

UDEBUG - Application Debugging

The Com-plete Application Debugger UDEBUG assists you in locating problem areas within an application and, where applicable, enables you to bypass a specific problem temporarily by altering various parts of the program's environment, such as storage contents, register contents or the actual program code itself.

Highlights of UDEBUG functionality include:

- Stop programs at specific points in their execution
- Alter storage
- Give instruction steps
- Trace MCALL
- Disassemble assembler code (not yet implemented)
- Modify assembler code (not yet implemented)
- Full screen conversational interface
- Powerful DSECT display facilities

This chapter covers the following topics:

- Overview
 - Restrictions on the use of UDEBUG
 - UDEBUG User Interface
 - UDEBUG Commands
-

Overview

What follows is a description of the terms associated with UDEBUG, followed by a description of how to use UDEBUG.

The UDEBUG Session

The UDEBUG session is simply the running of the program UDEBUG on a terminal attached to Com-plete. UDEBUG itself is a Com-plete application program with the ability to set dynamic hooks into another Com-plete application. In the case of a COM-PASS user, it must run as one of the user levels, in which case the application to be debugged may run on a different level or even a different terminal. In the case of a non-COM-PASS user, only one program may be run at a time. In this case, UDEBUG is this program and can only test an application to be debugged on a different terminal.

The Debugger

The Debugger is the session which is debugging an application program.

The Testing Terminal

The Testing Terminal is the terminal running the UDEBUG session. If the Testing Terminal is running COM-PASS, then there can be more than one UDEBUG session active on different levels. These different sessions can have their own individual Test Terminals and/or Test Levels.

The Test Terminal

The Test Terminal is the terminal running the program being debugged. A terminal can only be a Test Terminal for one UDEBUG user, even if the Test Terminal is running COM-PASS. However, if the Test Terminal is running COM-PASS, the Testing Terminal can test on more than one level. This simply requires that the Testing Terminal have a UDEBUG session active for each of the levels on the Test Terminal which are to be debugged.

The Test Level

The Test Level is the level running the program to be debugged. Only one debug session can be active on a particular Test Level at a time. When the Debugger is running COM-PASS and the Testing and Test Terminals are the same, the Test Level can be any level on the terminal except the level upon which the Debugger is running and level 0. If the Test and Testing Terminals are different, the Test Level can be any valid level for the Test Terminal. If the session with the program being debugged is not running COM-PASS, then the test level must be zero.

The Test Session

The Test Session is the Test Terminal / Level combination where testing is or will take place.

Breakpoints

A breakpoint is a point in a program's execution at which the program should be stopped to enable the Debugger to determine what the status of the program's operating environment is. At this point, the Debugger can change the program's environment, change the program itself or change the logical flow of the program.

UDEBUG Breakpoints

As stated previously, UDEBUG enables the user to stop the application program at certain points in the coding. This is done by setting breakpoints in the code. When a breakpoint is set at a certain point in the code, the Debugger receives control immediately before the instruction on which the breakpoint is set is executed. At this point, you can check the environment at the point at which the code was stopped and alter storage or the actual code before the code is executed. You can also restart the program being debugged at another address.

Breakpoints can be set to execute a certain number of times before giving control to the Debugger, and can also be set to stop giving control to the Debugger after it has been executed a certain number of times. They can be set in RESIDENTPAGE programs and in programs which are in the thread or will be loaded into the thread at some point in the execution of the program being debugged.

A breakpoint always has an owning TID/LEVEL combination and a Test TID/LEVEL combination. The owning combination identifies who set the breakpoint while the Test Combination identifies for which terminal/level combination the breakpoint has been set that is, the Test Terminal and the Test Level. If the program being debugged terminates for any reason, the breakpoints set by the Debugger will remain intact. If the Debugger terminates for any reason, the breakpoints are either deleted, or, if this is not possible, flagged for later deletion. Any user running on anything other than the Test Terminal Level combination who reaches the breakpoint is dispatched normally, though with extra CPU overhead to bypass the breakpoint.

When one or more breakpoints have been set, the Test Terminal and Level for that UDEBUG session will be unmodifiable until all the breakpoints are again deleted. The Debugger can then change the Test Terminal and/or the Test Level.

Warning About Breakpoints

Basically, the only way it can be known that a piece of storage will be executed is when the area of storage is fetched by instruction fetch processing. This applies even if the area is not even a valid instruction. Rather than restrict where a breakpoint may be set, UDEBUG simply ensures that the address where a breakpoint will be set is halfword aligned. This can conceivably cause problems, for example, if a breakpoint was set on the "to" address of an MVC instruction, it would change the location to where the MVC would take place. This will at best cause a storage exception or at worst, it will cause writing to an unknown storage area and enable the program to continue processing as if a certain piece of storage has been set. For this reason, care must be taken when a breakpoint is set to ensure that the breakpoint does not corrupt the program being tested.

Implicit Breakpoints

As stated previously, the user can set points in the program at which the program should stop. These are explicit breakpoints. We speak of implicit breakpoints when you have requested a function of UDEBUG which requires the program to be stopped at a specific point.

Currently, an implicit breakpoint is set for the following UDEBUG functions:

- Instruction Tracing
- MCALL tracing

Generally speaking, the implicit breakpoint only exists when the breakpoint is required, and it is deleted when the debugged program comes off the breakpoint. This means that, for example, during MCALL tracing, when the program issues a MCALL, an implicit breakpoint is set, causing the Debugger to get control. When the program is next dispatched and comes off the breakpoint, the breakpoint is deleted.

Breakpoints in Storage

It is also possible to set a breakpoint in storage obtained by an application program in the thread. This facility is available for high level language compilers that provide a trace function which simply branches to a storage area, in which there is simply a branch back. Another reason could be to enable the debugger to restart a program after a wild

branch. This type of breakpoint must be set with care and the following should be noted carefully:

- The breakpoint can only be set in an area which has been getmained in the thread. An attempt to set a breakpoint in a free area of the thread results in the breakpoint being flagged as invalid.
- Please be aware that setting a breakpoint in a storage area requires moving an SVC instruction to this location, overwriting the contents of the area for two bytes. By the same token, when the breakpoint is removed, the "instruction" moved from this area is moved back to leave the area in the state in which UDEBUG found it.
- It is your responsibility to ensure that the location the breakpoint is set is not altered while the breakpoint is active. If this occurs, the UDEBUG breakpoint SVC is deleted and therefore the breakpoint has no effect.
- When the Test Program terminates, any breakpoints that are set in storage are flagged as "Dormant". This means that the control block identifying the breakpoint is still there, but, the breakpoint is not set until the Debugger reactivates the breakpoint by issuing the AT command. This is to ensure that the test program can get the storage and build the environment before UDEBUG attempts to set the breakpoint again.

Redispatching a Breakpointed Program

When a program being debugged is on a breakpoint, if the Test and Testing Terminals are not the same, the only way to restart a breakpointed program is by issuing the UDEBUG command GO. However, if the Test and Testing Terminals are the same, you can issue the GO command or, if you have stacked the UDEBUG session, you can simply restart the program by using any of the commands from the COM-PASS screen to select the suspended Test Level.

MCALL Tracing

When MCALL tracing is active for a Test Session, when the program being debugged is about to execute an MCALL, the Debugger is given control. That is to say that the Debugger is given control BEFORE the MCALL is executed, giving the Debugger the chance to change the parameter options or even to bypass the MCALL altogether. Please note that when MCALL tracing is active, ANY program running on the Test Session will cause control to be passed to the Debugger when a MCALL is issued.

Instruction Tracing

Instruction tracing enables the Debugger to step through a program instruction by instruction. That is to say that once instruction tracing is activated, the Debugger gets control prior to the execution of every assembler instruction. As this is simply an implicit breakpoint, you can make any changes during an instruction trace which can be made at an explicitly defined breakpoint.

Instruction tracing can be active with no effect. That means that the Debugger may have set the option for the Test Session, but nothing happens. This is because instruction tracing can only be effected when the program to be debugged has terminated at a breakpoint, be it implicit or explicit. The UDEBUG code checks when coming off the breakpoint if the option is set or not. When it is set, it is then activated for the next instruction after the breakpoint.

The instruction tracing can only go as far as tracing within application programs. This means, that if the application issues a SVC call or an internal Com-plete nucleus call, this cannot be traced, as breakpoints cannot be set in these places. In these cases, UDEBUG sets the next "instruction trace" breakpoint at the instruction following the instruction that cannot be followed.

UDEBUG Symbols

To make programs more meaningful, various symbols or labels can be defined. In the same way, to make certain addresses and displays more meaningful in a UDEBUG test environment, symbols can be set in various ways. During the following description, reference is made to level 1, 2 and 3 symbols. How these categories related to UDEBUG is described later.

