

Debug Tool Mentor Workshop

Lab Exercises



Lab Exercise 1

Lab Setup: Create your lab files

In this exercise you will:

- Run the LABSETUP program to create your own copies of sample files that will be used during the Debug Tool exercises
 - Check that your files have been created
1. Log on to TSO (if you are not already logged on).
 2. Navigate to the ISPF command shell. On most systems, this is **Option 6** from the main ISPF menu.
 3. Run the LabSetup program:
 - a. On the command line, enter the lab setup command that your instructor or training coordinator has given to you. The command will have the format:

EX 'xxxx.ADLAB.INSTALL(LABSETUP)'

```
Menu List Mode Functions Utilities Help
-----
                                ISPF Command Shell
Enter TSO or Workstation commands below:
===> EXEC 'DNET249.MASTER.ADLAB.DATA(LABSETUP)'
```

```
Place cursor on choice and press enter to Retrieve command
=>
=>
->
```

4. You will see the message “**SETUP FOR HANDS-ON TRAINING**” message.
 - a. If you do not see this message, check the spelling of the command and try again.
 - b. Press the **ENTER** key repeatedly until the LabSetup program completes.
 - Note: Many screens full of messages will be displayed. Continue to press **Enter**. There may be times when three asterisks appear in the bottom left corner of the screen or the process just stops. Keep pressing Enter. You will end up at the Command Shell panel once again.

Debug Tool Mentor Workshop

Lab Exercises



5. Note: In the next steps, you will verify that your files have been created.
6. Navigate to the system 3.4 panel (Data Set List).
 - Type 3.4 on the command line, then **ENTER**:
7. Enter a Dsname level of: **your-tso-id.ADLAB**, then press **ENTER**.

```

Menu RefList RefNode Utilities Help
-----
Data Set List Utility
Option ==> _____ More: +
blank Display data set list          P Print data set list
      U Display UTOC information      PU Print UTOC information

Enter one or both of the parameters below:
Dsname Level . . . DNET249.ADLAB
Volume serial . . _____

Data set list options
  Truncate files 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  Enter "/" to select option
    
```

8. You should see a list of data sets that begin with your ID, with a middle qualifier of ADLAB.

```

Menu Options View Utilities Compilers Help
-----
DSLIST - Data Sets Matching DNET249.ADLAB          Row 1 of 16
Command ==> _____ Scroll ==> CSR
Command - Enter "/" to select action              Message          Volume
-----
DNET249.ADLAB.COPYLIB                             DMPU14
DNET249.ADLAB.CUSTFILE                            DMPU23
DNET249.ADLAB.DATA                                DMPU25
DNET249.ADLAB.DTCHD                               DMPU07
DNET249.ADLAB.DTLOG                               DMPU05
DNET249.ADLAB.DTMAP                              DMPU16
DNET249.ADLAB.EMI                                MIGRAT1
DNET249.ADLAB.EQI                                DMPU05
DNET249.ADLAB.JCI                                DMPU15
DNET249.ADLAB.LI                                 DMPU08
DNET249.ADLAB.LOAD                               DMPU19
DNET249.ADLAB.PROC                               DMPU09
DNET249.ADLAB.SOURCE                             DMPU21
DNET249.ADLAB.SYSDEBUG                           DMPU01
DNET249.ADLAB.TEMPLATE                           DMPU02

F1=Help      F2=$plit    F3=Exit     F4=$wapNext F5=Rfind    F7=Up
F8=Down     F9=$wap     F10=Left   F11=Right  F12=Cancel
    
```

It is OK if your list of files is different from the list shown here.

- a. If you have ADLAB data sets, then you are ready for the Debug Tool exercises.
- b. If you do not have the ADLAB datasets, then return to step 1. Ask for help if you aren't sure why you don't have the sample files.

Debug Tool Mentor Workshop

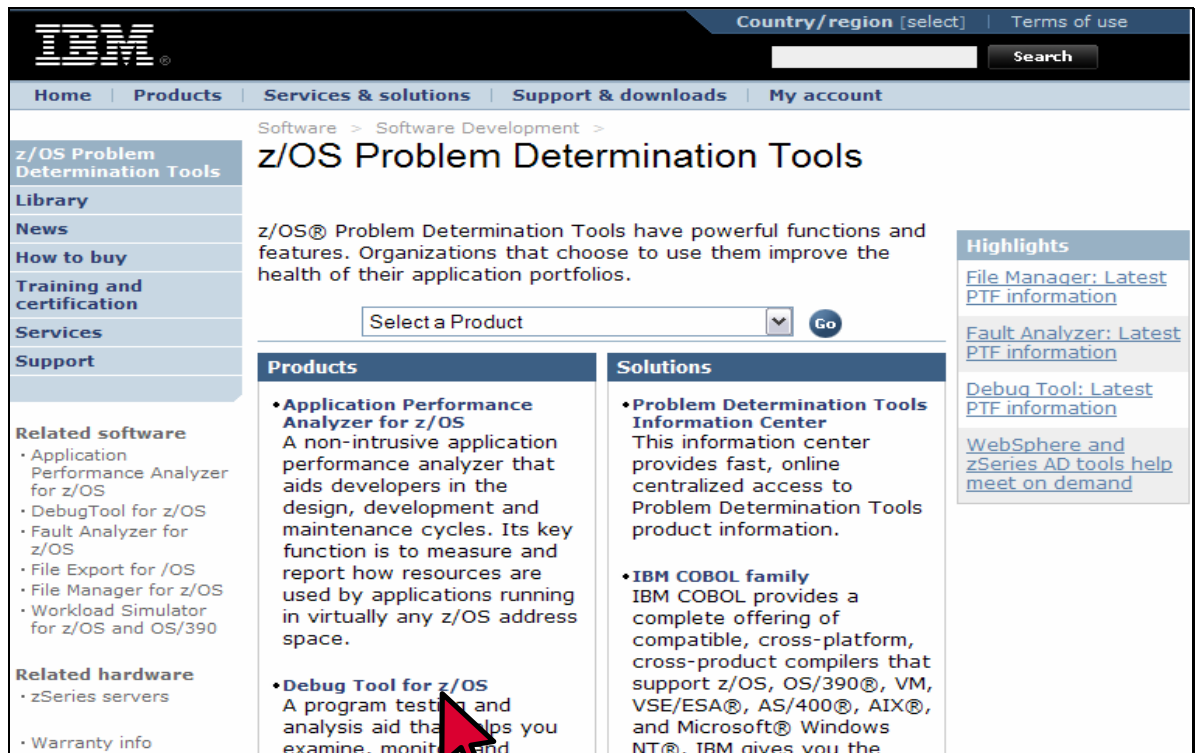
Lab Exercises



Lab Exercise 2

In this exercise you will:

- Use a web browser to locate the Debug Tool manuals on the IBM web site.
 - Open the “Debug Tool Reference and Messages” manual for your reference.
 - Learn how to download any of the Debug Tool manuals to your workstation.
1. Start an Internet Browser window.
 2. Open URL:
<http://www.ibm.com/software/awdtools/deployment>
 3. Click on the **Select a Product** pull-down.
 4. Click on **Debug Tool**



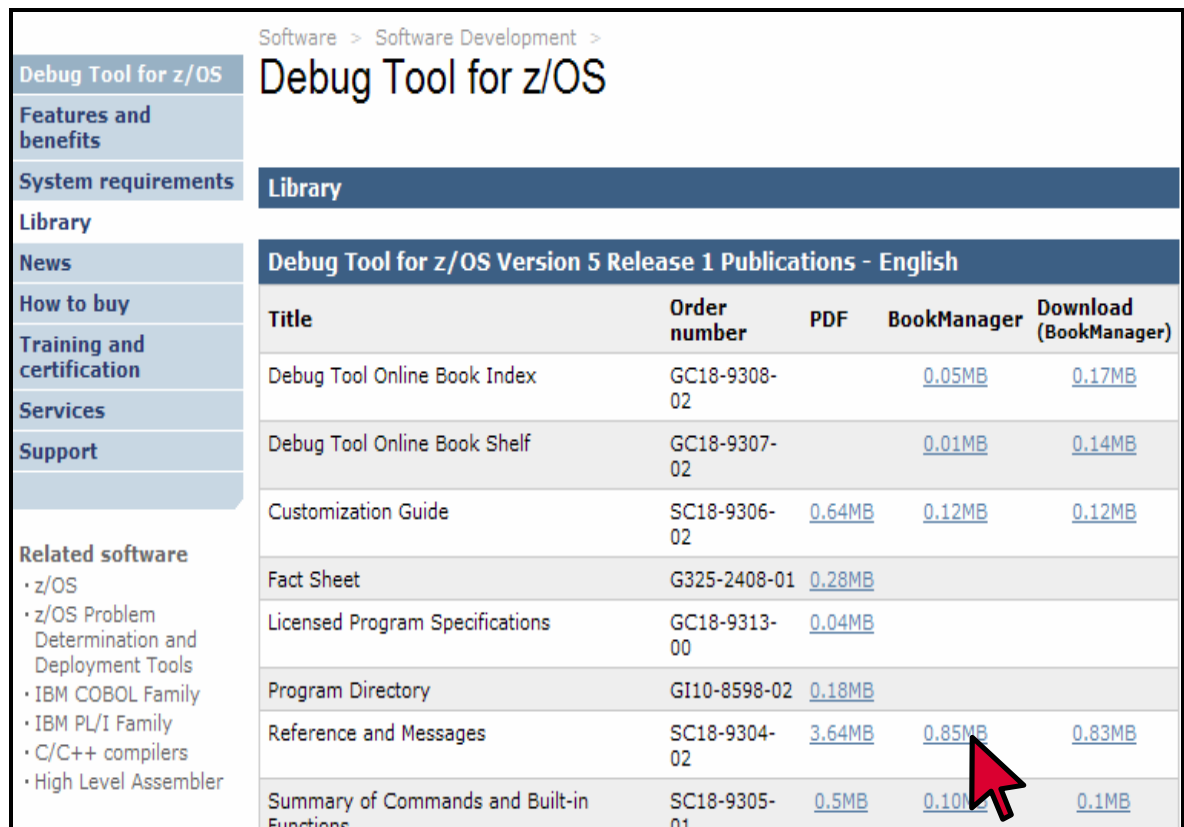
Debug Tool Mentor Workshop Lab Exercises



5. Click on **Library**



6. You can browse manuals online. For the “Reference and Messages” manual, click on the link under the BookManager column.



Debug Tool Mentor Workshop

Lab Exercises



- The contents page for the Reference and Messages manual appears. You can use the navigation links to get to specific sections.
- Click on the browser's BACK button until you are back to the Debug Tool Library page.
- You can download Debug Tool manuals to your workstation from this web site.
RIGHT CLICK on the link next to the "Reference and Messages" manual in the PDF column.

Software > Software Development > Debug Tool for z/OS

Library

Debug Tool for z/OS Version 5 Release 1 Publications - English

Title	Order number	PDF	BookManager	Download (BookManager)
Debug Tool Online Book Index	GC18-9308-02		0.05MB	0.17MB
Debug Tool Online Book Shelf	GC18-9307-02		0.01MB	0.14MB
Customization Guide	SC18-9306-02	0.64MB	0.12MB	0.12MB
Fact Sheet	G325-2408-01	0.28MB		
Licensed Program Specifications	GC18-9313-00	0.04MB		
Program Directory	GI10-8598-02	0.18MB		
Reference and Messages	SC18-9304-02	3.64MB		0.83MB
Summary of Commands and Built-in Functions	SC18-9305-01	0.5MB	0.10MB	0.1MB

- This will let you download the manual to your workstation. Click on **SAVE TARGET AS**.
- A dialog is displayed where you can select any valid directory on your workstation to download the manual in PDF format. If you would like a softcopy of the manual, you can download it now. Otherwise, press **Escape**.
- Close your Internet Browser window.

Lab Exercise 3

In this exercise you will:

- Submit a job to compile a COBOL program.
 - See an example of compiler options you use to prepare a program for debugging with Debug Tool.
 - See where the compiler stores information needed by Debug Tool.
1. Log on to TSO.
 2. Open the following dataset in an EDIT session:

'your-tso-id.ADLAB.JCL(BDTDEMO)'

```
EDIT          DNET074.ADLAB.JCL (BADSTAT) - 01.02          Columns 00001 00072
Command ==> |                                           Scroll ==> CSR
***** ***** Top of Data *****
000001 //YOUR-TSO-IDC JOB REGION=4M,
000002 // TIME=(1),MSGCLASS=H,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
000003 //*****
000004 //*          COMPILE
000005 //*****
000006 //COBCOMP EXEC PGM=IGYCRCTL,
000007 //          PARM='TEST (NONE, SYM, SEPARATE) , LIST, MAP, SOURCE, XREF, LIB, DYNAM,
000008 //          NORENT, NOOPT'
000009 //STEPLIB DD DISP=SHR, DSN=COBOL.V3R2.SIGYCOMP
000010 //SYSIN DD DISP=SHR, DSN=YOUR-TSO-ID.ADLAB.SOURCE (ADSTAT)
000011 //SYSLIB DD DISP=SHR, DSN=YOUR-TSO-ID.ADLAB.COPYLIB
000012 //          DD DISP=SHR, DSN=YOUR-TSO-ID.ADLAB.SOURCE
000013 //SYSPRINT DD DISP=SHR, DSN=YOUR-TSO-ID.ADLAB.LISTING (ADSTAT)
000014 //SYSDEBUG DD DISP=SHR, DSN=YOUR-TSO-ID.ADLAB.SYSDEBUG (ADSTAT)
000015 //SYSLIN DD DISP=(MOD, PASS) , DSN=&&LOADSET, UNIT=SYSALLDA,
000016 //          SPACE=(80, (10, 10))
000017 //SYSUDUMP DD SYSOUT=x
```

3. Customize the JCL to run on your system by making the following changes:
 - a. Add a JOB card.
 - b. The instructor may tell you about other customizations needed for your system.

Debug Tool Mentor Workshop

Lab Exercises



4. Notice the parameter string specified in the compile step. The TEST parm tells the compiler to prepare the program for debugging.

What are the sub-parameters specified in the TEST parm?

5. The SEPARATE sub-parameter tells the compiler to store information needed by Debug Tool in a “separate” file. The debug information will be written to the SYSDEBUG file.
6. Submit the job. (type SUB on the command line, then ENTER).
7. View the output of your compile and ensure that the return code from the compile step is 4 or less. If not, fix any errors and re-submit the job.
8. Browse data set ‘your-tso-id.ADLAB.SYSDEBUG(DTDEMO)’. This is the debug information that Debug Tool will use.

Debug Tool Mentor Workshop

Lab Exercises



Lab Exercise 4a

In this exercise you will:

- Use the TSO “CALL” command to run a program
- Specify a TEST run-time parameter to start Debug Tool.
- Get ready for exercise 4 (by starting a Debug Tool session).

IMPORTANT NOTES:

- **Do either exercise 4A (this exercise), or exercise 4B. You do not need to do both. This exercise (4A) is preferred. Only do exercise 4B if exercise 4A will not work on your system.**
- **This exercise will walk you through a fast and easy way to start a Debug Tool session to get ready for exercise 5. This is NOT the method you will normally use to start Debug Tool.**

1. Log on to TSO.
2. Navigate to the ISPF Command shell panel. (This is usually OPTION 6 from the main menu).
3. Next, you will run a program, *WITHOUT* starting Debug Tool. You will use the TSO “CALL” command, which can be used to run any program from a load library. Type in the following command:

CALL 'your-tso-id.ADLAB.LOAD(ADSTAT)'

Press **ENTER**.

```
ISPF Command Shell
Enter TSO or Workstation commands below:

===> call 'dnet074.adlab.load(adstat)'
```

```
Place cursor on choice and press enter to Retrieve command

=> call 'dnet074.adlab.load(adstat)' '/test'
=> call 'dnet074.adlab.load(adstat)'
=> call 'dnet074.demos.load(dtdemop)' 'test/'
=> call 'dnet074.demos.load(dtdemop)' 'test'
=> call 'dnet074.demos.load(dtdemop)' '/test'
=> ex 'dnet603.code.clist(musetup)'
=> ex 'dnet603.code.clist(mesetup)'
=> ex 'dnet603.code.clist(labdtcv)'
=> receive indataset('dnet074.adlab.data.xmit')
HELLO
GOODBYE
***
```


Debug Tool Mentor Workshop

Lab Exercises



- The program will run. You may see the program display “Hello” and “Goodbye” messages. Press **ENTER** to clear the messages.

Note: It’s OK if you do not see the “Hello” and “Goodbye” messages. They do not appear on certain TSO systems, depending on how your system is configured.

- Next, you will run the program *WITH* Debug Tool. Debug Tool will start when you pass the TEST parameter to Language Environment. Type in the following command:

CALL 'your-tso-id.ADLAB.LOAD(ADSTAT)' '/TEST'

Press **ENTER**.

```
ISPF Command Shell
Enter TSO or Workstation commands below:

===> call 'dnet074.adlab.load(adstat)' '/test'

Place cursor on choice and press enter to Retrieve command
```

- When the program executes, Debug Tool will appear on your TSO terminal.

