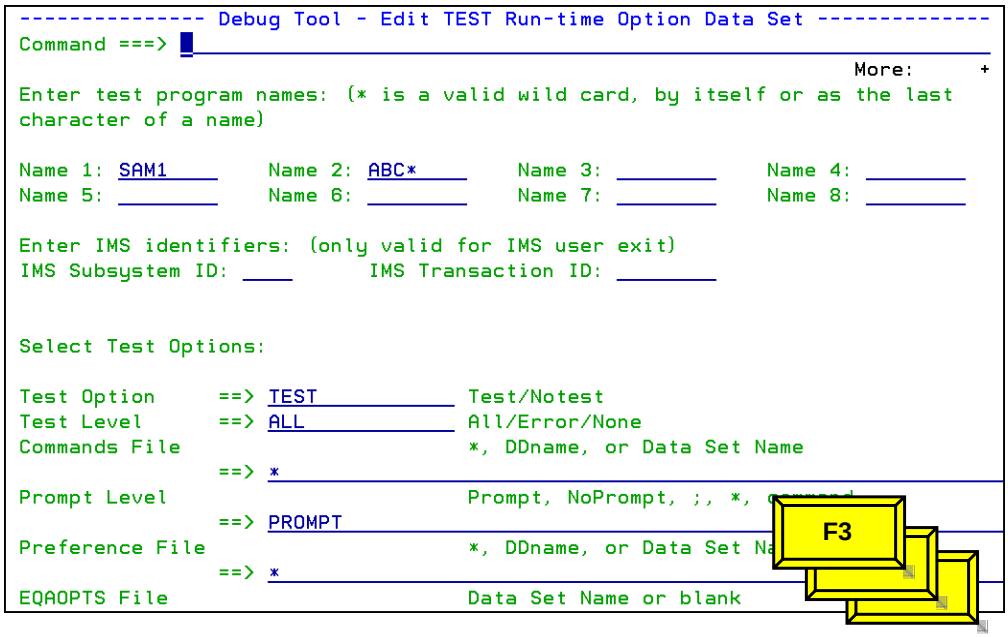


In this example, SAM1 is specified as the name of the main program that will trigger the debugger when it runs. You can specify up to eight program names. Notice that you can enter a partial program name using an asterisk as a wildcard. Scroll down with the F8 key.



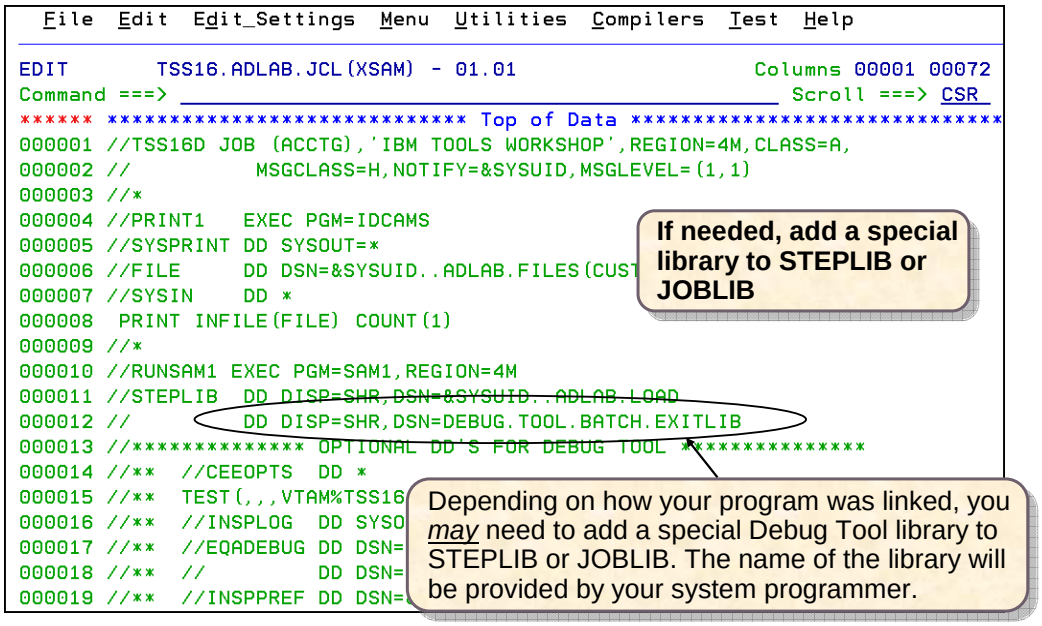
Since a graphical interface is being used, select the option named “Remote debug mode" with a slash. Specify the IP address of your workstation in the "address" field. Also specify the "port" field. These are the TCP/IP address and port number where the GUI debugging software is listening. That is all that is needed. Press Enter.