Standard Equates

A user can set up equates within a session using the EQUATE command (described later). This command enables an eight character identifier to be set to a particular address in storage. When this equate is later referenced alone or within an expression, UDEBUG uses the address to which the identifier was equated to resolve the request. An equate can also be given a length so that during disassembly or when listing the equate, UDEBUG "knows" the length the data that the equate describes. In UDEBUG terms, an equate is a level 1 type symbol.

Load Module Equates

Many load modules consist of more than one CSECT which can cause them to be very large. In any case, working with a linked module can be tiresome given the fact that initially, only the actual module name will be known. UDEBUG enables you to "know" each of the CSECT within a linked module, so that when the load module name is resolved, you can reference the various CSECTS within the module directly. This is achieved by using the LMODULE command (described later). This causes the load module to be read from a specified load library and an internal equate set up for each of the CSECTS within the load module.

The CSECT equates are set up relative to the start of the actual load module, so once the load module is not relinked, it can be loaded anywhere and UDEBUG succeeds in addressing the various CSECTS correctly. This is particularly useful in the case of a relocatable load module. In UDEBUG terms, the load module name which is loaded is a level 1 symbol, while the actual CSECT equates themselves are level 2 symbols.

Testran Defined Symbols

Current assemblers running under MVS enable you to produce what is called "testran" records. These records describe each of the various DSECTS and CSECTS that exist within a module and the various labels within the DSECT or CSECT. UDEBUG can read the testran symbols using the READ command (described later). The member name being read becomes a UDEBUG level 1 symbol, while each DSECT or CSECT becomes

a UDEBUG level 2 symbol. The fields defined within a DSECT or CSECT then become UDEBUG level 3 symbols.

The symbols defined as a result of the READ command can then be used to see the structure of DSECTS and/or CSECTS including field names, lengths and offsets. When a particular CSECT or DSECT is resolved, the contents of the various fields can also be seen. Resolution of CSECTS and DSECTS is discussed later.

Local and Global Symbols

The number of symbols you wish to define may run into thousands. It would not be practical to keep all these in the UDEBUG thread, therefore the concept of global and local symbols is available. A globally defined symbol is available to all UDEBUG users running within that Com-plete until it is deleted. In this way, commonly used control blocks and/or load modules can be defined globally.

You can define global symbols. However, in the case where one DSECT has changed while all of the "common" DSECTS available globally have not, you can simply define the changed DSECT locally. UDEBUG will always use local definitions before attempting to use global definitions. Therefore, with local symbols, you can effectively "front end" the globally defined symbols.

UDEBUG Symbol Levels

UEBUG "sees" all symbols as being of a certain level. This level determines whether a symbol is already defined or not. For example, a load module name is a level 1 type symbol and therefore no other load module of that name can appear on level one. Of course the same load module name can exist locally and globally as the global level 1 and the local level 1 are seen as being different.

As an example of how the second level functions, assume two NATURAL V21 linked nuclei named TSTNUC1 and TSTNUC2. Both of these have a CSECT named ACMDRIV and both can be defined, as the level 1 names are different. However, TSTNUC1 could not have the CSECT ACMDRIV linked twice (even though this makes no sense it is technically possible). In this case, UDEBUG only honours the first occurrence, and issues an error message about the second. In the same way, level 3 labels can only exist once within the level 1/2 combination, meaning, for example, that many DSECTS can have the same field name.

Resolution of Symbols

Where a symbol is not equated directly to an address, UDEBUG attempts to find where the address is. Firstly, UDEBUG searches for an equate for the highest level name for the symbol. For example, if a CSECT is referenced, UDEBUG attempts to find an equate for the load module or testtran member to which the CSECT relates. If no equate is found, UDEBUG attempts to find a using statement for the entity. If it is a level 3 field, a level 2 CSECT or DSECT is searched for. In the case of a level 2 DSECT or CSECT, the section itself is searched for. When symbols cannot be resolved either via an equate or a using, depending on the circumstances, you are either informed, or the area displayed relating to the symbol contains the "not resolved" UDEBUG character.

As various different modules and/or members used to create symbols may have the same symbols defined as a result of the LMODULE or READ command, the user has the ability to specify which particular symbol is being referred to. This is done as follows:

Lev1name.Lev2name

where

Lev1name	is the name of the module which was read by the LMODULE command or the member name read by the READ command.
Lev2name	is the symbol you want to reference.

If the symbol is entered on its own and exists more than once, the first occurrence is taken.

If you wish to always work with the same level 1 name as opposed to having to type in "Lev1name.Lev2name", if a Test Program name is entered, UDEBUG uses a symbol related to that program. When it fails to find this, it simply uses the first occurrence as before.

Addressing Mode

In systems capable of 31 bit addressing, there is always a question as to how to interpret an address. When a program is on a breakpoint, UDEBUG uses the AMODE in which the program is running to determine how it should interpret addresses. However, when the program is not at a breakpoint, all addresses are interpreted as 31 bit addresses.

Implicitly Defined Symbols

To avoid having to set certain standard symbols, UDEBUG sets some defaults at various points in the execution. The following symbols are set at startup and re-evaluated periodically to ensure that they are still correct.

DCOMREG	COMREG for the Com-plete where UDEBUG is running
DTIB	The TIB on which the UDEBUG session is running
Resident programs	For each resident program defined in the Com-plete system, UDEBUG sets up an equate for the name of the program, its address and length. As these are continuously updated, additions to the residentpage list via the PGM operator command, or any refreshes of programs should be reflected almost immediately.

The following symbols are defined as soon as a program is active on the test TID/Level. They are deleted as soon as the level is freed.

DUPCB	UPCB of the program being debugged.
THDS	Start of thread for the program being debugged.
THDE	End of thread for the program being debugged.
THXS	Start of thread extension for the program being debugged.
THXE	End of thread extension for the program being debugged.
programs	The root program name and any modules loaded into the thread will be set up as equates.

When a breakpoint is reached, UDEBUG deletes and sets the following symbols. If these names are already used, they are deleted and set as indicated below.

BPPSW	Breakpoint PSW address
BPRx	Breakpoint register, where 'x' is '0' to '9' or 'a' to 'f'.
BPTIBA	The TIB buffer related to the program under test.

Address Expressions

When attempting to address a storage area, UDEBUG can read various types of expression to enable the Debugger to find the address space. The given expression is evaluated and the result of that evaluation is used as the absolute address.

Of course, absolute addresses can also be used. All numeric data entered must be preceded by the hex character to indicate hexadecimal data, or the decimal character, which indicates a decimal number. Hex is the default. If the value is preceded by the relocate character, the resultant value is calculated relative to the relocated address as set by the RELOC command. The decimal, hex and relocate characters and how to set them are described later in this chapter. The following indications can be used in an expression.

Hex/decimal values	Absolute values
Hex/decimal addresses	Absolute addresses
Hex/decimal offsets	Relative offsets from a point set by the user
Symbols	Globally or privately defined symbols
Arithmetic operators	Plus (+) and Minus (-) enable the Debugger to add or subtract from addresses.
Brackets "(" and ")"	When brackets are specified, the result of the expression within the brackets is used as the address of the fullword. This value is then used in the expression in place of the brackets and its contents.

Note:

The arithmetic operators and brackets mentioned above can be customized to suit your requirements. This procedure is described later.

Storage Display and Modification

When a breakpoint is set, the program code will contain the UDEBUG SVC. In this case, the Debugger is not concerned with what is really at this point but what is logically at this point, that is, the replaced instruction. Therefore, when UDEBUG displays such a piece of storage, the logical contents of the storage are displayed, that is, the storage is displayed with the replaced instruction, and not the UDEBUG breakpoint SVC.

By the same token, if you wish to update a piece of storage and a breakpoint exists at the point where the modification is to take place, UDEBUG handles this by updating the breakpoint in such a way that the instruction to be executed is modified in core. This becomes apparent when the debugged program leaves the breakpoint.

Modifying thread storage is a little more complex. If UDEBUG is running with no program active on the test level, absolute addresses are treated as such, that is, if you enter an address which is contained in a Complete thread, you will see the actual data which is there at that point in time. However, when a program is active on the level, and you enter an address which is in the program's thread somewhere, this is resolved logically in the rolled out image of the thread.

There are cases where you receive the message that the storage is not available. This can occur if the image of the test program could not be written to the roll buffer; in this case contact your system programmer. It can also occur if the UDEBUG session was started on a test level AFTER that level was last active. In this case, Complete would not have known that the test level should have been retained in the roll buffer. This can be corrected by simply activating the test level once.