SUCCESS! Debug Tool appears.

```
COBOL LOCATION: ADSTAT initialization
Command ===> Scroll ===> CSR
MONITOR ---+---1---+---2---+---3---+---4---+---5---+---6 LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE: ADSTAT ---1---+---2---+---3---+---4---+---5---+ LINE: 1 OF 137
1 IDENTIFICATION DIVISION.
2 PROGRAM-ID. ADSTAT.
3 ENVIRONMENT DIVISION.
4 DATA DIVISION.
5 WORKING-STORAGE SECTION.
6

LOG 0 ---+---1---+---2---+---3---+---4---+---5---+ LINE: 41 OF 43
0041 IBM Debug Tool Version 5 Release 1 Mod 0
0042 07/07/2005 09:51:10 AM
0043 5655-M18 and 5655-M19: (C) Copyright IBM Corp. 1992, 2004
***** BOTTOM OF LOG *****
PF 1: ? 2: STEP 3: QUIT 4: LIST 5: FIND 6: AT/CLEAR
PF 7: UP 8: DOWN 9: GO 10: ZOOM 11: ZOOM LOG 12: RETRIEVE
```

Debug Tool Mentor Workshop

Lab Exercises



Note: You may see error messages in the log window (bottom window) as Debug Tool tries to open the Preferences file (INSPREF) and the Log file (INSPLOG). IGNORE these errors, since these files will not be needed yet.

7. At this point, you have an active Debug Tool session, and you are ready for exercise 5.

Lab Exercise 4b

In this exercise you will:

- Use the Debug Tool Setup Utility (DTSU).
- Create a “Setup File”.
- Add program and DD information to the Setup File needed to run the application.
- “Run” the setup file to execute the application and start Debug Tool.
- Get ready for exercise 4 (by having a Debug Tool session).

IMPORTANT NOTE:

- **Do either exercise 4A, or exercise 4B (this exercise). You do not need to do both. Exercise 4A is preferred, if it will work on your system.**

1. Log on to TSO.
2. Navigate to the Debug Tool Utilities panel.

```
----- Debug Tool Utilities -----
Option ==> █

0  Manage Job Card                                     More:  +
   For Program Preparation and Setup File Management

1  Program Preparation
   Compile old or new COBOL programs with newer compilers, convert old COBOL
   source into new COBOL source, use other compilers, and link edit.

2  Manage and Use Debug Tool Setup Files
   You can manage setup files and use them to run your program interactively
   with Debug Tool in TSO Foreground or submit your program to run in
   the background using MVS batch.

3  Code Coverage
   Measure code coverage in programs written in COBOL, PL/I, C/C++ and
   Assembler when compiled with specific IBM compilers and HLASM.

4  Manage IMS Programs

F1=Help      F2=Split      F3=Exit      F7=Backward  F8=Forward  F9=Swap
F12=Cancel
```

3. Select the “Manage and Use Debug Tool Setup Files” option.

Debug Tool Mentor Workshop

Lab Exercises



```
Option ==> 2
0  Manage Job Card
   For Program Preparation and Setup File
1  Program Preparation
   Compile old or new COBOL programs with
   source into new COBOL source, use other
2  Manage and Use Debug Tool Setup Files
   You can manage setup files and use them
   with Debug Tool in TSO Foreground or st
```

4. A panel will appear that requests the name of a “setup file”. A setup file is used to save information about an application. It contains information similar to JCL, such as the name of a program to run, and a list of datasets that the application needs.
5. Type in the name of a new setup file: **your-tso-id.ADLAB.DTSF** and member name **ADSTAT**.

```
----- Debug Tool Foreground - Edit Setup
Command ==>
Setup File Library:
Project . . . . DNET074
Group . . . . ADLAB
Type . . . . DTSF
Member . . . . ADSTAT (Blank or pattern for
                    (or existing or new m
```

6. Press **ENTER**. You have just created a new setup file.
7. The “Edit Setup File” panel will appear. This panel allows the specification of the program to run, and datasets to be allocated (similar to DD statements in JCL).

Debug Tool Mentor Workshop

Lab Exercises



```
EDIT - Edit Setup File 'DNET074.ADLAB.DTSF (ADSTAT)'          Row 1 to 1 of
Command ==> _____          Scroll ==> CS

Modify information and use the Run command to execute,
      or the Submit command to submit to Batch.
      Press HELP for a list of all available commands.

Load Module Name _____
Choose the format of your parameter string:
1 1 LE COBOL Default   - Program Arguments / Run-time Options
  2 Other LE Languages - Run-time Options / Program Arguments
  3 Non-LE Programs / OS/VS COBOL - Run-time Options / Program Arguments

_ Enter / to modify parameters _____

Cmd DD Name  Seq C DD Information (DSN/Sysin/Sysout/Dummy)      DISP
_____      _____ ***** Top of Data *****
***** Bottom of data *****
```

8. The name of the program you will execute is “ADSTAT”. Type **ADSTAT** in the Load Module Name field. Also ensure that **1** is specified next to the text: LE COBOL Default:

```
EDIT - Edit Setup File 'DNET074.ADLAB.DTSF (ADSTAT)'          F
Command ==> _____          Sc

Modify information and use the Run command to execute,
      or the Submit command to submit to Batch.
      Press HELP for a list of all available commands.

Load Module Name ADSTAT _____
Choose the format of your parameter string:
1 1 LE COBOL Default   - Program Arguments / Run-time Options
  2 Other LE Languages - Run-time Options / Program Arguments
  3 Non-LE Programs / OS/VS COBOL - Run-time Options / Program

_ Enter / to modify parameters _____
```

9. Next, you will add a STEPLIB DD statement so the system will know where to find the program. Follow these steps to insert it:
- Type **I** (for Insert) in the field under the Cmd label:, then **ENTER**.

Debug Tool Mentor Workshop

Lab Exercises



```
_ Enter / to modify parameter  
-----  
Cmd DD Name  Seq C DD Information  
i  _____  _____  *****  
*****
```

- In the line that was added, type in information for the new file:
 - DD Name: **STEPLIB**
 - Name: **'your-tso-id.ADLAB.LOAD'**
 - Disp: **SHR**

```
Cmd DD Name  Seq C DD Information (DSN/Sysin/Sysout/Dummy)  DISP  
-----  -----  -----  -----  
STEPLIB  1  'DNET074.ADLAB.LOAD'  SHR  
-----  -----  -----  -----  
***** Top of Data *****  
***** Bottom of data *****
```

10. Press **ENTER** to accept the file information.

Debug Tool Mentor Workshop

Lab Exercises



11. Next, you will run the program, *WITHOUT* starting Debug Tool.

- Type **RUN** on the command line, then **ENTER**.

```
EDIT - Edit Setup File 'DNET074.ADLAB.DTSF(ADSTAT)' RC 0
Command ==> RUN █ Scroll ==> CSR

Modify information and use the Run command to execute,
or the Submit command to submit to Batch.
Press HELP for a list of all available commands.

Load Module Name ADSTAT
Choose the format of your parameter string:
 1 LE COBOL Default - Program Arguments / Run-time Options
 2 Other LE Languages - Run-time Options / Program Arguments
 3 Non-LE Programs / OS/VS COBOL - Run-time Options / Program Arguments

_ Enter / to modify parameters _____

Cmd DD Name Seq C DD Information (DSN/Sysin/Sysout/Dummy) DISP
-----
STEPLIB 1 'DNET074.ADLAB.LOAD' SHR
***** Top of Data *****
***** Bottom of data *****

F1=Help F3=Exit F4=Run F7=Backward F8=Forward F10=Submit
F12=Cancel
```

12. The program will run. You will see the program display “Hello” and “Goodbye” messages.

- Press **ENTER** to clear the messages.

Note: It's OK if you do not see the “Hello” and “Goodbye” messages. They do not appear on certain TSO systems, depending on how your system is configured.

Debug Tool Mentor Workshop

Lab Exercises



13. Next, you will run the program *WITH* Debug Tool. Debug Tool will start when you pass the TEST parameter to Language Environment.

- Type `'/TEST'` in the parameters field .
- Type **RUN** on the command line, then **ENTER** .

```

EDIT - Edit Setup File 'DNET074.ADLAB.DTSF (ADSTAT)' RC 0
Command ==> RUN Scroll ==> CSR

Modify information and use the Run command to execute,
or the Submit command to submit to Batch.
Press HELP for a list of all available commands.

Load Module Name ADSTAT
Choose the format of your parameter string:
 1 LE COBOL Default - Program Arguments / Run-time Options
 2 Other LE Languages - Run-time Options / Program Arguments
 3 Non-LE Programs / OS/VS COBOL - Run-time Options / Program Arguments
_ Enter / to modify parameters '/TEST'

-----
Cmd DD Name Seq C DD Information (DSN/Sysin/Sysout/Dummy) DISP
-----
***** Top of Data *****
STEPLIB 1 'DNET074.ADLAB.LOAD' SHR
***** Bottom of data *****

F1=Help F3=Exit F4=Run F7=Backward F8=Forward F10=Submit
F12=Cancel
    
```

14. When the program executes, Debug Tool will appear on your TSO terminal.

SUCCESS! Debug Tool appears.

```

COBOL LOCATION: ADSTAT initialization
Command ==> Scroll ==> CSR
MONITOR --+---1---2---3---4---5---6 LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE: ADSTAT ---1---2---3---4---5--- LINE: 1 OF 137
1 IDENTIFICATION DIVISION.
2 PROGRAM-ID. ADSTAT.
3 ENVIRONMENT DIVISION.
4 DATA DIVISION.
5 WORKING-STORAGE SECTION.
6

LOG 0---1---2---3---4---5--- LINE: 41 OF 43
0041 IBM Debug Tool Version 5 Release 1 Mod 0
0042 07/07/2005 09:51:10 AM
0043 5655-M18 and 5655-M19: (C) Copyright IBM Corp. 1992, 2004
***** BOTTOM OF LOG *****
PF 1: ? 2: STEP 3: QUIT 4: LIST 5: FIND 6: AT/CLEAR
PF 7: UP 8: DOWN 9: GO 10: ZOOM 11: ZOOM LOG 12: RETRIEVE
    
```


Debug Tool Mentor Workshop

Lab Exercises



Note: You may see error messages as Debug Tool tries to open the Preferences file (INSPREF) and the Log file (INSPLOG). IGNORE these errors, since these files will not be needed this exercise.

15. At this point, you have an active Debug Tool session, and you are ready for exercise 5.

Debug Tool Mentor Workshop

Lab Exercises



Lab Exercise 5

In this exercise you will:

- Learn the basic Debug Tool commands.
- 1. Before you do this exercise, you must first do either exercise 4A (preferred) or exercise 4B to start a Debug Tool session. Before you continue, you must have a Debug Tool session started in program ADSTAT.
- 2. Get help with commands using the “?” command.
- Type **ZOOM LOG** on the command line, then **ENTER**.

```
COBOL LOCATION: ADSTAT initialization
Command ==> zoom log
MONITOR ---+---1---+---2---+---3---+---4---
***** TOP OF MONITOR ***
***** BOTTOM OF MONITOR *
```

- The log window will be displayed in full screen mode.
 - Type **?** on the command line and **ENTER**. (The “?” command by itself displays a list of all commands.)
- You can type “?” into any command to get a list of valid options for the command. This next command will display the different options of the “AT” command (used to set breakpoints):
 - Type **AT ?** on the command line, then **ENTER**. A list of the things that you can type after the word “AT” is displayed.

```
COBOL LOCATION: ADSTAT initialization
Command ==>
LOG 0 ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8---+---9---+---10---+---11---+---12---
0048 ENABLE LOADDEBUGDATA RUN
0049 The partially parsed command is:
0050 AT
0051 The next word may be one of:
0052 ( EVERY
0053 * EXIT
0054 %BLOCK FROM
0055 %LINE GLOBAL
0056 %STATEMENT LABEL
0057 block name LINE
0058 compile unit specification LOAD
0059 statement number OCCURRENCE
0060 APPEARANCE OFFSET
0061 CALL PATH
0062 CHANGE STATEMENT
0063 CURSOR TERMINATION
0064 DATE TO
0065 DELETE TOGGLE
0066 ENTRY
PF 1: ? 2: STEP 3: QUIT 4: LIST 5: FIND 6: AT/CLEAR
PF 7: UP 8: DOWN 9: GO 10: ZOOM 11: ZOOM LOG 12: RETRIEVE
```

Debug Tool Mentor Workshop

Lab Exercises



- PF10 is set to the ZOOM command. It will zoom in, and will also zoom out.
 - Press **PF10** to zoom out of the log window.
 - All three windows should be displayed again.
- The STEP command can be used to execute statements. Try these different methods:
 - Type **STEP** on the command line, then **ENTER**
 - PF2**
- Set breakpoints. Try these different methods:
 - Place the cursor on statement 31 in the Source window and then **PF6**.
 - Type **AT 32** on the command line (where 32 is a valid statement number) then **ENTER**.
 - Type **A** in the prefix area (the numbered area to the left of the program) of statement 33 in the Source window, then **ENTER**.
- You should have some breakpoints set now. Notice that the colors are reversed in the prefix area (on the left side of the screen) next to the statements with breakpoints. Also notice that all of your commands have been written to the log (the bottom window).

```
COBOL      LOCATION: ADSTAT  :> 29.1
Command ==> |                               Scroll ==> CSR
MONITOR ---+---1---+---2---+---3---+---4---+---5---+---6 LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE: ADSTAT ---1---+---2---+---3---+---4---+---5--- LINE: 29 OF 137
29  MOVE 'PROGRAM STARTING' TO PROGRAM-STATUS.
30  DISPLAY 'HELLO'.
31  PERFORM 100-GENERATE-ARRAY.
32  PERFORM 200-CALC-AVG-AND-SUM.
33  PERFORM 300-CALC-MIN-MAX-AND-RANGE.
34  PERFORM 400-SORT-ARRAY.

LOG 0 ---+---1---+---2---+---3---+---4---+---5--- LINE: 68 OF 71
0068 STEP ;
0069 AT 31 ;
0070 AT 32 ;
0071 AT 33 ;
PF 1: ?      2: STEP      3: QUIT      4: LIST      5: FIND      6: AT/CLEAR
PF 7: UP     8: DOWN     9: GO      10: ZOOM     11: ZOOM LOG  12: RETRIEVE
```

Debug Tool Mentor Workshop

Lab Exercises



7. You can display (List) your breakpoints:

- Type **LIST AT** on the command line, then **ENTER**.
- Your breakpoints are displayed in the log window. You *may* need to use PF11 (or the “Zoom Log” command) to enlarge the log window. If you do, you can use PF11 again to re-display all windows.

8. Clear breakpoints. Try these different methods:

- Place the cursor on statement 33, and press **PF6**. Notice that PF6 will toggle breakpoints on and off.
- Type **CLEAR AT 31** on the command line (where 31 is a valid statement number) then **ENTER**.
- Type **C** (for Clear) in the prefix area (the numbered area to the left of the program) of a statement with a breakpoint, then **ENTER**.
- To clear all breakpoints with a single command, type **CLEAR AT** on the command line, then **ENTER**.

9. Set breakpoints at statements 62 and 64. It’s OK if you have other breakpoints set. You can scroll forward (**PF8**) to see these statements.

```
COBOL      LOCATION: ADSTAT  :> 29.1
Command ==>
                                                    Scroll ==> CSR
MONITOR ---+----1----+----2----+----3----+----4----+----5----+----6 LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE: ADSTAT ---1----+----2----+----3----+----4----+----5----+ LINE: 60 OF 95
60      300-CALC-MIN-MAX-AND-RANGE.
61      MOVE 'CALCULATING MIN, MAX, AND RANGE' TO PROGRAM-STATUS
62      MOVE NUM(1) TO WORK-MIN.
63      MOVE NUM(1) TO WORK-MAX.
64      PERFORM VARYING SUB-A FROM 2 BY 1
65      UNTIL SUB-A > ARRAY-SIZE

LOG 0---+----1----+----2----+----3----+----4----+----5----+----6 LINE: 27 OF 30
0027 STEP ;
0028 STEP ;
0029 AT 62 ;
0030 AT 64 ;
PF  1: ?      2: STEP      3: QUIT      4: LIST      5: FIND      6: AT/CLEAR
PF  7: UP      8: DOWN      9: GO       10: ZOOM     11: ZOOM LOG  12: RETRIEVE
```

10.

Debug Tool Mentor Workshop

Lab Exercises



11. Run the program until it reaches a breakpoint.

- Type **GO** on the command line, then ENTER.
 - The program should be stopped at a breakpoint.
 - If the program ended, then you probably did not have a breakpoint set where it would be reached. If that happened, re-run the program (see exercise 4a or 4b) to start Debug Tool again. Re-run the program as many times as needed throughout this exercise.
- PF9 is set to the GO command. Make sure you have a breakpoint that will be reached. Then, press **PF9** to GO.

12. Practice setting, clearing, and running to breakpoints at statements until you are familiar with:

- the “AT #” and “CLEAR AT #” commands
- PF6 as a toggle to turn breakpoints on or off
- the “A” (At) and “C” (Clear) line commands
- PF9 or the “GO” command to run to a breakpoint

13. You can use the JUMPTO command to “jump” to a statement and continue execution at this statement. This is sometimes helpful to re-execute a few statements in the program. Try the following:

- Make sure you have breakpoints at statements 62 and 64.
- Run the program (GO or PF9) until it is stopped at statement 64 or later.
- Type **JUMPTO 61** on the command line, then ENTER.
- The program should be stopped at statement 61.

```
COBOL      LOCATION: ADSTAT :> 61.1
Command ==> █                               Scroll ==> CSR
MONITOR  --+---1---+---2---+---3---+---4---+---5---+---6 LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE:  ADSTAT  --+---1---+---2---+---3---+---4---+---5---+ LINE: 60 OF 95
60      300-CALC-MIN-MAX-AND-RANGE.
61      MOVE 'CALCULATING MIN, MAX, AND RANGE' TO PROGRAM-STATUS
62      MOVE NUM(1) TO WORK-MIN.
63      MOVE NUM(1) TO WORK-MAX.
64      PERFORM VARYING SUB-A FROM 2 BY 1
65      UNTIL SUB-A > ARRAY-SIZE

LOG 0--+---1---+---2---+---3---+---4---+---5---+ LINE: 30 OF 33
0030 AT 64 ;
0031 GO ;
0032 GO ;
0033 JUMPTO 61 ;

PF 1: ?      2: STEP      3: QUIT      4: LIST      5: FIND      6: AT/CLEAR
PF 7: UP     8: DOWN     9: GO      10: ZOOM     11: ZOOM LOG  12: RETRIEVE
```

Debug Tool Mentor Workshop

Lab Exercises



14. You can use the **GOTO** command instead of **JUMPTO**. It is similar to **JUMPTO**, but it does not stop at the statement. It jumps to the statement and immediately continues to run. The program will automatically run to the next breakpoint (or the end of the program).