The user exit data set was updated by pressing Enter

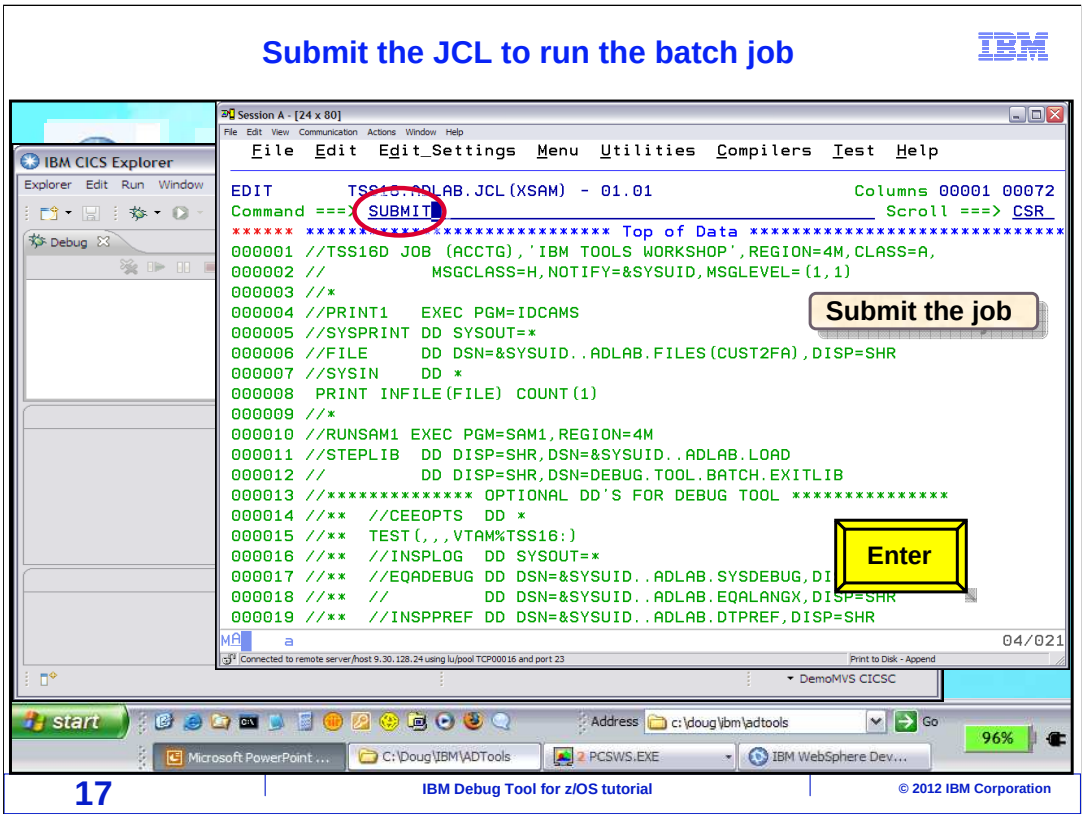


At this point, the name of the program that will trigger Debug Tool, and the location of the debugging interface have been specified. Press F3 several times to save the options in the user exit data set and exit from the Debug Tool utilities.

