IBM

Debug Tool Terminal interface

IBM's Interactive Debugger for applications running in z/OS

Basic Commands

For detailed descriptions of commands, command syntax, and command options, refer to the Debug Tool for z/OS Reference and Messages manual. A complete set of Debug Tool manuals can be obtained from the IBM Debug Tool website.
www.ibm.com/software/awdtools/debugtool/
select the "Library" link

Manuals:
Summary of Commands, User's Guide, Reference and Messages, and Customization Guide

For Use with Debug Tool Version 12

Work with windows and source

ZOOM or **Z**	Expand the source window to the full screen, or reduce an expanded window
ZOOM (with a cursor selected border location)	Expand the cursor selected window
POS 509	Position the source window to statement number 509
QUALIFY RESET or **QUA RES**	Position the source window to the current statement
FIND 'text' or **F** 'text'	Find the next occurrence of *text* in the source window
F5	Function key: repeat the last find command

Run the program

STEP or **STEP INTO**	Run one statement
F2	Function key: same as STEP
STEP 25	Run 25 statements, starting with the current statement, one at a time with step animation
GO	Run the program, starting with the current statement, until the next breakpoint or the end of the application
F2	Function key: same as GO command
RUNTO 630	Run the program starting with the current statement, until the next time it reaches statement 630 (or reaches a breakpoint or the end of the application)
R	Line command: same as a RUNTO command for the selected statement
JUMPTO 952	Jump to statement 952. Do not execute the current statement or any other statements. The program will be paused at statement 952.
GOTO 952	Same as a combination of JUMPTO 952 followed by a GO command

Commands for working with breakpoints

LIST AT	Display a list of all breakpoints in the log
FINDBP	Find the next statement breakpoint and position the source window to it
CLEAR AT	Clear all breakpoints in the current enclave

Set and clear statement breakpoints

A or **AT**	Line command: set a breakpoint at the selected statement
C	Line command: clear the breakpoint at the selected statement
F6 (with a cursor selected statement)	Function key: set a breakpoint at the cursor selected statement. If a breakpoint already exists at the statement, clear it.
AT 452	Set a breakpoint at statement 452
AT FROM 99 452	Set a breakpoint that will trigger starting with the 99th time that statement 452 is reached
CLEAR AT 452 or **CL AT** 452	Clear the statement breakpoint at statement 452
D	Line command: disable the breakpoint at the selected statement (but do not clear it)
E	Line command: enable the disabled breakpoint at the selected statement
AT *	Set a special breakpoint that will stop at all statements
CLEAR AT * or **CL AT ***	Clear the special AT * breakpoint

Set and clear change (watch) breakpoints

AT CHANGE var-name or **AT CHA** var-name	Set a breakpoint that will trigger when variable var-name changes
CLEAR AT CHANGE variable-name or **CLE AT CHA** variable-name	Clear the change breakpoint for variable-name

Set and clear program entry and exit breakpoints

AT ENTRY program-name	Set a breakpoint that will trigger when program (compile unit) program-name is entered
CLEAR AT ENTRY program-name	Clear the entry breakpoint for program-name
AT ENTRY *	Set a special breakpoint that will trigger at the entry of all programs
CLEAR AT ENTRY *	Clear the special AT ENTRY * breakpoint
AT EXIT program-name	Set a breakpoint that will trigger when program (compile unit) program-name is exited
CLEAR AT EXIT program-name	Clear the exit breakpoint for program-name

Make breakpoints conditional

To make a breakpoint conditional, code WHEN and a condition. Examples:

AT 502 WHEN CUSTID = ‘77409’	Set a breakpoint that will trigger at statement 502 if the condition is true when the statement is reached
AT CHANGE CUSTID WHEN CUSTID = ‘77409’	Set a breakpoint that will trigger when variable CUST-ID changes if the condition is true when it changes
AT CHANGE CUSTID WHEN BAL > 999	Set a breakpoint that will trigger when variable CUST-ID changes if the condition is true when it changes