- Make sure you have breakpoints at statements 62 and 64.
- Run the program until it is stopped at statement 64.
- Type **GOTO 61** on the command line, then **ENTER**.
- Notice that statement 61 executed, but Debug Tool did **NOT** stop at 61. A **GOTO** command will not stop at the target statement unless you have a breakpoint there. The program is stopped at statement 62, which was the next breakpoint that it reached.

15. Display the value of a program variable in the Log. Try these different methods:

- Type **LIST PROGRAM-STATUS** on the command line and **ENTER**. ('PROGRAM-STATUS' is a variable name.)
- Type **LIST** on the command line, then place the cursor directly on any variable that appears in the Source window, then **ENTER**.
- Place the cursor on a variable in the Source window, then **PF4**.

```

COBOL      LOCATION: ADSTAT :> 61.1
Command ===>                                     Scroll ==> CSR
MONITOR  --+---1---+---2---+---3---+---4---+---5---+---6 LINE: 0 OF 0
*****
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE: ADSTAT  --1---+---2---+---3---+---4---+---5--- LINE: 59 OF 137
59
60      300-CALC-MIN-MAX-AND-RANGE.
61      MOVE 'CALCULATING MIN, MAX, AND RANGE' TO PROGRAM-STATUS
62      MOVE NUM(1) TO WORK-MIN.
63      MOVE NUM(1) TO WORK-MAX.
64      PERFORM VARYING SUB-A FROM 2 BY 1

LOG 0 --+---1---+---2---+---3---+---4---+---5---+---6 LINE: 106 OF 109
0106 LIST PROGRAM-STATUS ;
0107 PROGRAM-STATUS = 'CALCULATING MIN, MAX'
0108 LIST ( WORK-MAX ) ;
0109 WORK-MAX = +0000000000
PF 1: ?      2: STEP      3: QUIT      4: LIST      5: FIND      6: AT/CLEAR
PF 7: UP     8: DOWN     9: GO      10: ZOOM     11: ZOOM LOG  12: RETRIEVE
    
```

Debug Tool Mentor Workshop

Lab Exercises



16. Display a variable in the monitor window. Try different methods:

- Type **MONITOR LIST PROGRAM-STATUS** then **ENTER**. Try abbreviating this command... For example: **MON LI variable-name**
- Type **MON LIS** (an abbreviation for MONITOR LIST) on the command line, then place the cursor on any variable that appears in the Source window, then **ENTER**.

```

COBOL      LOCATION: ADSTAT  :> 61.1
Command ==>
MONITOR  --+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6 LINE: 1 OF 2
***** TOP OF MONITOR *****
0001  1 PROGRAM-STATUS  'CALCULATING MIN, MAX'
0002  2 SUB-A          +00000000006
***** BOTTOM OF MONITOR *****

SOURCE:  ADSTAT  ---1---+---2---+---3---+---4---+---5--- LINE: 59 OF 137
59
60      300-CALC-MIN-MAX-AND-RANGE.
61      MOVE 'CALCULATING MIN, MAX, AND RANGE' TO PROGRAM-STATUS
62      MOVE NUM(1) TO WORK-MIN.
63      MOVE NUM(1) TO WORK-MAX.
64      PERFORM VARYING SUB-A FROM 2 BY 1

LOG 0 ---+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6 LINE: 101 OF 104
0101  MONITOR
0102  LIST ( PROGRAM-STATUS ) ;
0103  MONITOR
0104  LIST ( SUB-A ) ;
PF 1: ?      2: STEP      3: QUIT      4: LIST      5: FIND      6: AT/CLEAR
PF 7: UP      8: DOWN      9: GO       10: ZOOM     11: ZOOM LOG  12: RETRIEVE
  
```

17. Note: Most items that you can display in the log with a LIST command, you can also display in the monitor window with MONITOR LIST command.

18. Remove items from the Monitor Window:

- Type **C** (for Clear) next to a variable in the prefix area (the numbers on the left) of the Monitor window, then **ENTER**.

```

COBOL      LOCATION: ADSTAT  :> 62.1
Command ==>
MONITOR  --+-----1-----+-----2-----+-----3-----+-----4-----+
***** TOP OF MONITOR *****
C 01  1 PROGRAM-STATUS  'CALCULATING MIN, MAX'
0002  2 SUB-A          +00000000006
***** BOTTOM OF MONITOR *****
  
```

- To remove all items from the Monitor Window: Type **CLEAR MONITOR** on the command line, then **ENTER**. (Or abbreviate this command, such as : **CLE MON**)

Debug Tool Mentor Workshop

Lab Exercises



19. Using MONITOR LIST commands, as described in the last few steps, put the following variables in the Monitor Window:

- PROGRAM-STATUS
- WORK-MAX
- NUM

20. Modify a variable value.

- Tab to the data area of the WORK-MAX variable in the Monitor window.
- Change the data to 12345, clear out the rest of the field, then **ENTER**.
- Notice that a command was placed on the command line. **ENTER** to execute the command.

```
COBOL      LOCATION: ADSTAT  :> 61.1
Command ==>
MONITOR  ---+---1---+---2---+---3---+---4---
0001     1 PROGRAM-STATUS 'CALCULATING MIN, MAX'
0002     2 WORK-MAX      12345
0003     3 02 ADSTAT:>NUM
0004     SUB (1)        +00000000100
```

21. Automatically display active variables in the Monitor window.

- Turn on the Automonitor: Type **SET AUTO ON** on the command line, then **ENTER**.
- The “Automonitor” appears at the bottom of the Monitor window. You may have to scroll forward in the Monitor to see it. This will automatically display variables referenced by the current statement in the program.
- Use **PF2** (or the STEP command) a few times. Notice that as you step, the Automonitor changes to show variables referenced by the current statement.

22. You can set a breakpoint that will stop when a variable changes.

- First, clear ALL of your breakpoints:
 - Type **CLEAR AT** on the command line, then **ENTER**.
- For your reference, add variable SUB-A to the Monitor:
 - Type **MON LIST SUB-A** on the command line, then **ENTER**.

Debug Tool Mentor Workshop

Lab Exercises



- Set the breakpoint:
 - Type **AT CHANGE SUB-A** on the command line, then **ENTER**.
- Notice the value of SUB-A in the monitor.
- Press **PF9** (the GO command).
- Notice that the value of SUB-A has changed. Debug Tool has stopped at a statement immediately AFTER a statement that changed variable SUB-A.
- Tip: You can stop when a condition is true, using a “WHEN” clause. For example:
AT CHANGE SUB-A WHEN SUB-A > 2.
Here’s another example. If a program reads a file, and a variable named RECORD-KEY is in the record, you could stop when the program gets to the right record by setting a breakpoint such as:
AT CHANGE RECORD-KEY WHEN RECORD-KEY = ‘ABCD’

23. Clear the AT CHANGE breakpoint:

- For your reference, list your breakpoints. Type **LIST AT** on the command line, then **ENTER**.
- Notice that the AT CHANGE breakpoint is listed in the log.
- Type **CLEAR AT CHANGE SUB-A** on the command line, then **ENTER**.
- Type **LIST AT** on the command line again, then **ENTER**. Notice that the second LIST AT command in the log does not show the breakpoint.

24. You can set a breakpoint to stop at a sub-program. In this example program, the main program ADSTAT calls a sub-program named ADSORT.

- Set a breakpoint to stop when execution reaches (enters) sub-program ADSORT:
 - Type **AT ENTRY ADSORT** on the command line, then **ENTER**.
- Run the program:
 - Press **PF9** (the GO command) .
- The program should be stopped in sub-program ADSORT. If your Debug session ended, re-start the program and try again.
- You can step into the sub-program with **PF2** (or the **STEP** command) .

Debug Tool Mentor Workshop

Lab Exercises



25. Zoom in on the Source Window, by placing the cursor anywhere in the Source Window, then **PF10**.

26. Zoom back out with **PF10**.

27. Use the same method to Zoom in and out of the other windows.

- o Note: Instead of using a PF key, you could also use the **ZOOM** command (or just **Z**).

28. You can easily locate the current statement:

- First , scroll forward in the Source window by pressing **PF8** a few times.
- Type **QUALIFY RESET** on the command line, then **ENTER**. (can be abbreviated, such as **QUAL RES**).

29. You can copy one or more executed commands from the log window to the command line and re-execute them:

- Put the cursor on any logged command in the log window. Overtyping any character on that command.
 - Hint: You can overtype a character with the same character. For example, change **MOVE** to **mOVE**
- **ENTER**.
- The command is copied to the command line. You can make changes (if desired), and **ENTER** to execute.

30. You can copy one or more program statements to the command line, and execute them.

- Put the cursor on any **MOVE** or **COMPUTE** statement in the source window, and overtype any character on that statement.
 - Hint: You can overtype a character with the same character. For example, change **MOVE** to **mOVE**
- **ENTER**.
- The statement is copied to the command line. If the command is long, or if you overtype multiple statements, they may be placed into a temporary expanded command area below the command line. You can make changes (if desired).
- If the statement contains a period, you must remove it.

Debug Tool Mentor Workshop

Lab Exercises



-
- Press **ENTER** to execute the statement.

31. You can continue a long command with a dash (-). This can be helpful if you need to type a command that is too long to fit on the command line.

- On the command line type **MONITOR LIST -** then **ENTER**.
- Debug Tool will prompt for the rest of the command.
- On the command line, type any valid variable name, such as: **NUM** , then **ENTER**.

32. End the Debug Tool session:

- Type **QUIT** on the command line, then **ENTER**.
- When you receive the “Do you really want to ...” prompt, type Type **Y** on the command line, then **ENTER**.

Lab Exercise 6

In this exercise you will:

- Customize Debug Tool so that it will automatically save and restore your Breakpoints and Settings.
1. Note: Debug Tool will use two files to save Breakpoints and Settings. The files must be pre-allocated before Debug Tool will use them. The files are:
 - ***userid*.DBGTOOL.SAVESETS**
 - This is a sequential file where your Debug Tool settings will be stored.
 - ***userid*.DBGTOOL.SAVEBPS**
 - This is a PDSE (or PDS) file where breakpoints will be saved. The member names will be the names of the high-level program of each application.
 2. Run a sample batch job to allocate the files:
 - Edit the JCL in member '**your-user-id.ADLAB.JCL(DTSAVSET)**'.
 - Add a valid job card.
 - Submit the job.
 - Verify that the job ran correctly, and that the files were allocated.
 3. Start a Debug Tool session. If you aren't sure how to do this, refer to exercise 4A (preferred) or exercise 4B to start a Debug Tool session.
 4. In the next step, you will specify the settings that will make Debug Tool automatically restore settings and breakpoints at the beginning of each session, and automatically save settings and breakpoints at the end of each session.

Debug Tool Mentor Workshop

Lab Exercises



5. Enter each of these Debug Tool commands:

- **SET SAVE SETTINGS AUTO**
- **SET SAVE BPS AUTO**
- **SET SAVE MONITORS AUTO**
- **SET RESTORE SETTINGS AUTO**
- **SET RESTORE BPS AUTO**
- **SET RESTORE MONITORS AUTO**

Tip: You can abbreviate the commands:

```
SET SAV S A
SET SAV B A
SET SAV M A
SET RES S A
SET RES B A
SET RES M A
```

Tip: Use PF12 to retrieve commands.

6. Debug Tool will remember your settings and breakpoints now. Enter these commands to permanently save these settings:

- **SET AUTO ON** (turn on the Automonitor)
- **SET DEFAULT SCROLL CSR** (scroll based on you cursor position)
- **SET WARNING OFF** (allows you to update variable values, even in programs compiled with the OPTimize option)
- **SET PF16 "MONITOR" = MON LOCAL %CU LIST CURSOR** (allows you add a variable to the monitor with cursor selection and PF16)

7. Exit Debug Tool:

- Type **QUIT** on the command line, then **ENTER**.
- When you receive the “Do you really want to ...” prompt, type Type **Y** on the command line, then **ENTER**.

8. Restart a Debug Tool session (refer to exercise 3A or 3B if you aren't sure how).

9. Debug Tool should have automatically restored your settings. One indication is that you will see the message: “SETTINGS restored from *userid*.DBGTOOL.SAVESETS” in the log. If you aren't sure, you can use the command: **QUERY SET** to see a list of your current Debug Tool settings.

Debug Tool Mentor Workshop

Lab Exercises



10. Exit Debug Tool:

- Type **QUIT** on the command line, then **ENTER**.
- When you receive the “Do you really want to ...” prompt, type Type **Y** on the command line, then **ENTER**.

Tip: If you ever need to re-create your SAVE files, you can use the sample JCL in ‘userid.ADLAB.JCL(DTSAVSET)’ to easily re-allocate them. This job also enables the needed settings for auto save and restore.

Debug Tool Mentor Workshop

Lab Exercises



Lab Exercise 7

In this exercise you will:

- Learn to start Debug Tool for an application running in a batch job.
1. Log on to TSO.
 2. Open the following dataset in an EDIT session:

'your-tso-id.ADLAB.JCL(XSAM)'

```
SPF/E EDIT DNET074.ADLAB.JCL(XSAM) - 01.03           Columns 00001 00072
Command ==> _____ Scroll ==> CSR
***** ***** Top of Data *****
000001 /**      - - -  ADD A JOB CARD HERE  - - -
000002 /*******
000003 /**      RUN SAMPLE PROGRAM SAM1
000004 /**      INSTRUCTIONS FOR DEBUG TOOL:
000005 /**      1) PREPARE THE DEBUG TOOL INTERFACE (TERMINAL OR GUI)
000006 /**      2) ADD A JOB CARD
000007 /**      3) CUSTOMIZE AND UN-COMMENT ONE OF THE EXEC TEST PARMS
000008 /**      4) SUBMIT
000009 /*******
000010 //RUNSAM1 EXEC PGM=SAM1,
000011 /**          PARM='/TEST(,,MFI%TRMLU099:)',
000012 /**          PARM='/TEST(,,TCPIP&123.45.67.89%8001:)',
000013 /**          PARM='/TEST(,,VTAM%USERID:)',
000014 //          REGION=4M
000015 //STEPLIB DD DSN=&SYSUID..ADLAB.LOAD,DISP=SHR
000016 /**          DD DISP=SHR,DSN=DEBUG.V6R1.SEQAMOD (UNCOMMENT IF NEEDED)
000017 /**          //INSPREF DD DSN=&SYSUID..ADLAB.DTPREF,DISP=SHR
000018 /**          //INSPLOG DD DSN=&SYSUID..ADLAB.DTLOG,DISP=SHR
000019 //CUSTFILE DD DSN=&SYSUID..ADLAB.FILES(CUST2FA),DISP=SHR
```

3. Customize the JCL to run on your system by making the following changes:
 1. Add a valid job card as needed for your system(s).
 2. The instructor may have told you about other JCL customizations needed for your system.

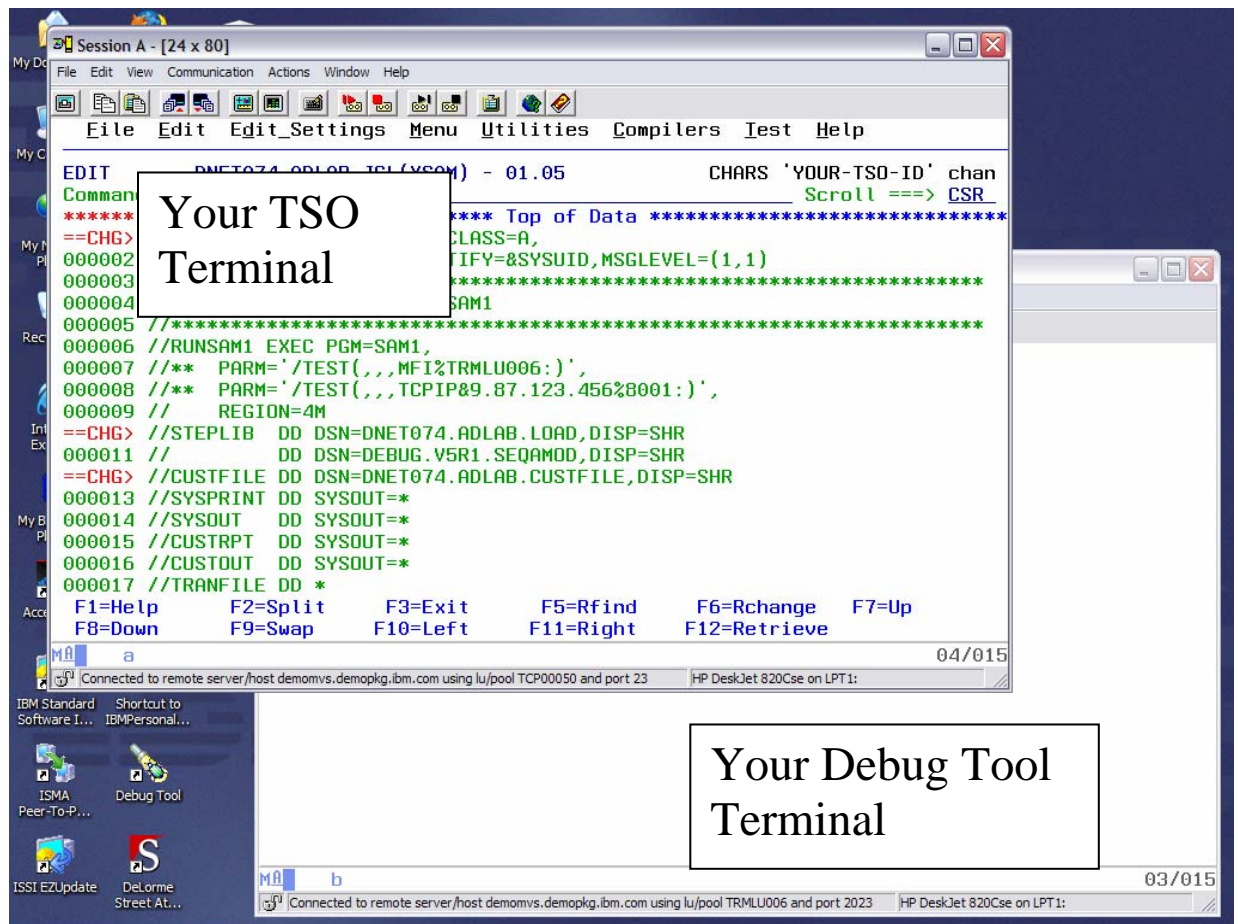
Leave your TSO edit session active.