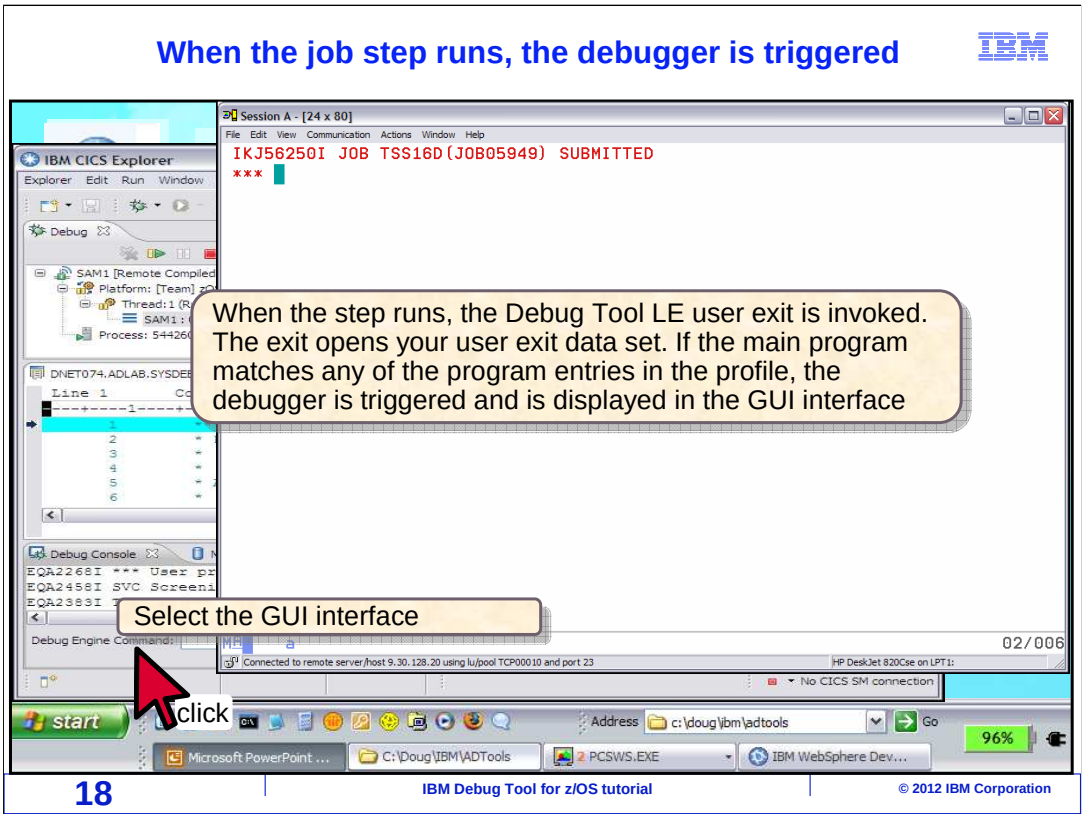
Next, open the JCL that will run the program in batch



Next, open the JCL that you will use to run your batch application in the editor. On some systems, you may need to add a special Debug Tool library to the STEPLIB or JOBLIB DD. This library enables the system exit that reads the 'User exit data set' to determine whether to start the debugger. If this library is needed, get its name from your system programmer or help desk.