Abend of a Test Program

When a program which is being tested abends, the Debugger is notified and can look into the thread. However, the program cannot currently be restarted after an abend. When a Complete dump is written to the Complete SD dataset, the thread will have been rebuilt to how it logically looked. This means that instructions upon which a breakpoint is set will have been rebuilt. It also means that bad breakpoints which were set in the middle of an instruction, for example, will also have been reset. If one of these bad breakpoints has caused the abend, it will not be obvious from the dump.

If any other dump of storage is taken other than a Complete dump, for example, if applymod 73 is on, the thread will be dumped as it was at the time of the abend, thus all UDEBUG SVC instructions will still be in place. If confusing results are being obtained, the problem can be further traced with such a dump as it will reflect the true status of storage at the time of the abend.

Confirmation Processing

UDEBUG provides full screen interfaces to display any information that the Debugger may require. In most of these screens, you can update some or all of the areas. When this occurs, if confirm processing is on, the changed field will be highlighted and protected and you are asked to confirm the change. If the CONFIRM UDEBUG command is then

issued, the updates are made as requested. If anything else is entered, the updates are forgotten. This facility can be turned on or off using the SET command.

If you want to use Confirm processing, but do not wish to have to go through two input operations, you can enter the CONFIRM command on the command line when the updates are made. This is then taken as confirmation that the updates should be done. The easiest way to do this is to set the CONFIRM command on a PF Key. In this way, the updates can be made and the PF Key pressed to cause the updates to be accepted. Any changes made inadvertently followed by an enter causes normal confirm processing to take place.

Restrictions on the use of UDEBUG

There are a number of restrictions and warnings that users must take into account while using UDEBUG. Some of these situations are actually disallowed by UDEBUG, however, in certain cases, UDEBUG cannot determine that this is the case and may accept and perform a certain task. However, the results will then not be as expected.

Execution of Breakpoints

A breakpoint may only be activated if the area of code where the breakpoint has been set is executed. When a breakpoint is requested by a user for a program, UDEBUG has no way of knowing whether an instruction will be executed or not as the only way to determine this is to execute the program. Where a breakpoint has been set and the code not executed, you should first ensure that that section of code is actually being executed in your application.

It would be possible to at least ensure that the address where the breakpoint is set an instruction. This was not implemented, as many people use an invalid hex instruction code to cause a program to abend. In some cases, users may wish to set their breakpoint on this 'instruction' to take another course of action. It is also possible that a breakpoint could be set on the address operator of an instruction which could represent a valid instruction code. This means that it is impossible to accurately determine if the storage at a location is actually an instruction so to avoid ambiguities whereby sometimes it would fail a request and other times not, it was decided not to include any checks whatsoever.

Note also that the setting of a breakpoint based on an offset into a module to be loaded into the thread can only occur when the module is actually loaded into the thread. In the event that the offset does not exist in the module, this will only be known when the module is loaded and as such, can only be marked as an invalid offset at that time. If such a breakpoint fails to be triggered, check if the breakpoint has been flagged as invalid using the BPLIST function.

Instruction Stepping

The instruction step functionality provides the facility to step through a program on an instruction by instruction basis. For each application program instruction, a breakpoint is triggered which will force the thread image to be rolled out of the thread. This will cause problems with programs which are event based whereby they issue a request for which they expect to be posted back when the request is completed.

If the ECB in use is inside the thread, it is likely that the serving posting back the indication that a request has completed will post the storage location in the actual thread storage area and not the actual ECB in the user's rolled out thread copy. This will result in the POST being lost to the expectant application due to the fact that it's copy of the ECB is never updated.

This may also cause other problems for the current occupier of the thread whose storage at the equivalent location will be overwritten by the post intended for the application program being debugged.

There is no way to know or protect against this. Therefore, application programs of this nature must be tested with extreme care and no breakpoints either explicit or implicit set between the requesting of a service and the wait for that service to complete.

UDEBUG User Interface

This section explains the user interface to UDEBUG. It describes the general layout of the screens along with the available options. The various UDEBUG maps are then shown and described in detail.

UDEBUG Session Startup

UDEBUG must be started with the command

***UDEBUG**

UDEBUG sets up the standard user environment as follows:

1. If you are a COM-PASS user, then the current terminal is set up as the Test Terminal if another user is not testing on it. In the case where another user is testing on the terminal or if you are not a COM-PASS user, the Test Terminal is set to 0 and you must set it to a valid terminal before testing can begin.
2. For a COM-PASS user, the next available level is set as the Test Level. This means that the UDEBUG level plus one is taken to the maximum levels available. When this is exceeded, it wraps around to take level 1. If a Test Session already exists on this level, or you are a non COM-PASS user, the Test Level is set to zero.
3. Symbols are set up for the following:
 - All residentpage programs (symbol names will be the program names)
 - COMREG (symbol name DCOMREG)
 - The UDEBUG Session TIB address (symbol name DTIB)
4. The hardcopy TIB for the Testing Terminal is set as the UDEBUG hardcopy device when available.
5. The UDEBUG nucleus is loaded.
6. The profile named after the your user ID is executed from the default profile DD/DLBL (COMDBPRF) if the DD/DLBL exists and a profile exists for you. For more details, see the description of the PROFILE command.
7. Any data entered after the *UDEBUG is assumed to be command data and is passed to the UDEBUG command handler.

When the above has completed, you will generally be presented with the UDEBUG session information screen UDB0. This will NOT be the case if data entered along with the *UDEBUG command or a command in the profile has a different map to be selected. For example, if a DUMP command was

contained in your profile, the first screen you will see is the storage display map UDB1.

The UDEBUG program can optionally be specified as a STARTUPPGM (see the sysparm by that name) which will cause UDEBUG to be attached with the user ID SYSUSR. You must then add a member name SYSUSR to the profile dataset which can contain UDEBUG commands to build global symbols that are available to all UDEBUG users. The profile must always end with the EOJ command to ensure that the program terminates cleanly. The following example will cause the Complete nucleus name COMPLETE to be read and each CSECT name in the nucleus to be set up as a global symbol.

```
LMOD COMPLETE * GLOBAL
EOJ
```

See also the description of the LMODULE command below.

Error and Information Messages

At UDEBUG startup and for various command combinations while in the UDEBUG session, more than one message may be required. Depending on the number of lines on the terminal, UDEBUG saves each message to the maximum available lines on the UDEBUG message screen UDB2. When more than one message is required, the first message is displayed in the UDEBUG map message line with '+++' as the message ID and not ZDB. To see the additional messages, simply enter MESSAGES on the command line. If more messages than can fit on the messages screen, this is indicated by the last message on the screen. All error messages following the last displayed message are discarded.

Program Function Keys

Generally, when ENTER-Key is pressed from a screen, anything typed on the command line is first interpreted and the relevant command executed. The screen handler then gets control to handle input from the screen (if any) and the screen is redisplayed with any appropriate messages and/or updates. This of course will not be the case if any of the commands entered cause the display screen to be changed.

When you press a program function key, either the user-specific keys are used, or the global PF keys for the system. Global PF keys are maintained by the system administrator, while you can add personal PF keys using the UPROF utility. When a PF key is defined, the data as defined for the PF key is taken as a command and passed to the UDEBUG command processor. If you press an undefined PF key, an appropriate message is displayed. The bottom line of the screen displays the first 5 characters of the command allocated to the PF key. When the PF key is not defined, no text appears under the PF key header. If the PF key is defined with the "display" option, the contents set for the PF key are displayed on the UDEBUG command line.

Customizing Characters and Options

For all functions, UDEBUG has default characters and options. However, you can set these as required. This can be done via your user profile. The following lists the operand name for the SET command which enables the character to be set together with the standard default for the character. When the operand ON is given for one of the following options, the default is set when possible. When OFF is specified, the character is disabled or x'00' is displayed when the option is a display option.

For the following characters, all of the characters must be unique within the list.

Operand	Default	Description
Decchar	Z	Identifies a numeric string as decimal
Hexchar	X	Identifies a string as hexadecimal
Relchar	#	Identifies a string as relative to the address set by the RELOC command
Parmdel	.	Delimits parameters within an operand, for example, NATCOM21.ACMDRIV identifies NATCOM21 as the level 1 and ACMDRIV as the level 2 symbol within an expression
Opdel	,	Operand delimiter. This character or blank must be used to delimit operands within a UDEBUG command
Cmddel	;	Command delimiter. This character delimits UDEBUG commands on the same command line
Pntldel	(Pointer left delimiter. This indicates the start of an expression which is used to get an address. The expression between this pointer and the Pntrdel delimiter is evaluated and the address that the expression points to is used as the result, as opposed to the result of the expression itself
Pntrdel)	Pointer right delimiter. This closes the procedure started by the Pntrdel character
Taddrid	*	When the current top address is to be used in an expression, this character can be used to avoid the need of retyping the address in full
Addchar	+	Indicates that two values are to be added together
Subchar	-	Indicates that two values are to be subtracted from each other

The following characters do not need to be unique as they are used in a different context to the above characters.