Debug Tool Mentor Workshop

Lab Exercises



4. Start a second terminal emulator window. We will call this your “Debug Tool terminal”. Your instructor will have shown you how do this in class. DO NOT LOG ON to this second emulator window. To make it work, your Debug Tool terminal must NOT be connected to ANY application. Not even a session manager.
 - Find the Terminal ID of your Debug Tool terminal. FYI... Technically, the “Terminal ID” is the “VTAM LU” (Logical Unit) ID.
 - What is the Terminal ID? _____
5. You now have 2 terminal emulator windows active:
 1. One where you are logged on to TSO, with the JCL open in an Edit session.
 2. Your Debug Tool terminal, where you are not logged on.



Debug Tool Mentor Workshop

Lab Exercises



6. Click on your TSO terminal. Notice the commented parameter strings in the lines immediately after the EXEC statement.

- UN-comment the first parameter line (This is the one that contains the string “MFI”).
- Change the terminal id in the TEST parm to the Terminal ID of your Debug Tool terminal. Be careful not to change the format of the TEST string. The line with the TEST parm should be similar to:

```
//      PARM= '/TEST( , , ,MFI%TRMLU012: ) ' ,
```

(Instead of TRMLU012, you should the name of your Debug Tool terminal specified.

Important Note: The special characters % and : are required!.

7. Your JCL should look similar to this example, although it will have your unique job card, your dataset names, and (most importantly) your Debug Tool Terminal ID:

```
SPF/E EDIT DNET074.ADLAB.JCL(XSAM) - 01.03                Columns 00001 00072
Command ==> _____ Scroll ==> CSR
***** ***** Top of Data *****
000001 //DNET074X JOB (ACCTG), 'IBM TOOLS WORKSHOP', REGION=4M, CLASS=A,
000002 //          MSGCLASS=H, NOTIFY=&SYSUID, MSGLEVEL=(1,1)
000003 //*****
000004 //*      RUN SAMPLE PROGRAM SAM1
000005 //*      INSTRUCTIONS FOR DEBUG TOOL:
000006 //*      1) PREPARE THE DEBUG TOOL INTERFACE (TERMINAL OR GUI)
000007 //*      2) ADD A JOB CARD
000008 //*      3) CUSTOMIZE AND UN-COMMENT ONE OF THE EXEC TEST PARMS
000009 //*      4) SUBMIT
000010 //*****
000011 //RUNSAM1 EXEC PGM=SAM1,
000012 //          PARM= '/TEST( , , ,MFI%TRMLU012: ) ' ,
000013 //**          PARM= '/TEST( , , ,TCPIP&123.45.67.89%8001: ) ' ,
000014 //**          PARM= '/TEST( , , ,VTAM%USERID: ) ' ,
000015 //          REGION=4M
000016 //STEPLIB DD DSN=&SYSUID. .ADLAB.LOAD, DISP=SHR
000017 //**      DD DISP=SHR, DSN=DEBUG.V6R1.SEQAMOD (UNCOMMENT IF NEEDED)
000018 //**      //INSPREF DD DSN=&SYSUID. .ADLAB.DTPREF, DISP=SHR
000019 //**      //INSPLOG DD DSN=&SYSUID. .ADLAB.DTLOG, DISP=SHR
```

8. Submit the job (use the **SUBMIT** command). Then click on your Debug Tool terminal.

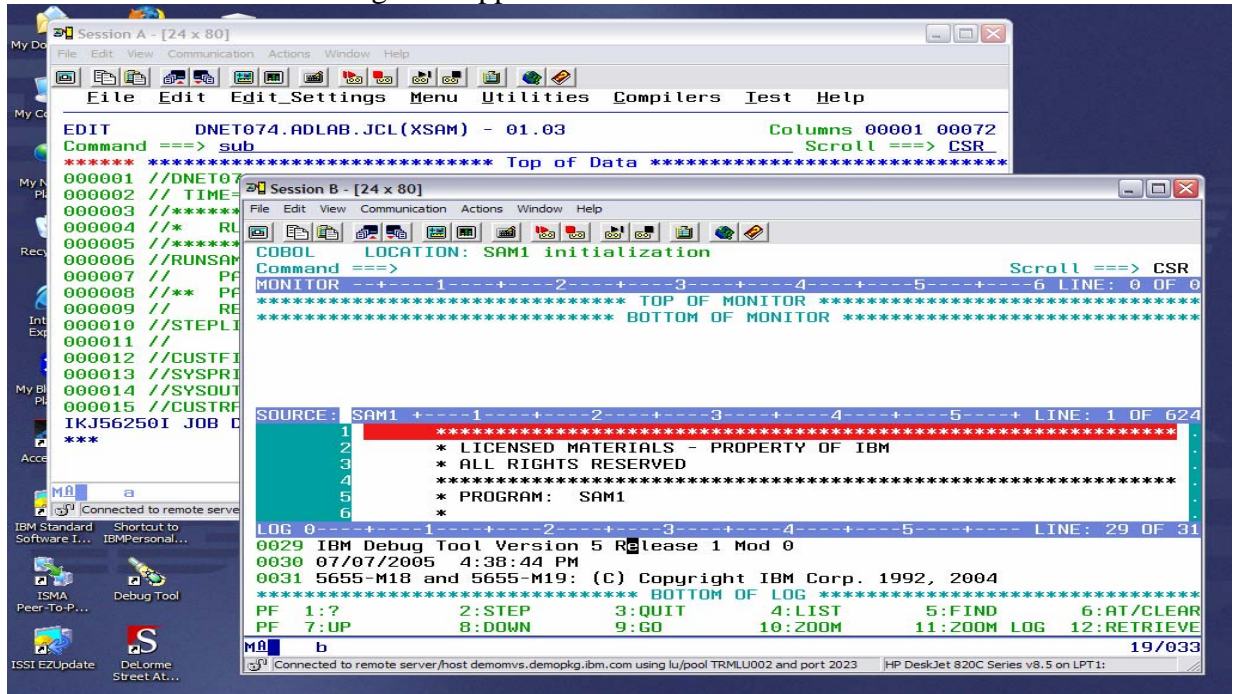
9. When the job runs, Debug Tool will automatically appear on your Debug Tool terminal.

Debug Tool Mentor Workshop

Lab Exercises



SUCCESS! Debug Tool appears.



- If Debug Tool does not appear, check your JCL for errors and try again. Use a JES viewing facility (such as SDSF) to check on the status of your job. Here are some common problems that you should check on:
 - The jobcard is coded incorrectly.
 - The job has a JCL error, due to an invalid dataset name, or some other error.
 - The Terminal ID was coded incorrectly.
 - The TEST parm was coded incorrectly.

10. Click on your Debug Tool session. Use STEP commands to verify that you can step into the program.

11. Use the QUIT command to end your Debug Tool session.

12. Click on your TSO window. Save your JCL changes. You will use this JCL again in later exercises.

Lab Exercise 8

In this exercise you will:

- Use the Debug Tool Setup Utility (DTSU) to start Debug Tool under TSO for a batch application.
 - Create a “Setup File”, which is used to run Debug Tool under TSO.
 - Copy information from existing JCL into the Setup File.
 - “Run” the setup file to start Debug Tool.
1. Log on to TSO.
 2. Open the following dataset in an EDIT session:

'your-tso-id.ADLAB.JCL(XSAMDTU)'

```
EDIT          DNET074.ADLAB.JCL(XSAMDTU) - 01.01          Colu
Command ==>
***** ***** Top of Data *****
000001 //YOUR-TSO-IDL JOB REGION=4M,CLASS=A,
000002 // TIME=(1),MSGCLASS=H,NOTIFY=8,SYSUID,MSGLEVEL=(1,1)
000003 //*****
000004 //*   RUN SAMPLE PROGRAM SAM1
000005 //*****
000006 //RUNSAM1 EXEC PGM=SAM1,
000007 //**  PARM='/TEST(,,MFI%TRMLU099:)',
000008 //**  PARM='/TEST(,,TCPIP&9.87.123.456%8001:)',
000009 //    REGION=4M
000010 //STEPLIB DD DSN=YOUR-TSO-ID.ADLAB.LOAD,DISP=SHR
000011 //**      DD DSN=DEBUG.V5R1.SEQAMOD,DISP=SHR
000012 //CUSTFILE DD DSN=YOUR-TSO-ID.ADLAB.CUSTFILE,DISP=SHR
000013 //SYSPRINT DD SYSOUT=*
```

3. In a few steps, you will “COPY” this JCL into a Debug Tool “setup file”. But first, customize the JCL by making the following changes:
 - a. Change all occurrences of “your-tso-id” to your actual TSO ID or personal file prefix.
 - b. The instructor may have told you about other JCL customizations needed for your system.

Note: You DO NOT need to customize the job card for this exercise.

4. Save your JCL changes and **exit from the editor**.

Debug Tool Mentor Workshop

Lab Exercises



5. Navigate to the Debug Tool Utilities panel.

```
----- Debug Tool Utilities -----
Option ==> █

More: +
0  Manage Job Card
   For Program Preparation and Setup File Management

1  Program Preparation
   Compile old or new COBOL programs with newer compilers, convert old COBOL
   source into new COBOL source, use other compilers, and link edit.

2  Manage and Use Debug Tool Setup Files
   You can manage setup files and use them to run your program interactively
   with Debug Tool in TSO Foreground or submit your program to run in
   the background using MVS batch.

3  Code Coverage
   Measure code coverage in programs written in COBOL, PL/I, C/C++ and
   Assembler when compiled with specific IBM compilers and HLASM.

4  Manage IMS Programs

F1=Help      F2=Split    F3=Exit     F7=Backward F8=Forward  F9=Swap
F12=Cancel
```

6. Select the “Manage and Use Debug Tool Setup Files” option.

```
----- Debug Tool Utilities -----
Option ==> 2█

0  Manage Job Card
   For Program Preparation and Setup File

1  Program Preparation
   Compile old or new COBOL programs with
   source into new COBOL source, use other

2  Manage and Use Debug Tool Setup Files
   You can manage setup files and use them
   with Debug Tool in TSO Foreground or su
```

7. A panel will appear that requests the name of a “setup file”. A setup file is used to save information about an application. A setup file contains information similar to JCL, such as the name of a program to run, and a list of datasets that the application needs.

Debug Tool Mentor Workshop

Lab Exercises



8. Type in the name of your setup file: **your-tso-id.ADLAB.DTSF** and member name **SAM1**.

```

----- Debug Tool Foreground - Edit Setup File -----
Command ==> █

Setup File Library:
Project . . . DNET074
Group . . . ADLAB
Type . . . DTSF
Member . . . SAM1 (Blank or pattern for member
                  (or existing or new member r

Other Data Set Name:
Data Set Name . . .
Volume Serial . . . (If not cataloged)
  
```

9. Press **ENTER**. You have just created a new setup file named “SAM1” in the dataset: your-tso-id.ADLAB.DTSF.

10. The “Edit Setup File” panel will appear. This panel allows the specification of the program to run, and datasets to be allocated (similar to DD statements in JCL).

```

EDIT - Edit Setup File 'DNET074.ADLAB.DTSF(SAM1)' Row 1 to 1 of 1
Command ==> █ Scroll ==> PAGE

Modify information and use the Run command to execute,
or the Submit command to submit to Batch.
Press HELP for a list of all available commands.

Load Module Name _____
Choose the format of your parameter string:
1 1 LE COBOL Default - Program Arguments / Run-time Options
2 Other LE Languages - Run-time Options / Program Arguments
3 Non-LE Programs / OS/VS COBOL - Run-time Options / Program Arguments

_ Enter / to modify parameters _____

Cmd DD Name Seq C DD Information (DSN/Sysin/Sysout/Dummy) DISP
_____ _ ***** Top of Data *****
***** Bottom of data *****
  
```

11. An easy way to get the needed information is to “COPY” it from existing JCL. That’s what we will do in the next few steps.

12. Type **COPY** on the command line, then **ENTER**.

Debug Tool Mentor Workshop

Lab Exercises



13. A panel will appear where you can specify the location of JCL that you want to copy. Type in dataset name: **your-tso-id.ADLAB.JCL** and member name **XSAMDTU**, then **ENTER**.

```

----- Debug Tool Foreground - Copy from Setup File or JCL -----
Command ==> █

Select data to copy into 'DNET074.ADLAB.DTSF(SAM1)'

Setup File or JCL Library:
Project . . . . DNET074
Group . . . . ADLAB . . . . .
Type . . . . JCL
Member . . . . XSAMDTU          (Blank or pattern for member selection list)
                                   (or existing or new member name)

Other Data Set Name:
Data Set Name . . . .
Volume Serial . . . .          (If not cataloged)
  
```

14. A “Copy from JCL” Panel will appear. “Select” the following statements by typing **S** next to them:

- The EXEC statement
- STEPLIB
- CUSTFILE
- SYSPRINT
- SYSOUT
- CUSTRPT
- CUSTOUT
- TRANFILE

```

----- Debug Tool Foreground - Copy from JCL Datas Row
Command ==> _____ S

Enter $* on the command line or on a Sel line to select all JCL
Enter $ on a Sel line to select that JCL statement.
Enter RESET to deselect all JCL statements.

Sel   JCL Image
____ //DNET074L JOB
____ /* from 'DNET074.ADLAB.JCL (XSAMDTU)'
$    //RUNSAM1 EXEC PGM=SAM1
$    //STEPLIB DD DSN=DNET074.ADLAB.LOAD,DISP=SHR
$    //CUSTFILE DD DSN=DNET074.ADLAB.CUSTFILE,DISP=SHR
$    //SYSPRINT DD SYSOUT=*
$    //SYSOUT DD SYSOUT=*
$    //CUSTRPT DD SYSOUT=*
$    //CUSTOUT DD SYSOUT=*
$ █ //TRANFILE DD *

____ *TRAN KEY          ACTION   FIELD NAME   VALUE
____ *-----
____ UPDATE 07025A      ADD      BALANCE      +00000001
____ UPDATE 11112A      ADD      BALANCE      +00000123
  
```

Debug Tool Mentor Workshop

Lab Exercises



15. Press **PF3**. The “Edit Setup File” panel appears again. You should see the DD statements you selected on the bottom of the panel. If not, use the COPY command again to copy information from JCL.

```

EDIT - Edit Setup File 'DNET074.ADLAB.DTSF(SAM1)'          Row 1 to 5 of 8
Command ==> |                                           Scroll ==> PAGE

Modify information and use the Run command to execute,
           or the Submit command to submit to Batch.
           Press HELP for a list of all available commands.

Load Module Name SAM1
Choose the format of your parameter string:
1 1 LE COBOL Default - Program Arguments / Run-time Options
2 Other LE Languages - Run-time Options / Program Arguments
3 Non-LE Programs / OS/VS COBOL - Run-time Options / Program Arguments

_ Enter / to modify parameters _

Cmd DD Name  Seq C DD Information (DSN/Sysin/Sysout/Dummy)  DISP
-----
***** Top of Data *****
CUSTFILE  1  'DNET074.ADLAB.CUSTFILE'  SHR
CUSTOUT   1  SYSOUT=*
CUSTRPT   1  SYSOUT=*
STEPLIB   1  'DNET074.ADLAB.LOAD'     SHR
  
```

16. You can modify the dataset information on the bottom of the panel by overtyping. You can also delete or insert files. Follow these steps to insert a file:

- Type an **I** (for Insert) next to an existing file, then **ENTER**.

```

Cmd DD Name  Seq
-----
i CUSTFILE  1
  CUSTOUT   1
  CUSTRPT   1
  
```

- On the new line that was added, type in information for the new file:
 - DD: **INSPREF**
 - Name: **'your-tso-id.ADLAB.DTPREF'**
 - Disp: **SHR**

```

Cmd DD Name  Seq C DD Information (DSN/Sysin/Sysout/Dummy)  DISP
-----
***** Top of Data *****
insppref  1  'dnet074.adlab.dtpref'  shr
CUSTFILE  1  'DNET074.ADLAB.CUSTFILE'  SHR
  
```

17. Press **ENTER** to accept the file information.

18. Next, you must specify a TEST run-time parameter. Keep in mind that the TEST parm makes Debug Tool start. If you RUN the setup file without a TEST parm, the application will run, but

Debug Tool Mentor Workshop

Lab Exercises



Debug Tool will not be invoked. The next few steps will show you an easy way to specify a TEST parm.