Monitor variables

SET AUTO ON	Turn on the automonitor. Variables referenced by the current statement display in the monitor window automatically.
SET AUTO ON BOTH	Turn on the automonitor. Variables referenced by both the current statement and previously displayed statements display in the monitor window automatically. This shows results automatically while stepping.
SET AUTO ON LOG or **SET AUTO ON BOTH LOG**	Turn on the automonitor. In addition to displaying variables in the monitor window, they are also displayed in the log. This automatically traces variable values referenced by every statement.
M	Line command in the source window: add all variables referenced by the selected line to the monitor
M*n* (such as M1, M2, …)	Line command in the source window: add the nth variable referenced by the selected line to the monitor
MONITOR LIST var-name or **MON LIST** var-name	Add variable-name to the monitor
MON LIST TITLED WSS	Add all variables to the monitor from COBOL working-storage
C	Line command in the monitor window: clear (remove) the selected item
H	Line command in the monitor window: display the value of the selected item in hexadecimal format
D	Line command in the monitor window: display the selected value in default format
CLEAR MONITOR	Clear all items from the monitor window

List variables in the log

L	Line command in the source window: display all variables referenced by the selected line in the log
L*n* (such as L1, L2, …)	Line command in the source window: display the nth variable referenced by the selected line in the log
F4 (with a cursor-selected variable in the source window)	Function key: display the cursor-selected variable in the log
LIST variable-name	Display variable-name in the log
LIST TITLED WSS or **LIST TITLED FS** or **LIST TITLED LS** or **LIST TITLED ***	Display all variables in the log from COBOL working-storage, file, or linkage section, or all variables

Change values of variables

Overtype the value of a variable in the monitor window to change the value	
MOVE 987 TO varx **MOVE ‘ZYX’ TO** var	Change the value of variables in COBOL programs
varx **=** 987 var **=** ‘ZYX’	Change the value of variables in PLI, C/C++, and assembler programs

Work with subprograms

STEP or **STEP INTO**	When the current statement calls or runs a sub-program, procedure, or function, step into it. (The sub must be compiled for debugging)
STEP OVER	When the current statement calls or runs a sub-program, procedure, or function, run it but do not show it in the debugger.
STEP RETURN	Run to the next program return. This is a quick way to run to the end of a sub-program.
LOAD progname	Load program progname. Display it in the source window if it is available.
QUALIFY PROGRAM progname	Display the source of progname in the source window
QUALIFY RESET or **QU RES**	Position the source window to the current program and current statement

End program testing

QUIT	Ends debugging. Prompts with a “Are you sure…?” message. If accepted, terminates the program.
QQ	Same as QUIT but without a prompt
QUIT DEBUG	Ends debugging but the program continues to run from the current statement
QUIT DEBUG TASK	Used in CICS only. Ends debugging but the program continues to run. The DTCN or CADP profile remains active.
QUIT ABEND	Ends debugging and terminates the program with a forced abend.

Playback (step backward in a program)

PLAYBACK ENABLE	Turn on the playback recorder. Consider the PLAYBACK ENABLE near the beginning of a program.
PLAYBACK START	Enter playback mode. The PLAYBACK ENABLE command must have been entered previously. STEP commands will step backward.
PLAYBACK FORWARD	Set the direction of STEP and RUNTO commands to forward
PLAYBACK BACKWARD	Set the direction of STEP and RUNTO commands to backward
PLAYBACK STOP	Exit playback mode, and return to normal debugging mode. The playback recorder remains on.
PLAYBACK DISABLE	Turn the playback recorder off

Work with program source

SET DEFAULT LISTINGS lib-name	Search library lib-name for a debug source file, if it has not already been found for the current program. Automatically search this library when new programs are encountered.
SET DEF LIST (lib1**,** lib2**,** … **,** libn)	Search library lib-name for a debug source file, if it has not already been found for the current program. Automatically search these libraries when new programs are encountered.
Code an **EQADEBUG** DD in JCL	An EQADEBUG DD can be used to specify a library concatenation for debug source files. It is an alternative to the “SET DEF LIST …” setting.
LDD csect-name **LDD** program-name	Load the LANGX file for the specified csect or program. Libraries specified by “SET DEF LIST …” or an EQADEBUG DD are searched.