Operand	Default	Description
DEFchar	*	When a UDEBUG command has more than one positional operand and you wish to use the default for one or more of the operands, this character can be used in the operand position
NOTRes	-	If UDEBUG attempts to display storage based on a symbol and the symbol cannot be resolved, this character is used to fill the area
NOTAlloc	.	If UDEBUG attempts to display storage and the storage is not allocated, this character is used to fill the area
NOTACocc	=	If UDEBUG attempts to display storage and you cannot access the storage, this character is used to fill the area

UDEBUG Screens

General Format

The following is the format of all UDEBUG screens on a 3270 model 2 device (24 lines by 80 columns). Each of the areas which appear on all screens are described once here.

```

Message line .....
Time          TID -tid"           Inst-ID       User -userid"   Date
              --- -Screen Name" ---

Screen Headings

Command line .....
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Function keys assignments .....
    
```

Message Line	This line is used by error and informational messages. UDEBUG messages have the format ZDBnnnnn, where nnnnn is a message number. If UDEBUG has more than one message to display, the first message is displayed in this line with an identifier of '+++nnnn'. This indicates that there are more messages which can be seen by displaying the Messages Screen.
Time	This field contains the time of day in the format HH:MM:SS that the screen was sent to the terminal.
Tid	This field contains the Terminal ID Number in Complete on which the UDEBUG session is running.
Instid	This field contains the Installation ID for the Complete running the UDEBUG session. This ID is set at Complete startup by the INSTALLATION sysparm.
Userid	This field contains the user ID of the user logged onto the terminal running the UDEBUG session.
Date	This field contains the date that the screen was sent to the terminal. If applymod 61 is not set, the date has the format: MM/DD/YY. If applymod 61 is set, the date has the format DD.MM.YY.
Screen Name	This field contains the title of the UDEBUG screen currently displayed. Each UDEBUG screen has a title (see the example screens later in this section).
ID	This is the UDEBUG map identifier for the screen currently displayed. This ID has the format UDBx, where x is an internal identifier for the screen.
Screen Headings	When the individual UDEBUG screens require headings for the various fields, they are displayed here.
Command Line	UDEBUG commands can be entered in this line. When there are no errors in a screen display, the cursor is positioned here.
Function Key Settings	You can use the globally defined PF keys or a set of PF keys customized set for yourself. When defined, UDEBUG takes the first five characters of the command defined for each PF Key and displays it here. If nothing is defined, nothing is displayed. The Enter key is always used to enter data.

The UDEBUG Session Information Screen (UDB0)

This screen is displayed as the result of the *UDEBUG call from COM-PASS and contains displays relevant data related to your UDEBUG session currently being run. You can modify various fields to change the criteria for the current session.

```

08:22:26      TID    17          COM-5.1.      User MBE      04/08/97
              ---  Session Information  ---
Test Information-  User Settings----- Miscellaneous-----

TID          17    MCALL Tracing      N    Defined Symbols      42
Luname      SHRDAEN  Instruction Tracing  N    Defined Breakpoints   0
Level       2      Confirm            N
Program     Bump Storage      N
            Default Character      *
            Hardcopy Luname
            Hardcopy TID          15

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help          Retur Confi Dump  Backp Forwp Messa BP    Go    Recal
    
```

Meaning of the information:

Test Information

This column contains data relevant to the Test Session:

TID	This is the number of the TID on which the user is testing. If this contains "0", no Test TID has currently been selected. You can alter this field only if no breakpoints have been set.
Luname	This is the logical unit name of the TID on which you are currently testing. If no Test TID has been selected, this is blank. You can alter this field only if no breakpoints have been set.
Level	This is the level number on which you are testing. If the Test TID is running COM-PASS, this is a number from 1 to 9. If the Test TID is not running COM-PASS, this is "0". You alter this field only if no breakpoints have been set.
Program	This is the name of the program or module which you are testing. Currently, you use this field to define which symbol to use in cases where there are duplicates. See the section on UDEBUG Symbols for more details. This can be altered at any time during the UDEBUG session.

User Information

This column displays the current settings for various UDEBUG options which you can modify:

MCALL racing	This indicates whether MCALL tracing is active for the TestSession or not. "Y" indicates that it is active and "N" indicates that it is not active. This can be altered at any time during the UDEBUG session. However, it will only be effective the next time a MCALL is issued from the Test Session.
Instruction Tracing	This indicates whether UDEBUG instruction tracing is active for the Test Session or not. "Y" indicates that it is active and "N" indicates that it is not active. This can be altered at any time during the UDEBUG session. However, it will only be effective if the Test Session is on a breakpoint or after the next time the Test Session reaches a breakpoint.
Confirm	This indicates whether UDEBUG Confirm processing is active for updates. "Y" indicates that it is active and "N" indicates that it is not active. This can be altered at any time during the UDEBUG session and is effective immediately.
Bump Storage	This indicates whether UDEBUG Bump processing is active for UDEBUG Screens. "Y" indicates that it is active and "N" indicates that it is not active. With UDEBUG Bump Processing active, on screens where there is more than one screen page of information, if you press ENTER-Key without typing in any data, the following screen of data is displayed. This can be altered at any time during the UDEBUG session and is effective immediately.
Hardcopy Luname	This is the name of the hardcopy device to which UDEBUG will route any hardcopy output it may generate as a result of UDEBUG commands. This can be altered at any time during the UDEBUG session and is effective immediately.
Hardcopy TID	This is the Terminal ID of the hardcopy device to which UDEBUG will route any hardcopy output it may generate as a result of UDEBUG commands. This can be altered at any time during the UDEBUG session and is effective immediately.

Miscellaneous

This column contains any other data relevant for the UDEBUG session. Fields in this column cannot be updated.

Defined Symbols	This is the number of symbols that are locally defined for this UDEBUG session. This will never exceed the maximum defined at Complete startup by the SYMTAB sysparm.
Defined Breakpoints	This is the number of breakpoints that are currently defined for this UDEBUG session. It does not include implicit breakpoints or breakpoints that have been flagged for deletion. It contains breakpoints that are dormant. Please refer to the section on Breakpoints for more details.

The UDEBUG Dump Storage Screen (UDB1)

This screen is displayed when you press the appropriate PF key (in our example above, PF6). The display shows your storage in hex and character format with the current breakpoint registers for reference, if the Test Session is currently on a breakpoint. It also enables you to change this storage.

08:23:44	TID	17	COM-5.1.	User MBE	04/08/97
			--- Dump Storage ---		UDB1
Address	Relative	Hex contents	-----		Char contents--- BP Regs-
00000000		040C0000 810B58F0	00000000	80000000	.. a..0 .
00000010		00FDD030 00000000	070E0000	00000000	.ü. ..
00000020		078C2000 0001DCF4	078C2000	0001D8B44... .Q.
00000030		00000000 00000000	070E0000	00000000	..
00000040		00000000 00000000	00000000	00FDD030	.ü.
00000050		00000000 00000000	040C0000	810AD4A8	.. a.My
00000060		040C0000 00CA5EF0	00080000	87DC51D0	.. .;0 . g..ü
00000070		00080000 87DC61B0	040C0000	810AF700	. g./... a.7
00000080		00000000 0011202	000400C8	00040011 H . .
00000090		0001D000 00052000	00D3D512	00000105	.ü .. LN. ..
000000A0		00000000 010B2F88	00000000	00000000	...h
000000B0		00000000 00000000	0001022F	00FD67E8Y
000000C0		28000000 00000000	00000000	00000000	.
000000D0		00000000 00000000	00000000	00000000	
000000E0		00000000 00000000	00000000	00000000	
000000F0		00000000 00000000	00000000	00000000	
00000100		00000000 00000000	00000000	00000000	Key 0FP
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---					
Help		Retur Confi Dump		Backp Forwp Messa BP Go Recal	

To cause the first address to be displayed to change, simply enter expressions on the command line or use cursor positioning. When the cursor is placed in either the Address or Relative line, this line becomes the top line and the top address is altered accordingly. When the cursor is entered in either of the contents fields, the byte on which the cursor has been placed becomes the top address for the screen.