19. Type a / (slash) on the panel next to the text “Enter / to modify parameters”. Then ENTER.

```

Load Module Name SAM1
Choose the format of your parameter string
1 1 LE COBOL Default - Program Argument
2 2 Other LE Languages - Run-time Options
3 3 Non-LE Programs / OS/VS COBOL - Run-t
/ Enter / to modify parameters

```

20. The “Modify Parameter String” panel appears. This panel allows you to modify the run-time parameters.

21. Press **PF3** to accept the default values.

22. The “Edit Setup File” panel appears again. Notice that a TEST parm string has been added.

23. At this point you are ready to run the application (and Debug Tool will start). But first, save your Setup File. Type **SAVE** on the command line, then ENTER.

24. Now, run the application. Type **RUN** on the command line, then ENTER.

```

EDIT - Edit Setup File 'DNET074.ADLAB.D
Command ==> RUN
Modify information and use the Run com
or the Submit command to submit

```

25. When the program executes, Debug Tool will appear on your TSO terminal.

SUCCESS! Debug Tool appears.

```

COBOL LOCATION: SAM1 initialization
Command ==>
MONITOR ---+---1---+---2---+---3---+---4---+---5---+---6 LINE: 0 OF 0
*****
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****
SOURCE: SAM1 +---1---+---2---+---3---+---4---+---5---+ LINE: 1 OF 624
1 *****
2 * LICENSED MATERIALS - PROPERTY OF IBM
3 * ALL RIGHTS RESERVED
4 *****
5 * PROGRAM: SAM1
6 *
LOG 0---+---1---+---2---+---3---+---4---+---5---+ LINE: 31 OF 34
0031 IBM Debug Tool Version 5 Release 1 Mod 0
0032 07/08/2005 11:53:27 AM
0033 5655-M18 and 5655-M19: (C) Copyright IBM Corp. 1992, 2004
0034 *** User preferences file commands end ***
PF 1:? 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CLEAR
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:RETRIEVE

```


Debug Tool Mentor Workshop

Lab Exercises



26. QUIT out of Debug Tool.

27. Exit from the Debug Tool Utilities panels.

Lab Exercise 9

In this exercise you will:

- Learn some helpful commands used to customize Debug Tool.
 - Add a log file to your JCL.
 - Create your own Preferences File. To do this you will:
 - start a Debug Tool session, enter commands to customize your session, and save your log file.
 - copy the log file into your preferences file.
 - start another Debug Tool session, using your new preferences file.
1. Log on to TSO.
 2. Open the JCL member 'your-tso-id.ADLAB.JCL(XSAM)' in an EDIT session.
 3. One way to use a Log file is to add it to your JCL with the special DD name "INSPLOG".
 - UN-comment the INSPLOG DD statement in your JCL.

```
000005 //*****
000006 //RUNSAM1 EXEC PGM=SAM1,
000007 //      PARM='/TEST(,,MFI%TRMLU006:)',
000008 //**  PARM='/TEST(,,TCPIP&9.87.123.456%8001:)',
000009 //      REGION=4M
000010 //STEPLIB  DD DSN=DNET074.ADLAB.LOAD,DISP=SHR
000011 //          DD DSN=DEBUG.V5R1.SEQAMOD,DISP=SHR
000012 //**  INSPREF DD DSN=DNET074.ADLAB.DTPREF,DISP=SHR
000013 //INSPLOG DD DSN=DNET074.ADLAB.DTLOG,DISP=SHR
000014 //CUSTFILE DD DSN=DNET074.ADLAB.CUSTFILE,DISP=SHR
000015 //SYSPRINT DD SYSOUT=*
```

4. Note: Another way to use a log file is to use the Debug Tool command:
SET LOG ON FILE 'file-name' after your Debug Tool session starts.
5. Start a Debug Tool session:
 - Prepare a Debug Tool terminal, then update and SUBMIT the JCL in 'your-tso-id.ADLAB.JCL(XSAM)'.
 - Refer back to exercise 7 if you are not sure how to start a Debug Tool session.
6. You must have a Debug Tool session open before you continue.

Debug Tool Mentor Workshop

Lab Exercises



7. Ensure that the setting to write to the log file is on:

- Type **SET LOG ON** on the command line, then **ENTER**.

8. You can open and close the windows.

- Type **CLOSE LOG** on the command line, then **ENTER**.
 - The Log window will disappear.
- You can still “Zoom” to the log window, even though it is closed. Type **ZOOM LOG** on the command line, then **ENTER**.
 - The Log window will appear.
- Type **ZOOM** on the command line, then **ENTER**.
 - The Monitor and Log windows will appear.
- Type **OPEN LOG** on the command line, then **ENTER**.
 - The Log window will reappear.
- Note: You can open or close any window with the commands: **OPEN MON**, **CLOSE MON**, **OPEN SOURCE**, **CLOSE SOURCE**, **OPEN LOG**, **CLOSE LOG**.

9. You can change the size of the windows.

- Type **SIZE** on the command line, then place your cursor where you want either the top or bottom of the Source window to be, then **ENTER**.
- You can also specify the number of lines in the **SIZE** command:
Type **SIZE 10 SOURCE** on the command line, then **ENTER**.

10. You can change the screen colors:

- Type **COLOR** on the command line, then **ENTER**.
 - The Color Settings screen will appear.
- If you want to, change the colors of some of the items shown. Then use **PF3** to return to the Debug Tool display and see the results.
- Try some different color combinations.

11. You can change the window layout:

Debug Tool Mentor Workshop

Lab Exercises



-
- Type **LAYOUT** on the command line, then **ENTER**.
 - Type the letters **M**, **L**, and **S** into the 3 boxes in layout #2. Use **PF3** to return to the Debug Tool display and see the results.
 - Try some different layouts.
12. To make the following steps easier, make sure that your log window is at least partially visible (open).
13. On the command line, enter the following commands one at a time:
- **SET AUTO ON** (To turn on the Automonitor)
 - **SET DEF SCROLL CSR** (To make Debug Tool scroll based on the cursor position)
 - **SET PF16 'Monitor' = MON LOCAL %CU LIST CURSOR** (To set PF16 to a cursor-selected MONITOR LIST function)
 - **PLAYBACK ENABLE** (To turn on the Playback recorder)
 - **SET FREQ ON** (To turn on the frequency counter)
14. Use the **Quit** command to end the Debug Tool session.
15. Click on your TSO window, and navigate to the editor.
16. Open dataset '**your-tso-id.ADLAB.DTLOG**' in an EDIT session (this is the log file you just generated):
17. The data in this file will become your new Preferences file. Make the following changes:
- Delete any lines that you don't want or need.
 - If you had any "mistakes" in your commands, fix them or delete them.
18. Save your changes and exit from the editor.
19. Using any method you choose, copy the contents of your Log file into your Preferences file.
Copy all records from: 'your-tso-id.ADLAB.DTLOG'
to: 'your-tso-id.ADLAB.DTPREF'.

Debug Tool Mentor Workshop

Lab Exercises




20. At this point, you have created a Preferences file! In the next steps, you will start a Debug Tool session using your new Preferences file.

21. Update your JCL to use the new preferences file.

- Click on your TSO window, and navigate to the editor.
- Edit the JCL in member 'your-tso-id.ADLAB.JCL(XSAM)'.

22. An easy way to use a Preferences file is to add it to your JCL with the special DD name "INSPREF".

- UN-comment the INSPREF DD statement in the JCL.



```
000005 //*****
000006 //RUNSAM1 EXEC PGM=SAM1,
000007 //      PARM=' /TEST(,,,MFI%TRMLU006:)',
000008 //**   PARM=' /TEST(,,,TCPIP&9.87.123.456%8001:)',
000009 //      REGION=4M
000010 //STEPLIB DD DSN=DNET074.ADLAB.LOAD,DISP=SHR
000011 //      DD DSN=DEBUG.V5R1.SEQAMOD,DISP=SHR
000012 █/INSPREF DD DSN=DNET074.ADLAB.DTPREF,DISP=SHR
000013 //INSPLOG DD DSN=DNET074.ADLAB.DTLOG,DISP=SHR
000014 //CUSTFILE DD DSN=DNET074.ADLAB.CUSTFILE,DISP=SHR
000015 //SYSPRINT DD SYSOUT=*

```

23. Note: Another way to use a Preferences file or a Command file is to use the Debug Tool command: USE 'file-name' after your Debug Tool session starts.

24. Verify that you have a Debug Tool terminal available, and that the Terminal ID in the TEST parm of your JCL is the name of that terminal.

25. Submit the JCL. Click on your Debug Tool window.

26. A Debug Tool session will appear on your Debug Tool terminal. If it does not, determine the problem and re-submit the job.

27. In Debug Tool, look in the log window. You should see that the commands from your Preferences file were automatically executed.

- If not, determine and fix the problem, exit from Debug Tool, and submit the job again.

28. Exit from Debug Tool.

29. Save your JCL changes.

Debug Tool Mentor Workshop

Lab Exercises



30. Take a look at the following examples of Preferences files. You may get some ideas about commands that you would like to have in your Preferences file. Browse these datasets:
- a. 'your-tso-id.ADLAB.FILES(**INSPREF**)
 - b. 'your-tso-id.ADLAB.FILES(**INSPRE2**)
 - c. 'your-tso-id.ADLAB.FILES(**INSPRE3**)

Debug Tool Mentor Workshop

Lab Exercises



Lab Exercise 10

In this exercise you will:

- Learn to continue running a program after it has abended.
7. Start a Debug Tool session:
- Prepare a Debug Tool terminal, then customize and SUBMIT the JCL in 'your-tso-id.ADLAB.JCL(XSAM)' that you customized in Exercise 7.
 - Refer back to exercise 7 if you are not sure how to start a Debug Tool session.
7. You must have a Debug Tool session open before you continue.
7. If you have any breakpoints, clear them.
- Type **CLEAR AT** on the command line, then **ENTER**.
 - This removes all breakpoints.
7. Run the program. It will abend (this sample program tries to do a calculation on some bad data)
- Type **GO** on the command line, then **ENTER**.
 - Debug Tool shows the abending statement.

```
COBOL      LOCATION: SAM2 :> 164.1
Command ==>
MONITOR --+---1---+---2---+---3---+---4---+---5---+---6 LINE: 1 OF 3
***** TOP OF MONITOR *****
0001  1 ***** AUTOMONITOR *****
0002  02 SAM2:>CUST-ACCT-BALANCE      X'7C7B5B6C50'
0003  02 SAM2:>WS-UPDATE-NUM      +000000001.23
***** BOTTOM OF MONITOR *****

SOURCE: SAM2 +---1---+---2---+---3---+---4---+---5--- LINE: 164 OF 181
164      COMPUTE CUST-ACCT-BALANCE =
165          CUST-ACCT-BALANCE + WS-UPDATE-NUM
166          COMPUTE TRAN-COUNT = TRAN-COUNT + 1
167
168          END-EVALUATE
169          WHEN 'ORDERS '
          EVALUATE TRAN-ACTION

LOG 0---+---1---+---2---+---3---+---4---+---5--- LINE: 37 OF 40
0037 The operating system has generated the following message:
0038 CEE3207S The system detected a data exception (System Completion
0039 Code=0C7).
0040 The current location is SAM2 ::> SAM2 :> 164.1.
PF 1:?      2:STEP      3:QUIT      4:LIST      5:FIND      6:AT/CLEAR
PF 7:UP     8:DOWN     9:GO      10:ZOOM     11:ZOOM LOG  12:RETRIEVE
```

Debug Tool Mentor Workshop

Lab Exercises



7. Display the variables referenced by the abending statement.
 - If the automonitor is not already on, then type **SET AUTO ON** on the command line, then **ENTER**.
 - This turns on the automonitor, and shows the variable values.
7. Notice that the variable CUST-ACCT-BALANCE has bad data (it caused the abend). Debug Tool automatically displays in Hex variables that contain bad data.
7. Fix the bad data. Overtyping the bad data with a valid number, like **1234** (and clearing the rest of the field). Then **ENTER** (to format a MOVE statement). Then **ENTER** again (to execute the MOVE statement).
7. The CUST-ACCT-BALANCE variable should show a valid data value.
7. Set a breakpoint at the statement immediately after the abending statement:
 - Type **AT 166** on the command line, then **ENTER**. (Verify that this is the statement after the abending statement).
 - You should have a breakpoint set as shown here:

```

COBOL      LOCATION: SAM2 :> 164.1
Command ==>
MONITOR  --+---1---+---2---+---3---+---4---+---5---+---6 LINE: 1 OF 3
***** TOP OF MONITOR *****
0001  1 ***** AUTOMONITOR *****
0002  02 SAM2:>CUST-ACCT-BALANCE      X'7C7B5B6C50'
0003  02 SAM2:>WS-UPDATE-NUM      +000000001.23
***** BOTTOM OF MONITOR *****

SOURCE:  SAM2 +---1---+---2---+---3---+---4---+---5--- LINE: 164 OF 181
164  COMPUTE CUST-ACCT-BALANCE =
165  CUST-ACCT-BALANCE + WS-UPDATE-NUM
166  COMPUTE TRAN-COUNT = TRAN-COUNT + 1
167  END-EVALUATE
168  WHEN 'ORDERS '
169  EVALUATE TRAN-ACTION

LOG 0---+---1---+---2---+---3---+---4---+---5--- LINE: 38 OF 41
0038  CEE3207S The system detected a data exception (System Completion
0039  Code=0C7).
0040  The current location is SAM2 ::> SAM2 :> 164.1.
0041  AT 166 ;

PF  1: ?      2: STEP      3: QUIT      4: LIST      5: FIND      6: AT/CLEAR
PF  7: UP      8: DOWN      9: GO       10: ZOOM     11: ZOOM LOG  12: RETRIEVE

```

7. Bypass the abend:
 - Type **GO BYPASS** on the command line, then **ENTER**.

Debug Tool Mentor Workshop

Lab Exercises



7. Debug Tool passes control to the next statement. It stopped there because of the breakpoint. At this point, theabend has been bypassed, and the program can continue to run.
7. Next, re-execute the statement that abended:
 - Type **JUMPTO 164** on the command line, then **ENTER**. (Verify that this is the line number of the abending statement).
7. Re-execute the statement that abended:
 - Type **STEP** on the command line, then **ENTER**.
7. At this point, you have bypassed theabend, repaired the data, and re-executed the abending statement! The program can continue running.
7. QUIT out of Debug Tool.

Lab Exercise 11

In this exercise you will:

- Set an AT ENTRY breakpoint to stop when a subprogram is reached.
 - Set a breakpoint in a subprogram.
 - Use the STEP RETURN command to return from a subprogram.
 - Use the QUALIFY PROGRAM command to select which program is displayed in the Source window.
1. Start a Debug Tool session:
 - Prepare a Debug Tool terminal, then customize and SUBMIT the JCL in 'your-tso-id.ADLAB.JCL(XSAM)' that you customized in Exercise 7.
 - Refer back to exercise 7 if you are not sure how to start a Debug Tool session.
 2. You must have a Debug Tool session open before you continue.
 3. This sample application will CALL a subprogram named SAM2. You will set an "AT ENTRY" breakpoint that will make Debug Tool stop when subroutine SAM2 is entered.
 - Type **AT ENT SAM2** on the command line, then **ENTER**.
 - Type **GO** on the command line, then **ENTER**.
 - Debug Tool stops when program SAM2 is entered.
 4. Step into the subprogram.
 - Type **STEP** on the command line, then **ENTER**.
 - Debug Tool steps into the subprogram.
 - **STEP** again to step into SAM2 processing.
 5. One way to return back to the calling program is to just continue to STEP until you get there. However, there is an easier way:
 - Type **STEP RETURN** on the command line, then **ENTER**.
 - The program continued to execute until it returned to the calling program. It stops at the statement after the CALL statement.

Debug Tool Mentor Workshop

Lab Exercises



6. This sample application will CALL the subprogram SAM2 more than once. Run the program, to get to the breakpoint that you already set in SAM2.
 - Type **GO** on the command line, then **ENTER**.
 - Debug Tool will stop again at the entry of SAM2.
7. Sometimes, you may want to work with programs that are not the current program. Next, you will tell Debug Tool to show you the main program (SAM1).
 - Type **QUALIFY PROG SAM1** on the command line, then **ENTER**.
 - SAM1 is displayed in the source window.
8. At this point, you could set other breakpoints in SAM1 and look at variables in SAM1.
9. Next, you will return to the active program.
 - Type **QUALIFY RESET** on the command line, then **ENTER**.
 - The active program (SAM2) is displayed.
10. QUIT out of Debug Tool.

Lab Exercise 12

In this exercise you will:

- Learn to start Debug Tool in a CICS environment using the DTCN transaction.
 - Use the Debug Tool Setup Utility (DTSU).
 - Create and save a debugging “profile” that will trigger a Debug Tool session.
 - Run a sample application and start Debug Tool.
1. Connect to CICS. If you normally log on to CICS, then log as well.
 - Your instructor may direct you to connect to a specific CICS region
 2. You may need to “route” to another CICS region. On some systems, you log on to a CICS “Terminal Owning Region” (TOR), and the demo transaction will run on a different region. In this sort of CICS environment, you can “route” to an “Application Owning Region” (AOR). One way to do this is with the CICS “CRTE” (CICS Route) transaction.
 - If needed, “route” to the correct Application Owning Region.
 - Clear the screen.
 - Type **CRTE SYSID=xxxx** and **ENTER** (where XXXX is the Sysid of the AOR where the demo transaction “CDAT” is defined).