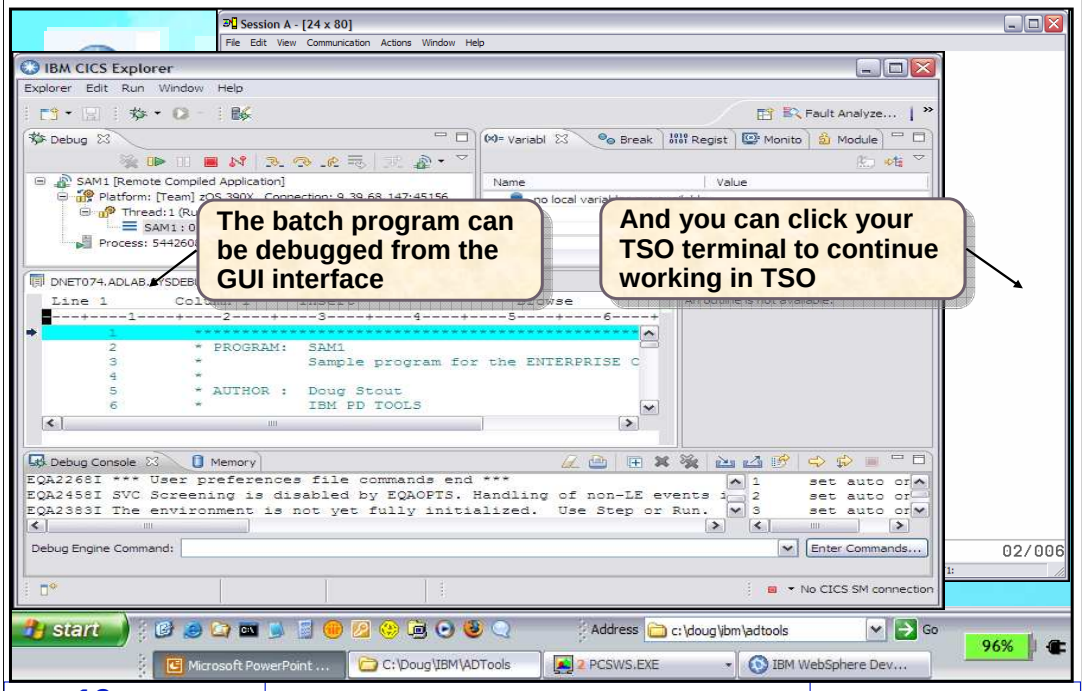
Submit the JCL to run the batch job.



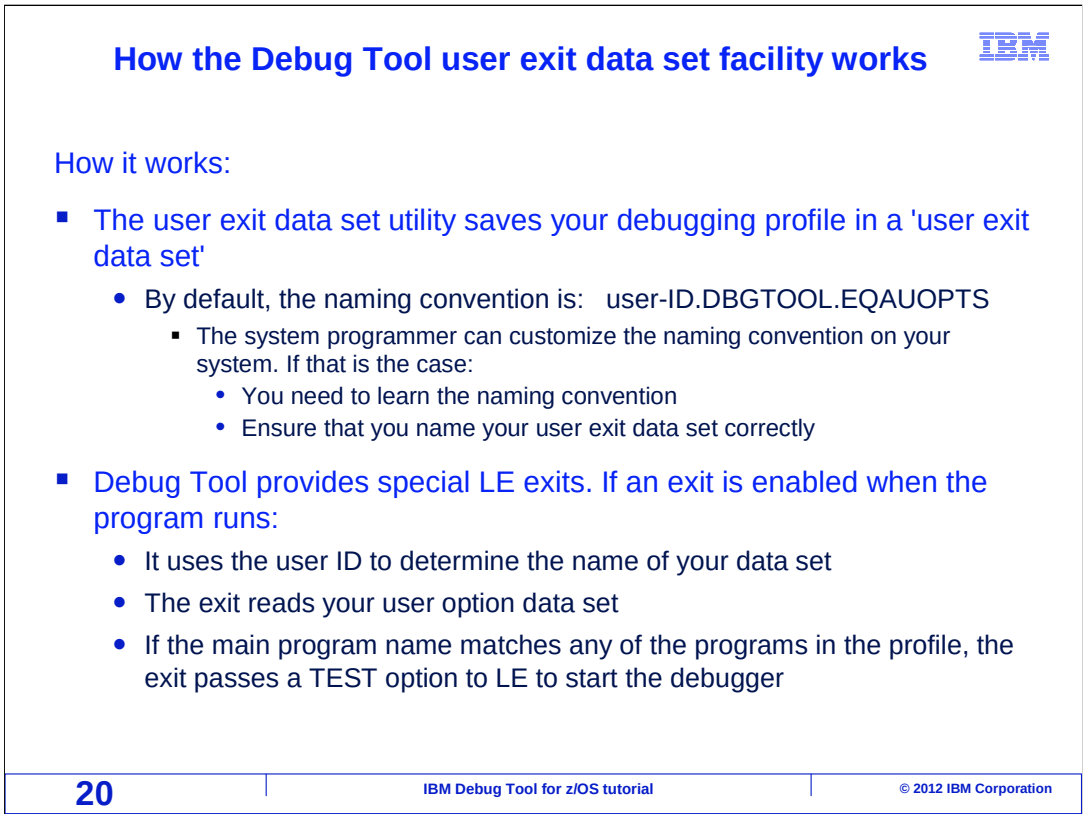
When the step that was modified in the batch job runs, the Debug Tool LE user exit runs automatically. The exit looks for your Debug Tool user exit data set. If the data set does not exist, no action is taken and the step runs normally. If the data set is found, it is read. If the highest level LE program does not match any of the program names in the profile, the step runs normally. But if there is a match on the program name, the exit automatically formats and passes a TEST option to Language Environment to trigger the debugger. The debugger is displayed by the GUI debugging software.