Meaning of the information by column heading:

Address

This is the address of the storage which is being displayed. The fields in this column cannot be altered.

Relative

This is the offset of the storage relative to the relocation address provided via the RELOC command. If the relocation address is 0, that is, if the relative address is equal to the actual address, nothing is displayed in this column. The fields in this column cannot be altered.

Hex Contents

This column contains the hex representation of the storage found at the address indicated in the Address column. For easier, it is represented as four eight byte areas, each representing four bytes of storage. When the storage does not exist, the field or portion of the field is filled with the UDEBUG "not allocated" character (default is ".") and is set using the SET command.

The hex data fields can be modified to cause the storage area itself to be updated by UDEBUG. In a case where storage does not exist, or only a portion of the four bytes to of a hex field exist, the field is protected. That means, that the field is only available for update when all four bytes exist.

Character Contents

This column contains the character representation of the storage found at the address indicated in the "Address" column. This output is translated according to the utility output translation table for the terminal. When the storage does not exist, the field or portion of the field is filled with the UDEBUG "not allocated" character (default is ".") and can be set using the SET command.

The data can be modified to cause the storage area itself to be updated by UDEBUG. In a case where storage does not exist, or only a portion of the sixteen bytes exists, the field is protected. That means that the field is only available for update when all sixteen bytes exist. Care must be taken when updating storage via this field, because for any update, all sixteen bytes are written back to the appropriate storage area. When the sixteen bytes contain non-character output and have been translated, it is possible that invalid hex values are written back to storage. It is therefore recommended that storage only be updated via the character representation when all sixteen bytes are valid characters.

BP Regs

This column contains sixteen 8 byte fields which display the hex representation of the contents of each of the breakpoint registers when the user is on a breakpoint. The top line represents R0, the next R1, and so on, up to RF. When you are not on a breakpoint, these fields are blank. These fields are not modifiable. In order to modify the breakpoint registers, you must go to the Breakpoint Information screen.

Key

This field gives information about the storage key of the page corresponding to the first address displayed on the screen. There are three positions to this field. The first is always filled and contains the storage protect key of the storage represented by "0" to "F". The second position indicates if the storage is fetch protected or not. When the storage is fetch protected, it contains the character "F", and when it is not fetch protected, it contains a blank. The third position contains a "P" if the storage area is page protected, and a blank if it is not. If the storage being displayed on the screen relates to two pages, the information only relates to the first page displayed.

The UDEBUG Error Messages Screen (UDB2)

When UDEBUG has more than one message to be displayed, only the first is displayed on the message line. You can then go to this screen by pressing the appropriate PF key (in our example, PF9) to see the other messages. Each information line on the display contains one message. No input options are available on this screen.

Meaning of the information according to column header:

Breakpoint data

This column contains information relating to the breakpoint on which the test program is currently sitting, together with information about the state of the user.

Addressing Mode	This indicates the address mode that the user program is running in. Under non-XA capable systems, this is always 24. You can optionally change the addressing mode when the test program is on a breakpoint. When this is done, the next time the test program is dispatched, it will have the new address mode as set by you.
State	This indicates the state that the user program is running in.
	S Supervisor state
	P Problem Program state
	You can change this, causing the test program to be in the specified state the next time it is dispatched.
Protect Key	This indicates the protect key that the user program is running in. This contains a value from "0" to "F", depending on the protect key found in the PSW. You can change this, causing the test program to be in the specified set protect key the next time it is dispatched.
Exception Masks	These fields indicate whether various exceptions will cause a program interrupt or not. Possible values:
	N if the exception occurs, it is ignored Y an exception will cause a program interrupt. These exceptions relate to the four exception masks found in the PSW. You can change one or more of these settings to cause the program to run with the exception condition active or inactive the next time it is dispatched.
Condition Code	These field contains the last condition code set by the test program. It can be a number from 0 to 3. You can set this condition code to one of the desired values to cause the program to see a different condition code the next time it is dispatched.

Id	This field contains the identifier of the breakpoint on which the test program is sitting. For an explicit or user-defined breakpoint, this is the ID specified on, or generated by the AT command. For implicit breakpoints such as those for instruction tracing or MCALL tracing, the following values may be found here:	
	INSTTRCE	Indicates an instruction tracing breakpoint.
	BP-MCALL	Indicates an MCALL breakpoint.
	BP-OSSVC	During user program processing, some OS SVCs area is trapped and satisfied by Com-plete. When MCALL tracing is active, these will also cause a breakpoint to occur before the request is satisfied.
	*ABEND**	Indicates a user program abend breakpoint.
Information	For various breakpoints, additional information is provided in this field as follows:	
	BP-MCALL	The MCALL request being issued is displayed.
	BP-OSSVC	The OS Macro name associated with the SVC or the SVC number is displayed.
	*ABEND**	The abend code is displayed.

BP Contents

This column contains the contents of the PSW and registers for the breakpoint as indicated on each line. The PSW cannot be explicitly changed on this screen. However, alterations in the Breakpoint Data column or a parameter on the GO command will obviously cause the PSW to be altered. The register contents can be altered, the changes take effect when the test program is next dispatched.

Data

This column shows the storage at the location pointed to by the PSW or register on the appropriate line. The register contents are interpreted according to the address mode of the user program. If the address does not exist, the UDEBUG "not allocated" character is displayed.

The UDEBUG Symbol Display Screen (UDB5)

This screen displays locally and globally defined symbols as a result of the SYMBOLS command. As the screen must display many different symbols, all the headings do not make sense for all symbols. A number of examples are therefore given, one for a display of level 1 symbols, one for a display of level 2 symbols and one for a display of level 3 symbols. A description of the fields as they are displayed for the first example with any differences noted for the subsequent examples.

Apart from the FORWPAGE and BACKPAGE commands, the top line to be displayed can be selected by placing the cursor anywhere on the desired line and pressing ENTER.

Level 1 Example Display:

16:51:03	TID	13	COM-5.1.	User	JPO	02/03/97
--- Symbol Display ---				UDB5		
Type	D/Csect		Label	Equ/Lmod/Memb		Scope G
Symbol--	Mult---	Type	Lngrh-	Disp----	Addr----	Contents-----
U2DBMAP4	1	EQU	2600	00000000	04CE45D8	47F0F060E4F2C4C2D4C1D7F4F4F5F4 >
U2DBMAP5	1	EQU	6384	00000000	04CE2710	47F0F060E4F2C4C2D4C1D7F5F4F5F4 >
U2DBMAP6	1	EQU	3496	00000000	04D29220	47F0F060E4F2C4C2D4C1D7F6F4F5F4 >
U2DBNXTO	1	EQU	352	00000000	0470A040	47F0F060E4F2C4C2D5E7E3D6F4F5F3 >
U2DBOFF	1	EQU	1024	00000000	04CE14E8	47F0F060E4F2C4C2D6C6C640F4F5F3 >
U2DBOSIO	1	EQU	1136	00000000	00337240	47F0F060E4F2C4C2D6E2C9D6F4F5F3 >
U2DBOSLD	1	EQU	1816	00000000	04CE18E8	47F0F060E4F2C4C2D6E2D3C4F4F5F3 >
U2DBPRLG	1	EQU	264	00000000	047002A8	47F0F060E4F2C4C2D7D9D3C7F4F5F3 >
U2DBPROF	1	EQU	1328	00000000	04715340	47F0F060E4F2C4C2D7D9D6C6F4F5F3 >
U2DBREAD	1	EQU	2752	00000000	04BE1330	47F0F060E4F2C4C2D9C5C1C4F4F5F3 >
U2DBSET	1	EQU	1392	00000000	047003B0	47F0F060E4F2C4C2E2C5E340F4F5F3 >
U2DBSTOR	1	EQU	1384	00000000	04714658	47F0F060E4F2C4C2E2E3D6D9F4F5F3 >
U2PRINT	1	EQU	1104	00000000	00706608	47F0F060E4F2D7D9C9D5E340F4F5F0 >
U2STHD	1	EQU	328	00000000	0003C018	47F0F060E4F2E2E3C8C44040F4F5F0 >
CCOMBLKS	1	MEM	0	00000000		-
COMPLETE	1	LMOD	130 K	00000000		-----
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---						
Help		Retur		Confi	Dump	Backp Forwp Messa BP Go Recal

Meaning of the information by column heading:

Type

This field can be used to search for a specific type of level 1 object or restrict the display to certain types of objects. The following values are valid for this field:

Equ	Equates
Lmod	Load modules
Memb	Members

D/Csect

When displaying symbols that have been built as a result of the reading of Testran output from the assembler, there will be one or more sections defined. You can select which section is to be displayed by entering the name in this field. The third example in this section shows an example of selecting the DCOMREG DSECT built from a member called CCOMBLKS.