Part 1: Become familiar with the demo “CDAT” transaction

3. Start the sample transaction:
 - Clear the screen.
 - Type **CDAT** and **ENTER**.
4. The CDAT birthday application screen is displayed. Take a minute to become familiar with this application.
 - It is a pseudo-conversational CICS application written in COBOL.
 - All application functions are performed under transaction id “CDAT”.
 - The main program is named CDAT1.
 - If you run the “B” function, CDAT1 performs an EXEC CICS LINK to run subprogram CDAT2.
 - If you run the “R” function, program CDAT1 performs a CALL to run subprogram CDAT3.

Debug Tool Mentor Workshop

Lab Exercises



```
ADTOOLS BIRTHDAY/RETIREMENT SAMPLE APPLICATION

  YYYYYMDD  <== PLEASE ENTER BIRTHDATE IN YYYYYMDD FORMAT

  B  <=== ENTER REQUEST
      B : SEE YOUR BIRTHDAY           (LINK TO PROGRAM CDAT2)
      R : CALCULATE RETIREMENT        (CALL PROGRAM CDAT3)
      C : CLEAR AND START OVER
      @ : ABEND WITH SOC7

F3/F12/CLEAR TO TERMINATE, ENTER TO PROCESS
```

5. Familiarize yourself with the CDAT application.
 - Enter your birth date, select the **B** function, and **ENTER** .
 - Enter your birth date, select the **R** function, and **ENTER** .
 - Select the **C** function, and **ENTER** .
 - **PF3** or **CLEAR** to exit the application.

Part 2: Use the DTCN transaction to create a Debugging Profile

6. Start the Debug Tool transaction:
 - Clear the screen.
 - Type **DTCN** and **ENTER** .
7. The DTCN screen is displayed. Review the options on the screen. DTCN is used to define a debugging “profile”. The profile defines the application that you want to debug, and where the debugger is to be displayed.
 - You define the application that you want to trap for debugging. If needed you can use a wildcard of * (asterisk) to make entries generic. For example, you can specify a program name of: ABC*. If you leave any entry blank, then it will match any name.

Debug Tool Mentor Workshop

Lab Exercises



For example, leave the transaction field blank to trap any transaction code. You can specify any combination of:

- The CICS terminal ID where the application will run (usually your terminal).
 - The default is your terminal ID.
 - **Important note:** To debug a “background” transaction (a transaction that is not executed from a terminal), leave the terminal ID blank.
 - The Transaction ID
 - Up to 8 program names where the debugging session will *start*.
 - **Important note:** Once the debugging session starts, Debug Tool will follow the application logic into other programs as well, not just programs in the list.
 - The user ID (usually your ID)
 - The Network name. This is the VTAM LU name of the CICS terminal. In most cases this is not used, but it gives you an alternate method of identifying the terminal (in addition to the CICS terminal ID).
 - The IP address of the network node that initiated a “background” transaction. In some cases, the IP address is known to CICS, and this information can be used to identify a specific instance of a transaction.
- You also define where the Debug Tool session is to be displayed. You can specify either an MFI (3270 interface) session, or a GUI debugger such as WDDz.
- To define an MFI (3270 interface) session:
 - Specify “Session type”: MFI
 - Specify “Port number”: (leave blank)
 - Specify “Display id”: a CICS terminal id (usually your terminal)
 - To define an GUI interface (such as WDDz) session:
 - Specify “Session type”: TCP
 - Specify “Port number”: the TCP port number you set up in your GUI debugging software (usually 8000 or 8001).
 - Specify “Display id”: the TCP address or TCP netname of your workstation

Debug Tool Mentor Workshop

Lab Exercises



```
DTCN                      Debug Tool CICS Control - Primary Menu                      CICSACB4

Select the combination of resources to debug (see Help for more information)
Terminal Id      ==> 2002
Transaction Id  ==>
Program Id(s)   ==>           ==>           ==>           ==>
                ==>           ==>           ==>           ==>
User Id         ==> DNET074
NetName        ==>
IP Name/Address ==>

Select type and ID of debug display device
Session Type    ==> MFI                      MFI, TCP
Port Number     ==>                      TCP Port
Display Id      ==> 2002

Generated String:  TEST(ALL, '*', PROMPT, 'MFI%2002: *')

Repository String: No string currently saved in repository

Profile Status:   No Profile Saved. Press PF4 to save current settings.

PF1=HELP 2=GHELP 3=EXIT 4=SAVE 5=ACT/INACT 6=DELETE 7=SHOW 9=OPTION
```

8. Specify information for your debugging profile. Notice the defaults: your terminal ID is assumed for the application to be debugged, and also for where Debug Tool will be displayed. Your User ID is the default for the application to be trapped.
- Type **CDAT1** in the first program id field.
 - **IF YOU USING THE MFI (3270 INTERFACE):**
 - Leave the rest of the fields unchanged. The default settings will trap only when a transaction is running on your terminal and with your user ID.
 - **IF YOU ARE USING A GUI DEBUGGER (WDDZ, etc.):**
 - **Start the GUI debugger software on your workstation now.**
 - Type **TCP** in the “Session type” field.
 - In the “Port number” field, specify the TCP port address you configured (or took the default for) in your GUI debugger. This is normally **8000** or **8001**.
 - In the “Display ID” field, type the TCP address of your workstation, where the GUI debugger software is listening.

Debug Tool Mentor Workshop

Lab Exercises



```
DTCN                Debug Tool CICS Control - Primary Menu                CICSACB4

Select the combination of resources to debug (see Help for more information)
Terminal Id        ==> Z002
Transaction Id     ==>
Program Id(s)     ==> CDAT1      ==> █          ==>          ==>
                  ==>          ==>          ==>          ==>
User Id           ==> DNET074
NetName           ==>
IP Name/Address   ==>

Select type and ID of debug display device
Session Type      ==> MFI                MFI, TCP
Port Number       ==>                  TCP Port
Display Id        ==> Z002

Generated String:  TEST(ALL, '*', PROMPT, 'MFI%Z002: *')

Repository String: No string currently saved in repository

Profile Status:   No Profile Saved. Press PF4 to save current settings.

PF1=HELP 2=GHELP 3=EXIT 4=SAVE 5=ACT/INACT 6=DELETE 7=SHOW 9=OPTION
```

9. Next, you will review the settings on the LE TEST parm. Debug Tool is started when the TEST parm is passed to LE. By setting up a debugging profile in DTCN, CICS will automatically pass the TEST parm at the appropriate time when your application executes.

- Notice that a TEST parm has already been generated. It is displayed on the bottom of the screen. You can make changes to it.
- Press **PF9**.

10. A screen is displayed where you can modify the options in the TEST parm. For example, if you want to automatically run a Command File or Preferences file, you can type in the names of the files here. These fields need fully qualified file names, without quotes. Note: These files do NOT need to be defined to CICS. However, the CICS region must have read authorization to the files.

- In CICS, it is recommended change the “Test Level” option to ERROR. With the default, Debug Tool will stop at every EXEC CICS RETURN in the program, which many people do not prefer.
 - Type **ERROR** in the “Test Level” field.
 - Press **PF3** to return.

Debug Tool Mentor Workshop

Lab Exercises



```
DTCN                      Debug Tool CICS Control - Menu 2                      CICSACB4

Select Debug Tool options

Test Option      ==> TEST                      Test/Notest
Test Level       ==> error█                    All/Error/None
Commands File    ==> *
Prompt Level     ==> PROMPT
Preference File  ==> *

Any other valid Language Environment options
==>

PF1=HELP 2=GHELP 3=RETURN
```

11. The DTCN main panel is displayed again. Save your profile:

- Press **PF4**.
- Verify that a “Debug Tool profile saved” message was displayed. At this point, Debug Tool is activated, and will trap your application the next time it runs.

12. Notice some of the other PF keys available:

- You can delete your profile with PF6.
- You can see a list of active profiles in the CICS region with PF7.
- You can temporarily inactivate and reactivate your profile with PF5.
- You can delete the profile with PF6. But don’t do this now!

13. Exit from DTCN:

- Press **PF3**.
- A blank screen is displayed.

Part 3: Run the demo transaction and start Debug Tool

14. Run the demo transaction again:

- Clear the screen.
- Type **CDAT** and **ENTER**.

15. Debug Tool should start !

- If Debug Tool did not start, use the DTCN transaction again to review your profile. Some common mistakes are:
 - The wrong program name was entered (should be CDAT1).
 - For the MFI, you had the wrong terminal ID coded (it should be your term id).
 - For WDDz, you had the wrong TCP address or port coded.
 - You forgot to SAVE the profile (PF4).
 - You accidentally “Inactivated” the profile with PF5. If so, activate it by pressing PF5 again.
- Ask you instructor for help if you can’t get a Debug Tool session started.

Part 4: Practice using Debug Tool with the demo transaction

16. Here are some other things to try:

- a. Run the program to completion several time to see the interaction between the application and Debug Tool. As the program executes and terminates, you will see both application screens and the debugger. Practice running the CDAT transaction to become familiar with this interaction.
 - In the MFI interface, run the program with the GO command.
 - In WDDz, run the program with the RESUME button.
- b. Try setting a breakpoint at the entry of subprogram CDAT2, then run until program CDAT2 is trapped. CDAT2 is executed as a subprogram when you select the B function on the CDAT application screen.
 - In the MFI interface, set the breakpoint with the command: AT ENTRY SAM2.
 - Or in WDDz, to set the entry breakpoint:
 - Right-click in the breakpoints window
 - Select “Entry” breakpoint.

Debug Tool Mentor Workshop

Lab Exercises



-
- Set a breakpoint in load module CDAT2, csect CDAT2, entry point CDAT2.
- c. Change the DTCN profile to trap program CDAT2 instead of CDAT1. As you run the transaction, notice that Debug Tool does not start for the main menu in CDAT, but only when program CDAT2 is executed.

Lab Exercise 13

In this exercise you will:

- Learn to start Debug Tool in a CICS environment using the CADP transaction.
 - Create and save a debugging “profile” that will trigger a Debug Tool session.
 - Run a sample application and start Debug Tool.
1. Connect to CICS. If you normally log on to CICS, then log as well.
 - Your instructor may direct you to connect to a specific CICS region
 2. You may need to “route” to another CICS region. On some systems, you log on to a CICS “Terminal Owning Region” (TOR), and the demo transaction will run on a different region. In this sort of CICS environment, you can “route” to an “Application Owning Region” (AOR). One way to do this is with the CICS “CRTE” (CICS Route) transaction.
 - If needed, “route” to the correct Application Owning Region.
 - **Clear the screen.**
 - **Type CRTE SYSID=XXXX and ENTER** (where XXXX is the Sysid of the AOR where the demo transaction “CDAT” is defined).

Part 1: Become familiar with the demo “CDAT” transaction

3. Start the sample transaction:
 - **Clear the screen.**
 - **Type CDAT and ENTER.**
4. The CDAT birthday application screen is displayed. Take a minute to become familiar with this application.
 - It is a pseudo-conversational CICS application written in COBOL.
 - All application functions are performed under transaction id “CDAT”.
 - The main program is named CDAT1.

Debug Tool Mentor Workshop

Lab Exercises



- If you run the “B” function, CDAT1 performs an EXEC CICS LINK to run subprogram CDAT2.
- If you run the “R” function, program CDAT1 performs a CALL to run subprogram CDAT3.

```
ADTOOLS BIRTHDAY/RETIREMENT SAMPLE APPLICATION

YYMMDD <== PLEASE ENTER BIRTHDATE IN YYMMDD FORMAT

B <=== ENTER REQUEST
  B : SEE YOUR BIRTHDAY           (LINK TO PROGRAM CDAT2)
  R : CALCULATE RETIREMENT        (CALL PROGRAM CDAT3)
  C : CLEAR AND START OVER
  @ : ABEND WITH SOC7

F3/F12/CLEAR TO TERMINATE, ENTER TO PROCESS
```

5. Familiarize yourself with the CDAT application.
 - **Enter** your birth date, select the **B** function, and **ENTER**.
 - **Enter** your birth date, select the **R** function, and **ENTER**.
 - **Select** the **C** function, and **ENTER**.
 - **PF3** or **CLEAR** to exit the application.

Debug Tool Mentor Workshop

Lab Exercises



```
CADP      -      CICS Application Debugging Profile Manager      -      CICSACB3

Create Compiled Debugging Profile ==> █          for DNET427

CICS Resources To Debug (use * to specify generic values e.g. *, A*, AB*, etc.)
Transaction      ==>                               Applid      ==> CICSACB3
Program          ==>                               Userid       ==> DNET427
Compile Unit     ==>                               Termid      ==> 0053
                                                        Netname     ==> TCP00053

Debug Tool Language Environment Options
Test Level       ==> All                            (All,Error,None)
Command File     ==>
Prompt Level     ==> PROMPT
Preference File  ==>

Other Language Environment Options
==>
==>
==>
==>

Enter=Create PF1=Help 2=Save options as defaults 3=Exit 10=Replace 12=Return
```

8. Notice where you can modify the options in the TEST parm. For example, if you want to automatically run a Command File or Preferences file, you can type in the names of the files here. These fields need fully qualified file names, without quotes. Note: These files do NOT need to be defined to CICS. However, the CICS region must have read authorization to the files.

- In CICS, it is recommended change the “Test Level” option to ERROR. With the default, Debug Tool will stop at every EXEC CICS RETURN in the program, which many people do not prefer.

9. Fill in the information for your profile:

- **Type your initials** in the “Create Compiled Debugging Profile” field so that your profile has a name. You can give your profile any name, it just has to be unique (not already used by an existing profile).
- **Enter cdat*** as a wild card selection for the programs you want to test using the Debug Tool.
- **Type Error** in the “Test Level” Field.
- **Press ENTER** to create the profile

Debug Tool Mentor Workshop

Lab Exercises



```
CADP      -      CICS Application Debugging Profile Manager      -      CICSACB3

Create Compiled Debugging Profile ==> rlja      for DNET427

CICS Resources To Debug (use * to specify generic values e.g. *, A*, AB*, etc.)
Transaction      ==>
Program          ==> cdat*
Compile Unit     ==>
Applid          ==> CICSACB3
Userid          ==> DNET427
Termid          ==> 0053
Netname         ==> TCP00053

Debug Tool Language Environment Options
Test Level       ==> Error      (All,Error,None)
Command File     ==>
Prompt Level     ==> PROMPT
Preference File  ==>

Other Language Environment Options
==>
==>
==>
==>

Enter=Create PF1=Help 2=Save options as defaults 3=Exit 10=Replace 12=Return
```

10. Return to the main panel.

- **PRESS “PF12”** to return.

```
CADP      -      CICS Application Debugging Profile Manager      -      CICSACB3

List Debugging Profiles      (A=Activate,I=Inactivate,D=Delete,C=Copy)

  Owner  Profile  S Tran Program Compile Unit Applid  Userid  Term Type
  ────  ────  ── ──── ────  ────  ────  ────  ────
█ DNET427 RLJA   I *   CDAT*   *           CICSACB3 DNET427 0053 Comp

1 profile(s) filtered by owner
Enter=Process PF1=Help 2=Filter 3=Exit 4=View 5=Create Comp 6=Create Java
9=Set display device 10=Edit 11=Sort
```


Debug Tool Mentor Workshop

Lab Exercises



Notice that your new profile is displayed in the list now. It has an “T” (inactive) in the S (Status) column. When you first create a profile, it is inactive. Activate the profile so Debug Tool will be invoked when any program starting with CDAT begins:

- **Enter the A line command next to your profile.**
- **PRESS ENTER.** This will bring up this screen:

```
CADP      -      CICS Application Debugging Profile Manager      -      CICSACB3

Set Compiled Debugging Display Device (checked at PROFILE activation time)

Debugging Display Device
Session Type          ==>  3270          (3270,TCP)
3270 Display Terminal ==>  0053

TCP/IP Name Or Address
==>
==>
==>
==>
Port                  ==>  00000

Type of socket communication ==> Single          (Single,Multiple)

Display this panel on LE profile activation ==> YES

Enter=Save and return PF1=Help 3=Exit 12=Cancel
```

- On this screen, you select the 3270 session or TCP session where you want Debug Tool to appear once the program starting with CDAT executes. By default, Debug Tool will appear on your current CICS terminal. For purposes of this exercise leave it as the default.
- **Press Enter.** This will save the updated profile information and return to the primary CADP screen

Debug Tool Mentor Workshop

Lab Exercises



```
CADP      -      CICS Application Debugging Profile Manager      -      CICSACB3

List Debugging Profiles      (A=Activate,I=Inactivate,D=Delete,C=Copy)

  Owner   Profile  S Tran Program  Compile Unit  Applid  Userid  Term  Type
  ────   ────   ── ──── ────   ────   ────   ────   ────   ────
█ DNET427 RLJA    A *   CDAT*    *           CICSACB3 DNET427 0053 Comp

1 activate(s) processed
Enter=Process PF1=Help 2=Filter 3=Exit 4=View 5=Create Comp 6=Create Java
                      9=Set display device 10=Edit 11=Sort
```

11. Notice some of the other PF keys that are available:

- You can filter the profile with PF2. This can be important if there are multiple profiles created by a lot of different people. Pressing PF2 repeatedly will filter by owner, by status, or display all profiles.
- You can view a profile with PF4. This is done by placing the cursor next to the profile you want to view and pressing PF4.