IBM

Debug Tool
Terminal interface

IBM's Interactive Debugger for applications running in z/OS

"How To" quick reference and Notes

For detailed descriptions of commands, command syntax, and command options, refer to the Debug Tool for z/OS Reference and Messages manual. A complete set of Debug Tool manuals can be obtained from the IBM Debug Tool website.
www.ibm.com/software/awdtools/debugtool/
select the "Library" link

Manuals:
Summary of Commands, User's Guide, Reference and Messages, and Customization Guide

For Use with Debug Tool Version 12

How to bypass an Abend condition

If an abend occurs, you are notified with a message in the log. If the program is stopped at an abend, and you STEP or GO, the application will abend. To continue without abending:

GO BYPASS	This command bypasses the statement where the abend occurred, passes control to the next logical statement, and stops there.

How to call Fault Analyzer to capture a fault entry

CALL %FA	Invoke IBM Fault Analyzer for z/OS to capture a fault entry based on the current state of the application. Control is returned to the debugger after the fault entry has been captured, and debugging can continue.

Files that can be used by the debugger, and commands to use them

Preferences File (DD name INSPPREF)	A file that contains a series of Debug Tool commands (a script) that runs automatically when the debugger starts. It is typically used to customize debugging settings and the environment for the developer.
Command File (DD name INSPCMDS)	A file that contains a series of Debug Tool commands (a script) that runs automatically when the debugger starts. It runs after the preferences file completes, if there is one. It is typically used to run a series of commands to control execution of the test session and programs.
Log File (DD name INSPLOG)	A file where Debug Tool writes messages that are written to the log window., such as results of various Debug Tool Commands.
SET LOG ON FILE file-name **OLD**	Command that opens file-name of the log file. All log messages occurring after this command is issued are written to the file.
USE file-name	Command to run Debug Tool commands (a script) contained in the specified file.
LANGX file	Debugging information for OS/VS COBOL, VS COBOL II, or Assembler
Save settings file	Allows saving/restoring of SETTINGS between debugging executions. The default naming convention is user-id.DBGTOOL.SAVESETS, but may be customized on each system. File attributes: sequential, RECFM=VB, LRECL=3204 or more, BLKSIZE=any
Save breakpoints and monitors file	Allows saving/restoring of breakpoints and MONITOR values between debugging sessions. The default naming convention is user-id.DBGTOOL.SAVEBPS, but may be customized on each system. File attributes: PDS or PDSE (Library), RECFM=VB, LRECL=3204 or more, BLKSIZE=any
SET SAVE SETTINGS AUTO	Automatically save current settings to the save settings data set when the debugger ends
SET SAVE BPS AUTO	Automatically save current breakpoints to the save breakpoints and monitors data set when the debugger ends
SET SAVE MONITORS AUTO	Automatically save current monitors to the save breakpoints and monitors data set when the debugger ends
SET RESTORE SETTINGS AUTO	Automatically restore settings from the save settings data set when the debugger starts
SET RESTORE BPS AUTO	Automatically restore breakpoints from the save breakpoints and monitors data set when the debugger starts
SET RESTORE MONITORS AUTO	Automatically restore monitors from the save breakpoints and monitors data set when the debugger starts

How to invoke the debugger: Batch LE program, connecting to GUI debugger

Insert a CEEOPTS DD statement with TEST run-time option in the JCL in the step or steps to be debugged. Syntax:

```
  //CEEOPTS DD *
  TEST(,,,TCPIP&address%port:)
    address = the IP address of your workstation, and
    port = the listening port number configured in the GUI
    TCPIP  directs the debugger to use a GUI
```