Label

This field enables the user to specify criteria which determine the first symbol in a list to be displayed. You can specify an absolute name, which must exist to be displayed, or a generic string, by entering a prefix following by an asterisk. For example, if you enter "ABCDE", the label ABCDE must exist, but if you enter "ABCDE*" in this field, the first symbol found starting with the letters ABCDE are the first to be displayed.

Equ/Lmod/Memb

Here you can enter the name of the level 1 symbol with which you want to work. The display depends on what sort of symbol is first found. For a more restrictive search, you can specify a type in the TYPE field. For example, if an Equate exists for ABCDE and ABCDE is also a load module which has been processed via the LMOD command, two level one symbols will exist. The first may be the Equate and the second the Load module. If this is entered with no type, a list of equates is displayed starting with ABCDE. However, if it is entered with the type LMOD, the level two items for the load module ABCDE are displayed as in the example for LMOD COMPLETE in this section.

Scope

This indicates the scope of the first symbol displayed on the screen. Possible values:

G	Global symbol
L	Local symbol.

This is purely for information and cannot be changed.

Symbol

This column contains the name of the symbol for which information is being displayed on the same line.

Mult

This is the multiplication factor. For non-level 3 symbols, it is meaningless and is set to 1. For level 3 symbols, this contains the multiplication factor for the storage type being displayed. An example of where this is meaningful can be seen on the display for the DSECT DCOMREG.

Type

This describes the type of symbol being displayed. The following are the possible mneumonics and what they indicate.

MEM	Member name read using the READ command.
LMOD	Load module read using the LMODULE command.
EQU	Equate defined explicitly or implicitly.
USNG	Using statement defined by the USING command.
CSCT	Code Section (CSECT).
DSCT	Data Section (DSECT).
COMN	Common Section.
INST	Program instruction.
CCW	Channel Command Word.
CHAR	Character data.
DBCS	Double Byte Character Set data.
HEX	Hexadecimal data.
BIN	Binary data.
FW	Fullword data.
HW	Halfword data.
FLPS	Floating point (short) data.
FLPL	Floating point (long) data.
FLPE	Floating point (extended) data.
ADDR	Address type data.
Y	'Y' type data.
S	'S' type data.
VCON	V-Constant data.
PD	Packed decimal data.
ZD	Zoned decimal data.

Length

This shows the length that the data was defined with. For example, a load module's length is the total length of all CSECTs in the load module. When the length is greater than 9999 bytes this is represented in Kbytes, indicated by a K. If it is greater than 9999 KB, it will be represented in Mega Bytes and indicated by an M.

Disp

This shows the displacement from the base area. For a field in a DSECT or CSECT, it is the displacement from the section. For a module within a load module, it is the displacement from the first module.

Meaning of the information according to column header:

Number

This is the number assigned to the breakpoint when it is defined with the AT command. Each defined breakpoint is allocated a sequential number which is unique within that particular UDEBUG session. If the breakpoint is an implicit breakpoint, this field shows "0".

ID

This is the ID of the breakpoint as specified, or defaulted on the AT command. For implicit breakpoints, this ID indicates the purpose of the breakpoint.

Status

This field indicates the status of the breakpoint as follows:

Reset	The breakpoint has been defined but is not currently set. This occurs when breakpoints are to be set in thread storage as Com-plete must first rollin the test session to physically set the breakpoint.
Set	The breakpoint has been defined and the UDEBUG SVC has been set in the appropriate place.
Active	The test program is currently sitting on the breakpoint.
Dormant	The breakpoint has been defined. However, due to the termination of the test program, it could not be automatically set again. To reactivate the breakpoint, simply issue an AT command for the same breakpoint.
Deleted	The breakpoint has been deleted logically. However, it could not be physically deleted. This can occur in a number of cases, for example, when the test program is sitting on the breakpoint. In this case, the breakpoint is physically deleted when the test program is next activated. In other cases, physical deletion will take place as soon as is possible.
Invalid	The breakpoint has been defined. However, when it was defined it was not possible to determine that it was valid. This occurs for breakpoints set within the test program's thread. A breakpoint is marked as invalid if it is defined as an offset from a program and the program is not that large, or if it is defined in a storage area in thread and the storage area has not been acquired by the test program.

Module

This is the name of the module within which the breakpoint is set or is to be set when it can be determined. When the string "\$THREAD" appears in this column, it indicates that the breakpoint has been requested in thread at an offset from the start of the thread. When the breakpoint is set, if it exists within a module, the breakpoint will be updated to

reflect the module name which will then be seen here.

Program

Where module is made up of one or more programs, the LMODULE command can be issued to build symbols for the various CSECTs contained in the module. When this is available, the name of the CSECT in the module where the breakpoint is set is displayed here.

Offset

This is the offset from the module or program where the breakpoint is set. If a program name exists, it is the offset from the program. If only a Module name exists, it is the offset from the module. If the module name is '\$THREAD\$', it is the offset from the start of the user program area in thread.

Exec'd

This is the number of times that the breakpoint has been executed.

Maxexec

This is the maximum number of times that the breakpoint should be executed. After this limit is reached, the breakpoint no longer causes the test program to give control to the debugger.

Preexec

This is the number of times a breakpoint should be executed without giving control to the debugger. After it has executed this number of times, the debugger receives control.

UDEBUG Commands

Entering Commands

More than one command can be entered at a time by using the command delimiter (default is semi-colon ";"). Each operand for a command must be separated by a blank or the operand delimiter (default is comma ","). The commands are processed in turn from left to right. As PF keys and profiles are passed through the same command processor, this applies equally when defining them. All commands and operands can be abbreviated to a short form that makes them unique. In the following descriptions, the upper case characters indicate the minimum which must be entered to uniquely identify a command or option.

Recalling Commands

Each command is entered is stored in a UDEBUG buffer. This means that the commands can be recalled using the UDEBUG command RECALL. When first entered, the RECALL command redisplay the last command entered. If entered again immediately, it displays the command entered previous to that and so on. The location pointer for RECALL processing is reset as soon as a command other than RECALL is entered. A maximum of 30 commands are buffered, the oldest is lost when the buffer fills. Please note also that this processing also applies to PF keys and Profiles.

Command Description

In the following description of the commands, the shaded command format indicates the shortest possible abbreviation of the command keyword. Keywords typed in capitals must be entered as is. Parameter keywords in italics must be substituted with valid values.

AT - Set a Breakpoint

This command enables you to explicitly set a breakpoint. Before this command can be issued, a valid Test Terminal and Level must have been set.

AT *offset/addr* *program* *bpid* *execute-no* *ignore-n0*

Where:

offset/addr	Determines where the breakpoint will be set. This expression is first checked to determine if it is within a residentpage program. If so, any program name entered is ignored. If it is not within a residentpage program, it is taken to be an offset into breakpoint program name.
program	If an offset is provided for the breakpoint name, a program name is required from which this offset is taken. If program is not supplied, or the default character is specified, a previously set Test Program is used. If this is not set, then the breakpoint cannot be set and an error message is issued. If a program name is established, the program must be cataloged to Com-plete.
bpid	Optional. This is an eight character ID which you can supply to identify the breakpoint. This is displayed whenever the breakpoint is referenced in any way. If no ID is specified, the breakpoint ID defaults to "BPnnnnnn", where nnnnnn is the zoned representation of the internal breakpoint number.
execute-no	Optional. The number of times the breakpoint is to be executed, after which time it will become dormant (defined but ignored whenever hit). This must be a number less than x'7FFFFFFF'. If it is not specified, it will default to x'7FFFFFFF', which in practice means that it is executed every time it is hit.
ignore-no	Optional. The number of times the breakpoint is to be ignored before being taken. This must be a number less than x'7FFFFFFF'. If this is not specified, the breakpoint is taken the first time that it is hit.

Examples:

1. AT 0 UCTRL

This causes a breakpoint to occur when the program UCTRL is started on the test level.

2. AT RESPGM+50 * RESBP

This adds a breakpoint with an ID of "RESBP" which is hit when a program on the test level hits the instruction at offset x50 into the resident program "RESPGM".

Notes:

1. When an error in the parameter value occurs, the breakpoint is NOT set, even if the parameter is an optional one.
2. If an offset/program combination is provided, it cannot be ensured at the time it is defined that the offset is valid. This can only be established when the program is loaded for the Test Session ,when it can be verified if firstly the module length can contain the offset and secondly that a valid instruction exists at this location. If it is found to be invalid, the breakpoint entry is flagged as having an invalid offset, which will be seen if the breakpoint is displayed.
3. If a new address and/or amode is specified, the amode in which the user program is to be dispatched must be consistent with the restart address. For example, a 31 bit mode address cannot be specified if the restart amode is specified or defaults to 24 bit mode.
4. The amode parameter only applies to operating systems that are capable of running in 31 bit mode.
5. Breakpoints can only be deleted by the Terminal/Level combination which added them.
6. When a UDEBUG session terminates either normally or abnormally, all breakpoints set by that session are removed.
7. If the EOJ command is contained in a profile, it will cause immediate termination of the execution of the profile AND of your session.
8. Global symbols that need to be defined each time Com-plete comes up can be defined using this functionality. If UDEBUG is started as a Com-plete STARTUPPGM, it will have a user ID of SYSUSR. If a profile name of SYSUSR exists on the COMDBPRF defined dataset, this will be executed when the program is attached. To avoid an abend from this task when it finishes, the EOJ command must be the last command in the profile.
9. When the TESTSTRAN outout is created in a PDS, the assembler generally punches out the module text records following them. For this reason, the READ command simply reads the member until it finds the first non-TESTSTRAN record when it finishes. If the command finishes correctly the number of TESTSTRAN records read will be indicated in the message.
10. Global symbols can be defined at startup using the SYSUSR profile as explained for the PROFILE command.

BACKPAGE - Page Backward on the Current Screen

On screens where information can be scrolled, this command causes the display to scroll backward one logical page. On screens where scrolling is not necessary, it has no effect.

BA

Note:

This command is usually assigned to PF7.

BP - Show Breakpoint Information

This command causes the breakpoint information screen to become the current screen. If the user program is not on a breakpoint, an error message is issued and the command rejected.

BP

Note:

This command is usually assigned to PF10.

BPLIST - Give a Breakpoint List

This command causes the Breakpoint List screen to become the current screen.

BPL

C - Relocate the Top of Screen Address (24 bit mode)

This command causes UDEBUG to take the fullword pointed to by the top address on the current screen, clear the high order byte and make this address the new top address.

c

CONFIRM - Confirm a Previously Entered Update

When an update is made in full screen mode, and confirm processing is active, the screen processor highlights and protects the changed fields, and requests that you confirm your changes. This command indicates that you wish the changes to take place. If this command is NOT the next command issued after such an update request, UDEBUG clears any record of the update request.

co

DELETE - Delete Symbols

With this command, you can delete a symbol defined using the EQUATE, LMODULE or READ UDEBUG commands.

DE type name g|l

Where:

type	Is Equate, Lmodule or Member depending on what is to be deleted. Note that when Lmodule or Member is specified, all symbols created for that symbol will also be deleted.
name	Is the name of the Equate, Lmodule or Member which you wish to delete.
g l	Optional. Specify either "g" for global delete of the symbol, or "l" for local delete. Local is the default.

Example

1. DE Equate MYPOINT

This command causes the equate MYPOINT to be deleted.

2. DE Lmodule MYLOAD

This command causes the load module MYLOAD and all CSECT symbols relating to this load module to be deleted.

DUMP - Dump Storage

This causes the dump storage display to become the current screen. If no parameter is used, the previous top address from the last DUMP command or DUMP processing is used.

DU expression

Where:

expression	is an expression determining the new address to be used as the top of screen.
------------	---

Example

DU DCOMREG

This causes the data around the the address where COMREG is found to be displayed.

EOJ - Terminate the UDEBUG Session

This command causes the UDEBUG session to be terminated. All breakpoints owned by this TID/Level combination are either deleted or flagged for deletion.

EOJ

EQUATE - Define Private Symbols

With this command, you can define symbols within your private symbol table area.

EQ name expression length

Where:

name	Required. The name of the symbol. Maximum length is 8 characters.
expression	Optional. The expression indicating where the symbol should be equated to. If this is not specified, the address at the top of the screen address is taken by default.
length	Optional. The length of the symbol being defined. If this is not specified, the symbol length is zero.

Example

1. EQ MYCOM DCOMREG

This causes the symbol MYCOM to be defined with a pointer to the address described by DCOMREG which is the address of Com-plete's COMREG area.

2. EQ PSA 0

This causes the symbol PSA to be defined with a pointer to the address 0.

FORWPAGE - Page Forward on the Current Screen

On screens where information takes up more than one screen page, this command scrolls the display forward one logical page. On screens where scrolling is not necessary, it has no effect.

FOR

Note:

This command is usually assigned to PF8.

GO - Restart a Breakpointed User Program

When the program being debugged is sitting on a breakpoint, this command causes it to be restarted at the next instruction. A new restart address can also be supplied as a parameter to the GO command, as well as a new address mode for the user.

GO *expression amode*

Where:

expression	Optional. The expression indicating the address where the program to be debugged is to restart. If this is not specified, the program is restarted at the breakpointed instruction in the AMODE it was in when the breakpoint was hit.
amode	Optional. Possible options are 31 or 24 to force the program to be restarted in the appropriate addressing mode. If this is not specified, the AMODE the program was in when the breakpoint was taken is used.

Example

GO *+8 24

This causes execution to continue at the current address at the top left of the screen plus 8 bytes in 24 bit mode.

Notes:

1. If a new address and/or amode is specified, the amode in which the user program is to be dispatched must be consistent with the restart address. For example, a 31 bit mode address cannot be specified if the restart amode is specified or defaults

- to 24 bit mode.
2. The amode parameter only applies to operating systems that are capable of running in 31 bit mode.
 3. Breakpoints can only be deleted by the Terminal/Level combination which added them.
 4. When a UDEBUG session terminates either normally or abnormally, all breakpoints set by that session are removed.
 5. If the EOJ command is contained in a profile, it will cause immediate termination of the execution of the profile AND of your session.
 6. Global symbols that need to be defined each time Com-plete comes up can be defined using this functionality. If UDEBUG is started as a Com-plete STARTUPPGM, it will have a user ID of SYSUSR. If a profile name of SYSUSR exists on the COMDBPRF defined dataset, this will be executed when the program is attached. To avoid an abend from this task when it finishes, the EOJ command must be the last command in the profile.
 7. When the TESTRAN outout is created in a PDS, the assembler generally punches out the module text records following them. For this reason, the READ command simply reads the member until it finds the first non-TESTRAN record when it finishes. If the command finishes correctly the number of TESTRAN records read will be indicated in the message.
 8. Global symbols can be defined at startup using the SYSUSR profile as explained for the PROFILE command.

HELP or ? - Provide UDEBUG Help

This will cause the Com-plete help utility to be called to display the available UDEBUG help information.

HE?

Note:

This command is usually assigned to PF1.

LMODULE - Read a Load Module and Equate its CSECTS

Under MVS and FACOM, when a load module is created, a list of CSECTS contained in the load module are placed in formatted records at the start of the module with details of offsets and lengths. The LMODULE command causes UDEBUG to read these records and set up symbols, indicating the offset and length of the CSECT names. When the original module name is resolved either via an EQUATE or using the CSECTS within, the module will be also be addressible.

LM name dd scope

Where:

name	Required. The name of the load module to be read.				
dd	Optional. The name of DD which should be used to locate the appropriate module. If this is not specified, the default COMPLIB will be used.				
scope	Possible options: <table border="0"> <tr> <td>LOCAL</td> <td>the symbols will only be set up for the UDEBUG session issuing the READ command. When this session is terminated, the symbols are lost.</td> </tr> <tr> <td>GLOBAL</td> <td>the symbols defined as a result are available to all users of UDEBUG under that Com-plete, and remain for the lifetime of the Com-plete region.</td> </tr> </table>	LOCAL	the symbols will only be set up for the UDEBUG session issuing the READ command. When this session is terminated, the symbols are lost.	GLOBAL	the symbols defined as a result are available to all users of UDEBUG under that Com-plete, and remain for the lifetime of the Com-plete region.
LOCAL	the symbols will only be set up for the UDEBUG session issuing the READ command. When this session is terminated, the symbols are lost.				
GLOBAL	the symbols defined as a result are available to all users of UDEBUG under that Com-plete, and remain for the lifetime of the Com-plete region.				

Example

LM COMPLETE * GLOBAL

This will cause the Com-plete nucleus module COMPLETE to be read from the current default LOADDD (COMPLIB, unless you change it), and a CSECT equate set up for each CSECT found in the load module. The resultant symbols are available globally.

MESSAGES - Show Current List of Messages

This causes the UDEBUG messages screen to become the current screen. You can issue this command if there is more than one message to be displayed.

ME

Note:

This command is usually assigned to PF9.