12. You can also use Line Commands to modify the profiles

- You can temporarily inactivate a profile by placing the letter “I” next to the profile you want to inactivate. Do not do this now.
- You can delete a profile by placing the letter “D” next to the profile you want to delete. But don’t do this now!

12. Exit from CADP:

- **Press “PF3”**. A blank screen is displayed.

Part 3: Run the demo transaction and start Debug Tool

13. Run the demo transaction again:

- **Clear the screen**
- **Type CDAT .**
- **ENTER .**

14. Debug Tool should start !

- If Debug Tool did not start, use the CADP transaction again to review your profile. Some common mistakes are:
 - The wrong program name was entered (should be CDAT* or CDAT1).
 - For the MFI, you had the wrong terminal ID coded (it should be your term id).
 - For WDDz, you had the wrong TCP address or port coded.
 - You accidentally “Inactivated” the profile with I command. If so, activate it by entering “A” next to the profile and pressing enter again.
- Ask you instructor for help if you can’t get a Debug Tool session started.

Part 4: Practice using Debug Tool with the demo transaction

15. Here are some other things to try:

- d. Run the program to completion several times to see the interaction between the application and Debug Tool. As the program executes and terminates, you will see both application screens and the debugger. Practice running the CDAT transaction to become familiar with this interaction.
 - In the MFI interface, run the program with the GO command.
 - In WDDz, run the program with the RESUME button.
- e. Try setting a breakpoint at the entry of subprogram CDAT2, then run until program CDAT2 is trapped. CDAT2 is executed as a subprogram when you select the B function on the CDAT application screen.
 - In the MFI interface, set the breakpoint with the command: AT ENTRY SAM2.
 - Or in WDDz, to set the entry breakpoint:

Debug Tool Mentor Workshop

Lab Exercises



- Right-click in the breakpoints window
 - Select “Entry” breakpoint.
 - Set a breakpoint in load module CDAT2, csect CDAT2, entry point CDAT2.
- f. Change the CADP profile to trap program CDAT2 instead of CDAT1. As you run the transaction, notice that Debug Tool does not start for the main menu in CDAT1, but only when program CDAT2 is executed.

Debug Tool Mentor Workshop

Lab Exercises

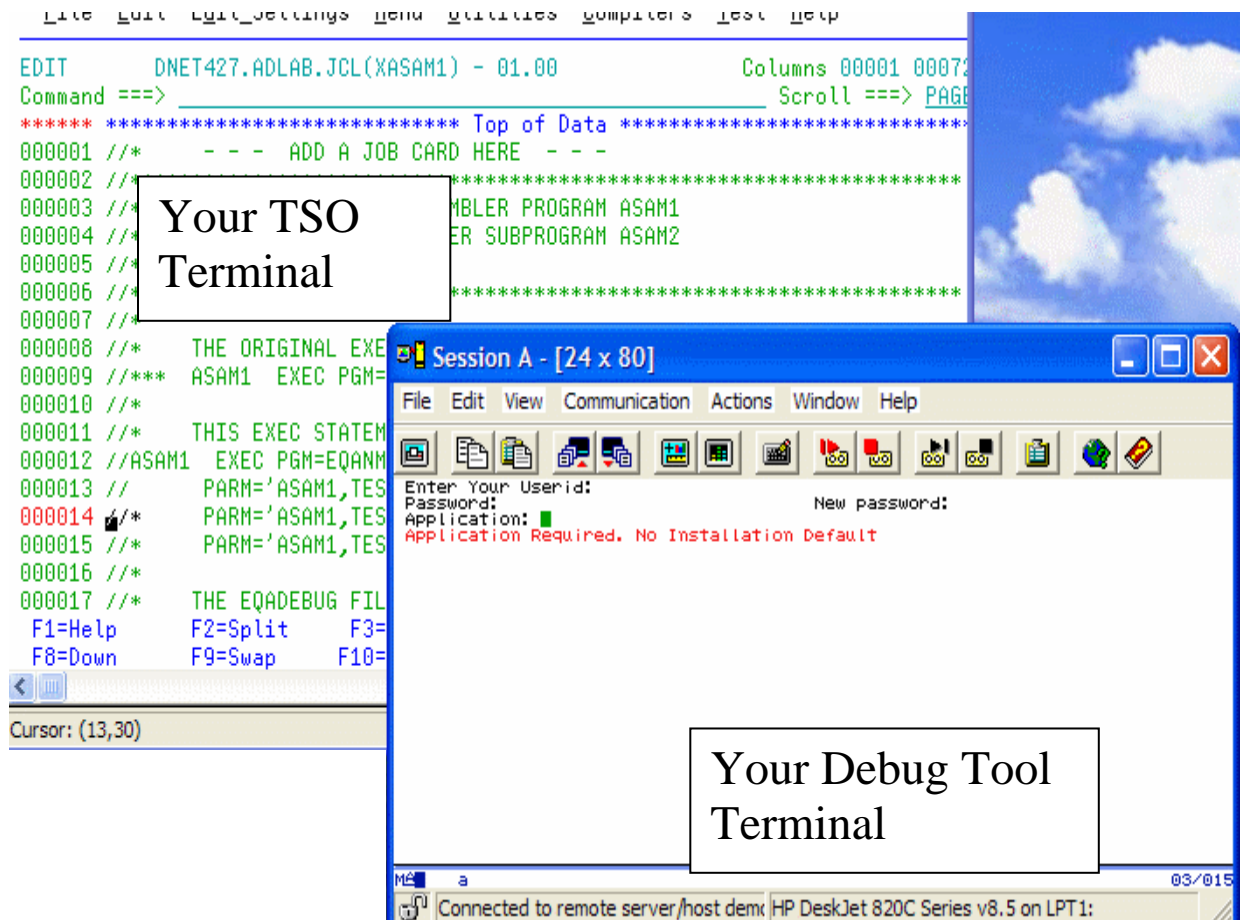


16. Start a second terminal emulator window. We will call this your “Debug Tool terminal”. Your instructor will have shown you how do this in class. DO NOT LOG ON to this second emulator window. To make it work, your Debug Tool terminal must NOT be connected to ANY application. Not even a session manager.

- Find the Terminal ID of your Debug Tool terminal. FYI... Technically, the “Terminal ID” is the “VTAM LU” (Logical Unit) ID.
- What is the Terminal ID? _____

17. You now have 2 terminal emulator windows active:

1. One where you are logged on to TSO, with the JCL open in an Edit session.
2. Your Debug Tool terminal, where you are not logged on.



Debug Tool Mentor Workshop

Lab Exercises



18. Click on your TSO terminal. Notice the parameter string on the line immediately after the EXEC statement.

- The first parameter is the name of the assembler program, which is ASAM1 in this example. To start Debug Tool for an assembler main program, you execute Debug Tool (program EQANMDBG) on the EXEC statement, and pass the name of the assembler program as the first parm.
- Change the terminal id in the TEST parm to the Terminal ID of your Debug Tool terminal. Be careful not to change the format of the TEST string. The line with the TEST parm should be similar to:

```
//      PARM='ASAM1,TEST(,,,MFI%TRMLU006:)',
```

(Instead of TRMLU006, you should use the name of your Debug Tool terminal specified.

Important Note: The special characters % and : are required!.

19. Your JCL should look similar to this example, although it will have your unique job card, your dataset names, and (most importantly) your Debug Tool Terminal ID:

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT      DNET427.ADLAB.JCL(XASAM1) - 01.05          Columns 00001 00072
Command ==>                                     Scroll ==> PAGE
***** ***** Top of Data *****
==MSG> -Warning- The UNDO command is not available until you change
==MSG>          your edit profile using the command RECOVERY ON.
000001 //DNET427I JOB (ACCTG),'IBM PD Tools Group',REGION=4M,CLASS=A,
000002 //          MSGCLASS=H,NOTIFY=DNET427,MSGLEVEL=(1,1)
000003 //*          - - - ADD A JOB CARD HERE - - -
000004 //*****
000005 //*          RUN SAMPLE NON-LE ASSEMBLER PROGRAM ASAM1
000006 //*          CALLS ASSEMBLER SUBPROGRAM ASAM2
000007 //*
000008 //*****
000009 //*
000010 //*          THE ORIGINAL EXEC STATEMENT IS COMMENTED:
000011 //***          ASAM1 EXEC PGM=ASAM1
000012 //*
000013 //*          THIS EXEC STATEMENT WILL START DEBUG TOOL
000014 //ASAM1 EXEC PGM=EQANMDBG,
000015 //          PARM='ASAM1,TEST(,,,MFI%TRMLU006:)'
F1=Help    F2=Split    F3=Exit    F5=Rfind    F6=Rchange  F7=Up
F8=Down    F9=Swap     F10=Left  F11=Right  F12=Cancel
```

20. Submit the job (use the **SUBMIT** command). Then click on your Debug Tool terminal.

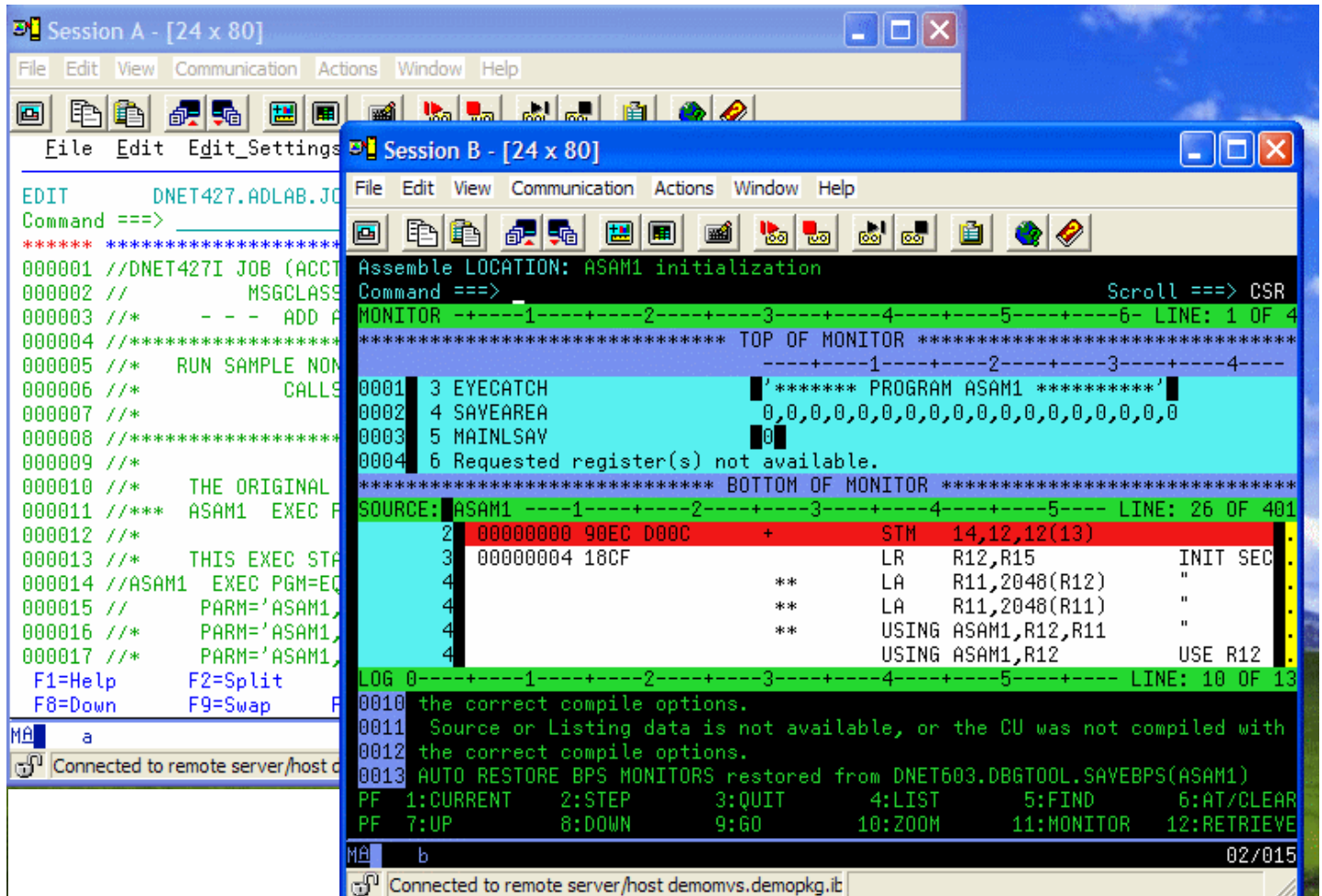
21. When the job runs, Debug Tool will automatically appear on your Debug Tool terminal.

Debug Tool Mentor Workshop

Lab Exercises



SUCCESS! Debug Tool appears.



- If Debug Tool does not appear, check your JCL for errors and try again. Use a JES viewing facility (such as SDSF) to check on the status of your job. Here are some common problems that you should check on:
 - The job card is coded incorrectly.
 - The job has a JCL error, due to an invalid dataset name, or some other error.
 - The Terminal ID was coded incorrectly.
 - The TEST parm was coded incorrectly.

22. Click on your Debug Tool session. Use STEP commands to verify that you can step into the program.

23. Use the QUIT command to end your Debug Tool session.

Debug Tool Mentor Workshop

Lab Exercises



24. Click on your TSO window. Save your JCL changes. You will use this JCL again in later exercises.

Lab Exercise 15

In this exercise you will:

- Learn the basic Debug Tool commands used in debugging Assembler.
33. Before you do this exercise, you must first do exercise 14 to start a Debug Tool session. Before you continue, you must have a Debug Tool session started in program ASAM1.
34. Get help with commands using the “?” command.
- Type **ZOOM LOG** on the command line, then **ENTER**.

```
Assemble LOCATION: ASAM1 initialization
Command ==> zoom log
MONITOR -+---1---+---2---+---3---+---4---+---5---+---6- LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****
SOURCE: ASAM1 ---1---+---2---+---3---+---4---+---5---+--- LINE: 0 OF 0
```

- The log window will be displayed in full screen mode.
 - **Type ?** on the command line and **ENTER**. (The “?” command by itself displays a list of all commands.)
- You can type “?” into any command to get a list of valid options for the command. This next command will display the different options of the “AT” command (used to set breakpoints):
 - **Type AT ?** on the command line, then **ENTER**. A list of the things that you can type after the word “AT” is displayed.

Debug Tool Mentor Workshop

Lab Exercises



```
Assemble LOCATION: ASAM1 initialization
Command ==>
LOG 0-----1-----2-----3-----4-----5----- LINE: 18 OF 36
0018 Possible RACF error, invalid member name, etc.
0019 The partially parsed command is:
0020 AT
0021 The next word may be one of:
0022 ( EXIT
0023 * FREE
0024 %BLOCK FROM
0025 %LINE GLOBAL
0026 %STATEMENT LABEL
0027 block name LINE
0028 compile unit specification LOAD
0029 statement number OCCURRENCE
0030 APPEARANCE OFFSET
0031 CALL PATH
0032 CHANGE STATEMENT
0033 CURSOR TERMINATION
0034 DELETE TO
0035 ENTRY TOGGLE
0036 EVERY
```

35. PF10 is set to the ZOOM command. It will zoom in, and will also zoom out.

- **Press PF10** to zoom out of the log window.
- All three windows will be displayed again.

36. The SET ASSEMBLER ON and SET DISASSEMBLY ON commands can be used to set up the Assembler Debug Tool environment.

Note: Consider the types of programs you will be debugging when deciding which setting to use. The following guidelines can help you decide which command to use:

- You must turn on EITHER the ASSEMBLY or DISASSEMBLY setting to work with assembler programs.
- The ASSEMBLER or DISASSEMBLY settings are mutually exclusive. You can only be in Assembler or Disassembly mode, not both.
- An advantage of DISASSEMBLY is that if you do not have source information saved for Debug Tool to use (a LANGX file), then the DISASSEMBLY setting allows you to see the machine instructions in the program. However, even in DISASSEMBLY mode, source statements will be displayed if you have a LANGX file.
- The SET ASSEMBLER ON setting enables these Debug Tool commands with assembler programs:
 - AT APPEARANCE *
 - AT APPEARANCE *program-name*

Debug Tool Mentor Workshop

Lab Exercises



- AT ENTRY name
 - DESCRIBE PROGRAM
 - LIST *assembler-variable-name* and MONITOR LIST *assembler-variable-name*
 - LIST NAMES PROGRAMS
 - QUERY SOURCE
- The SET DISASSEMBLY ON setting gives you ALL OF THE CAPABILITIES of the SET ASSEMBLER ON setting, PLUS enables the commands:
 - STEP INTO (to step into an assembler subroutine)
 - AT ENTRY *
 - Many people prefer the DISASSEMBLY setting, since it is more powerful. You may want to choose the ASSEMBLER setting if you want to avoid accidentally stepping into, or having breakpoints in, system routines and other assembler routines.
 - If you are debugging in ASSEMBLER mode and later decide you want to debug in DISASSEMBLY mode, you can enter the SET DISASSEMBLY ON command after you enter the SET ASSEMBLER ON command.

Type **SET DISASSEMBLY ON** on the command line and hit **Enter**.

37. Note: In the next step, you will “load” source information for the program into debug tool. Before you do that, you must tell Debug Tool the name of the PDS where your EQALANGX files have been stored. (EQALANGX files contain source information in a format that Debug Tool can use. They must be created when you assemble the program). The best ways to give Debug Tool the name of your EQALANGX library (or libraries) are:

- Type a “SET DEFAULT LISTINGS *library-name*” command, or
- Code an EQADEBUG DD statement in the execution JCL. IMPORTANT: The JCL that you submitted to start Debug Tool contained this DD statement.

38. THE LDD program name loads the source information for the program into Debug Tool environment. Specifically, it is loading an EQALANGX file that was created when the program was assembled.

Enter LDD ASAM1 on the command line and hit **Enter**. This loads the source information that’s in member ASAM1 from the EQALANGX library (PDS). You’ll see the program source information as shown here:

Debug Tool Mentor Workshop

Lab Exercises