The debugging software 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.



That was an example of how to start Debug Tool for a batch application using the Debug Tool 'User Exit data set' utility and remote GUI debugging software. At this point, you may be comfortable with the process, and have all of the information you need to start the debugger. The rest of this section goes into technical details about how it works. You can opt to skip the rest of this section, if you are not interested in the details.

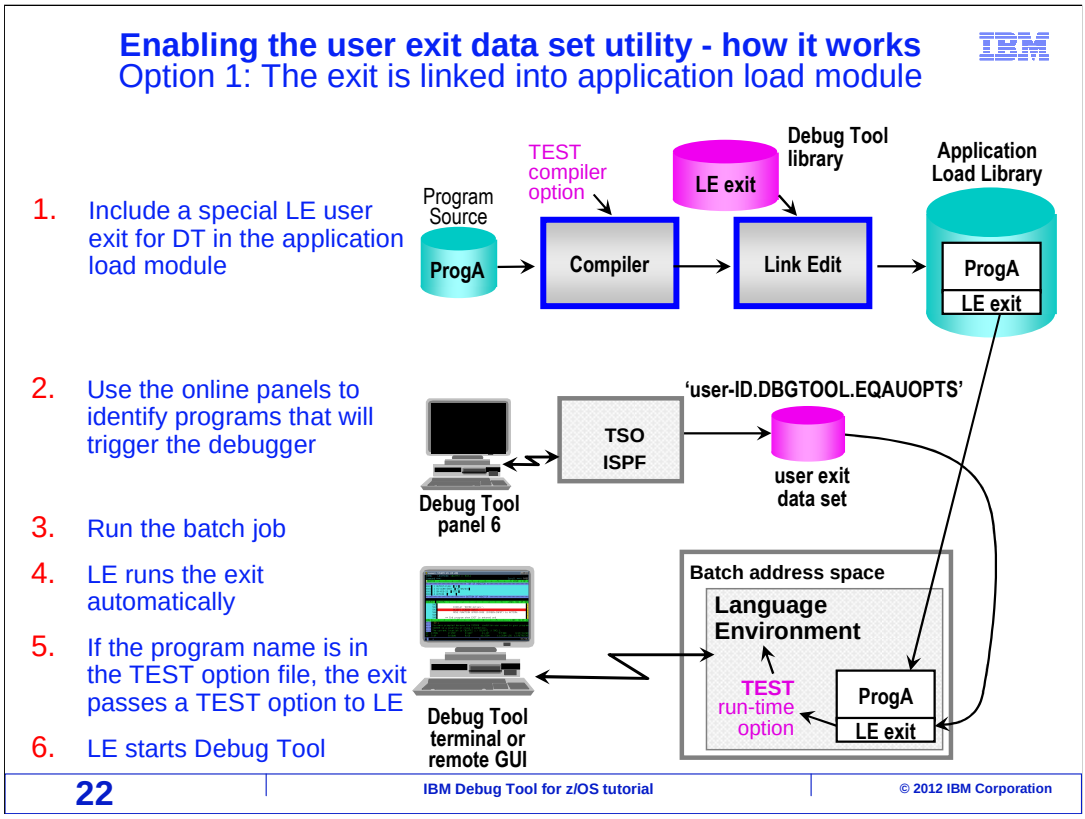
Here is how it works. The 'User exit data set' utility saves your Debug Tool profile in a file. The test option file has to be named using a specific naming convention. By default, user ID.DBGTOOL.EQAUOPTS. However, your systems programmer can customize the naming convention. Use the default naming convention unless you have been instructed otherwise by your system programmer.