OFF - Remove Breakpoints

This command deletes a breakpoint, or flags a breakpoint for deletion, indicating that deletion has been deferred for some reason. In the latter case, the system ensures that the breakpoint is physically deleted at the appropriate time.

OFF bpid|ALL

Where:

bpid	This is either the eight byte breakpoint ID for the breakpoint, or the internal numeric ID by which the breakpoint to be removed is known.
ALL	All breakpoints are to be removed.

Example

OFF MYBPID

This causes the breakpoint ID "MYBPID" to be deleted.

Notes:

1. Breakpoints can only be deleted by the Terminal/Level combination which added them.
2. When a UDEBUG session terminates either normally or abnormally, all breakpoints set by that session are removed.
3. If the EOJ command is contained in a profile, it will cause immediate termination of the execution of the profile AND of your session.
4. Global symbols that need to be defined each time Com-plete comes up can be defined using this functionality. If UDEBUG is started as a Com-plete STARTUPPGM, it will have a user ID of SYSUSR. If a profile name of SYSUSR exists on the COMDBPRF defined dataset, this will be executed when the program is attached. To avoid an abend from this task when it finishes, the EOJ command must be the last command in the profile.
5. When the TESTRAN outout is created in a PDS, the assembler generally punches out the module text records following them. For this reason, the READ command simply reads the member until it finds the first non-TESTRAN record when it finishes. If the command finishes correctly the number of TESTRAN records read will be indicated in the message.
6. Global symbols can be defined at startup using the SYSUSR profile as explained for the PROFILE command.

PROFILE - Execute a Profile

Profiles must be added as members of a PDS, and this PDS must be allocated to Com-plete with a specific DD name. Profile members can contain any valid UDEBUG command that can be entered on the UDEBUG command line. At UDEBUG startup, the PROFILE command is issued implicitly for you to enable you to customize the environment automatically.

PRO name dd

Where:

name	Optional. The name of the profile to be executed. This must exist as a PDS member in the dataset pointed to by the applicable DD name. If this is not specified, your user ID is used as the profile name.
dd	Optional. The name of DD which should be used to locate the appropriate profile member. If this is not specified, the default COMDBPRF is used.

Example

PROfile TESTPROF TESTDD

This causes each record in the member TESTPROF from the DD/DLBL name TESTDD to be read and passed in turn to the UDEBUG command processor.

Notes:

1. If the EOJ command is contained in a profile, it will cause immediate termination of the execution of the profile AND of your session.
2. Global symbols that need to be defined each time Com-plete comes up can be defined using this functionality. If UDEBUG is started as a Com-plete STARTUPPGM, it will have a user ID of SYSUSR. If a profile name of SYSUSR exists on the COMDBPRF defined dataset, this will be executed when the program is attached. To avoid an abend from this task when it finishes, the EOJ command must be the last command in the profile.
3. When the TESTSTRAN outout is created in a PDS, the assembler generally punches out the module text records following them. For this reason, the READ command simply reads the member until it finds the first non-TESTSTRAN record when it finishes. If the command finishes correctly the number of TESTSTRAN records read will be indicated in the message.
4. Global symbols can be defined at startup using the SYSUSR profile as explained for the PROFILE command.

READ - Read TESTSTRAN Symbols

The various assemblers can produce what is called TESTSTRAN output for the module they are assembling when the TEST parameter is specified. TESTSTRAN records contain the details of all CSECTs and DSECTs in the assembled module. UDEBUG can read these from the applicable DD name and build tables containing the applicable information. In this way, DSECTs can be displayed online and it is hoped to be able to disassemble a module to provide the most readable output possible.

REA name dd scope

Where:

name	Required. The name of the member to be read containing the TESTSTRAN records.				
dd	Optional. The name of DD which should be used to locate the appropriate member. If this is not specified, the default COMDBTXT is used.				
scope	Possible options: <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; padding: 5px;">LOCAL</td> <td style="padding: 5px;">the symbols will only be set up for the UDEBUG session issuing the READ command. When this session is terminated, the symbols are lost.</td> </tr> <tr> <td style="padding: 5px;">GLOBAL</td> <td style="padding: 5px;">the symbols defined as a result are available to all users of UDEBUG under that Com-plete, and remain for the lifetime of the Com-plete region.</td> </tr> </table>	LOCAL	the symbols will only be set up for the UDEBUG session issuing the READ command. When this session is terminated, the symbols are lost.	GLOBAL	the symbols defined as a result are available to all users of UDEBUG under that Com-plete, and remain for the lifetime of the Com-plete region.
LOCAL	the symbols will only be set up for the UDEBUG session issuing the READ command. When this session is terminated, the symbols are lost.				
GLOBAL	the symbols defined as a result are available to all users of UDEBUG under that Com-plete, and remain for the lifetime of the Com-plete region.				

Example

MYSYMS * GLOBAL

This causes the member MYSYMS to be read from the current default TEXTDD (COMDBTXT unless you change it) and the Testran records interpreted from that member. The resultant symbols are available globally.

Notes:

1. When the TESTRAN outout is created in a PDS, the assembler generally punches out the module text records following them. For this reason, the READ command simply reads the member until it finds the first non-TESTRAN record when it finishes. If the command finishes correctly the number of TESTRAN records read will be indicated in the message.
2. Global symbols can be defined at startup using the SYSUSR profile as explained for the PROFILE command.

RECALL - Display the Command Entered Last on the Command Line

This command causes the command entered last to be redisplayed on the command line. When entered repeatedly without intervening commands, it will cause UDEBUG to progressively display the previous commands to the last command displayed.

REC

Note:

This command is usually assigned to PF12.

RELOC - Set the Relocation Base Address

During dump and disassembly processing, you can see the actual address of displayed data, together with a relative address. By default these are equal. However, this command enables you to set the base address against which the relative address will be calculated.

Note:

This is only effective when on the DUMP storage screen or the DISASSEMBLY screen. Also, a separate relocation factor can be set for the DISASSEMBLY screen without effecting the DUMP screen and vice versa.

REL *expression*

Where:

expression	Optional. The expression indicating the address to which the current address should be made relative. If this is not specified, the top address for the screen is taken by default.
------------	---

Example

RELOC DCOMREG

This causes all relative address to be displayed/used relative to Com-plete's COMREG address.

SESSINFO - Show the Current Session Information

This causes the UDEBUG session information screen to be displayed.

SES

SET - Set Various UDEBUG Options

The SET command enables you to set options and various character and default values for the UDEBUG session. This facility is provided primarily to enable you to customize your UDEBUG session via your Profile.

SET option name ON|OFF value

Where:

option	The option to be turned on or off. Valid options are:
	MTRACE MCALL tracing
	ITRACE Instruction tracing
	BUMP bumping of screens when ENTER is pressed without data being entered
	CONFIRM Confirm processing

name	The name of a default character or value which is to be changed. The characters and values that can be changed are:	
	PROFDD	Default profile DD/DLBL name for PROFILE command.
	TEXTDD	Default text DD/DLBL name for READcommand.
	LOADDD	Default load DD/DLBL name for LMODULE command.
	DEFCHAR	Character to indicate default for positional operands for a command.
	NOTRES	Character to indicate unresolved storage.
	NOTALLOC	Character to indicate unallocated storage.
	NOACCESS	Character to indicate unavailable storage.
	DECCHAR	Character to indicate decimal values.
	HEXCHAR	Character to indicate hexadecimal values.
	RELCHAR	Character to indicate relative values.
	TADDRID	Character to represent top address on screen.
	CMDDEL	Character to delimit commands in a line.
	OPDEL	Character to delimit operands in a command.
	PARMDEL	Character to delimit parameters in an operand.
	PNTLDEL	Pointer indicator left delimiter.
PNTRDEL	Pointer indicator right delimiter.	
ADDCHAR	Character to indicate addition.	
SUBCHAR	Character to indicate subtraction.	
ON/OFF	Indicates if the specified option is to be turned on or off. ON is the default. If ON is specified for a character or value, the default is set. OFF is invalid for default values. However, for character defaults, OFF causes the character not to be used.	
value	The character or string to which the character or value should be set.	

Examples

SET ITRACE	Sets instruction tracing on
SET ITRACE OFF	Sets instruction tracing on
SET DECCHAR Y	Sets the character to identify decimal to "Y"
SET TEXTDD	Sets the default Text DD/DLBL name to the default (COMDBTXT).

SYMBOLS - Show the Symbol Display Screen

This causes the UDEBUG symbol display/list screen to become the current screen.

SY

X - Relocate the Top of Screen Address (31 bit mode)

This command causes UDEBUG to take the fullword pointed to by the top address on the current screen, clear the high order bit and make this address the new top address.

x