Example:

```
    //CEEOPTS DD *
    TEST(,,,TCPIP&123.45.67.89%8001:)
```

How to invoke the debugger: Batch LE program, connecting to a Terminal Interface Mgr (TIM) terminal

Insert a CEEOPTS DD statement with TEST run-time option in the JCL in the step or steps to be debugged. Syntax:

```
  //CEEOPTS DD *
  TEST(,,,VTAM%userid:)
    userid = your user ID
    VTAM%user-id:  directs the debugger to use the terminal
      interface manager. It will connect to the TIM terminal
      where userid is logged on.
```

Example:

```
  //CEEOPTS DD *
   TEST(,,,VTAM%USRX001:)
```

How to invoke the debugger:

Batch non-LE program, connecting to a GUI debugger

Change the program name on the EXEC statement to EQANMDBG, and code an EQANMDBG DD statement with the program name and a TEST option.

For example, if the EXEC statement in the run JCL looks like:

```
  //STEP10  EXEC  PGM=MYPROG,PARM='ABC,123'
```

Replace the EXEC statement with:

```
  //STEP10  EXEC  PGM=EQANMDBG,PARM='ABC,123'
  //EQANMDBG DD *
  MYPROG, TEST(,,,TCPIP&address%port:)
  /*
```

How to invoke the debugger: Batch non-LE program, connecting to a Terminal Interface Mgr (TIM) terminal

Change the program name on the EXEC statement to EQANMDBG, and code an EQANMDBG DD statement with the program name and a TEST option.

For example, if the EXEC statement in the run JCL looks like:

```
  //STEP10  EXEC  PGM=MYPROG,PARM='ABC,123'
```

Replace the EXEC statement with:

```
  //STEP10  EXEC  PGM=EQANMDBG,PARM='ABC,123'
  //EQANMDBG DD *
  MYPROG,TEST(,,,VTAM%userid :)
  /*
```

How to invoke the debugger: Debugging batch programs under TSO

The Debug Tool Setup Utility can optionally be used to debug batch programs under TSO. It is on the Debug Tool utility menu in ISPF.

How to invoke the debugger: CICS programs

Use the **DTCN** or **CADP** transaction to create a debugging profile for CICS applications, depending on which of these is installed on your systems.

The DTCN transaction is used to define a profile to start the debugger for one or more CICS programs, based on program name, transaction id, user id, and other criteria. DTCN is a feature of IBM Debug Tool for z/OS.

There is an optional graphical user interface for DTCN (an Eclipse plug-in) so you can set debugging profiles from a workstation without using a terminal.

The CADP transaction is used to define one or more profiles to start the debugger for CICS programs, based on program name, transaction id, user id, and other criteria. CADP is a feature of CICS.

The Language Environment TEST option

The LE TEST option is used to invoke the debugger.

It has five sub-options, separated by commas and a colon:
TEST(test-level , command-file , prompt , connection : preferences-file)

test-level is not typically coded. It is used to control when the debugger will automatically stop as a program runs. (default = all conditions and abends)

command-file can be used to specify the DDname or file name of a script file containing debugger commands that will run automatically.

prompt is not typically coded. (default = display the debugger when triggered)

connection controls where the debugger displays:
VTAM%user-id: = Connect to the TIM terminal where user-id logged on
MFI%terminal-id: = Connect to the non-TIM terminal named terminal-id
TCPIP%workstation_tcpip_address**%**port_id   = Connect to GUI debugging software such as the Debug Tool Eclipse plug-in

preferences-file can be used to specify the DDname or file name of a script file containing debugger commands that will run automatically. The preferences file (if specified) runs before the command file (if specified).

Example: TEST(,,,VTAM%USER123:)

Notes:

Use **NAMES EXCLUDE/INCLUDE** to reduce storage footprint especially in CICS, and to completely eliminate programs/non-executable load modules from Debug Tool consideration

Use **CALL %VER** to display WA the version and level of Debug Tool being used

Set up a log file so you have a record of your debugging session. If it isn't needed, no harm is done, but if you need it, then you do not have to recreate the debugging session to get the log. A **SET LOG ON FILE file-name OLD** command will open a log file.