Debug tool provides special exits for Language Environment. If one of these exits is enabled when your program runs, it uses the user ID of the job to determine the user ID portion of the file name. The exit program then reads your test option data set. If it finds a match on the program name in your profile, the exit passes a TEST option to Language Environment, which causes the debugger to start.

Enabling the Debug Tool LE user exit

- There are two ways to enable a Debug Tool LE exit program
  - Option 1: A supplied Debug Tool exit is link-edited into the application load module
  - Option 2: A supplied Debug Tool exit is made available in STEPLIB or JOBLIB in run-time JCL
    - The system programmer must set up a special library that Debug Tool users can add to their STEPLIB or JOBLIB concatenation

There are two ways to enable a Debug Tool LE exit program. The first way is to link-edit one of the supplied Debug Tool exits into an application load module. Then it will be enabled automatically whenever the program runs. The second way is to make the exit available by adding a special library to STEPLIB or JOBLIB of the run-time JCL. The system programmer must set up this special library so that Debug Tool users can use it.



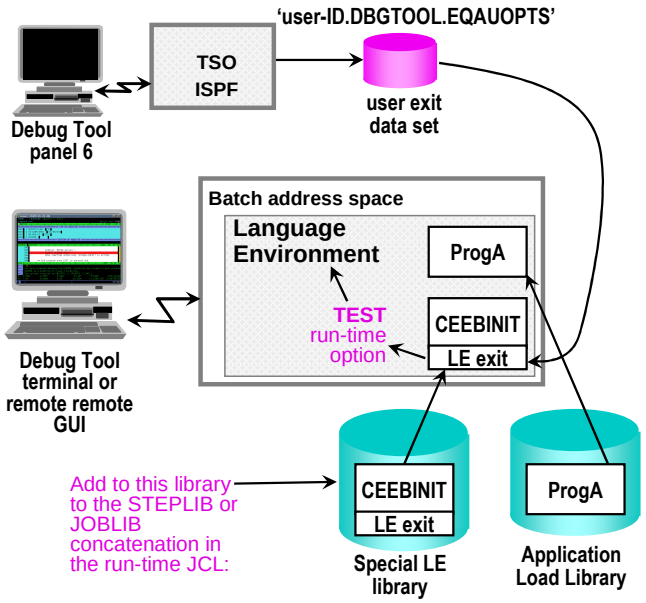
This chart shows how the supplied LE exit works if your organization has chosen to link edit the exit into your application load modules. In the first step, the LE exit is included into your application load module when the program is compiled and linked. This could be made part of the standard compile process for all of your batch programs.

In the second step, the Debug Tool 'User exit data set' panels are used to identify the name of the programs that will trigger the debugger, and the terminal or workstation where the debugger will be displayed. These settings are stored in the file. In the third step, the job is submitted.

When the program runs, Language Environment automatically runs the exit program because it was included in the load module. This capability is a normal part of Language Environment initialization. The exit uses the user ID of the job to determine the name of the user exit data set. The exit opens the file and reads the options stored there. If there is match on the program name, then the exit passes a TEST option to Language Environment, and Language Environment starts Debug Tool.

Enabling the user exit data set utility - how it works
Option 2: STEPLIB or JOBLIB to the exit

1. The system programmer prepares a special library with the exit
2. Use the utility to identify programs that will trigger the debugger
3. Add the special library to STEPLIB in run-time JCL
4. Run the batch job
5. LE runs the exit automatically
6. If the program name is in the TEST option file, the exit passes a TEST option to LE
7. LE starts Debug Tool



This chart shows how the supplied LE exit works if your organization has chosen NOT to link edit the exit directly into your application load modules, but instead has chosen to make the exit available by adding it to your STEPLIB concatenation. In the first step, the system programmer prepares a special library that contains the exit module.