```
Assemble LOCATION: ASAM1 :> 2
Command ==> Scroll ==> CSR
MONITOR -+----1----+----2----+----3----+----4----+----5----+----6- LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE: ASAM1 ---1---+----2---+----3---+----4---+----5---+ LINE: 1 OF 401
***** TOP OF SOURCE *****
2 00000000 ASAM1 AMODE 31
2 ASAM1 CSECT
2 * *****
2 * SAMPLE PROGRAM SAMASM1
2 * AUTHOR: DOUG STOUT
LOG 0---+----1----+----2----+----3----+----4----+----5----+ LINE: 13 OF 33
0013 AUTO RESTORE BPS MONITORS restored from DNET603.DBGTOOL.SAVEBPS(ASAM1)
0014 LDD ASAM1 ;
0015 The CU specified for the LOADDEBUGDATA command is already an assembler or
0016 OS/VS COBOL CU.
PF 1:CURRENT 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CLEAR
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:MONITOR 12:RETRIEVE
```

5. Use the STEP command to enter into the program ASAM1:
 - Type **STEP** on the command line, then **Enter**.
 - Enter **PF2**.
39. Set breakpoints. Try these different methods:
 - Place the cursor in the Source Window and then **Enter PF10**. This will open the Source Window.
 - Place the cursor on statement 3 in the Source window and then **Press PF6**.
 - Type **AT 5** on the command line (where 5 is a valid statement number) then **ENTER**.
 - Type **A** in the prefix area (the numbered area to the left of the program) of statement 12 in the Source window, then **ENTER**.
40. You should have some breakpoints set now. Notice that the colors are reversed in the prefix area (on the left side of the screen) next to the statements with breakpoints.

Debug Tool Mentor Workshop

Lab Exercises



```

Assemble LOCATION: ASAM1 :> 2
Command ==>
SOURCE: ASAM1 -----1-----2-----3-----4-----5----- LINE: 26 OF 401
2 00000000 90EC D00C + STM 14,12,12(13)
3 00000004 18CF LR R12,R15 INIT SEC
4 ** LA R11,2048(R12) "
4 ** LA R11,2048(R11) "
4 ** USING ASAM1,R12,R11 "
4 USING ASAM1,R12 USE R12
4 00000006 50D0 C03C ST R13,SAVEAREA+4
5 0000000A 41D0 C038 LA R13,SAVEAREA ADDRESS
6 0000000E 47F0 C080 B MAINLINE GO AROUND
7 00000018 DS 0D
7 00000018 EYECATCH DC CL32'***** PROGRAM ASAM1
8 00000038 SAVEAREA DC 18F'0'
9 *
9 00000080 MAINLINE DS 0F
9 00000080 45E0 C178 BAL R14,OPENFILS
10 00000084 45E0 C0A0 BAL R14,MAINLOOP
11 00000088 45E0 C218 BAL R14,CLOSFILS
12 *
12 0000008C D21D C2A2 C618 MVC STATUS,=C'RETURNING TO CALLI
PF 1:CURRENT 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CLEAR
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:MONITOR 12:RETRIEVE
  
```

41. You can display (List) your breakpoints:

- Press **PF10** (Zoom) to un-zoom and display all 3 windows again.
- Type **LIST AT** on the command line, then **ENTER**.
- Notice that your breakpoints are displayed in the log window.
- Press **PF10** (Zoom) again to zoom the source window.

42. Clear breakpoints. Try these different methods:

- Place the cursor on statement 12, and **Press PF6**. Notice that PF6 will toggle breakpoints on and off.
- **Type CLEAR AT 5** on the command line (where 5 is a valid statement number) then **ENTER**.
- **Type C** (for Clear) in the prefix area for source line 3 (the numbered area to the left of the program), then **ENTER**.
- To clear all breakpoints with a single command, you can type **CLEAR AT** on the command line, then **ENTER**.

Debug Tool Mentor Workshop

Lab Exercises



43. Set breakpoints at statements 22 and 25. It's OK if you have other breakpoints set. With the Source Window zoomed open (place the cursor in the Source Window and use the PF10 key), you can scroll forward (**PF8**) to see these statements.

```
Assemble LOCATION: ASAM1 :> 2
Command ==>
SOURCE: ASAM1 -----1-----2-----3-----4-----5----- LINE: 65 OF 401
22 000000B8 41D0 C038          LA   R13,SAVEAREA
23          ***** LINK EP=ASAM2,(DATALEN,INREC,HEX
23          * FOLLOWING LINES COMMENTED OUT WHEN LOAD
23          ***** LOAD EP=ASAM2
23          ***** LTR R15,R15
23          ***** BNZ LOADERR
23          ***** LA R13,SAVEAREA
23          ***** LA R1,PARMLIST R1 = PARM L
23          ***** LR R15,R0 R0 = ADD OF
23          ***** BALR R14,R15 BRANCH TO S
23          CALL ASAM2,(DATALEN,INREC,HEXTOP
23          + SYSSTATE TEST
23          + CNOP 0,4
23 000000BC 47F0 C0C4          B   **8
24 000000C0          +IHB0005B DC V(ASAM2)
25          + IHB0PLTX ASAM2,(DATALEN,INREC,HEX
25          + SYSSTATE TEST TE
25 000000C4 4110 C0CC          LA  1,IHB0007
26 000000C8 47F0 C0DC          B   IHB0007A
PF 1:CURRENT 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CLEAR
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:MONITOR 12:RETRIEVE
```

44. Run the program until it reaches a breakpoint.

- **Type GO** on the command line, then **ENTER**.
 - The program should be stopped at a breakpoint.
 - If the program ended, then you probably did not have a breakpoint set where it would be reached. If that happened, re-run the program to start Debug Tool again. Re-run the program as many times as needed throughout this exercise.
- PF9 is set to the GO command. Make sure you have a breakpoint that will be reached. **Press PF9** to GO.

45. Practice setting, clearing, and running to breakpoints at statements until you are familiar with:

- the “AT #” and “CLEAR AT #” commands
- PF6 as a toggle to turn breakpoints on or off
- the “A” (At) and “C” (Clear) line commands
- PF9 or the “GO” command to run to a breakpoint

Debug Tool Mentor Workshop

Lab Exercises



46. You can use the JUMPTO command to “jump” to a statement and continue execution at this statement. This is sometimes helpful to re-execute a few statements in the program. Try the following:

- Make sure you have breakpoints at statements 22 and 25.
- Run the program (GO or PF9) until it is stopped at statement 25 or later.
- **Type JUMPTO 21** on the command line, then **ENTER**.
- The program should be stopped at statement 21.

```
Assemble LOCATION: ASAM1 :> 21
Command ==>
Scroll ==> CSR
MONITOR +-----1-----2-----3-----4-----5-----6- LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE: ASAM1 ---1---2---3---4---5--- LINE: 62 OF 401
20 000000AE 4780 C16C BE MAINLEX IF YE .
21 * * CALL SUBPROGRAM TO GET HEX CHARACT .
22 000000B2 D21D C2A2 C636 MVC STATUS,=C'CALLING SUBPROGRA .
23 000000B8 41D0 C038 LA R13,SAVEAREA .
23 ***** LINK EP=ASAM2,(DATALEN,INREC,HEX .
* FOLLOWING LINES COMMENTED OUT WHEN LOAD .

LOG 0---1---2---3---4---5--- LINE: 40 OF 43
0040 GO ;
0041 GO ;
0042 GO ;
0043 JUMPTO 21 ;

PF 1:CURRENT 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CLEAR
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:MONITOR 12:RETRIEVE
```

47. You can use the GOTO command instead of JUMPTO. It is similar to JUMPTO, but it does not stop at the statement. It jumps to the statement and immediately continues to run. The program will automatically run to the next breakpoint (or the end of the program).

- Make sure you have breakpoints at statements 22 and 25.
- Run the program (GO or PF9) until it is stopped at statement 25.
- **Type GOTO 21** on the command line, then **ENTER**.
- Note that statement 21 executed, but Debug Tool did NOT stop at 21. A GOTO command will not stop at the target statement unless you have a breakpoint there. The program is stopped at statement 22, which was the next breakpoint that it reached.

Debug Tool Mentor Workshop

Lab Exercises



48. Display the address and length of a program in the Log.

Type **DESCRIBE ATTRIBUTES ASAM1** on the command line and **ENTER**.

In the log window you will see the address and length information of ASAM1.

```
Assemble LOCATION: ASAM1 :> 21
Command ==>                               Scroll ==> CSR
MONITOR --+---1---+---2---+---3---+---4---+---5---+---6- LINE: 0 OF 0
***** TOP OF MONITOR *****
***** BOTTOM OF MONITOR *****

SOURCE: ASAM1 ---1---+---2---+---3---+---4---+---5--- LINE: 62 OF 401
20 000000AE 4780 C16C          BE    MAINLEX          IF YE .
21                                *    * CALL SUBPROGRAM TO GET HEX CHARACT .
21 000000B2 D21D C2A2 C636    MVC    STATUS,=C'CALLING SUBPROGRA .
22 000000B8 41D0 C038          LA    R13,SAVEAREA .
23                                **** LINK EP=ASAM2,(DATALEN,INREC,HEX .
23                                * FOLLOWING LINES COMMENTED OUT WHEN LOAD .

LOG 0---+---1---+---2---+---3---+---4---+---5---+--- LINE: 56 OF 59
0056 DESCRIBE ATTRIBUTES ASAM1 ;
0057     ATTRIBUTES for ASAM1
0058     Its address is 00007280 and its length is 1742
0059     CSECT
PF 1:CURRENT  2:STEP  3:QUIT  4:LIST  5:FIND  6:AT/CLEAR
PF 7:UP       8:DOWN  9:GO   10:ZOOM 11:MONITOR 12:RETRIEVE
```

16. Display the address and length of a variable in the Log Window.

Enter **DESCRIBE ATTRIBUTES SAVEAREA** on the command line and then **Enter**.

(Note: Try DE AT SAVEAREA which is an abbreviation of the same command).

17. Display a variable in the monitor window. Try different methods:

- Type **MONITOR LIST SAVEAREA** then **Enter**. Try abbreviating this command... For example: **MON LI SAVEAREA**
- Type **MON LIS** (an abbreviation for MONITOR LIST) on the command line, then place the cursor on any variable that appears in the Source window (for example EYECATCH), then **Enter**.

Debug Tool Mentor Workshop

Lab Exercises



```

Assemble LOCATION: ASAM1 :> 21
Command ==>
MONITOR +-----1-----2-----3-----4-----5-----6- LINE: 1 OF 4
***** TOP OF MONITOR *****
0001 3 SAVEAREA          0,26216,0,-2147453730,12858868,30024,30024,27
0002                    95,26152,533725264,533732064,7,533727512,5337
0003                    1096,26216,48763,44668,-2147454336
0004 4 EYECATCH          '***** PROGRAM ASAM1 *****'
***** BOTTOM OF MONITOR *****
SOURCE: ASAM1 ---1---2---3---4---5--- LINE: 62 OF 401
20 000000AE 4780 C16C          BE MAINLEX          IF YE .
21 * * CALL SUBPROGRAM TO GET HEX CHARACT .
21 000000B2 D21D C2A2 C636     MVC STATUS,=C'CALLING SUBPROGRA .
22 000000B8 41D0 C038          LA R13,SAVEAREA
23 * * LINK EP=ASAM2,(DATALEN,INREC,HEX .
23 * FOLLOWING LINES COMMENTED OUT WHEN LOAD .
LOG 0-----1-----2-----3-----4-----5----- LINE: 68 OF 71
0068 LIST
0069 The cursor is not positioned at a variable name.
0070 MONITOR
0071 LIST EYECATCH ;
PF 1:CURRENT 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CLEAR
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:MONITOR 12:RETRIEVE
  
```

18. Note: Most items that you can display in the log with a LIST command, you can also display in the monitor window with MONITOR LIST command.

19. Remove items from the Monitor Window:

- **Type C** (for Clear) next to a variable in the prefix area (the numbers on the left) of the Monitor window, then **ENTER**.

```

MONITOR +-----1-----2-----3-----4-----5-----6- LINE: 1 OF 4
***** TOP OF MONITOR *****
0c01 3 SAVEAREA          0,26216,0,-2147453730,12858868,30024,30024,27
0002                    95,26152,533725264,533732064,7,533727512,5337
0003                    1096,26216,48763,44668,-2147454336
0004 4 EYECATCH          '***** PROGRAM ASAM1 *****'
  
```

- To remove all items from the Monitor Window: **Type CLEAR MONITOR** on the command line, then ENTER. (Or abbreviate this command, such as : **CLE MON**)

20. Using MONITOR LIST command, as described in the last few steps, put the following variables in the Monitor Window:

- MAINLSAV
- SAVEAREA

21. Modify a variable value.

- Tab to the data area of the MAINLSAV variable in the Monitor window.

Debug Tool Mentor Workshop

Lab Exercises



- **Change** the data to 1, clear out the rest of the field, then **ENTER**.
- The value in MAINSLAV is now 1.

```

MONITOR +-----1-----2-----3-----4-----5-----6- LINE: 1 OF 5
***** TOP OF MONITOR *****
-----1-----2-----3-----4-----
0001 3 SAVEAREA          0,26216,0,-2147453730,12858868,30024,30024,27
0002                    05,26152,533725264,533732064,7,533727512,5337
0003                    1096,26216,48763,44668,-2147454336
0004 4 EYECATCH          /***** PROGRAM ASAM1 *****/
0005 5 MAINSLAV          1
  
```

22. Along with variables, Debug Tool will allow you to monitor and list the General Purpose Registers and special variables associated with General Purpose registers.

Type **LIST REGISTERS** on the command line and hit enter. In the log window you will see:

```

LOG 0-----1-----2-----3-----4-----5-----6- LINE: 76 OF 79
0076 GPR 0 - 3 = 00007548 00007548 00006BCB 00006628
0077 GPR 4 - 7 = 1FD00050 1FD01AE0 00000007 1FD00918
0078 GPR 8 - 11 = 1FD01718 00006668 0000BE7B 0000AE7C
0079 GPR 12 - 15 = 80007280 00007288 80007328 00C435F4
  
```

Type **MO LIST REGISTERS** on the command line and hit **ENTER**. In the log window you will see the General Purpose Registers in the Monitor Window and the values in the General Purpose Registers which can be modified like a variable. Try both LIST and MONITOR LIST commands on Storage locations as part of this exercise.

```

MONITOR +-----1-----2-----3-----4-----5-----6- LINE: 3 OF 4
-----1-----2-----3-----4-----
0003                    1096,26216,48763,44668,-2147454336
0004 4 EYECATCH          /***** PROGRAM ASAM1 *****/
0005 5 MAINSLAV          1
0006 6 GPR 0 - 3 = 00007548 00007548 00006BCB 00006628
0007   GPR 4 - 7 = 1FD00050 1FD01AE0 00000007 1FD00918
0008   GPR 8 - 11 = 1FD01718 00006668 0000BE7B 0000AE7C
  
```

23. Add a single variable in the Monitor window.

Type MON LIST %GPR1 and Enter.

Debug Tool Mentor Workshop

Lab Exercises



Notice that register 1 was added to the Monitor Window. You may have to scroll forward in the monitor window to see it (put your cursor in the monitor window, then press PF8). You can modify register values in the Monitor Window when they have been added this way.

```
MONITOR  +-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6 LINE: 6 OF 10
          +-----1-----+-----2-----+-----3-----+-----4-----
0006  6 GPR 0 - 3 = 00007548 00007548 00006BCB 00006628
0007  GPR 4 - 7 = 1FD00050 1FD01AE0 00000007 1FD00918
0008  GPR 8 - 11 = 1FD01718 00006668 0000BE7B 0000AE7C
0009  GPR 12 - 15 = 80007280 000072B8 80007328 00C435F4
0010  7 %GPR1                X'00007548'
```

24. You can display storage in the Log Window with a LIST Storage command. Try these commands.

- **LIST STOR(INREC)** to display 16 bytes of storage (the default) starting with the address of the INREC variable.
- **LIST STOR(X'7848')** to display 16 bytes of storage (the default) starting with an address.
- **LIST STOR(X'7848',64)** to display 64 bytes of storage starting with an address.

25. You can display storage in the monitor window with a MON LIST STORAGE command. Try these commands:

- **MON LIST STOR(INREC)** to display 16 bytes of storage (the default) starting with the address of the INREC variable.
- Note: You can change any of the above LIST STOR(...) commands to a MON LIST STOR(...) command to view the same result in the monitor window, instead of the log window.

26. Automatically display active variables in the Monitor Window.

- Turn on the Automonitor: **Type SET AUTO ON** on the command line, then **ENTER**.
- The “Automonitor” appears at the bottom of the Monitor window. You may have to scroll forward in the Monitor to see it. This will automatically display variables referenced by the current statement in the program.

Debug Tool Mentor Workshop

Lab Exercises



- Use **PF2** (or the STEP command) a few times. Notice that as you step, the Automonitor changes to show variables referenced by the current statement.

27. End the Debug Tool session:

- **Type QUIT** on the command line, then **ENTER**.
- When you receive the “Do you really want to ...” prompt, **Type Y** on the command line, then **ENTER**.

Note: If you were to just continue to run the program with the GO command (or PF9), the program would eventually terminate on its own, and the Debug Tool session would end when the program terminates.