CALL %HOGAN – invoke HOGAN application (CICS)
CALL %DUMP – invoke LE dump
CALL %FA – invoke Fault Analyzer (dump)
CALL %CEBR – invoke CICS temp storage browser
CALL %CECI – invoke command interpreter

DTCXXO – CICS transaction to TURN ON SUPPORT for non-LE assembler and/or OS/VS COBOL in CICS (Must issue this transaction in order to debug non-LE assembler or OS/VS COBOL under CICS) (Use DTCXXF to turn support "off")

Working with program function (PF) keys

QUERY PFKEYS	Display a list of the current function key settings in the log
SET KEYS ON	Displays the function key settings on the bottom two lines of the screen
SET KEYS OFF	Turns off the function key display
SET KEYS 12	With "SET KEYS ON", displays function keys 1 - 12
SET KEYS 24	With "SET KEYS ON", displays function keys 13 - 24
SET PF16 "Monitor" **= MONITOR LIST %CU LOCAL**	Set the F16 key to the command "MONITOR LOCAL %CU LIST". The function key display will show PF16 as "Monitor".

Default function key settings

F1 / 13	HELP
F2 / 14	STEP
F3 / 15	END
F4 / 16	LIST
F5 / 17	FIND
F6 / 18	AT/CLEAR
F7 / 19	UP
F8 / 20	DOWN
F9 / 21	GO
F10 / 22	ZOOM
F11 / 23	ZOOM LOG
F12 / 24	RETRIEVE

IBM

Debug Tool
Terminal interface

IBM's Interactive Debugger for applications running in z/OS

Commands used to work with storage and registers and assembler programs

For detailed descriptions of commands, command syntax, and command options, refer to the Debug Tool for z/OS Reference and Messages manual. A complete set of Debug Tool manuals can be obtained from the IBM Debug Tool website.
www.ibm.com/software/awdtools/debugtool/
select the "Library" link

Manuals:
Summary of Commands, User's Guide, Reference and Messages, and Customization Guide

For Use with Debug Tool Version 12

Set an AT CHANGE breakpoint based on a storage area

AT CHANGE %STORAGE (X'12B4C',20)	Set a change breakpoint to watch the storage area beginning at address 12B4C for a length of 20 bytes (Note: X'12B4C' is assembler syntax. For C it is 0x12B4C. For COBOL it is H'12B4C')

Display storage in the MEMORY window

ZOOM MEM	Display (zoom in to) the memory window Note: ZOOM again will zoom out of the memory window.
MEM variable-name	Position the memory window to the address of variable-name
MEM X'A500' **MEM X'A500'+20** **MEM X'A500'+X'B6'** **MEM X'A500'-32**	Position the memory window to the specified address or offset Note: If the address has more than 8 significant hexadecimal digits, it is taken as a 64-bit address. If it has 7 or 8 significant digits, it is a 31-bit address. Otherwise, it is a 24-bit address.
MEM %GPR12->	Position the memory window to an address pointed to by register 12

Display storage in the log or monitor

Note:
LIST … will display an item in the log.
MONITOR LIST … will display an item in the monitor.

LIST STOR(var,20**)** **MONITOR LIST STOR(**var,20**)**	Display 20 bytes of storage beginning at the address of variable var
LIST *or* **MONITOR LIST** *followed by one of:* **STOR(X'5F000',64)** *or* **STOR(X'5F000'->+256,64)** *or* **STOR(X'5F000'->+X'100')**	Display storage at an address or offset
LIST *or* **MONITOR LIST** *followed by one of:* **STOR(R1->,16)** *or* **STOR(%GPR1-> ,16)** *or* **STOR(%GPR1->+256,16)**	Display 16 bytes of storage at the address pointed to by a register, or an offset of a register address

Modify storage

1. MON LIST *var* to display the variable in the monitor **2.** Overtype the value of the variable in the monitor	Follow these steps to modify the value of a variable using the monitor window
1. ZOOM MEM to display the memory window **2. MEM** *address* to position to the address **3.** Overtype hexadecimal values in the memory window	Follow these steps to modify storage using the memory window
A1 = 1 *(note: decimal 1)* **A1 = 'Text'** **A1 = X'123C'** **A1 = A1 + 5**	Replace variable A1 with a value or expression
STORAGE(X'5F000',4) = 256	Update 4 bytes of storage at an address with the binary equivalent of decimal 256
STOR(X'5F000',4) = X'100'	Update 4 bytes of storage at an address with a right-justified hexadecimal value
STOR(X'5F000') = X'00000100'	Update 4 bytes of storage at an address with a hexadecimal value
STOR(X'5F000') = 'Some Text'	Update 9 bytes of storage at an address with a text string
%GPR8->+8 <l'x> = x	Assign the value of X to the 4 bytes at offset 8 from the contents of R8
%GPR2->+6 <14> = R8->+0	Move a string of 14 bytes pointed to by the contents of R8 (where R8 is an equated register in the program) to 6 bytes past the location pointed to by register 2
%GPR6->+0 <X'20'> = X'00	Set 32 bytes pointed to by register 6 to zero. Note: specify the length of the receiving storage within < >

Display registers in the log or monitor

Note:
LIST … will display an item in the log.
MONITOR LIST … will display the item in the monitor.

LIST REG	Display the sets of different types of registers in the log or monitor
MON LIST REG	Display all general purpose registers in the log
LIST *or* **MONITOR LIST** *followed by one of:* **64BIT REG** or **SHORT FLOAT REG** or **LONG FLOAT REG**	Display all of different types of registers in the monitor or log
LIST %GPR12 or **MONITOR LIST %GPR12**	Display general purpose register 12 in the log or monitor
LIST %GPRGn or **MONITOR LIST %GPRG**n	Display a 64-bit general purpose register in the log or monitor
LIST %FPRn or **MONITOR LIST %FPR**n	Display a short-precision floating point register in the log or monitor in hexadecimal format
LIST %FPRDn or **MONITOR LIST %FPRD**n	Display a short-precision floating point register in the log or monitor in decimal format
LIST %FPRBn or **MONITOR LIST %FPRB**n	Display a short-precision floating point register in the log or monitor in binary format
LIST %EPRn or **MONITOR LIST %EPR**n	Display a extended-precision floating point register in the log or monitor in hexadecimal format
LIST %EPRDn or **MONITOR LIST %EPRD**n	Display a extended-precision floating point register in the log or monitor in decimal format
LIST %EPRBn or **MONITOR LIST %EPRB**n	Display a extended-precision floating point register in the log or monitor in binary format

Modify the contents of a register

1. MON LIST %GPR*n* (or one of the other register types) to display the register in the monitor **2.** Overtype the contents of the register in the monitor	Follow these steps to modify the contents of a register using the monitor window
%GPR1 = x'1afc3' **%GPR12 = 10** **%GPR5 = %GPR5 + 1**	Replace the contents of a register with a value or expression Note: The other register types can be modified: %GPRGn (64-bit general purpose) %FPRn (floating point) %EPRn (extended floating point)

Display the address, length, and type of a variable

DESC ATTR var	List (describe) the attributes of variable var in the log

Display the program PSW (program status word)

LIST %PSW	Display the PSW in the log
MON LIST %PSW	Display the PSW in the monitor

Display the address of a program or module

DESC PROG pgm1	List (describe) the attributes of program ASAM1, including it's address in storage
DESC LOAD lmod1	List (describe) the attributes of load module LMOD1, including it's address in storage
LIST %EPA	Display the entry point address of the current program
LIST %AMODE	Display the current addressing mode
LIST %BLOCK	Display the name of the current block point (a CSECT is a block, for example)