In the second step, the Debug Tool 'User exit data set' panels are used to identify the name of the programs that will trigger the debugger, and the terminal or workstation where the debugger will be displayed. These settings are stored in the file.

In the third step, the JCL to run the job is edited. The special load library is added to the STEPLIB or JOBLIB concatenation. In the fourth step, the modified JCL is submitted. When the program runs, Language Environment automatically runs the exit program that was made available in STEPLIB or JOBLIB. The exit uses the user ID of the job to determine the name of the user exit data set. The exit opens the file and reads the options stored there. If there is match on the program name, then the exit passes a TEST option to Language Environment, and Language Environment starts Debug Tool.

Enabling the user exit data set utility - how it works
Option 2: STEPLIB to the exit

- Example:
  - Note: Get the name of the library from your system programmer

```
//STEP5  EXEC PGM=MYPROG
//STEPLIB DD DSN=MY.PROGRAM.LIB,DISP=SHR
//        DD DSN=DBGTOOL.BATCH.LOADLIB,DISP=SHR
```

- Tip: You can override STEPLIB in a step that is in a PROC without changing the PROC
  - This example adds or overrides the third library in the STEPLIB concatenation of the step named ASTEP in PROC99

```
//RUN99  EXEC PROC99
//ASTEP.STEPLIB DD
//        DD
//        DD DSN=DBGTOOL.BATCH.LOADLIB,DISP=SHR
```


Here are examples of coding your JCL if you are using the method where you add a special library to STEPLIB or JOBLIB. In the first example, the library is added to the existing STEPLIB concatenation.

That is fine, if you're JCL is self-contained. However, your JCL may run a JCL PROC, and the STEPLIB that you need to modify may be hard coded in the PROC. If that is the case, you can code JCL as shown in the second example. That example assumes that STEPLIB is already coded in a step named ASTEP in a proc named PROC99. In this example the first two DD statements in the overridden STEPLIB concatenation remain unchanged, and a third library is either added or overridden.

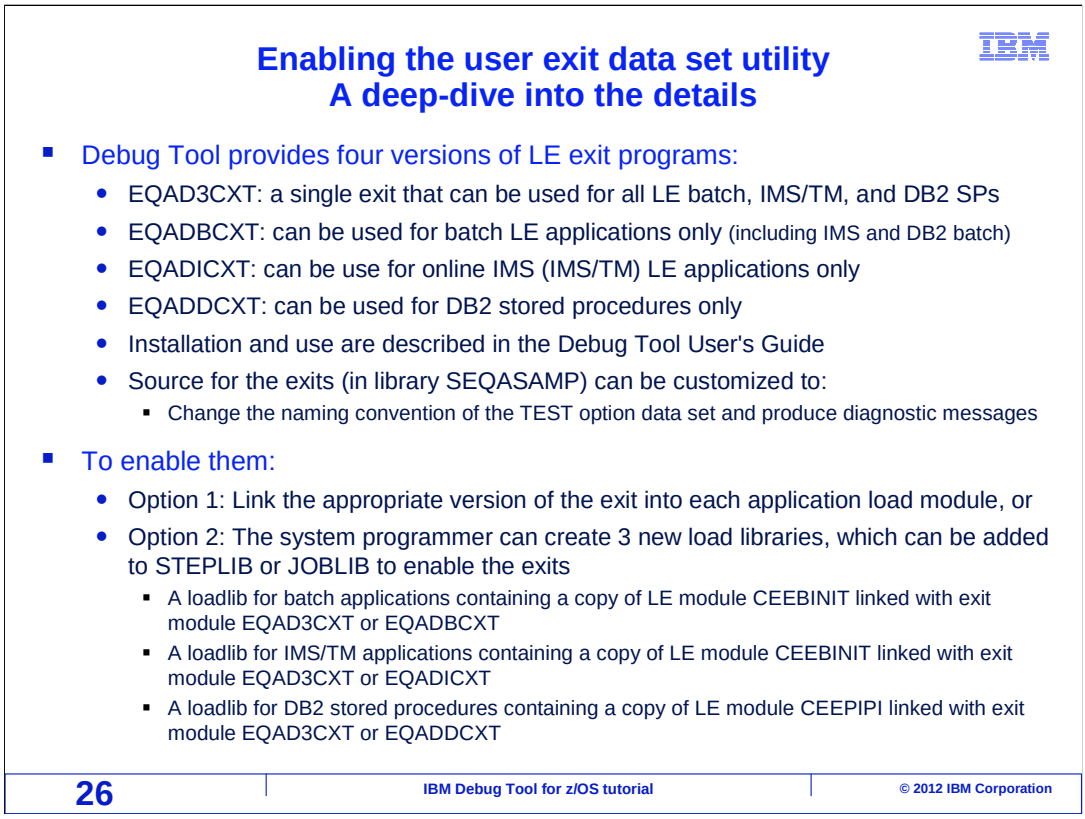
Comparing the two ways to enable the user exit data set utility

- In option 1, a special Debug Tool module is link-edited into the application load module
  - Advantage:
    - No JCL changes are needed at run-time to start Debug Tool !
  - Considerations:
    - The exit module always runs once each time the main program starts, but incurs negligible overhead when not debugging
    - It may be preferred, but not technically necessary, to remove the exit module before promoting to production
- In option 2, the exit is made available at run-time in STEPLIB or JOBLIB
  - Advantages:
    - No change to the compile process (to link the exit into the application module)
    - You can control when the exit runs with JCL
  - Consideration:
    - Run-time JCL must be modified to start Debug Tool

Two options have been described for enabling the Language Environment exit that Debug Tool supplies. If it is up to you to choose which option to use, here is a comparison.

In option one, the supplied Language Environment exit is link edited directly into the application load module. The advantage to this method is that no JCL changes are needed at runtime to start the debugger. You just submit your JCL without any modifications whatsoever. You control whether to start the debugger strictly by specifying program names in the 'user exit data set' panel. A consideration for this method is that the exit will always run once each time a main program is executed, although this incurs negligible overhead. Even so, it may be preferred to remove the exit module before promoting the application module to production, to eliminate any possible risk of inadvertently triggering the debugger during a production run.

In option 2, the exit is made available at run time by adding a special library to STEPLIB or JOBLIB in your JCL. The advantages of this option are that no changes are needed to your compilation processes, and you can control when the exit runs with changes to your JCL. The consideration is that your JCL requires this simple modification before you can trigger the debugger.



If you are curious about exactly how to implement the supplied Language Environment exits, here is a deep-dive into the details. Debug Tool provides three exit programs that work with the 'user exit data set' utility. A module named EQADBCXT is used to start Debug Tool for **batch** Language Environment applications. A module named EQADICXT can be used to start Debug Tool for **online IMS/TM applications**, and a module named EQADDCXT is used to start Debug Tool for **DB2 stored procedures**. These modules are shipped in Debug Tool library SEQAMOD. Source code for the exits is also shipped with Debug Tool in the SEQASAMP sample library. The source code can be modified and assembled to change the naming convention for user exit data sets, and to request that the exit produce messages as an aid to the installer.

To enable the exits, one option is to link edit the appropriate version of the exit into each application load module. The exit is enabled because it is present in the module. The second option to enable the exits is to create new load libraries, which the Debug Tool user can add to their STEPLIB or JOBLIB. First, create a load library for batch applications. It will contain module CEEBINIT, from the Language Environment SCEERUN library, with the supplied exit module EQADBCXT link-edited into it. Also, create a load library for IMS/TM applications. It will contain module CEEBINIT, from the Language Environment SCEERUN library, with the supplied exit module EQADICXT link-edited into it. Finally, create a load library for DB2 stored procedure programs. It will contain module CEEPIPI, from the Language Environment SCEERUN library, with the supplied exit module EQADDCXT link-edited into it.

That is the end of this section, which described a program running in batch, triggering the debugger by using the 'user exit data set' utility, and displaying the debugger on a remote GUI 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_DTv12s05DebuggingBatchGUIExit.ppt

This module is also available in PDF format at: ../DTv12s05DebuggingBatchGUIExit.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers