colesoft™

ZXDC®

THE REAL QUICK REFERENCE

2024 Edition Including Z/XDC, c/XDC and Dump/XDC)

Bob Shimizu

ZXDC®

PREFACE
PROPRIETARY LEGEND

z/XDC® and its documentation (collectively, "Product"), including copies thereof, are the property of Mainchord, LLC DBA Colesoft ("Owner"). Use of the product is licensed from ColeSoft Marketing, Inc. ("Licensor").

The Product may be used only by those organizations that are licensed by Licensor for such use and only in the manner so licensed. The Product may not be published, reproduced, distributed or made available to third parties for any purpose without the expressed written permission of Owner or Licensor. However, a reasonable number of copies may be made of the documentation (including the copyright notices thereon) as is necessary for the legitimate use of the Product within a licensed organization ("Customer").

Except as may be otherwise expressed in a signed agreement between Licensor and Customer, Owner and Licensor make no representations or warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, the warranty of freedom from rightful claims by way of infringement and the like, and any warranty as to accuracy.

WARNING! z/XDC® is a powerful tool for dynamically locating and correcting malfunctions in actively executing user programs and operating system routines. Accordingly, it is inherent in its design, that unless the use of this Product is properly controlled, then under certain conditions a malicious or careless user can use the Product to alter, subvert, counterfeit, damage or otherwise disturb the normal execution of user programs or system routines including, under certain conditions, both its own and system security routines. Except as may be otherwise expressed in a signed agreement between Licensor and Customer, Owner and Licensor make no representations or warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, the warranty of freedom from rightful claims by way of infringement and the like, and any warranty as to accuracy.

Therefore, even if advised of the possibility of loss or damages, under no circumstances shall Owner or Licensor be liable for any loss or damage whatsoever (including death) arising from the Product, whether such loss or damage be direct, indirect, consequential, special or otherwise. Further, neither Owner nor Licensor shall be obligated to indemnify in any manner against any person or organization for any loss of any kind or nature which the person or organization may experience, arising out of the use or misuse of the Product.

TRADEMARKS

TFS™, XDC-TFS™, CDF™, XDC-CDF™, FASM™, base/XDC™, c/XDC™, asm/XDC™ and **dump/XDC™** are trademarks of Mainchord, LLC.

Extended Debugging Controller®, XDC®, and z/XDC® are registered trademarks of Mainchord,LLC.

Other brand and product names referenced in this document are trademarks or registered trademarks of their various holders. Use of their names herein is for identification purposes only.

Contents

Chapter 1 - Welcome to Cole Software

What are z/XDC, c/XDC and dump/XDC?

Cole Software's products are designed to help people who must work in complex, intricate solutions that they either support for their in-house user communities, or provide to their downstream customers.

z/XDC Allows you to work with object code, whether or not it comes with ADATA (Assembler source) maps or not. You can literally go anywhere/do anything with z/XDC and its companion products.

c/XDC Allows you to do the same range of functions, but in a C-centric environment. While you're looking at your C DWARF data (your C source code), ALL of z/XDC's power remains available, under the covers.

dump/XDC Extends z/XDC and c/XDX into the world of dump-reading. Again, you can see your Assembler ADATA or C DWARF in this environment. ALL of z/XDC is STILL available, except that you can't set breakpoints or trace the code.

Here is what our tools can do, in a single sentence: *Using Cole Software's debugging technology, a knowledgeable user may more easily find and fix "bugs" in arbitrary object code within any csect, within any module (or compilation unit) running under any Request Block within any Task (or Service Request Block) within any address space on the system, including much of the internals of the z/OS operating system.*

That's range, and it's why z/XDC has helped generations of people to get more done in less time with less frustrating effort.

Our tools exist in the space between your outer level of comprehension, and what's REALLY going on in the system. Ideally, we want that gap to be as small as possible. We think our tools can help you in that effort.

How To Contact Our Support Structure

Cole Software keeps fairly normal business hours, but it's not unusual for one of us to be up and monitoring our emails many hours of the night and on holidays. So, we're generally available if you're up and working.

First up: The most authoritative reference on our products is contained in Dave Cole's Help Subsystem. This subsystem comes with elaborate references and hyperlinks to help you learn about z/XDC at a very granular level.

To access the online Help Subsystem, start a z/XDC session. From a TSO READY prompt, issue "XDCCALL IEFBR14". Then enter HELP on the command line. You're in.

Our website contains a lot of useful, self-help information including the entire body of our documentation (such as the User's Guide, the Installation Guide and the Commands reference), all in .pdf format. However the "formal" body of Cole Software's documentation is merely a compendium of all those blasted ISPF panels from the online Help Subsystem, which is decidedly non-linear to read. The good news here is that you can use Acrobat's search function to drill down to what you need. Again, there isn't a better reference than the one Dave has written.

Telephone support: You are welcome to reach us directly at our Charlottesville, VA USA headquarter. The telephone number is (540) 456-8210. Mine (in Prescott, AZ) is (800) 932-5150, or (928) 771-2003.

Email support: Our engineering guys monitor **support@colesoft.com**. That's the best way to get to the smart guys.

I can be reached at **bob@colesoft.com**. I can offer first-level support, within my limited range of capabilities.

I am no engineer. I'm capable of gross simplification. I have less than optimal levels of understanding. Sometimes I'm just plain WRONG. Dave Cole has genially put up with me, because he knows what I'm trying to do: make the tools easier to approach and use. He'll fix that on the back end, whenever you have questions.

Why This book?

To function as a senior software engineer in z/OS requires that you be a “watch-maker”, able to embrace deep complexity. Dave and his team are such guys, as are many of our senior customers. The world needs these guys. Perhaps you are, or will become a watch-maker. You’ll have a good job for life.

In order to survive in the z/OS environment our users need the intricacy and complexity that our products afford. There just isn’t anything else out there that comes close.

This, however, can bewilder the new or casual user. Not all of us are watch-makers yet, but in time perhaps you will become one. Our products can greatly help the new user to more quickly grasp z/OS internals.

So, our tools help people at all levels of this game, but HOW can we make using z/XDC simpler and easier to approach? That’s where I come in.

Since 1988 it’s been my mission to make z/XDC and it’s companion products easy to comprehend and begin to use. It’s why I have become your product educator and the author of this and other books on our products.

Blanket statement: Dave Cole will send me anywhere in the world to offer in-person (or virtual) product education *at his cost.* Keep that in mind.

And that’s why this book. After completing the Railroad Track Reference, I am presenting you with the REAL Quick Reference. I’ve tried to include ONLY what you need and nothing that you don’t.

Know this: I have inserted the railroad track diagrams of the most important XDC commands in this book. Don’t be put off with their complexity. The outer ranges of our commands are for watch-makers. The rest of us can get along quite well with the simplest parameters for our commands.

Example: You don’t need all the twiddly bits of the FORMAT command to see your code. Merely issue “FORMAT <address-expression> (or “F” for that matter). Simpler yet, merely issue Where. Bang. You’re right there.

What Is An Address-Expression?

z/XDC makes heavy use of address expressions within the command set. What use would the “hammer” be to me if I didn’t know to hit THAT NAIL? In this case the nail head IS our address expression. The better you get with concocting address expressions, the better you will with our tool.

VERY basically, address expressions are composed of bases, operators and offsets, just as in your Assembler class or textbook.

A base can be the ILC portion of the Program Status Word (the next instruction to execute), a virtual address, a Register value, a module name or csect name, the contents of a control block (or where it points to), an Equate that we create in storage dynamically or one that Dave has created for you.

An operator can be plus (“+”) or minus (“-”) some offset from the base. You can also use indirect operators: “%” for 24-bit addressing, “?” for 31-bit addressing or “!” for 64-bit addressing.

An offset is merely a hexadecimal number.

So, putting all these components together.....

The DISPLAY command produces a “dump” of storage at some address expression. The minimum abbreviation of DISPLAY is “D”. If I were to enter “D 0” (D-space-zero) I’d get a dump starting a location zero in storage. Right?

Likewise the FORMAT command forces a disassembly, either of pure object code or ADATA or DWARF (if available to me). The minimum abbreviation of FORMAT is “F”. So, if I issue “F PSW!” then I will see z/XDC disassembler with the PSW’s ILC points to, with 64-bit addressing.

“D RW14!” will treat the contents of the 64-bit General Register 14 as a pointer into storage, and will give me a “dump” of storage there. But you COULD issue “D RW14!!” and see the SECOND iteration of pointers hung off of Register 14.

I give a decent treatment of address expressions in my classes and in the Primers. Dave's much better explanation can be find at Help ADdressing (the caps indicate the minimum abbreviation).

For advanced Addressing concepts please visit Help ADdressing Functions.

Using their advanced knowledge of z/OS architecture, Dave and his team come up with some very interesting address expressions to precisely identify and exploit known places in z/OS. I don't pretend to understand what they mean, but usually our customers say, "AHA!" and go off on their own, armed with their cool new address expression.

Enough of the preamble. Let's get to it. This book is organized by function, not by alphabetic order. So please feel free to dive into this booklet, read about an area of inquiry and get right to work.

This booklet is organized into short chapters, thusly:

- **The REALLY SHORT list of z/XDC Commands**
- **The Shortcut Commands**
- **The Point and Shoot Commands**
- **Administrative** (Beginning and ending a session)
- **Seeing your local environment** (Display, Format, List, Map, DMap, Using, Find and Scanlog).
- **Trapping and Tracing** (How to stop programs and how to follow their execution)
- **Changing Things** (Changing your program or its environment)
- **Working in other Address Spaces** (Using Foreign Address Space Mode or Cross Domain Facility to debug batch jobs)
- **Help** (Just how does this blasted tool work?)
- **Wrap-up**

Again, don't be dismayed by the extensive railroad track diagrams for each command. Each command can be used with the simplest of parameters - usually an address-expression. You probably don't need the advanced parameters - yet.

If you understand the advanced parameters already, you don't need THIS book. Congratulations: You're already a watch-maker.

The REALLY SHORT list of z/XDC commands

Here are the commands that you can't live without, grouped by function.

Administrative:

XDCCALL	Begin a z/XDC session.
END	Terminate a z/XDC session.
PROfile	Fiddle with your session characteristics (PFkey settings, z/XDC's default actions and MUCH MORE).
Set	Influence how z/XDC interprets your environment.

Seeing your code and local environment:

Where	Display the current executable statement or instruction within your program.
Equate	Create equates at any place in storage. Equates can map a singled address, or may span storage.
Format <addr-expr>	Dis-assemble anywhere.
Display <addr-expr>	Obtain a "dump" of memory wherever you like.
List Regs	See the registers. You may see ALL types of registers including control registers.
List PSw Format	Decode the Program Status Word
List TAsks	See the Task Structure in the address space.
List RBs	See the Request Block structure for any task.
List Subpools	See the organization of memory
List PGms	See all the programs running under any task.
List Variables	Examine variables in the C programming environment
List Vstack	Display the current pool of Global, Local and subroutine variables (for c/XDC).
FInd	Search memory.
Set MapLibs	Prepare to load ADATA source listings for your program (somewhat obsoleted by AUTOMAP).
Map <addr-expr>	Load maps for your program. We support ADATA and DWARF (for C).
Set Qualifier <addr-expr>	Establish a "default" module.csect.
SCanlog	After the appropriate Format command, search for text strings in your source listing.
DMap	Load dsect maps for your program (or a control block).
Using	Establish an origin address for a dsect.

Trapping and Tracing:

AT <addr-expr>	Create a permanent breakpoint.
AD <module.csect+offset>	Set a deferred breakpoint in a module that isn't loaded yet.
T	Trace a single instruction step.
T BY	Trace to the next successful branch statement.
T BNC	Trace within the current program. Do NOT follow branches into subroutines.
HOok <addr-expr>	Create a z/XDC session wherever you like! Think of Hook as the Super-breakpoint.
STEP	Follow the execution of a C program.
Off	Delete a breakpoint.
GO	Let your program execute to the next breakpoint, or abend.

Changing Things:

REAd <dsn>	Run a z/XDC "Script" of commands.
COPY	Move the contents of storage elsewhere.
Zap	Overwrite any program or storage location from the command line or as a shortcut command.
GEtmain	Obtain a "chunk" of storage from any subpool for any amount of space.
LOad <module>	Load a copy of a module into storage.
POst (addr-expr>	Post an Event Control Block.
Jump	The "J" line command. Force the PSW to point wherever you like.
Set PSw	Alter the contents of the Program Status Word.

Other Address spaces:

Set ASid <jobname>	Switch to another address space and look around.
Set ASid Home	Switch back to your home address space.
Cross Domain Facility	Not a command. "CDF" is how you work on a batch job. See the section!
DISC;GO	How to terminate a CDF session without your TSO session locking up.

The Help Subsystem:

Help Debugging	Learn how to deploy z/XDC in different situations.
Help COmmands <name>	Research the particulars of any z/XDC command.
Help Scripts	See the READ command scripts we've provided in the product.
Help SUpport	Contact us when you get "stuck".

The Shortcut commands

z/XDC's command language is subject to abbreviation wherever possible, perhaps because we at Cole Software are such poor typists! We also want to save time in our work. z/XDC's displays often show an underscore character out to the left of each line in Column one of the terminal screen. You can use this field to enter "shortcut commands". These commands will save you a lot of time.

If you are ever unsure of what Shortcut Command you can put in the underscored area, you can enter a question mark ("?") and z/XDC will read you the list of available commands, which can vary with context. Thus, you may receive a list of allowed commands that looks like this: "A CU D E F H I J K L LO M N O P S SH T W X Z ? * | \ /". That's clear as mud, isn't it? Here is a table of the Line Commands and what they can do for you:

A	Sets a permanent breakpoint at "this" line in the program by way of the "AT" command.
CU	In dump/XDC (only), this forces the relocation of the Current eXecution Unit.
D	Performs a context-sensitive DISPLAY command. Performs an EWHERE command (displays the user's Error Level Abend Address).
E	Performs an EWHERE command (displays the user's Error Level Abend Address).
F	Performs a FORMAT command, disassembling object code. With preparation can incorporate ADATA or DWARF data into the resulting display. A classic z/XDC command.
H	When put to the right of any z/XDC error message "H" will drill right into the Help Subsytem and will display the zXDC message in its entirety.
I	Performs a LIST LOCATION command. Not particularly useful (Hey, I don't write this product!).
J	Performs a "jump" operation, forcing the Instruction Length Counter portion of the PSW wherever you want it.
K	Sets a dynamic HOOK at this point. Creates a debugging session right here/right now regardless of any other recovery environment. Think of Hook as the "Super-breakpoint".
L	Performs a LIST command. This is context-sensitive. For example, if used in the output of a LIST TASKS display, "L" will issue LIST RBS for that sub-task. Play around with it.
LO	Performs a "LIST OPERANDS" command, showing you the operands of this instruction or statement at THIS point in time. The further away from the PSW you get, the less accurate this command will be.
M	Performs a mapping operation, retrieving ADATA or DWARF data and incorporates it into the display after the next FORMAT command.
O	Deletes a breakpoint definition.
P	Purges a previously saved note. Can only be used in the LIST NOTES command display.
Q	Issues a "SET QUALIFIER" command, establishing a default module and csect. Thereafter you may refer to labels in the code as ".label-name". The leading dot is significant.

S	Performs a Select operation in the Help and Profile Subsystems, or performs a “SWITCH” command in the LIST Tasks display.
SH	Displays one line of storage at any place. Best used in a display, or “watch” window so as to save vertical space in your display.
T	Sets a transient breakpoint “here”. The breakpoint will stop execution after a GO command (from the command line) and the transient breakpoint will be deleted.
W	Performs a WHERE command, showing you precisely WHERE your code is executing.
X	Enables, or disables a breakpoint definition. Turns the breakpoint on or off “for now” until you change your mind.
Z	Enter this, tab over from column one, and wherever the cursor stops, you have an opportunity to ZAP storage, register displacements or opcodes.
?	Available all the time. Will show you what shortcut commands are available in the current context.
*	For c/XDC. Shows storage in EBCDIC format.
|	For c/XDC. Shows storage in ASCII format.
\	For c/XDC. Show a variable in null-terminated format.
/	For c/XDC. Show a variable in length-terminated format.

Some of the shortcut commands are very useful. Others are for specialized environments (such as C) or aren’t particularly useful at all.

There you have it. Let’s move on.

The Point and Shoot Commands

z/XDC has a lot of point-and-shoot capabilities that will save you LOADS of time. These are context-sensitive displays that utilize the cursor, and the “%”, “?” and “!” characters. For general reference, please remember that:

- The percent character (“%”) denotes 24-bit mode;
- The question mark character (“?) denotes 31-bit mode;
- The exclamation mark character (“!”) denotes 64-bit mode;
- The D character will produce a DISPLAY (a dump).
- The F character will produce a formatted display. You get object code OR ADATA OR DWARF if z/XDC can find it in your search order.

Things to explore:

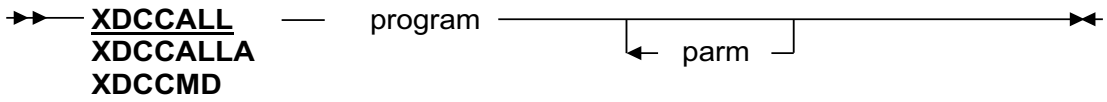
1. Put the cursor on any word of memory in the working or display window(s) that looks like code and press Enter. z/XDC will FORMAT (disassemble) memory at that point. Cool!

2. Put the cursor over any part of the display, enter a D-space and press Enter. The memory pointed to under the “D-space” is DISPLAYED for you in hex-EBCDIC format.
3. Put the cursor over any part of the display, enter an F-space and press Enter. The memory pointed to under the “F-space” is FORMATed for you.
4. One can merely type %, or ? or ! as well. z/XDC will establish whether you want a Display command or a Format command to be processed, based on whether the screen you’re working from resulted from a Display or Format command in the first place.
5. One can also use D or F together with %, ? and ! in any combination.
6. Put the cursor on any word of memory in a window, enter % and press Enter. z/XDC will display that word as a 24-bit pointer. This works in a register display as well.
7. Put the cursor on any word of memory in a window, enter ? and press Enter. z/XDC will display that word as a 31-bit pointer. This works in a register display as well.
8. Put the cursor on any word of memory in a window, enter ! and press Enter. z/XDC will display that word as a 64-bit pointer. This works in a register display as well. Be advised that in Foreign Address Space Mode, z/XDC will default to 24-bit mode because it cannot tell which PSW you may refer to. In this case, use D or F with a % or ? or ! as appropriate.
9. Put the cursor on the first digit of a machine instruction’s base/displacement field, type in a %, a ? or a ! and press Enter. The memory used by the instruction is interpreted according to the width specified (24-, 31-, or 64-bits).

For a REAL time-saver, try these point-and-shoot commands on registers! Hey, that’s neat!

You can also use “D_“ and “F_“ anywhere on the screen as well as on the one-byte area at the left of the display (“D-space” and “F-space”). If you put the cursor on any interesting area of storage (or a register in the lower display window), and type “D_“ or “F_“ then z/XDC will take the address over which you typed, and issue a Display or Format command for that address. This makes following pointer chains much easier. Try it!

Chapter 2 - Administrative Commands



Invoke z/XDC from the TSO READY prompt, or in the Batch.

program The name of a module.
parm The parameters of the module, separated from one another by spaces.

Remarks: This is how you invoke z/XDC, either interactively or in batch mode.

If you wish to execute APF-authorized, then you use the XDCCALLA form of the command. In fact, many z/XDC users use "XDCCALLA IEFBR14" merely to gain APF-authorization for the purpose of investigating system routines, or code running in the Private Areas of other address spaces.

XDCCALL is also used to invoke z/XDC in batch programs. To do that, alter your EXEC statement from

"// EXEC PGM=myprogram, PARM='parm1 parm1 parm2 …"

to

// EXEC PGM=XDCCALL, PARM='my program parm1 parm2 …"

If you need to invoke your program APF-authorized Use the XDCCALLA command in your EXEC statement.

When invoked, XDCCALL does several things:

- XDCCALL(A) loads into a task in the address space (TSO Private Area or batch address space);
- It ATTACHes your program as a subtask.
- It specifies that XDCCALL is the ESTAI for your program;
- It uses the #DIE command to create a S0C1 abend at the entry point of your code;
- It writes a "DEAD" message to your TSO display, or to the Operator Console if in batch.

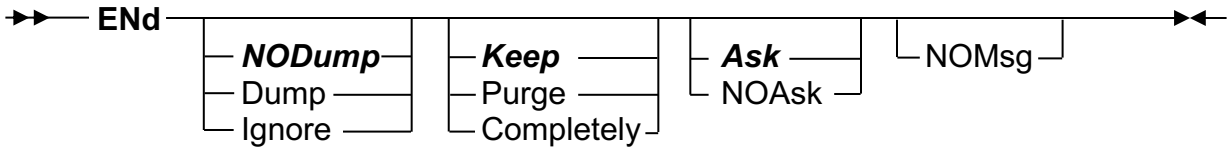
If in TSO, you can begin debugging. If in batch you'll use Option 3 from the z/XDC Startup Panel, and select your batch job (with an "S") to establish a connection to your batch job.

Use the XDCCMD alias of XDCCALL to establish a debugging session on a TSO Command processor. In this case, z/XDC passes parameters, if any, to the user's program in the format of a command buffer pointed to by a CPPL (pointed to by R1).

This command may not be used in Foreign Address Space Mode.

Examples: XDCCMD MYTSOCOMMAND ← the user wishes to work on a TSO command in the Foreground.
XDCCALLA IEFBR14 ← The user wishes to launch an APF-authorized z/XDC session for some cool purpose.

More Info: There is no reference for the XDCCALL "command" under HELP COMMANDS. Frank says, "That's because XDCCALL isn't a command. It's a callable module." So, please issue HELP XDCCALL for more information. See also Help Debugging Batchjobs.



Terminate the z/XDC session.

NODump	A default action. z/XDC will suppress the printing of a dump.
Dump	z/XDC will print a dump of the session.
Ignore	Prevents z/XDC from altering the dump flag in Recovery Termination Manager. A dump may or may not be printed depending upon how the dump flag has been affected by other processes.
Keep	A default action. Retains breakpoints, maps and equates. Useful if the user expects z/XDC to regain control under another task or ESTAE/I.
Purge	Removes all breakpoints, maps and equates before terminating ALL of the user's session(s). A synonym of "Completely".
Completely	Removes all breakpoints, maps and equates before terminating ALL of the users' session(s).
Ask	A default action. z/XDC will ask the user if he or she really wants to end the session.
NOAsk	Bypasses the termination prompt. z/XDC proceeds with termination of the session.
NOMsg	z/XDC purges all messages that may be pending and have not yet been displayed.

Remarks: ENd allows a fair amount of "tailoring" to meet specific situations. My personal favorite is "END COMPLETELY" ("z/XDC, go away. Don't prompt me. I know what I'm doing").

Examples: END ← Prior to ending the debugging session, z/XDC will prompt the user to confirm with "Y" ("I want to end this session.") or "N" ("NO! I do not want to end this session.").
END NOASK ← z/XDC ends the session without prompting the user.
END COMP ← z/XDC terminates ALL debugging sessions for this user.

More Info: Enter Help COmmands ENd.

PROfile

Access the Profile Facility.

This command accepts no parameters

Directly invoke the PROFILE facility's panel structure. This form of the PROFILE command accepts no parameters.

Remarks: A great deal of z/XDC's session and window characteristics can be controlled from the Profile Facility.

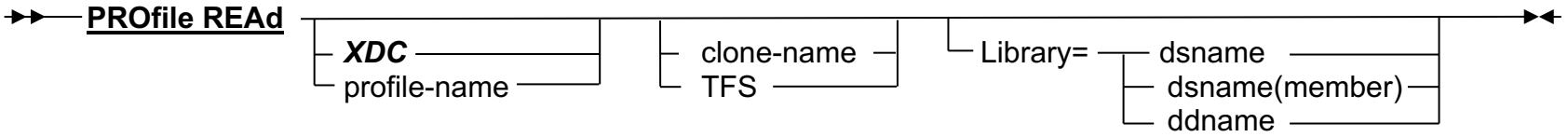
Behind the Profile facility, there are a huge number of SET commands that affect session characteristics. Profile allows you to review and change them symbolically, rather than having to know about the SET commands. It's a great time-saver.

Profiles control PFKey settings, screen colors, display window layouts, session logging and a great many other things. You can create a desired session layout and save this by name (via the Profile SAVE command). Later, you can recall this or any other profile by its unique name (via the PROfile REAd command). If desired, you may recall the "factory default" Profile at any time by issuing "PROfile RESet".

If you use c/XDC, then a very good profile to invoke is PROFILE RESET CWIDE (for wide terminal geometries), or PROFILE RESET C80 (for narrow terminal geometries).

You can have many named profiles, each of which is suitable for a particular debugging scenario (as many as will fit in the user's ISPF profile library). If the user wishes to use his or her favorite profiles when working on batch jobs via the Cross Domain Facility, then you must add a DD card (XDCPROF or ISPPROF) to the batch job's JCL that points to the ISPF profile library containing the desired profile(s).

More Info: Enter Help COmmands PROfile. See subsequent pages in this reference to learn about Profile's other parameters.



Retrieve a previously-saved profile for use in the debugging session.

XDC Default action. If no profile-name is invoked, then the default system profile is read.

profile-name The name of a previously saved profile, 1 to 5 characters in length. The profile name may consist of alpha, numeric or the national characters ("**$**", "**#**", and "**@**"). Why only five characters? That's because z/XDC appends the prefix "XDC" (or the current "clone name") to all of your profiles. If profile-name is omitted, then the user's current default profile is loaded.

clone-name z/XDC allows different versions of itself to have names other than "XDC". If you have created a clone of your z/XDC system (such as "X21" for Release z2.1, or "X1D" for Release z1.13) then you may invoke a profile that was defined under that clone-name.

TFS z/XDC writes profiles only to profile libraries (such as ISPPROF), but can **read** installation-specific default profiles from profile libraries **and** tables (such as ISPTLIB). Specifying "TFS" tells z/XDC to load profile-name from ISPTLIB.

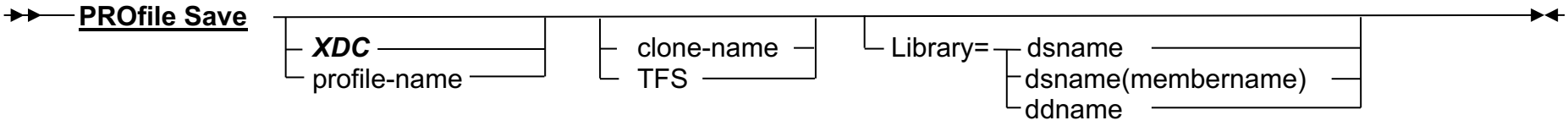
Library= If you want to load a profile from other than in the standard ISPC allocations, then you may specify one.

- **Dsname** Any FB-80 partitioned dataset.
- **Dsname (member)** Normally, profiles may be loaded only from library members named "xxxPNAME" (in your case, probably "XDCPNAME"). This allows for different "clone names" for z/XDC. However, using this parameter blows that away, and as long as the contents of (member) are z/XDC-generated Profiles, you are good to go. Because you have some compelling reason to do so, or you just prefer living a complex life.
- **DDname** If you have a profile library allocated to some arbitrary ddname, you can load your profile from this allocation instead of the dfailt (//XDCPROF, //ISPPFOR and //ISPTLIB).

Remarks: If you have created an ungainly mess on your screen, and you wish to get our "factory default" profile back, you may enter "PROfile RESet".

Examples: PRO REA TEST ←The user wishes to recall a previously-defined profile that is named "TEST1".
PROFILE REA FNOOB X1D ←The user wishes to recall profile "FNOOB" from the X1D clone of z/XDC .

More Info: Enter Help COmmands PROfile REAd. See also Help PRofiles Tlibprofiles. See also Help Commands LISt PROfile (which displays your current profile).



Store the current Profile for use in a later debugging session.

XDC If the word "XDC" or if this parameter is omitted, then current profile settings are saved as the user's default.

profile-name The name that you wish to assign to the profile, from 1 to 5 characters in length. The profile name may consist of alpha, numeric or the national characters ("**$**", "**#**", and "**@**").

clone-name z/XDC allows different versions of itself to have names other than "XDC". If you have created a clone of your z/XDC system (such as "Z21" for Release z2.1, or "X1D" for Release z1.13) then you may save a profile under that clone-name.

TFS z/XDC writes profiles only to profile libraries (such as ISPPROF), but can **read** installation-specific default profiles from profile libraries **and** tables (such as ISPTLIB). Specifying "TFS" tells z/XDC to load profile-name from ISPTLIB.

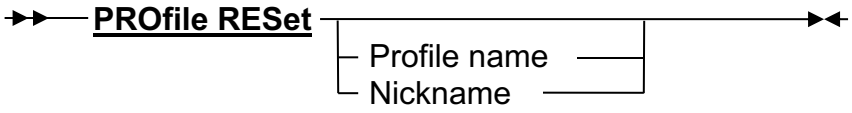
Library=

- **dsname** The name of a library other than the standard ISPPROF allocation.
- **dsname(membername)** The name of any desired member within dsname. When membername is specified, neither profilename or clonename may be given.
- **ddname** Cause the profile to be saved into any DD allocation other than the defaults (XDCPROF, ISPPROF or ISPTLIB).

Remarks: Profile is pretty cool. Imagine setting a hook or a breakpoint in a module and associating a PROFILE READ command with it. When the GO command executes the hook, or hits a breakpoint, your screen changes to the profile layout that is perfect for working on that part of your code.

Examples: **PRO S TEST1** ← The user wishes to save the existing session definitions under the name of "TEST1".
PROFILE SAVE FARFEL TFS ← The user wishes to save the existing session definitions under the name of "FARFEL" into ISPTLIB as a system default.

More Info: Enter Help COmmands PROfile Save. See also Help COmmands PROfile.



Summon the z/XDC "factory default" profile for use in the debugging session.

Remarks: z/XDC's "factory default" profiles are hard-coded within the product itself. In order to react to the proliferation of larger screen geometries and the advent of High Level Language support in z./XDC a number of new default profiles have been created.
If no name or nickname is provided, z/XDC will issue PROFILE RESET XDC, and give you the "factory default".

Profile name Can be any of the four names below:

A80	Intended for the Assembler environment with column width of 80 columns.
AWIDE	Intended for the Assembler environment with column width of 100 or more columns.
C80	Intended for the IBM C/C++/Metal C environment with column width of 80 columns.
CWIDE	Intended for the IBM C/C++/Metal C environment with column width of 100 or more columns.

Nickname Can be any of the four "nicknames" below:

ASM	z/XDC will configure the profile for Assembler using A80 or AWIDE depending on what it "senses".
C	z/XDC will configure the profile for C using C80 or CWIDE depending on what it "senses".
CEE	A synonym for nickname "C".
XDC/omitted	Pretty cool: z/XDC will configure for ASM or C depending on what it "senses".

More Info: Enter Help COmmands PROfile RESet. See also, the PROfile REAd command.

Issue a TSO command from within a z/XDC debugging session.

Command The desired TSO command
parameter(s) the parameters for the TSO command, if any.

Remarks: This command will cannot be used when using z/XDC to debug a batch job. When the TSO command is entered, z/XDC will suspend its operation until the specified TSO command completes. Then the current debugging session is resumed.

Examples: TS LISTD 'SYS1.LPALIB' ← A LISTD command is issued. Debugging of the current debugging session resumes when the LISTD command completes.
TSO ISPF ← The current debugging session is suspended and an ISPF session is started. When the ISPF session ends, the current debugging session is resumed.

More Info: Enter Help COmmands TSo.

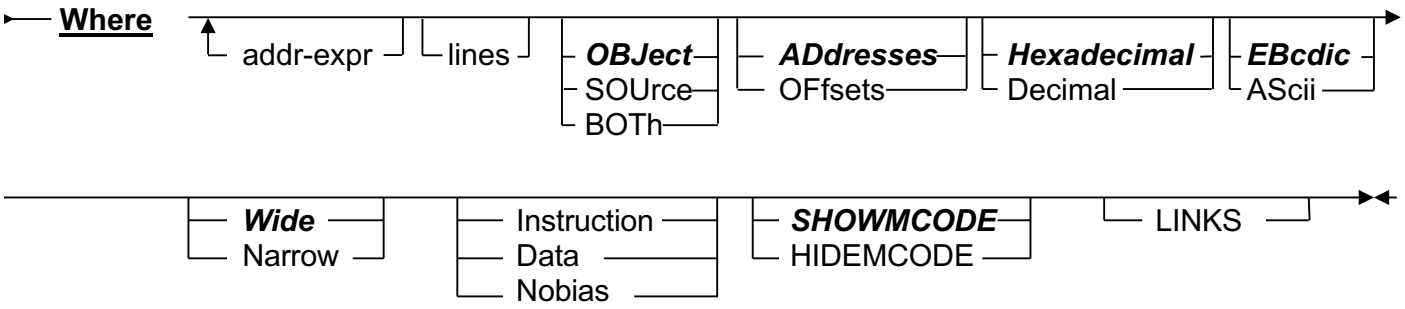
Here are the Set commands you may find most useful:

Set ACCessto	Establish access to a dataspace
Set LOCks	Toggle system locks (used only with SRB debugging) See Help DEBugging SRBmode
Set ILC	System default. Prevents a zap command from overtyping an instruction with an instruction of different length. To over-ride this, issue “Set NOILC”. And go with God.
Set Maplibs	Establish linkage between “this” piece of object code and the original ADATA (for Assembler) or DWARF (for C language debugging).
Set REFRprot	Directly control the way in which refreshable modules are loaded (Key 8 vs. Key 0 storage).
Set TRace	Control the scope and action of the Trace command. Also changes how z/XDC presents the contents of the working window.
Set FOrmat	Change the way in which z/XDC interprets and presents your debugging session.
Set WIndow CReate/DElete	Dynamically create or delete “Watch windows” on the display panel. May only be used with a PFKey.
Set Zap	“Normal” is the system default. Prevents the user from overwriting common storage. To turn this off, issue Set Zap Sprot. And go with God.

You can always research any z/XDC command by starting a z/XDC session and entering Help COmmands <command-name>.

In the case of the Set commands, you may issue “Help COmmands Set”. All available Set commands will be displayed.

Chapter 3 - Seeing your Code, and The Local Environment



Cause z/XDC to FORMAT the instruction pointed to by the Program Status Word. In other words, "Where am I?" Shortcut command equivalent, "W".

Note: **Just issue "W" from the command line or any acceptable space in column one of the display. You don't need all the optional parameters to make this command work.**

addr-expr The location in memory where the disassembly is to begin. If addr-expr is omitted, then the next screen-full of memory is disassembled. If addr-expr is omitted and other parameters are desired, then a comma must be used before stating the other parameters.

lines The number of lines the user wishes to see. The upper limit is 1000. If omitted, z/XDC will fill the current window.

SOurce *OBject* BOth If a source level map is available, source code can be shown as part of the formatted display. OBject is z/XDC's normal output. BOth should be obvious. These settings have no

Produce a "dump" of virtual memory.
Short-cut equivalent: "D"

addr-expr	The location in memory where the dump is to begin. If addr-expr is omitted, then the next screen-full of memory is displayed. If addr-expr is omitted and other parameters are desired, then a comma must be used before stating the other parameters.
Lines=	The number of lines the user wishes to see. If omitted, z/XDC will fill the current window. LINES= will generate from 0-10,000 lines of display.
Bytes=	Will display either a number of bytes given in hexadecimal, or a decimal number, if suffixed with an "N" (example: BYTES=200N).
ADDresses	The virtual address of each line of the display will appear on the left side of the panel.
Offsets	The offset from the start of the data's object (load module, csect or dsect) will appear on the left side of the panel.
***Ebcdic*/Ascii**	Memory will be interpreted in EBCDIC format (default) or ASCIIi format.
***WIDe*/NARrow**	The display command will show 32 bytes of storage OR 16 bytes of storage per line.
omitted	If no parameters are given, z/XDC will display the NEXT LOGICAL screen's worth of memory – like a "scrolling" function.

Remarks: This is one of the core commands in z/XDC. Display presents a "dump-like" display of raw storage.

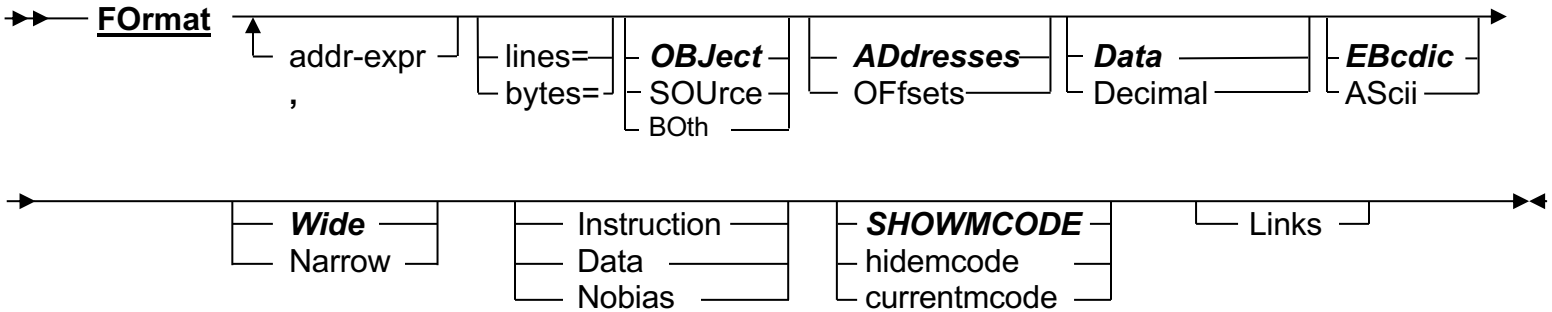
There is one field you'll see in Display that you won't see in a true dump: z/XDC shows you the key structure of memory you're looking at.. "8F" would mean "Key-8, Fetch-protected". "0S" would mean "Key-0, Store-protected storage".

If you're interested in seeing a disassembly of object code instead of a mere display of storage, use the FORMAT command.

Examples:

D ,AS ← The next logical screen full of memory will be displayed in ASCII format.
D MYPROG.MYCSECT+24D 20 ← At offset 24D into MYPROG.MYCSECT 20 lines of memory will be displayed.
D PSW? Lines=20 ← A Display command will be executed where the PSW points in 31-bit mode and 20 lines of information will be shown.
D PSW? B=10 ← A Display command will be executed where the PSO points to in 31-bit mode and X'10' bytes will be shown. That's sixteen decimal, Cowboy.
D R12% ASC ← A Display command will be executed where R12 points to in 24-bit mode and the memory will be shown in ASCII format.

More Info: Enter Help COmmands DISPlay.



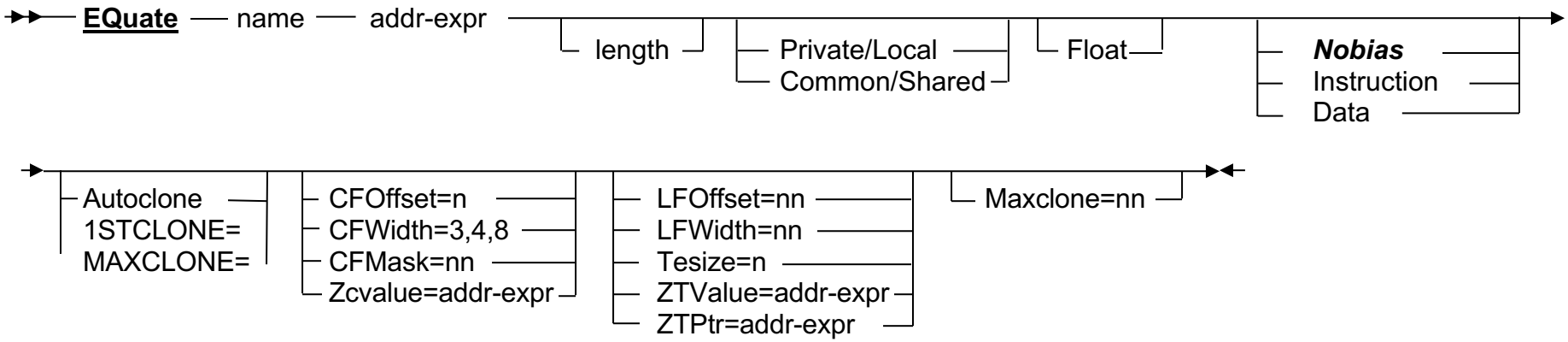
Disassemble memory at a specified location.
Short-cut command equivalent: "F"

addr-expr	The location in memory where the disassembly is to begin. If addr-expr is omitted, then the next screen-full of memory is disassembled. If addr-expr is omitted and other parameters are desired, then a comma must be used before stating the other parameters. The comma is a "place-holder".
Lines=	The number of lines the user wishes to see. The upper limit is 10,000. If omitted, z/XDC will fill the current window.
Bytes=	Specifies the number of bytes to be displayed in hexadecimal, or decimal if suffixed with "N" (example: 100N=100 decimal bytes).
Source/*Object*/BOth	If a source level map is available, source code can be shown as part of the formatted display. OBject is z/XDC's normal output. BOth should be obvious. These settings have no true default, as they can be set in the Profile facility (Under the SET FORMAT command).
ADdresses	The virtual address of each line of the display will appear on the left side of the panel.
OFfsets	The offset from the start of the data's object (load module, csect or dsect) will appear on the left side of the panel.
Hexadecimal/Decimal	Displacement fields of machine instructions will be displayed as hexadecimal or decimal numbers as desired. Operative ONLY when source statement support is not being used.
Ebcdic/AScii	Memory will be interpreted in EBCDIC or ASCII format, as desired.
Wide/Narrow	Shows 32 bytes of memory (default) per line, or 16 bytes per line.
Instruction/*Data*/Nobias	z/XDC makes a number of decisions in the Format process, many of which are dependent on a great many other variables. Consequently, the Format command may, under certain circumstances, mis-interpret data as instructions if the format command is attempted within data rather than object code. The Instruction, Data and Nobias parameters affect z/XDC's determination in the Format command. Please see Help Commands Format for a thorough understanding of how to use these parameters.
***SHOWMCODE*/HIDEMCODE/CURRENTMCODE**	z/XDC will SHOW or HIDE macro expansions - IF – z/XDC is displaying source statements at the time.
Links	When storage is being formatted via a disassembly, and if a line of display shows a snippet of data that is 3, 4, or 8 bytes long, then LINKS will attempt to resolve that snippet as a storage pointer. Then it will display that resolution as a comment on the display line. This can be useful, but is VERY CPU-EXPENSIVE.

Remarks: **This is one of the most important z/XDC commands**, one that will be used in every session. Use the Display command if what you really want is a "dump" of raw storage.

Examples:
F MYPGM.MYCSECT.MYLABEL 999 ← z/XDC will disassemble memory at the specified location for 1000 lines. PF7 and PF8 can then be used to scroll forward and backwards in the disassembled code.
F ← The next logical screen full of data will be disassembled.
F PSW? Li=10 ← z/XDC will produce a FORMAT'd panel starting where the Program Status Word points to, in 31-bit addressing mode. Only ten lines will be displayed.

More Info: Enter Help COmmands FOrmat.



Create an Equate in memory at a specified address.

name A name of up to 63 characters in length. Names may include alphahanumeric or national characters (@,#,$._). If autocloning is desired then name becomes a "root" and subsequent copies of the equate will receive a suffix of "#nn" where "nn" is incremented for every instance of the equate.

addr-expr The location where the equate is to be placed.

length The amount of memory the equate is to "span", in hexadecimal numbers. In other words, equates may represents areas of storage as well as points.

Private A synonym of Local. Forces the equate to be treated as if it occurs only in the Private Area, even if it's actually assigned to a Common Area.

Common A synonym of Global and Shared. The equate will be owned by all address spaces even if it resolves to a private storage location.

Float The addr-expr is saved in unresolved form and is resolved whenever the equate is referenced. Thus, the equate "floats" as the addr-expr changes.

Nobias Default behavior. z/XDC will treat the memory pointed to by the equate as pure data.

Instruction z/XDC will consider the equate as pointing to an instruction, and will attempt to disassemble memory at the equate whenever it is referenced.

Data z/XDC will NOT consider the equate as pointing to an instruction.

Autoclone Directs z/XDC to create a list of named equates that refer to pointers in a chain, or elements in a queue according to Autoclone's own parameter list. If all other parameters are omitted, then z/XDC assumes these defaults: CFOFFSET=0, CFWIDTH=4, CFMASK=7FFFFFFF, ZCVALUE=0 and MAXCLONE=10.

1STCLONE= Normally z/XDC will number the first cloned equate as "1". You may force any initial value from 0 to 999999..

MAXCLONE= Sets a limit on the number of autocloned equates. When 1STCLONE=1, the range can be from 1 to 999999.

CFOFFSET=n The offset of the entry's chain field, starting from the beginning of the dsect. Can be expressed as a hexadecimal number OR a decimal number if followed directly by an "N". That is, "10" means 16, and "10N" means 10. The default offset is 0.

CFWIDTH=n Defines the width of the chain field. Permitted values are 3, 4, or 8. The default width depends on whether or not the CFMASK option is used. If CFMASK is not given, then the default width is 4.

CFMASK=n Defines a selection mask that is AND'd against the chain field's value to eliminate bits that are not part of the chain pointer. The width of the mast must match the width of the chain field (as defined by the CFWIDTH parameter).

ZCVALUE= the "zero chain field" value. z/XDC always considers that a chain has ended when it finds a zeroed chain field. However, you can also give an address expression that will be compared to the contents of each entry's chain field. A match identifies the chain's last entry.

You can also use the keywords "NEGATIVE" or "NOT POSITIVE" as operands of the ZCVALUE parameter. Both of these keywords cause any zero or negative value to signify the end of the chain. When testing for a negative value, the CFMASK= parameter is ignored. z/XDC checks the field's hi-order bit. If it is ON, then the chain is ended.

LFOFFSET=n If the table entries have a length field, then this operand provides the offset of that field from the start of each table entry. As with CFOFFSET the number can be expressed as a hexadecimal number OR a decimal number if followed directly by an "N". That is, "10" means 16, and "10N" means 10. The default offset is 0.

LFWIDTH= If the table entries have a length field, then this operand provides the width of that field. LFWIDTH may range in value from 1 to 8. There is no default value for this parameter. If LFOFFSET= is stated, LFWIDTH must also be stated.

TESIZE= If the table has fixed-length entries, or if the entries are variable, but have a fixed section of constant size, then this parameter provides the length of the fixed section of the entry. Tesize is either pure hexadecimal, or a number followed directly by the character "N". The default value is 0.

Display all of the breakpoints that are currently defined for the z/XDC session.

This command accepts no parameters.

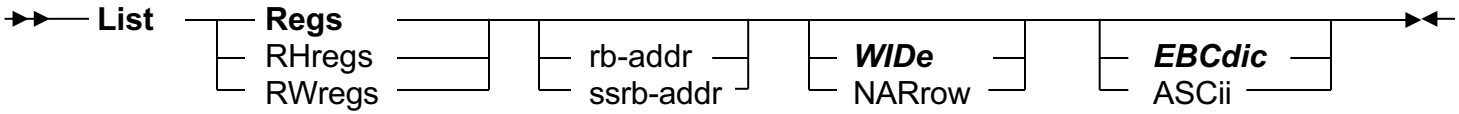
Remarks: The List Breakpoints command displays the following information about all currently defined breakpoints:

- The breakpoint's unique name;
- Location, both as a raw virtual address and as relative to the nearest relevant load module, map or equate (if any);
- The associated conditional expression, if any;
- Associated automatic z/XDC commands that are associated with the breakpoint, if any;
- Whether or not the breakpoint is enabled or disabled (via the "X" line command);
- The number of times the breakpoint has been "hit". A zero ("0") indicates that the breakpoint has never been taken.
- For deferred breakpoints, the number of times the breakpoint has been cloned, and the number of cloning errors (if any).
- Whether or not execution location displays will roll or scroll during tracing (SET TRACE SCROLL keeps the current instruction at the top of the display).

This command may not be used in Foreign Address Space Mode, because z/XDC is not in control of any program when in FASM. You get "look-don't-touch" access.

Examples: L B

More Info: Enter Help COmmands LISt BReakpoints.

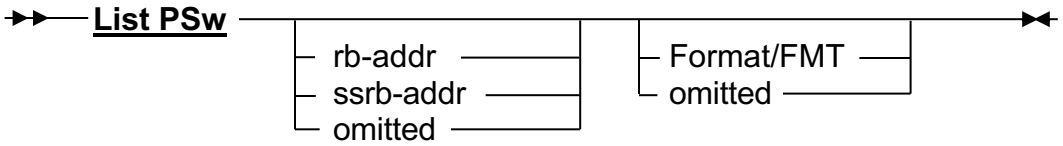


Display the set of Error-level General Registers.

Regs	Display the General Registers. In 64-bit addressing mode, this will show only the low-order half of the 64-bit General Registers.
RHregs	In 64-bit addressing mode, this will show only the high-order half of the 64-bit General Registers.
RWregs	In 64-bit addressing mode, this will show the entirety of the 64-bit General Registers.
RB-addr-expr	z/XDC will display the register set for the module running under the given Request Block. If you issue the appropriate LIST RBS command you may then use the RB#n equates that LIST RBS generates.
Ssrb-addr	z/XDC will display the register set for a suspended SRB or SSRX (an extension of the SSRB that resides in 64-bit storage).
WIDe	Default. z/XDC will display eight words of data per line in its display.
NARrow	z/XDC will display four words of data per line in its display.
EBCdic	Default. The display will be shown in EBCDIC format.
ASCii	The display will be shown in ASCII format.

Remarks: In order to use the RB-addr-expr parameter conveniently, issue LIST RBs for the desired task. That will allow the use of the equate RB#n in this command.

More Info: Enter Help LISt REGisters. See also Help COmmand LISt Generalregisters Registerset.



Display the retry-level Program Status Word for the current or any other Request Block (or Suspended SRB) in the old, 8-byte format.

rb-addr An address expression that resolves to the desired Request Block. If you have previously issued LIST RBS you may use the RB#n equate for this parameter.

ssrb-addr The address of any suspended SRB or SSRX(an extension of the SSRB that resides in 64-bit sotrage), located in any accessible address space in the system. If you have previously issued LIST SSRBS you may use SSRB#n or SSRX#n as a parameter to this command.

omitted z/XDC displays the retry-level PSW for the current program.

Format/FMT Causes z/XDC to translate various fields in the PSW for the user. These fields are:

- The PER bit;
- Dynamic address translation (ON/OFF), I/O (ON/OFF), External interrupts (ON/OFF);
- Protection Key;
- Execution State (Problem state/Supervisor state);
- Address Space Control Mode (Primary, Access Register, Secondary, Home), Condition Code;
- Fixed point overflow (ON/OFF), Decimal Overflow (ON/OFF), Exponent underflow (ON/OFF), Significance (ON/OFF);
- Addressing mode (24/31-bit) and;
- The instruction Address.

2nd parm omitted Formatting is not performed.

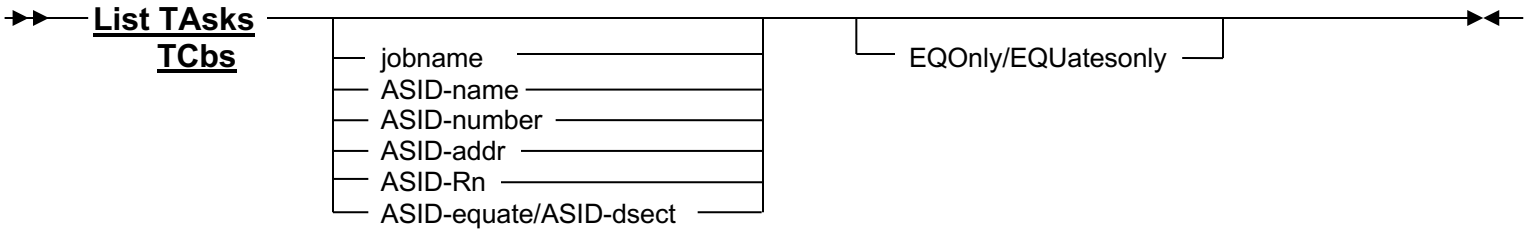
Remarks: This is one of the central commands in the z/XDC debugging tool. You'll use it just about every time you use the product.

The LIST PSWE command will show the full 16-byte format (when executing in the 64-bit environment). All the parameters remain the same.

Examples: L PS F ← Displays the older 8-byte PSW for the current program, and displays the state of the fields.

LI PSW RB#2 ← Displays the PSW for the code running under RB#2 for the current TCB. This command will only work after both the List TAsks and List RBs commands have been entered.

More Info: Enter Help COmmands LISt PSw.



Display the organization of Task Control Blocks within any accessible address space.

jobname Qualify the display by job name, or TSO userid.
ASID-name A keyword, like “HOME”, “PASID”, “ESASID”, “IASID”, etc.
ASID-number The number assigned to the address space in the Address Space Vector Control Table.
ASID-addr An address-expression that points to an ASID number.
ASID-Rn A register that contains and ASID number.
ASID-equate/ASID-dsect An equate or a dsect that has been assigned to represent storage in an address space.
omitted Default. Displays tasks in the current address space.
EQUonly/EQUatesonly This variant of the command does not display the result, but DOES create the relevant Automatic equates that List Tasks normally does. It might, for example, be very useful in a script.

Remarks: The List TAsks command is one of z/XDC’s most powerful commands. If the List TAsks and then the List RBs commands are entered in sequence, the user can gain a very good understanding of where a program is, and why. List TCbs is an alias for this command.

When entered, List TAsks gives an easy-to-understand graphic display that includes:

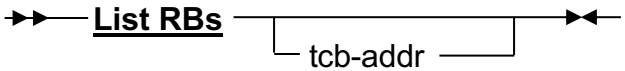
- An equate named “TCB#1-n” for each entry that is created automatically to “name” each Task Control Block;
- The program name running under each Task Control Block;
- The hierarchy of each task by indentation;
- The current task, identified by highlighting;
- The abend type for the current task.

Some of the parameters (those having to do with ASID-etc.) can only work when z/XDC is running APF-authorized.

Note: For dump/XDC users: This command, in conjunction with the LIST RBs command for any sub-task offers the dump/XDC user the ability to alter the Current eXecution Unit with the “CU” shortcut command. Thus, if you prefer to have THIS TASK/THIS RB be the scope of your inquiry, you have a handy too with which to do so.

Example:
L TA ← The task structure for the current address space is shown.
List TA JES2 ← If z/XDC is running APF-authorized, this will show the task structure for the JES2 address space.
L TA 1 ← Displays the task structure for the Master Scheduler Address Space.

More Info: Enter Help COmmands LIS TAsks. This command is really cool. You should read about it.



Display the Request Blocks associated with the current, or any given Task Control Block.

tcb-addr The virtual address of a Task Control Block. If you have previously issued LIST TASKS, then you may use a TCB#n equate for this parameter.
omitted z/XDC displays the Request Block structure for the current task.

Remarks: For convenience, the List TAsks command should be issued prior to List RBs. Then, if you want to easily query the RB structure under any sub-task, you can put the shortcut command "L" next to any TCB# equate, and press Enter. Nice!

[List TAsks establishes equates in memory pointing to each Task Control Block (TCB#1-n, where n is the number of tasks in the address space) that can be used in the List RBs command.]

In fact, issuing List TAsks and List RBs one after the other on entry to a debugging session is a very good way to determine "where" the program is and why.

Elements displayed by List RBs include:

- The type of RB (PRB, SVRB, Etc.);
- The manner in which it was created (ATTACH, PGM-CHK, etc.);
- The name of the routine associated with it (load-module-name, SVC-n, abend-code, etc.);
- and the resume address relative to the nearest relevant module, map or equate (if any).

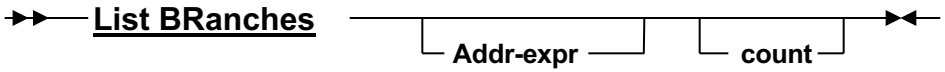
Pro tip: If you end up somewhere and you don't know how you got "here", try this command sequence:

```
LIST BEA;;LIST TASKS;;LIST RBS
```

By stacking these three commands you get a good idea of what's going on. The double semi-colons create blank lines between each command's output for readability. You're welcome.

Examples:
L RB TCB#2 ← Show all the Request Block information for the second Task Control Block in the address space.
L RBS 21C%+7C! ← Show the Request Blocks queued from the Jobstep TCB.

More Info: Enter Help COmmands List RBs. See also Help COmmands LISt TAsks.



View the contents of z/XDC's Branch History Table.

To be more in agreement with the Help System: The list branches command displays a list of recent user-program machine instructions that a) have been executed by the user program, b) are branch-type instructions and c) that z/XDC KNOWS about.

addr-expr A "filter" that, when given, includes branch history table entries only with this branch-from or branch-to address (in the target address space).

count A decimal number in the range from 1 to 4096 that sets the upper limit of Branch History Table entries to be displayed. If outside this range, the number will be treated as an address expression. If TWO counts are given, the second will be parsed as an address expression, not a count.

Remarks: **In order to work, your processor must have BEA support enabled.** You may issue the LIST FEATURES command to find out if BEA is enabled.

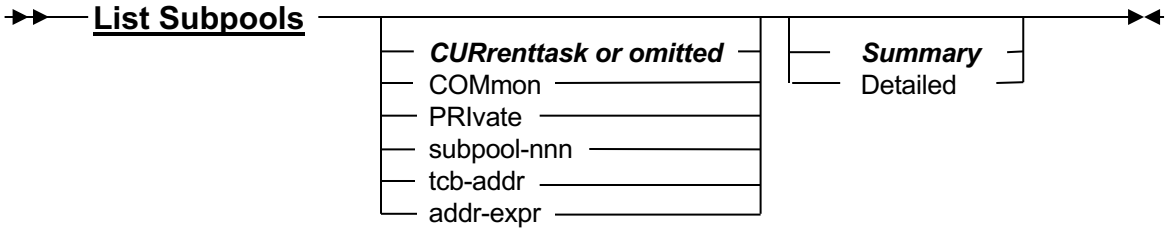
Every time that z/XDC gets control, it checks to see whether or not the user program's current retry level instruction is a branch-type instruction. If so, then an entry is added to its Branch History Table. NOT every branch that the user program executes will be recorded in the Branch History Table. Only those branches at which z/XDC receives control will be recorded.

Note: If the Breaking Event Address field is available for the current processor, then a LIST BEA command will automatically be included in the output of LIST BRANCHES.

This command may not be used in Foreign Address Space Mode.

Examples: *LI BR mymod.mycsect.myloop* ← *investigates all branches from, or to this point* in the program.
LI BR ← show all branch statements at which z/XDC received control.

More Info: Enter Help COmmands LISt BRAnches.



Display the organization of storage subpools within the Home Address Space.

CURrenttask	Default. Since it's the default, this parameter can be ignored. Displays subpool information for the current task in the address space.
COMmon	Displays subpool extent information for the Systems Common Storage Area and System Queue Area. "Global" is a synonym.
PRIvate	Displays subpool extent information for the Private Area in the address space. "Region" and "Local" are synonyms.
Subpool-id	A numeric value up to three digits in width. z/XDC will return a display for only that subpool extent number (example: "78" or "230").
TCB-addr	The starting address of a Task Control Block. z/XDC will return a display for the given task. If you execute the List TAsks command, you may thereafter use the equates generated automatically by the List Tasks command (i.e. "TCB#n").
addr-expr	z/XDC will display the subpool extent that the address expression resolves within (example, "PSW!" or "R15?").

Summary	Default. z/XDC displays totals for all subpool allocations.
Detailed	z/XDC displays a single line for each and every extent of subpool allocation.

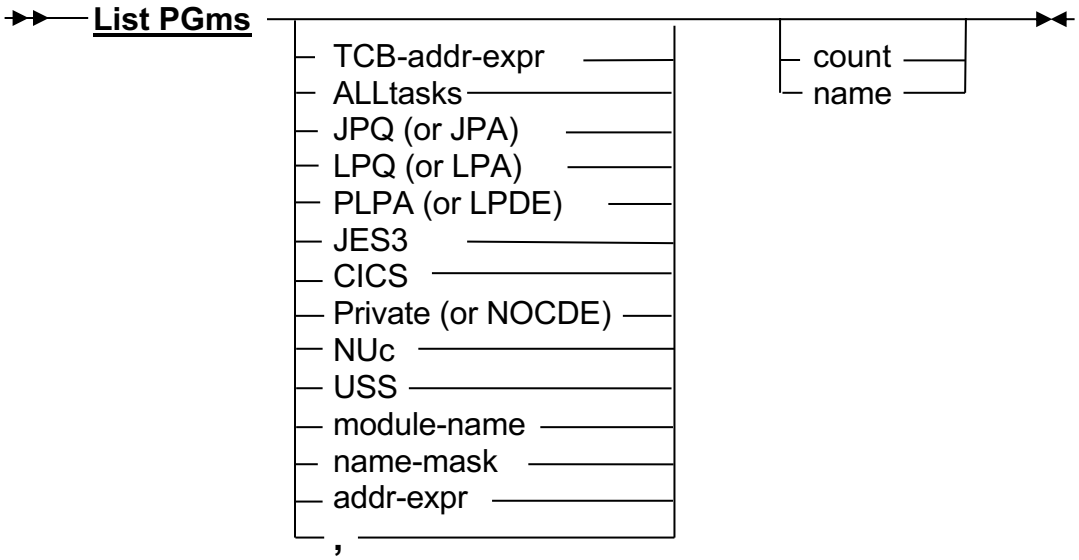
Remarks: When issued, this command will display the starting and ending addresses of each subpool extent, the amount of space allocated, the amount remaining unused, the Key of the extend and the ownership of each extent by task control block number.

This command can be very useful for tracking the allocation of memory by a program (or perhaps its mis-allocation).

Examples:

L S 78	←z/XDC will return the allocations of memory within subpool 78.
L S TCB#3	← If a LIST TASKS command has been executed, you may use the TCB#n equate and see subpool allocations for any given task.

More Info: Enter Help COmmands LIS SUbpools. See also Help COmmands LISt SUbpools Reports.



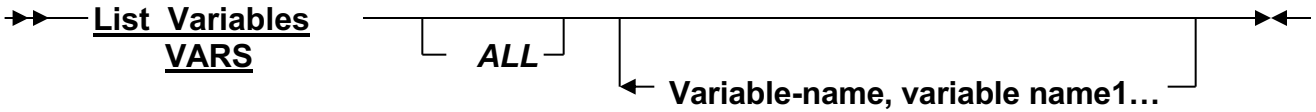
Display the list of load modules in memory for any given task, or locate a module via an address-expression.

1st parm omitted	Default. Display only load modules running under the current TCB. You can't omit this parameter when in Foreign Address Space Mode. **Note:** When the first parameter is omitted, and a second parameter stated, you must use a comma (",") to indicate the omission.
TCB-addr-expr	The queue of load modules associated with any given TCB in any accessible address space If you've previously issued LIST TASKS, then you may use any TCB#n equate as this parm.
ALLtasks	Displays all queues of load modules within all TCBs in this address space. You may NOT abbreviate this parameter to "ALL". It could end up confusing you. Dave says so…
JPQ	Displays the queue of load modules in the Target Address space's Job Pack Queues. An alias is "JPA".
LPQ	Displays the queue of load modules in both Systems' Dynamic Link Pack Area (DLPA) and the Classic Link Pack Area (LPA). This also includes the Modified Link Pack Area (MLPA) and the Fixed Link Pack Area (FLPA). An alias for this parm is "LPA".
PLPA	Displays the queue of load modules in the system's Pageable Link Pack Area. An alias is "LPDE".
JES3	Displays all load modules loaded by JES3's internal services. This command DOES work under Foreign Address Space Mode (wherein one issue the SET ASID command).
CICS	Displays the queue of load modules loaded via CICS services.
Private	A "Privately Loaded" load module is one that has been read into storage in such a way that its location is not described by system control blocks (such as via a directed load or by using the LOAD macro with GLOBAL=YES). Thus, such modules do not have either CDEs or LPDEs. This parameter displays a report of all mapped Privately Loaded load modules and program objects – IF- they have previously been MAP's or DMAP'd. An alias is "NOCDE".
NUc	Displays the system Nucleus (IEANUC01).
USS	Can also be stated as "Pathname", if you like typing. Displays the same list of programs that the ALLTASK operand shows – EXCEPT – that only programs that have a path name are shown.
module-name	Searches all address space and system queues are searched for this name.
Name-mask	Use "?" to substitute a single character in a queue's name. Use "*" to substitute one to many characters in a queue's name.
addr-expr	Returns the name of the module that addr-expr resolves into.
count	Suppresses the display of the first "count-1" entries on the list.
Name	Suppresses the display until "name" appears
2nd parm omitted	No messages are suppressed from the display.

Remarks: This command helps the user to locate and count modules in terms of their occurrence in memory. There are some rules in which parameters may be stated. Please see the Help Subsystem for pertinent details.

Examples: L PG 21c%+7C% ← This shows the load modules associated with the Jobstep TCB.
LIST PGMS CICS ← If the command is issued within a CICS address space, z/XDC will display all modules loaded by CICS (It does NOT display any load modules located in the region's Job Pack Queue).
L PG PSW! ←This shows all uses and aliases of the currently executing program.

More Info: Enter Help COmmands LISt PGms , Help COmmands LIST PGms Syntax and Help COmmands LIst PGms Report.



Display all, or individual C variables.

ALL Default (or omitted). c/XDC displays all variables from all variable pools in only the current stack frame. This is the default action.

Variable-name The name(s) of individual variables. If the ALL parameter is given in conjunction with other variable names, then "ALL" is treated as a variable name.

Remarks: This command is central to c/XDC (and our support for other High Level Languages, in the future). If individual variable names are given, then c/XDC will display one or more lines for each variable to include:

- Simple scalar variables;
- Array names;
- Selected array elements;
- Structure names;
- Structures within arrays within structures within…;
- Unions.

The variables value will be displayed, formatted according to the variable's type.

Individual variables may be examined in several ways using new shortcut commands that you put in column 1 of the display line for each variable.

- "D" or "F" - Perform a Display of the variable. It will be shown in raw hexadecimal storage format. Format works the same way in this situation. Note that if you want to change a variable, you MUST first Display it (the "D" shortcut command), then use the "Z" shortcut command to alter it.
- You may use the "L" shortcut to "drill into" a variable, and "L" again to drill out. This behavior is most useful in the Working Window, and acts differently in a Watch, or Display Window.
- Asterisk ("*") – Display the variable in EBCDIC format.
- Vertical bar ("|") – Display the variable in ASCII format.
- Back-slash ("\") – Display a string, up to its null termination.
- Forward-slash ("/") – Display the entirety of a string, up to its length termination.

The default value in the "factory" Profile allows a LIST VAriables upper limit of 32767 lines of display(!). The List Variables Recurse Limit (per structure) is also 32,767 lines.

Examples:

L VAR ← You see 'em all. Depending upon your program's complexity, his may result in a certain amount of processing time and a large display.

L VAR FRED ← You see only variable "FRED".

L VARS array[0][1][2-4] ← This variant drills down into a two- or three-dimensional array. It's expressed as a C programmer might expect, and in this case, c/XDC will display the "X" dimension as "0", the "Y" dimension as 1, and the "Z" dimension to show the 2nd, third and fourth elements in the array. There's more in the c/XDC Primer.

More Info: Enter Help COmmands LISt VARiables. See also Help SHortcutcommands.

List VStack

Review the contents of the Variable Stack Pools.

This command accepts no parameters

Remarks: This command is implemented in support of c/XDC. If you don't have the c/XDC feature installed, this command won't help you.

An LE compliant subroutine generally has access to up to three variable pools:

- A Global Pool of constants;
- a Subroutine Pool of variables declared for the current routine, subroutine or function;
- a Local Pool of variables declared only for the current block of code.

As one subroutine calls another, new Subroutine Pools and Local Pools are created, and the variables declared in the calling routine are generally unavailable to the called routine. The management of these variable pools is accomplished through structures known as Lanquage Environment Stack Frames.

The LIST VSTACK command displays information about these Stack Frames as follows:

Stack Frames are displayed from oldest to newest.

For each frame:

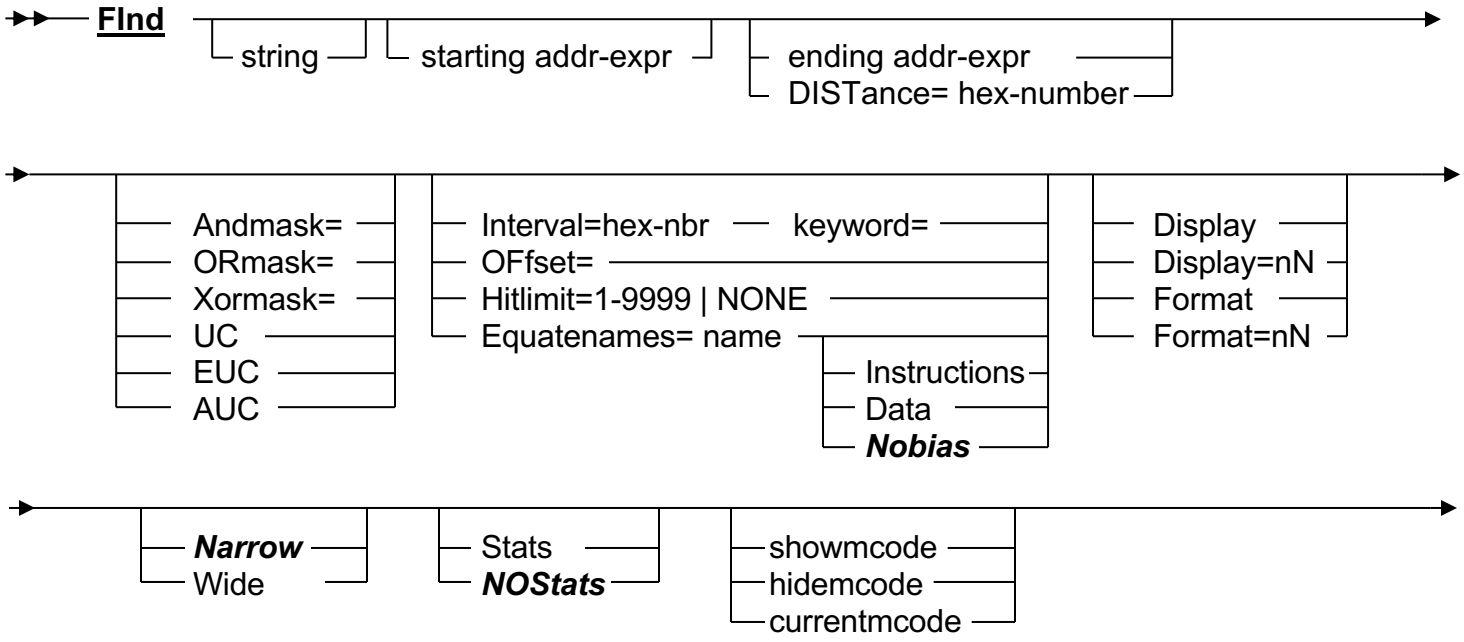
- The resume address is displayed for the routine that owns the frame;
- Information is displayed about each variable pool that is owned by the frame;
- Shortcut Entry Fields are provided that can be used to display resume address locations or variable pool storage.

For each pool, an automatic equate is generated conveying the following information:

- The sequence number of the stack frame;
- The type of variable pool (Global, subroutine or local);
- The location of the variable pool;
- The length of the variable pool.

If you're an experienced z/XDC Assembler user, then you will see a similarity – even a parallel - between the List VStack and List TAsks commands.

More Info: Enter Help List VStack.



Search for a specific string of data sequentially from a given point in memory.
Helper Dialog: "FIND ?"

string	That which you seek(!). The string can be Hexadecimal, EBCDIC (enclosed in single quotes), ASCII or a mixture of all. If string is omitted, z/XDC uses the previously defined string. Thus, in searching for multiple occurrences of the same "thing", one can merely re-issue the FIND command without parameters.
Starting addr-expr	The location in memory where the user wishes to begin the search. The search will proceed "upwards" in memory from this point.
Ending Addr-expr or DISTance=	Where the user wishes the search to end. DISTance= is a hex number ranging from 1 to two-times-ten to the 64th power minus one. If DISTance is specified, z/XDC will begin with Starting addr-expr and will stop when DISTance is reached.
Andmask=/Ormask=/Xormask=	Boolean arithmetic/Masking operations to be performed on the target data before comparison. The user supplies a mask of any length.
UC	Cause what is found to be upper-cased depending upon the current format set in the SET FORMAT command.
EUC	Cause what is found to be upper-cased in EBCDIC format.
AUC	Cause what is found to be upper-cased in ASCII format.
Interval=	Normally, FIND operates on every sequential byte within the range of storage to be searched. Interval cause the find to be executed at "every nth byte" instead. Can be a pure hexadecimal number or a decimal number suffixed by "n" (ex: "10N").
Keyword=	Equivalent to Interval. Keyword can be "BYTE", "HWORD", "FWORD", "DWORD", "QWORD", "HALFWORD" , "FULLWORD", "DOUBLEWORD", "QUADWORD", "PAGE".
Offset=	Either a hexadecimal number or a decimal number suffixed by an "N". If an Interval is specified that is greater than 1, then the offset is used to test for a match at the specified offset within the interval.
Hitlimit=	Normally, FIND stops at the first "hit" in sequential storage. Hitlimit causes FIND to run until a predetermined number of "hits" are found. This may be a decimal number up to 1000.
=NONE	The FIND command will show ALL "hits" within the range of storage to be searched.

Equatenames= An equate name to be generated automatically whenever a match is found. Start with a "root name", such as "PTR" or "THING" . Must consist of alphanumerics or national (@ # $ and underscore) characters. The first character must not be numeric. May be up to 63 characters in length – *including the root name*.

Instructions/Data/Nobias Whether the memory mapped by the equatename generated above will be treated as instructions, as data or as it occurs.

Display Specifies that the result of the Find will be shown in "dump" format (via a Display command).

Display=nN Causes the Display command to show only a number of lines. The number is specified as a decimal number, suffixed by "N".

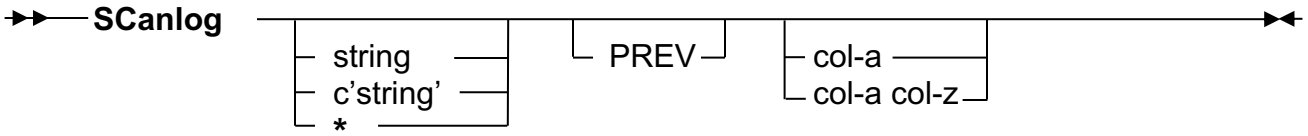
Format Specifies that the result of the Find will be shown in disassembled mode (via a Format command).
Format= Causes the Format command to show only a number of lines. The number is specified as a decimal number, suffixed by "N".
Narrow The displayed result of the Find command will be 16 bytes wide, and will include an EBCDIC translation of memory.
Wide The displayed result of the Find command will be 32 bytes wide.
Stats Requests an additional display of statistics z/XDC keeps during the Find operation.
NOStats Requests no display of Find statistics.
Showmcode/hidemcode/currentmcode: Controls whether macros will be displayed if z/XDC is being used with source statement displays.

Remarks: This command is extremely useful for locating specific things in memory. In particular, it can be used to locate modules by their "eye-catchers". Find has been re-engineered to help users to cope with the vast amounts of memory available within 64-bit address spaces**.**

Helper Dialog: Enter "FIND ?" ("Find – space- ?) and you'll get a "Helper Dialog" that presents a screen with all the FIND parameters. You can fill the panel in without having to memorize all the parameters!

Examples: FI 'DAVID B. COLE' PSW? ← Presumes 31-bit addressing (the "?"). Starts where the Program Status Word "is" now, and searches sequentially in memory for the characters "DAVID B. COLE'.
FI ← Continues a previously defined search for a given item from the current location in memory.

More Info: Enter Help COmmands FInd.



Search the session log for a particular string of characters.

String The character string to be searched for. May be enclosed in quotes. If the string contains spaces or commas, then quotes MUST be used.

c'string' C indicates that the string you're looking for is to be case-sensitive. You must enclose the case-sensitive string with either single or double quotes on each end of the string.

- String may contain any characters, including blanks, commas and the quote character.
- If a quoted string does contain the quote character, then it must be double as in the word don''t. This, however, will make the search case insensitive. So I suppose I wouldn't try to include phrases in my string that are conjunctions or are possessive.

***** Or omitted. Search for a previously stated character string again. That is, reissue the search for the "same string" again.

PREV Forces z/XDC to search for the string "upward" or "previously" in the log.

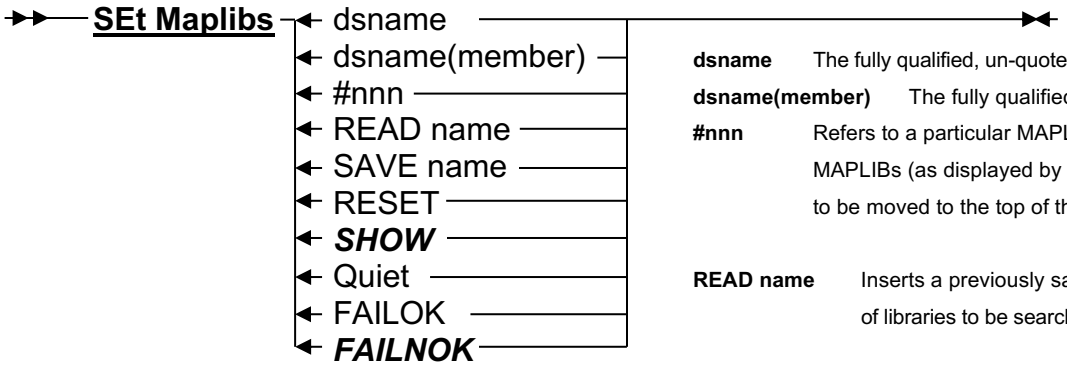
col-a Column areas to scan. If col-a is stated without col-z then SCanlog will search for the string value beginning ONLY in column-a.

col-z If col-a is and col-z are both specified, then the string will be found if the first character falls between the two column values.

Remarks: Once set, column limits remain unless reset. That is, if the user wishes to use other column definitions later they must be entered separately.

Examples: SC 'COLE' PREV ← Searches for the character string "COLE" upward in the session log.
SC 'COLE' PREV 24 33 ← Searches for the character string "COLE" upward in the session log, and occurring only within columns 24-33.

More Info: Enter Help COmmands SCanlog. See also Help COmmands SYntax Characterstrings.



dsname The fully qualified, un-quoted name of a sequential or partitioned dataset.

dsname(member) The fully qualified, un-quoted name of a member of a PDS.

#nnn Refers to a particular MAPLIB's unique number in the list of active MAPLIBs (as displayed by the List MAPlibs command). Causes #nnn to be moved to the top of the search order in the list. You must preface this parameter with "#".

READ name Inserts a previously saved MAPLIBs list into the front of the currently active list of libraries to be searched for ADATA information. It does not replace the active list.

Create or add to the list of datasets that z/XDC will search for ADATA (source statement) information.

SAVE name Causes the currently defined list of MAPLIBs to be saved under a name, so that they can be "READ" later.

name For both the READ and SAVE parameters, "name" is 1-8 characters, including "_", "@", "#" and "$".

RESET The currently active MAPLIBS list is cleared. Previously saved lists of MAPLIBs are not deleted.

***SHOW*/Quiet** Determines whether the list of defined MAPLIBs will be displayed when the List MAPlibs command is issued, or not. SHOW is the default. NOSHOW is a synomym for Quiet.

FAILOK/*FAILNOK* If SEt Maplibs is being executed from within a script that is processed by z/XDC's READ command, then this parameter controls whether the READ command will abort on error.

Remarks: z/XDC automatically maps your program if it can find the ADATA anywhere in your search order. SET AUTOMAP ON is active in the "factory default" profile that comes with the product, so once you create a default MAP library(s) you won't have to either issue SET MAPLIBS or the MAP command again. It's a great convenience, but if the ADATA dataset cannot be found in your search order, then you will find SET MAPLIBS very useful.

After SEt Maplibs is issued, z/XDC's MAP command will read source statements for use in Formatted displays. Lists of MAPLIBs can be saved in the user's profile as the user's default. Lists of MAPLIBs may also be established by the systems programmer as a system-wide default.

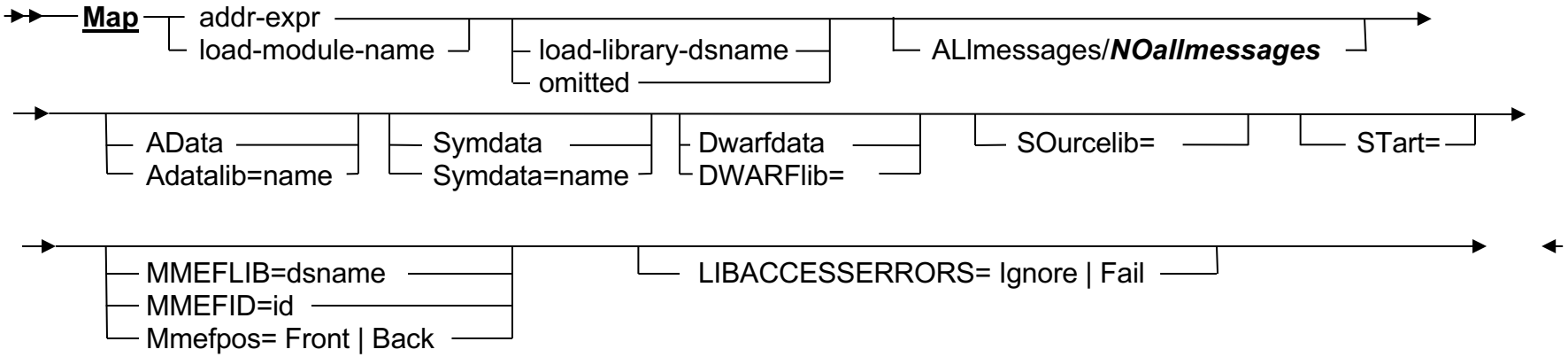
Here is the command sequence to make a set of maplibs your personal default, in your default profile:

1. DELETE MAPLIB DEFAULT ← This wipes out anything you had already!
2. SET MAPLIBS <as.many.datasets.as.you.like>
3. SET MAPLIB SAVE DEFAULT
4. PROFILE SAVE

Another way to avoid having to issue SET MAPLIBS is to add a DD name of XDCMAPLB to your JCL. z/XDC will search this dataset name for ADATA and will automatically map it if the dataset contains your ADATA.

Examples: S M DBCOLE.XDCZ17.XDCADATA ← When a subsequent MAP command is issued, this dataset will be searched for ADATA information.

More Info: Enter Help COmmands SEt Maplibs. For important overview information, enter Help MAPs AData.



Build symbol maps or source image maps for High Level Language load modules.
Helper dialog: "MAP ?"
Shortcut command equivalent: "M"

addr-expr	An address expression that resolves to a place in the desired load module. z/XDC will return symbol or source image information for the module that the address expression occurs in. You can state module.csect if you prefer. When the module is mapped, the addr-expr is examined again and a csect map is loaded.
load-module-name	If stated as a pure load module name, then ONLY a module map will be loaded. To load a csect, end the module name with a dot, or use an address expression that resolves inside the csect or specify a csect name (ie, "module.csectname").
load-library-dsname	A load library from either a PDS or PDSE as a catalogued, fully qualified, UN-quoted name. May NOT be the name of an ADATA library or GOFF file. Two-byte variable symbols are allowed for each qualifier. See Help COmmands SYNtax DSNames
2nd parm omitted	z/XDC will attempt to located symbol or source image information on its own.
Allmessages/noallmessages	Map can generate a LOT of messages as it searches for your map. The default action, "noallmessages" surpresses them.
AData	This parameter coerces the Map command to create ADATA information (SYM data is not considered).
Adatalib=name	A dataset or library name where z/XDC is to search for ADATA. If unspecified, then the current MAPLIBS list is searched. See Help MAps AData MAPlibs for more information.
Symdata	This parameter coerces the Map command to search for SYM records, rather than ADATA (ADATA is not considered).
Symdata=name	The name of a load library that contains the load module to be mapped, using SYMDATA.
Dwarfdata	Coerces z/XDC to load DWARF Data. ADATA and SYM data are not considered.
DWARFlib=	The name of a partitioned dataset (PDS or PDSE) that contains the a member whose name matches the csect to be mapped. It must contain DWARF data.
SOurcelib=	A synonym for DWARFlib=libr. This parm will assist with setting up a LIBRARYLISTS redirect (see Help COmmands LIBrarylists).
STart=	The load point of a privately loaded module (one that has no CDE). When you use "ST=", the first parameter of the command must be the name of the load module, not an address-expression. For example "MAP MYMOD START=24CDC".

Using MAP with XDCMMEF program output:

When you receive a dump from a down-stream or customer site, and when that dump was generated on a system other than your own, you often have to guess at the layout of the foreign system's storage. The freeware program XDCMMEF available on the Cole Software website (https://www.colesoft.com/support/module-extract-utility) can be downloaded and run on the foreign system. This extracts "boundary" information (no content is included) and is highly compressed. If the people at the foreign system send you the output of XDCMMEF, then dump/XDC can fold this into the dump, giving you a much clearer picture of the foreign system's layout. NORMALLY, dump/XDC can fold XDCMMEF's output into the dump seamlessly. Other times you may have to help it along. The following parameters help you to accomplish that.

MMEFLIB=ds	Causes dump/XDC to create a DEFINE MMEFDSN to add to the MMEF datasets list.

MMEFID=id This operand can be used to supply a specific ID value for the MMEF Definition. It is used to create a short label for later use in DDNTRANS tokens. See Help COmmands DEFine Mmefdsn for more information.

MMEFPOS= Front | Back Establishes the position of the MMEFDSN "token" in the queuing order.

LIBACESSERRORS= Ignore | Fail If MAP encounters errors in trying to map from a MMEFDSN library, then Ignore (the default) will not display an error. If Fail is in effect then the error tells the user that a library is missing.

Remarks: This important command helps z/XDC to make raw object code far more intelligible to the user. It would be worth your time to research this command thoroughly.

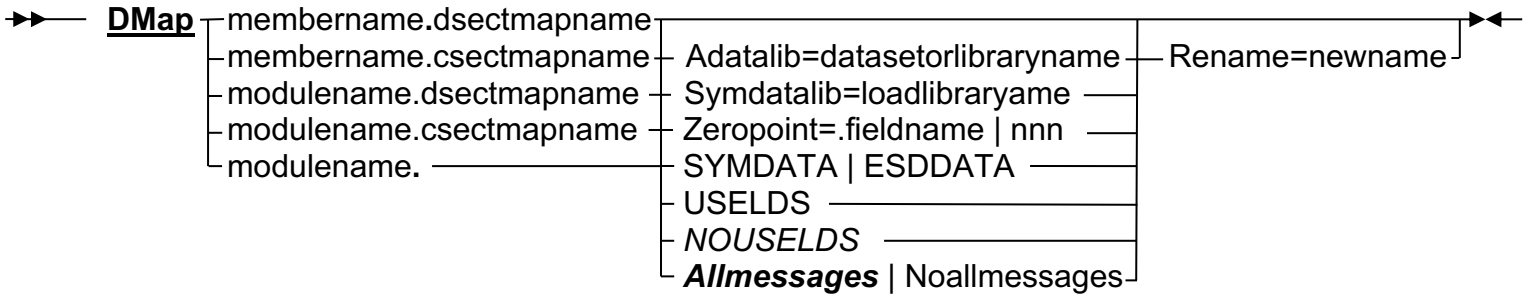
The Map command can make a formatted display look just like a source listing, including original mnemonics and comments. In the absence of ADATA information being made available to it via the SET MAPlibs command, z/XDC will load a linkedit map and ESD entries as well as ADATA or SYM record information (if it is available).

There is an AUTOMAP function available for z/XDC, c/XDC AND dump/XDC. If ADATA or DWARF files are established for any program or compilation unit AND if SET AUTOMAP ON or YES is in effect then the MAP command happens without your intervention. This is hugely cool! You can investigate and change the automap setting either with an interactive command (SET AUTOMAP <whatever>) or issue PROFILE and navigate to Display/Change AUTOMAP PRINT and Other Settings.

Helper dialog: You can issue "MAP ?" and a Helper dialog will appear. The MAP ? Dialog presents you with all the fields used by MAP, allowing you to fill in the fields as you care to. It's a REAL convenience!

Examples: SET MAPLIBS MYUSERID.ADATA PROJECT.ADATA ← Tells z/XDC where to find ADATA for a module.
M PSW! ←The map command brings the Adata information into the z/XDC display, allowing the user to view original source code.

More Info: Enter Help COmmands Map. See also Help COmmands Map Syntax. See also Help Whatsnew z112 Helperdialogs. See also Help COmmands SEt Automap. See also Help MAps Privatelyloaded.



Load a DSECT map into memory.
Helper dialog: DMAP ?

membername.dsectmapname	The name of a PDS dsect in ADATA or SYMDATA format. If membername name is omitted, then the load module scanned by any prior DMap command will be used again.
membername.csectmapname	The name of a CSECT that you wish to load as a dsect. This is useful in situations where one wishes to "preload" a csect in order to set a deferred breakpoint or hook at a label (.lablename) inside the csect, rather than using a hex offset.
modulename.dsectmapname	The name of a loadmodule and dsect to be loaded – OR a **.csectmapname** which will be loaded as a dsect.
modulename.	When you specify "modulename-dot", it forces z/XDC to LOAD a new dsect map – NOT clone an existing dsect map.
Adatalib= | Symdatalib=	Libraries to be searched for either ADATA or SYMDATA versions of a csect.
Zeropoint=	You may establish a "beginning" field using .fieldname or an offset using a hex value (or decimal with an "N" suffix).
SYMDATA | ESDDATA	These two interchangeable parameters both force z/XDC to IGNORE ADATA information. I don't know why I'd want that.
USELDS | *NOUSELDS*	This are specific to dump/XDC. You can force DMAP to "Use Local Datasets" in a dump/XDC session, OR ignore XDCMMEF and DSECTLIB tokens in a dump/XDC session.
***Almessages* | Noallmessages**	You can choose to view or discard messages that are generated during z/XDC's "hunt" through the search order.
Rename=newname	Allows you to load the dsect with a different name of your choosing. For example, if you loaded a dsect map for a particular TCB, you could name it JSTEPTCB to denote the Jobstep Task dsect map. Now it stands out.

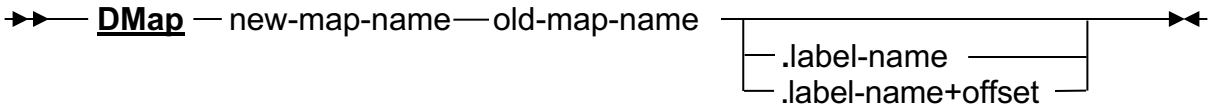
Remarks: The DMap command has two forms. This form loads a copy of the SYM (or ADATA) records for a dsect into memory. The "clone" form of the DMap command (appears before this command in this book) will make a copy of an existing dsect definition with another name.

IMPORTANT: When you have loaded a dsect into memory *you must subsequently issue a Using command* to "place" it at a specific virtual address. See the Using command. In order to make ADATA available to the DMap command one may need to issue the Set MAPLibs command.

Helper Dialog: **Enter "DMAP ?" and a "Helper Dialog" will appear. This will allow you to fill in all the command's parameters without having to memorize them. This is a real convenience.**

Examples: DM XDCMAPS.TCBFIX ← This command loads a copy of the TCB dsect from the XDCMAPS module supplied with the z/XDC product.

More Info: Enter Help COmmands DMap Load. Also, Help COmmands Using and/or Help COmmands SEt MAplibs



Create a “clone” (an exact copy) of an existing dsect map, assigning it a new name.
Helper dialog: DMAP ?

new-map-name The name you wish to assign to the copy of the existing dsect map.
old-map-name The name of the existing dsect map you have previously loaded with a DMap load command.
.label-name The name of a field label within the new dsect where you would like to establish a “zero point”. The dot (“**.**”) must be used.
+offset The hexadecimal offset within the new dsect where you would like to establish a “zero point”. You can use a negative offset also (“-offset”).

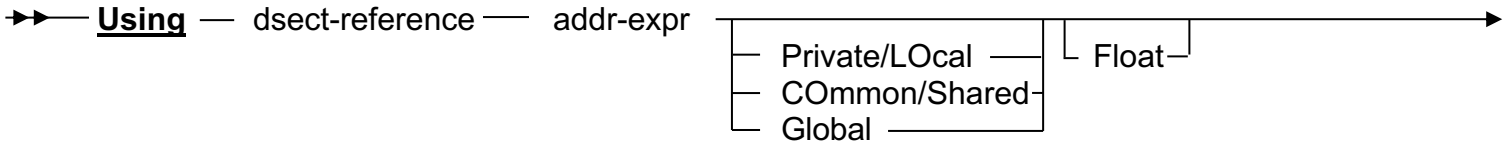
Remarks: This command gives you the option of establishing a “zero point” within the dsect map that will be referenced in a subsequent Using command.

Recent update: Enter “DMAP ?” and a “Helper Dialog” will appear. This will allow you to fill in all the command’s parameters!

IMPORTANT: When you have loaded a dsect into memory you must subsequently issue a Using command to “place” it at a specific virtual address. See the Using command.

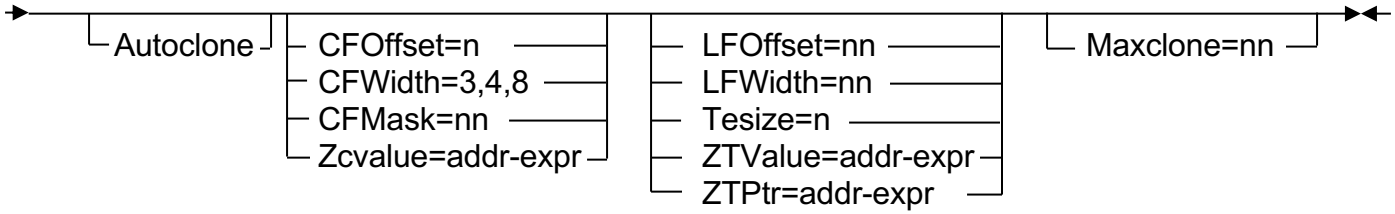
Examples: DM NEWTCB TCBFIX.TCBRBP ← This creates a copy of the existing dsect map TCBFIX and establishes label.TCBRBP as its zero point.
DM TCBFRED,.TCBRBP ← This searches all maps for the label .TCBRBP. The first map found to contain this label is TCBFIX. A new dsect named TCBFRED is created but with the zero point set to .TCBRBP.

More Info: Enter Help COmmands DMap Clone. Also, Help COmmands Using.



Place a dsect in memory at a fixed address or set it to "float" with the resolution of addr-expr.
Helper Dialog: "USING ?"

dsect-reference	A field in the dsect that will be used as the origin address. Sometimes the origin address is NOT the first field in the dsect.
addr-expr	The address in memory where the user wishes to place the dsect.
PRIVATE/LOCAL	"PRIVATE" and "LOCAL" are aliases of each other. When used, they force the dsect to be treated as residing in the Private Area, even if it is actually assigned to a Common Area location.
COMMON/SHARED	A common or shared dsect is owned by ALL spaces, address spaces, data spaces and real storage. You can override z/XDC's determination of a dsect's placement by using this parameter. Common/shared dsects are not displayed when real storage or data space storage is displayed. Use "GLOBAL" for that purpose.
Global	A global dsect is owned by ALL spaces: address spaces, data spaces and real storage.
omitted	GLOBAL, COMMON, SHARED, PRIVATE and LOCAL all omitted. If the dsect resolves to a common storage location then it is treated as being Common. Otherwise it is treated as being private. This may be your "best" choice in most cases.
Float	The Float parameter specifies that the origin address of the dsect will move in memory according to the resolution of addr-expr. This is very handy for correctly placing dsects that might map an input or output record, or any other data structure that may "move" in memory.



Autoclone	Directs z/XDC to create a list of named equates that refer to pointers in a chain, or elements in a queue according to Autoclone's own parameter list. If all other parameters are omitted, then z/XDC assumes these defaults: CFOFFSET=0, CFWIDTH=4, CFMASK=7FFFFFFF, ZCVALUE=0 and MAXCLONE=10.
CFOFSET=n	The offset of the entry's chain field, starting from the beginning of the dsect. Can be expressed as a hexadecimal number OR a decimal number if followed directly by an "N". That is, "10" means 16, and "10N" means 10. The default offset is 0.
CFWIDTH=n	Defines the width of the chain field. Permitted values are 3, 4, or 8. the default width depends on whether or not the CFMASK option is used. If CFMASK is not given, then the default width is 4.
CFMASK=n	Defines a selection mask that is AND'd against the chain field's value to eliminate bits that are not part of the chain pointer. The width of the mast must match the width of the chain field (as defined by the CFWIDTH parameter).
ZCVALUE=	The "zero chain field" value. z/XDC's default is to consider a zeroed chain field the end of the chain. However, you can also give an address expression that will be compared to the contents of each entry's chain field. A match identifies the chain's last entry. You can also use the keywords "NEGATIVE" or "NOT POSITIVE" as operands of the ZCVALUE parameter. Both of these keywords cause any zero or negative value to signify the end of the chain. When testing for a negative value, the CFMASK= parameter is ignored. z/XDC checks the field's hi-order bit. If it is ON, then the chain is ended.
LFOFFSET=n	If the table entries have a length field, then this operand provides the offset of that field from the start of each table entry. The number can be expressed as a hexadecimal number OR a decimal number if followed directly by an "N". That is, "10" means 16, and "10N" means 10. The default offset is 0.
LFWIDTH=	If the table entries have a length field, then this operand provides the width of that field. LFWIDTH may range in value from 1 to 8. There is no default value for this parameter. If LFOFFSET= is stated, LFWIDTH must also be stated.

TESIZE= If the table has fixed-length entries, or if the entries are variable, but have a fixed section of constant size, then this parameter provides the length of the fixed section of the entry. Size, as above is either pure hexadecimal, or a number followed directly by the character "N".

ZTVALUE= Either ZTVALUE, ZTPTR or MAXCLONE can be used to provide end-of-table information. ZTVALUE= can be used when the table entries are defined by a length field. As with the ZCVALUE parameter, you can give ZTVALUE an address expression, a NEGATIVE or NOTPOSITIVE parameter. If an address expression is given, the address expression is resolved and compared with the contents of the length field. A match identified the last entry of the table. If NEGATIVE is given as a parameter, z/XDC will assume that the last table entry contains a negative value (the hi-order bit is ON). If NOTPOSITIVE is given as a parameter, z/XDC will assume that the length field of the last table entry will contain either a negative number or zeroes.

ZTPTR= ZTPTR can be used to provide a table's ending address. z/XDC will recognize the end of the table when it attempts to create an autocloned equate that is located at or beyond the ending address.

MAXCLONE= MAXCLONE sets the maximum number of equates to be created through autocloning. The default is 10. The maximum is greater than you will ever need.

Remarks: USING is one of z/XDC's critical commands. You must learn to use it in order to get the most out of the product. Be advised that there are lots of "bells and whistles" in the Autocloning parameter, which cannot be discussed completely herein due to space considerations. Also, be advised that use of the FLOAT operand can be CPU intensive, and can impact z/XDC's performance. Use the FLOAT parameter with some consideration.

The Autoclone parameter works with the EQUATE command as well as the USING command, with the same parameters.

Issue "USING ?" and you'll get a "Helper Dialog" with blank spaces for all the Using command's parameters. You just fill it in and press Enter!

This command may be used in Foreign Address Space Mode.

Examples: Using ctrblock.ptrfield 243DF ← place the origin address (.ptrfield) at virtual address 243DF.
Using mymap.origin R12!!! F ← place the origin address (.origin) wherever the third level of 64-bit indirection from the contents of register 12 point to (so that "mymap" "floats in memory).

More Info: Enter Help COmmands USing and Help Commands USing Autocloning.

Chapter 4 - Setting Breakpoints (Traps) and Performing Traces

z/XDC (and c/XDC) have robust mechanisms for setting breakpoints (AKA “Traps”), and performing traces. There are two distinct ways in which to do this: By either using X’00’ to cause a S0C1 abend OR by using TRAP2 instructions. z/XDC doesn’t “care” which method you use, but you may care.

The classic X’00’ breakpoint causes Recovery Termination Manager to be involved. This adds to the execution path, and CAN slow things down (not appreciably to a human most of the time). TRAP2 instructions happen at the hardware level, which shortens the execution path.

You may alter your breakpoint setting either in any profile, or by using the commands SET TRACE ZERO or SET TRACE TRAP2.

- Breakpoints may be Permanent (extant until explicitly removed) or Transient (used only one time).
- Breakpoints may be Deferred (scheduled for some time in the future). Deferred breakpoints may also be Permanent or Transient.
- Any number of breakpoints may be set at the same time. They will then be placed into the same “family” (which you probably don’t care about).

To set a Permanent breakpoint you may issue “AT <address-expression>” from the command line OR you may use the “A” shortcut next in column one of a FORMAT’d display.

To set a Transient breakpoint you may issue “T <address-expression>” from the command line OR you may use the “T” shortcut next in column one of a FORMAT’d display.

Permanent breakpoints are defined in this book later on under the “AT” command. Transient breakpoints are NOT documented here, because the syntax of both AT and T are exactly the same.

Once you have set your breakpoint, you may issue the “GO” command to let the program run to your breakpoint.

Traces, on the other hand, follow the execution path of your program allowing you to watch every aspect of a program's logic. The Trace command is documented herein under "Trace". There are seven Traces, which you'll read about further on.

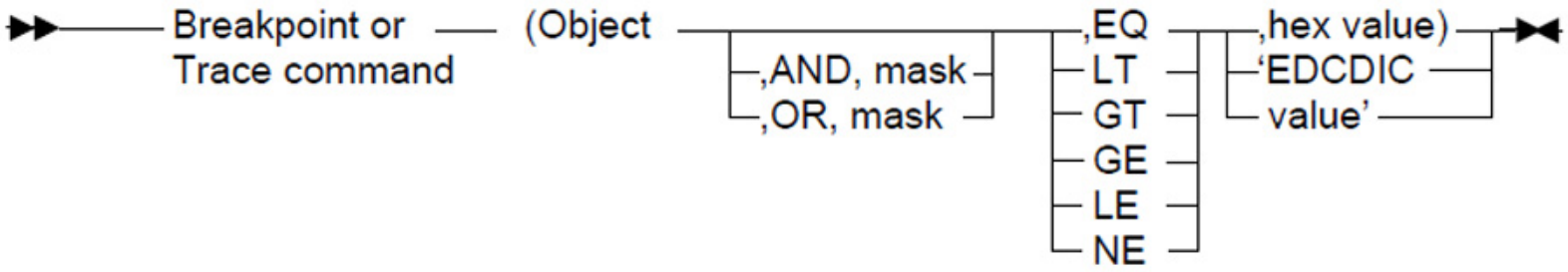
Conditional Statements

Both Breakpoints and Traces accept a conditional statement, of either the counting variety, or the value-testing variety. Using the latter, you can establish Breakpoints or perform Traces that ONLY stop the program when an external condition exists. Going into the mechanics of value-testing conditions is almost beyond the scope of this book (it keeps getting LONGER as I think of more things to say!). Here, however is the anatomy of a value-testing breakpoint (which you can find in the z/XDC Primer).

A Value-testing Conditional Statement:

Value-testing Conditions

Value-testing Conditions allow comparison of the contents of an object (located at some address expression) to a constant that is supplied in the Conditional Expression. All relational operators (Equal, greater-than, less-than, greater-than-or-equal-to, etc.) are supported. You can even enter a bit mask and do the comparison at the bit level by way of "AND-ing" or "Or-ing" bit strings against the supplied bit mask. Here is the syntax of a Value-testing Conditional Expression:



Both "Object" and "value" are four-bytes wide.

As always, Dave Cole says it best himself. Please see:

- Help Breakpoints
- Help Commands Syntax Breakpoints
- Help Breakpoints Conditional
- Help Commands Syntax Breakpoints Conditions

Introducing HOOKs

HOOKs are like the “Uber-Breakpoint”, a huge advance for z/XDC. z/XDC (and c/XDC) both depend upon being the most-recently-defined Recovery Environment. If your code or a subsequent program establishes its own Recovery Environment (through ESTAE, ESTAI, ESTAEX, ARR or FRR), then our product (respecting system integrity) is over-ridden and the debugging session is lost.

HOOK body-slams any other Recovery Environment out of the picture, establishing a debugging session right here/right now. HOOKs are as useful as they are powerful. You don’t want to set a HOOK in a system routine, or any normally running program that YOU DEPEND UPON for your own survival. So, use HOOK with caution.

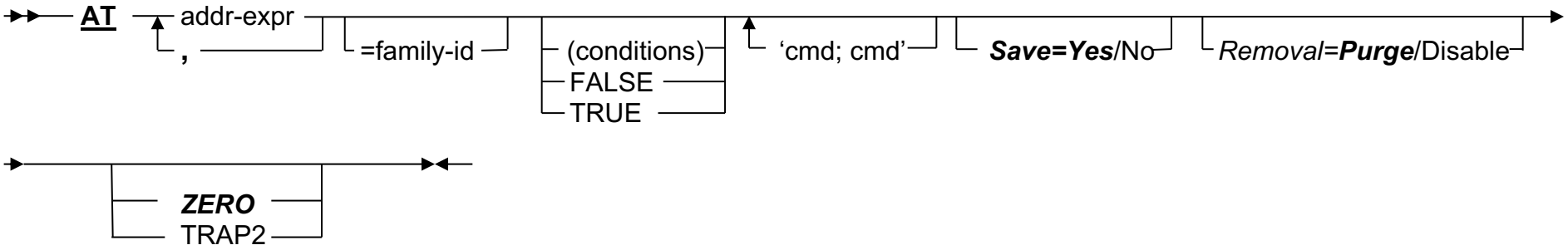
HOOKs come in two “flavors”: Dynamic, and Static.

A Dynamic HOOK is set interactively with “HOOK <address-expression>”, the “K” shortcut command and the “HD command” (which sets up a scheduled HOOK. A Static HOOK is set by invoking the #XDCHOOK macro, or (in C language) the cxdchook() function call.

Static HOOKs are great for interrupting complex programs, such as a transaction processor. You really don’t care to debug all of CICS (even though you could with z/XDC), but you DO care when your transaction loads. That’s a great scenario for a Static HOOK.

Dave’s resources:

- Help Hooks
- Help Commands HOok



Set a permanent breakpoint at a specified location in virtual memory in order to stop program execution at that location.
Short-cut equivalent: "A"

addr-expr	Specify the location where you wish to place the breakpoint. You may specify multiple address expressions separating them with a comma OR a space. Multiple breakpoints created at the same time will be assigned to the same family (a numeric value).
=family-id	Assign this breakpoint to a particular family. This parm may appear anywhere in the command string, prefaced with the equal sign ("**=**") and a family number ranging from 1-999.
(conditions)	Specify a Counting or Value-Testing conditional statement (See Conditional Expressions in this booklet or type Help COmmands Syntax Breakpoints CONditions). **FALSE** will bypass the trap, or cause the Trace to advance one instruction. **TRUE** will cause the trap to be accepted or the Trace operation will stop. FALSE/TRUE were introduced in order to cause a TRACE WATCH to stop program execution – which it couldn't do originally.
'cmd'	Specify other z/XDC commands that you would like to have executed WHEN the breakpoint is reached. Enclose the command string within single quotations. Separate individual commands within a string using a semi-colon.
Save=	When a conditional expression or an associated command string is stated, z/XDC normally saves them for us as defaults in subsequent breakpointing commands. Omitting this parameter defaults the action to Save=Yes. Save=No turns this off.
Removal=	Default action is "Purge". This would certainly apply to transient breakpoints (which are automatically deleted when executed once). "Disable", on the other hand would deactivate the breakpoint (changing the op-from X'00' or X'01FF' back to its original value). This retains the breakpoint, and in the LIST BREAK command's output you will see that the breakpoint is "DISABLED". You can enable or disable a breakpoint using the "X" shortcut command next to any breakpoint. PURGE will cause the breakpoint to be automatically deleted when it is executed. In this case it would be just as easy to create a TRANSIENT breakpoint.
ZERO	This will create a "classic" breakpoint of X'00'. It will cause a S0C1 abend as z/XDC has done for decades.
TRAP2	This invokes z/XDC's TRAP2 support, in which case z/XDC will create a X'01FF' at the breakpoint and will use the TRAP2 facility to create breakpoints. Note: for more information on z/XDC's TRAP2 support, enter "Help Whatsnew z22 Trap2".

Remarks: This is one of z/XDC's core commands; one that you will use in almost every session. TRAP is an alias of this command. You can also specify "ATX" to create a "hard-to-remove" breakpoint (that must be OFF'd explicitly). This command may not be used in Foreign Address Space Mode.

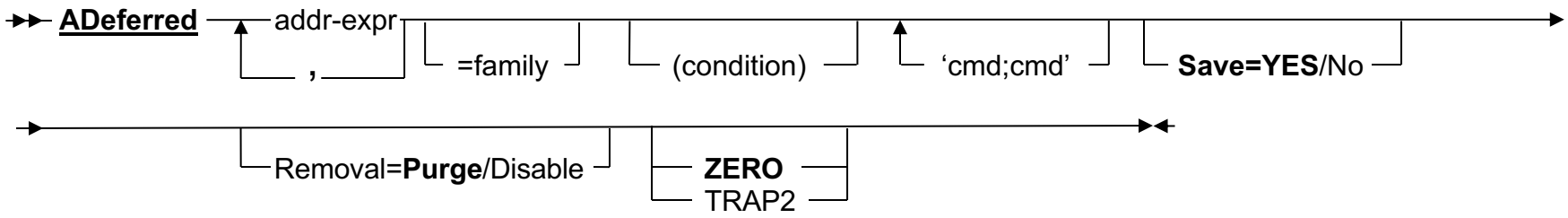
Examples: **AT R14? +4 + 6 .JOBTOP1 +2:LREGS:F R15?** ←Set six breakpoints, then execute a List Registers and a Format at the location pointed to by the contents of R15 whenever the breakpoint(s) is/are taken.

AT myprogram.mycsect+23C ← Set a single breakpoint at offset 23C into the csect "mycsect" of the module "myprogram".

AT .loopstart (R1,NE,7FFFFFFF):ALARM WHOA! IT HAPPENED! ← Set a breakpoint at a label in the program that will only stop execution if the contents of Register One are no longer equal to X'7FFFFFFF' Then, automatically create an alarm condition to alert the programmer.

AT R5+10 (R3+3,AND,02,EQ,02) Set a breakpoint that will not be accepted until the 31st bit (i.e., bit number 30) of R3 is on.

More Info: Enter Help COmmands AT. For conditions see HELP Breakpoints Conditional and Help Commands Syntax Breakpoints Conditions.



Create a deferred breakpoint in the desired module (permanent for the life of the session or until manually removed).

addr-expr Address expression(s) that names a module, a module.csect and (if desired) the offset where the deferred breakpoint is to be placed. The "base" of the addr-expr MUST be the primary name of a load module. Separate multiple address expressions with commas OR spaces.

=family A breakpoint family may be specified (a series of breakpoints named alike). The breakpoint will then be assigned to that family of breakpoints.

(condition) A conditional expression may be specified to further control under which situations the breakpoint is to be taken.

'cmd;cmd' An additional z/XDC command (or commands) that are to be executed when the breakpoint is taken. New for z/XDC Release z2.1, you no longer may use colons to associate commands with traces or breakpoints. Instead you enclose the command string within single quotations and separate commands within a string with a semi-colon. See Help Whatsnew Z21 Syntaxchange! for more information.

Save= When a conditional expression or an associated command string is stated, z/XDC normally saves them for us as defaults in subsequent break-pointing commands. Omitting this parameter defaults the action to Save=Yes. Save=No turns this off.

Removal= Default action is "Purge". This would certainly apply to transient breakpoints (which are automatically deleted when executed once). "Disable", on the other hand would deactivate the breakpoint (changing the op-from X'00' or X'01FF' back to its original value). This retains the breakpoint, and in the LIST BREAK command's output you will see that the breakpoint is "DISABLED". You can enable or disable a breakpoint using the "X" shortcut command next to any breakpoint. PURGE will cause the breakpoint to be automatically deleted when it is executed. In this case it would be just as easy to create a TRANSIENT breakpoint.

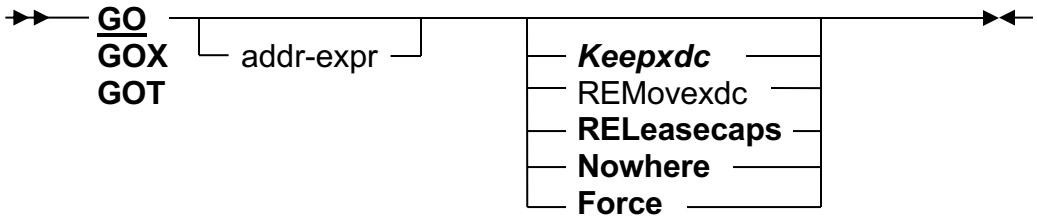
ZERO This will create a "classic" breakpoint of X'00'. It will cause a S0C1 abend as z/XDC has done for decades.

TRAP2 This invokes z/XDC's TRAP2 support, in which case z/XDC will create a X'01FF' at the breakpoint and will use the TRAP2 facility to create breakpoints.
Note: for more information on z/XDC's TRAP2 support, enter "Help Whatsnew z22 Trap2".

Remarks: This command creates a breakpoint in a module that will be loaded into memory sometime in the future. This is a "scheduled" breakpoint. A breakpoint created by ADeferred remains in existence for the life of the debugging session or until it is explicitly removed. If you desire a one-time-only deferred break-point, see TDEFERRED.
This command may not be used in Foreign Address Space Mode.

Examples:
AD MYPGM+3DC ← Create a Deferred breakpoint at offset "+3DC" into module MYPGM whenever it loads.
AD MYPGM.MYCSECT+3DC ← Create a Deferred breakpoint at offset "+3DC" into module MYPGM and csect MYCSECT whenever it loads. You would have had to issue a DMAP command for MYPGM prior to this so that z/XDC can "find"csect MYCSECT.
AD =5 MYPROG+3DC +4 +8 (R1,EQ,00) :L REGS:L AREGS ← OK, this is a toughie. Set a Deferred breakpoint into Family "5" at MYPROG+3E8 but only if Register one is equal to all zeroes. If and when this Transient Deferred breakpoint fires, then issue "LIST REGS and LIST AREGS.

More Info: Enter Help COmmands ADeferred. For conditions see HELP Breakpoints Conditional and Help Commands Syntax Breakpoints Conditions.



Release the program being debugged for processing by the operating system.

addr-expr The desired resume address. If omitted, then z/XDC uses the address pointed to by the retry-level Program Status Word (PSW).

KEEPxdc This is the default. All current instances of z/XDC will remain and the current recovery environment will remain unchanged.

REMovexdc If z/XDC got control in this debugging session via a HOOK, then this parameter will remove the current instance (only) of z/XDC from your program's recovery environment. If other instances of z/XDC are pending in the recovery stack they will remain in effect. This parm is intended to remove instances that have been established dynamically via the Hook command (or the old Hook script). It is NOT a comprehensive way to remove z/XDC under all circumstances. Read and understand the Help Subsystem's instructions before use of this parameter.

RELeasecaps This ends a debugging session without also ending the program being debugged. The program resumes but all z/XDC-related processes are terminated, including all maps, all breakpints all equates and all dsects.

Nowhere This causes z/XDC to resume the program and return to z/XDC right away, which allows Recovery Termination Manager to reschedule z/XDC. This parameter is intended to work with z/XDC's recently added SET AUTH command, which allows a user to make any debugging session APF-authorized (subject to system security rules). See Help COMmands SEt AUTH for more information.

Force When using TRAP2 instructions for trapping and tracing with z/XDC, the Trap Save Area can become corrupted. In this case message DBC799 will be displayed. It is then incumbent upon the user to restore the proper values to the PSW, General Registers and AR15. One may then issue the GO FORCE command to resume the debugging session. Note: It is beyond the scope of this book to discuss how the Trap Save Area corruption can occur, but an excellent overview can be found by entering "Help Messages DBC799".

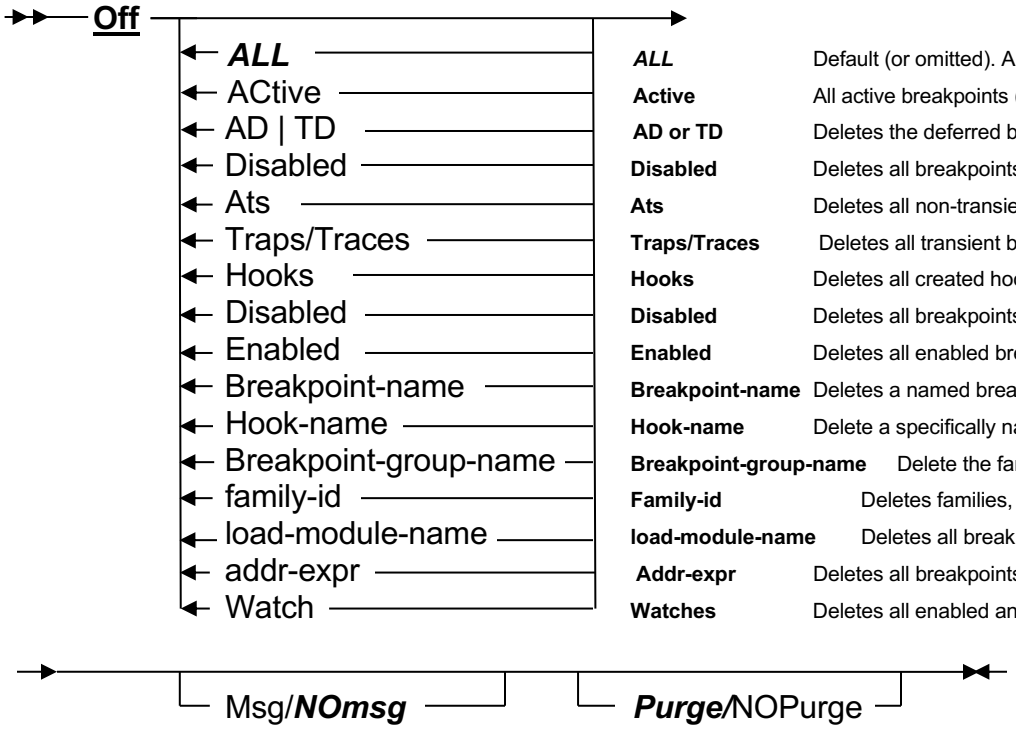
Remarks: This is a CORE command in z/XDC. The user's program will resume execution, and will stop at the next breakpoint or abend. GO, GOX and GOT are equivalent, with minor differences. With GOX, z/XDC will attempt to restore whatever information was being displayed prior to the GOX command. GOT causes the user program to be resumed without clearing the display screen.

We recommend that you read the Help Subsystem on this command, especially with attention paid to the various parameters above. There are a lot of moving parts here.

This command may not be used in Foreign Address Space Mode.

Examples:
GO R10! ← The user's program is resumed NOT where the PSW points to, but where the contents of R10 point to (in 64-bit mode).
GO REM ← IF z/XDC got control of the debugging session by means of a Hook, then the program is resumed and z/XDC is removed from the picture.
GO N ← Used in conjunction with the SET AUTH command, this can help a previously unauthorized debugging session in another address space to become APF-authorized.

More Info: Enter Help COmmands GO.



ALL	Default (or omitted). All breakpoints are deleted.
Active	All active breakpoints (except ATX breakpoints and HOOKs) are removed.
AD or TD	Deletes the deferred breakpoints created by ADeferred and TDeferred commands.
Disabled	Deletes all breakpoints that have been previously disabled (via the "X" line command).
Ats	Deletes all non-transient breakpoints (those defined by AT and ADeferred commands).
Traps/Traces	Deletes all transient breakpoints (those defined by TRACE, TRAP and TDeferred).
Hooks	Deletes all created hooks.
Disabled	Deletes all breakpoints that have been previously disabled (via the "X" line command).
Enabled	Deletes all enabled breakpoints.
Breakpoint-name	Deletes a named breakpoint such as "AT0000nc" or "TR0000nc".
Hook-name	Delete a specifically named HOOK.
Breakpoint-group-name	Delete the family of breakpoints such as "AT0004".
Family-id	Deletes families, such as "*0003" regardless of their "AT" or "TR" prefix.
load-module-name	Deletes all breakpoints in the load-module (except ATX breakpoints).
Addr-expr	Deletes all breakpoints at this address.
Watches	Deletes all enabled and disabled Trace Watch commands.

Remove breakpoint(s) with any specified level of granularity.

Line mode equivalent: "O"

msg/NOmsg Enable or suppress z/XDC's messages as it executes OFF.

Purge/Nopurge Purge (default) will remove the breakpoint(s) from existence. Nopurge will merely disable them, leaving their definitions intact. You may later re-enable them with a subsequent "X" short-cut command.

Remarks The OFF command removes breakpoints with a surprising amount of granularity. However, keep in mind that one can also use the "O" short-cut command to accomplish the same result from a LIST BREAK display and/or wherever a breakpoint definition appears in a formatted z/XDC display. Also, you may use the "X" Line Command to "toggle on/off" a breakpoint definition, retaining the breakpoint for later use without deleting it.

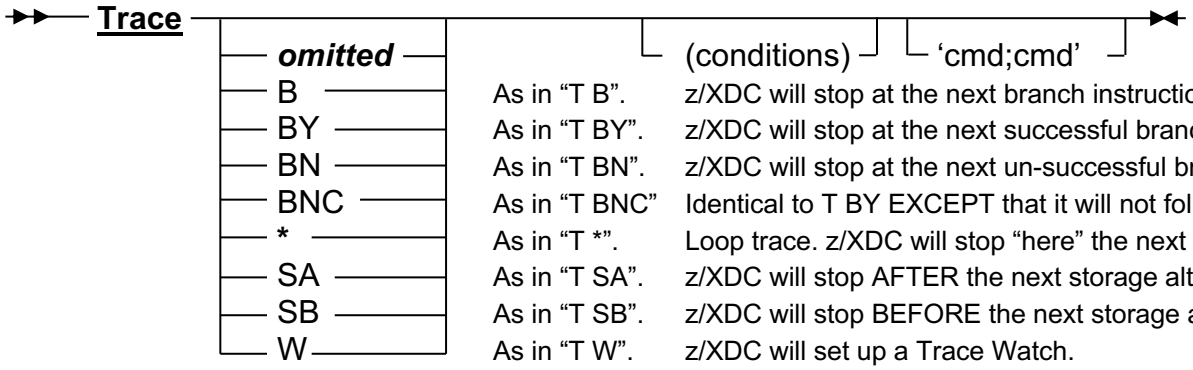
Examples:

O ←All breakpoints are deleted.

O AC ←All non-deferred breakpoints are deleted (except ATX breakpoints – which are persistent until explicitly deleted by name or the "O" shortcut command).

OF HOOK ← All dynamically created Hooks are deleted (not static ones created by the #XDCHOOK macro or cxdchook() function).

More Info: Enter Help COmmands Off. See also Help COmmands SEt Breakpoints. See also Help COmmands LISt BREakpoints.



Follow the execution path of a program.

Omitted z/XDC will return control after a single instruction.

(conditions) A counting condition or a value-testing condition.

'cmd;cmd' Another or many z/XDC commands to be executed when the trace takes effect. New for z/XDC Release z2.1, you no longer may use colons to associate commands with traces or breakpoints. Instead you enclose the command string within single quotations and separate commands within a string with a semi-colon. See Help Whatsnew Z21 Syntaxchange! for more information.

Remarks: Trace is one of the core commands with in z/XDC, so much so that the "factory default" Profile has PF9 set to "T", and PF10 set to "T BY".

Neat feature: Whenever z/XDC "stops" at a Branch-type instruction it automatically evaluates the branch. If the branch will be taken, then the branch target is displayed next to the instruction within parentheses. If the branch will fall through, then z/XDC displays "(NO BRANCH)". Nice.

More information on the parameters of the Trace command may be found above in this booklet in the section entitled, "z/XDC's Trace Commands".

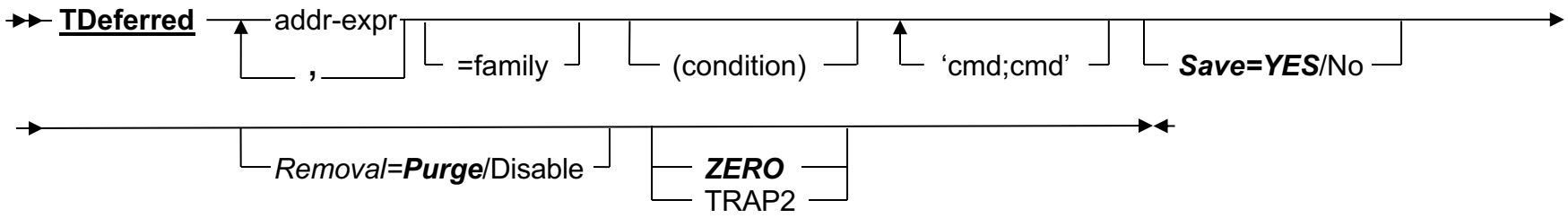
A Trace Watch is a special case that sets up a conditional statement that is not tied to any breakpoint or trace, but is evaluated WHENEVER z/XDC gains control. Trace Watch is the very next command discussed in this book.

Dave wrote T BNC because sometimes in his AUTOTRACE script, z/XDC would go down into the system and never resurface (down the "rabbit hole"). This often made the AUTOTRACE Script useless. Now he's incorporated the T BNC command into a new script called AUTOTBNC. See Help Scripts Ourscripts AUTOTbnc for more information.

This command may not be used in Foreign Address Space Mode.

Examples: T ← Execute to the next sequential instruction
T BY 'LIST PSW F:LIST REGS:RETRIEVE' < ← Trace to next successful branch. When there, also show the PSW (broken down by fields) as well as the general registers. Finally a RETREIVE command is processed which puts this command string back on the command line. Then all you have to do is keep pressing Enter to keep running the command. I LOVE THIS.

More Info: Enter Help COmmands TRACe, and Help Commands TRAce Watch. See also Help COmmands SYntax Breakpoints.



Create a transient deferred breakpoint in the desired module.

addr-expr Address expression(s) that names a module, a module.csect and (if desired) the offset where the deferred breakpoint is to be placed. The "base" of the addr-expr MUST be the primary name of a load module. Separate multiple address expressions with commas OR spaces.

=family A breakpoint family may be specified (a series of breakpoints named alike). The breakpoint will then be assigned to that family of breakpoints.

(condition) A conditional expression may be specified to further control under which situations the breakpoint is to be taken.

'cmd;cmd' An additional z/XDC command (or commands) that are to be executed when the breakpoint is taken. New for z/XDC Release z2.1, you no longer may use colons to associate commands with traces or breakpoints. Instead you enclose the command string within single quotations and separate commands within a string with a semi-colon. See Help Whatsnew Z21 Syntaxchange! for more information.

Save= When a conditional expression or an associated command string is stated, z/XDC normally saves them for us as defaults in subsequent break-pointing commands. Omitting this parameter defaults the action to Save=Yes. Save=No turns this off.

Removal= Default action is "Purge". This would certainly apply to transient breakpoints (which are automatically deleted when executed once). "Disable", on the other hand would deactivate the breakpoint (changing the op-from X'00' or X'01FF' back to its original value). This retains the breakpoint, and in the LIST BREAK command's output you will see that the breakpoint is "DISABLED". You can enable or disable a breakpoint using the "X" shortcut command next to any breakpoint. PURGE will cause the breakpoint to be automatically deleted when it is executed. In this case it would be just as easy to create a TRANSIENT breakpoint.

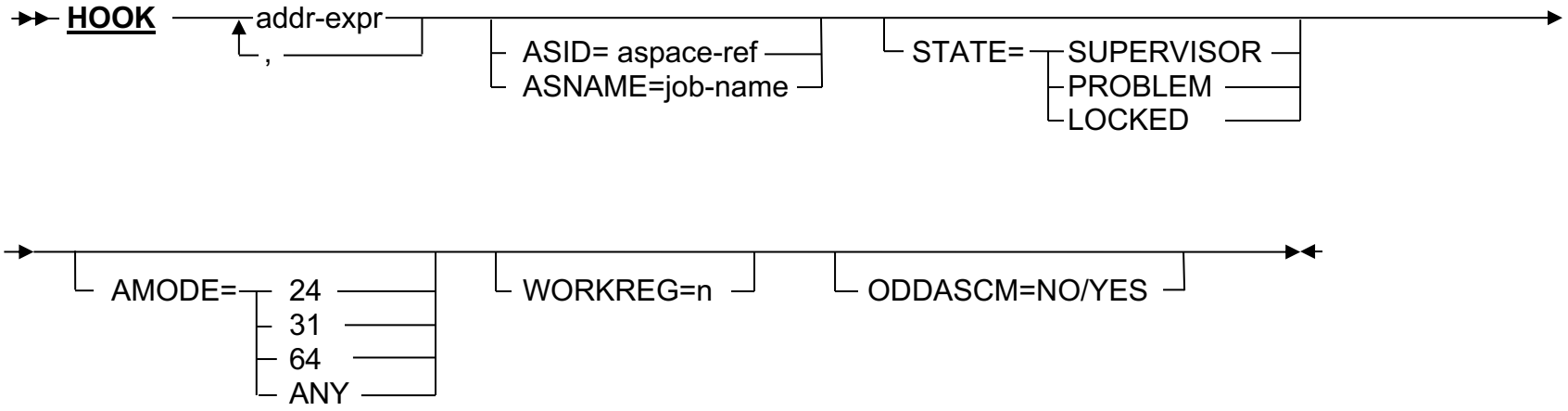
ZERO This will create a "classic" breakpoint of X'00'. It will cause a S0C1 abend as z/XDC has done for decades.

TRAP2 This invokes z/XDC's TRAP2 support, in which case z/XDC will create a X'01FF' at the breakpoint and will use the TRAP2 facility to create breakpoints.
Note: for more information on z/XDC's TRAP2 support, enter "Help Whatsnew z22 Trap2".

Remarks: This command creates a breakpoint in a module that will be loaded into memory sometime in the future. This is a "scheduled" breakpoint. A breakpoint created by ADeferred remains in existence for the life of the debugging session or until it is explicitly removed. If you desire a one-time-only deferred break-point, see TDEFERRED.
This command may not be used in Foreign Address Space Mode.

Examples:
AD MYPGM+3DC ← Create a Deferred breakpoint at offset "+3DC" into module MYPGM whenever it loads.
AD MYPGM.MYCSECT+3DC ← Create a Deferred breakpoint at offset "+3DC" into module MYPGM and csect MYCSECT whenever it loads. You would have had to issue a DMAP command for MYPGM prior to this so that z/XDC can "find"csect MYCSECT.
AD =5 MYPROG+3DC +4 +8 (R1,EQ,00) :L REGS:L AREGS ← OK, this is a toughie. Set a Deferred breakpoint into Family "5" at MYPROG+3E8 but only if Register one is equal to all zeroes. If and when this Transient Deferred breakpoint fires, then issue "LIST REGS and LIST AREGS.

More Info: Enter Help COmmands ADeferred. For conditions see HELP Breakpoints Conditional and Help Commands Syntax Breakpoints Conditions.



Create a debugging session regardless of prior existing recovery environments
Shortcut command equivalent: "K"

addr-expr An address expression that names a module, a module.csect and (if desired) the offset where the deferred Hook is to be placed. The "base" of the addr-expr MUST be the primary name of a load module.

ASID= Specify a specific address space that currently exists. ASID can be a 4-digit hexadecimal number (presumably from the Address Space Vector Table), a 1-8 character job name, or a keyword such as HOME, PASID, etc. For more information see HELP COMMANDS SYNTAX ASIDS.

ASNAME= Specifies the address space by name. Here's an important distinction: If you use ASNAME=jobname and the address space does not yet exist, AND if it will exist in common storage, then the hook is created anyway. Conversely, if you use ASID=jobname and the address space doesn't yet exist, the hook command will fail.

STATE= Omit this operand to have the Hook set in the current execution state. Use this operand only if the state of the Hook Point is going to be different from the execution state of the current code at the moment you set the deferred Hook.

AMODE= Omit this operand to have the Hook set in the current addressing mode. Use this operand only if you wish the hook to be placed in an addressing mode that is different from the current addressing mode. ANY directs z/XDC to use the current addressing mode.

WORKREG=n A register that can be used by the generated Hook processing code that is specific to this HDEFERRED request. You need this if a) you wish to hook address that are above the bar, or b) you wish to hook code that may be executing in secondary or in home ASC-MODE (in which case you will also need to specify ODDASCM=YES). The register must range from 0-F and the Workreg will be set to all zeroes.

ADDASCM=NO/YES Indicates that the hook needs to support code that is executing in secondary or home ASC-MODE. Failure to use this parm when executing in secondary or home ASC-MODE will result in a S0D3 abend. So, if you specify ODDASCM then you MUST also specify WORKREG

Remarks: The HOOK command creates a z/XDC session "on the fly".

This is an extremely cool command! z/XDC's HOOK facility (of which HDeffered is a part) completely solves the conflict between z/XDC's ESTAE and ones defined in your own programs. Consider HOOK as the "uber-breakpoint", which WILL give you control wherever you want no matter what other recovery structures are in place.

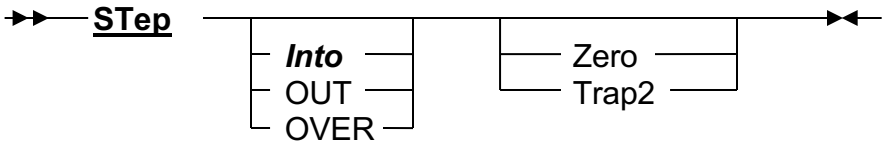
[How it was: Back in the old days, if you set a breakpoint "over there" in storage and entered GO, by the time execution reached the breakpoint any number of other ESTAEs could have been issued. Then, z/XDC lost control of the debugging session, and your application went down with a S0C1. Bad juju. HOOK solves this problem.]

Hook is available in four ways: As an interactive command, or via the Shortcut command "K", as a deferred HOOK (see Help COmmands HDeffered) or as a macro call ("#XDCHOOK").

With Release z2.2, Hook's scope has been greatly expanded. It is no longer SVC-based, and can now be used in PC routines, SRB routines, FRR-protected code, Locked code, Disabled code as well as in its traditional environments.

Examples: Hook mypgm.mycsect ← A Hook will be placed so that an z/XDC session will initialize when the code executes.
HOOK +A ← Set a hook X'10' bytes beyond where z/XDC is looking "now" (the location of the Current Display Pointer. Could just as easily be done with a "K" next to any line of code.

More Info: Enter Help Debugging Hooks. See also Help COmmands LISt Hooks. You REALLY should familiarize yourself with this powerful command. There are important considerations well beyond the scope of this book.



Execute one High-Level Language statement (such as C, C++ or Metal C).

omitted Default action. Execute a single statement. If the next statement happens to be a subroutine, execution will begin there.
INto Step INTO a subroutine. If execution is NOT at a subroutine, c/XDC merely issues a STEP command, returning control at the next statement.
OUT Step OUT OF a subroutine.
OVER Allow a subroutine to execute, and return control after its execution
Zero Perform the STep command and force z/XDC to use a X'00' to do so.
Trap2 Perform the STep command and force z/XDC to use a TRAP2 instruction (X'01FF') to do so.

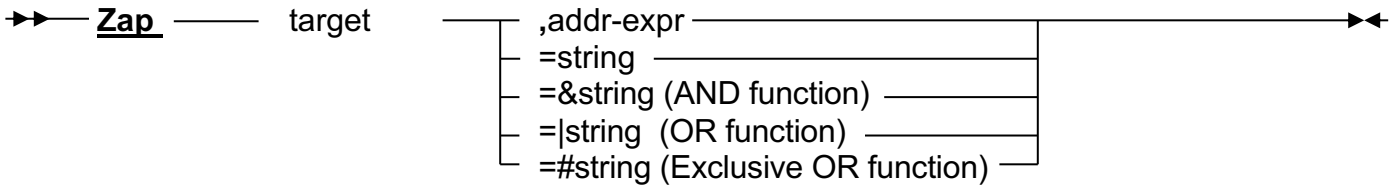
Remarks: The Step command allows you to trace a single High Level Language statement, which can encompass many machine instructions. It's one of the most useful features in c/XDC. Remember, c/XDC's interaction with High-Level Languages doesn't mean that z/XDC isn't available to you. You can still SET FORMAT BOTH to see raw object code AND source statements together and you can still issue TRACE, and breakpoint commands.

At this writing STEP INTO is a system default, and Dave Cole gives a pretty convincing argument why this is so in Help COmmands STep. He also says that if you want to change this behavior you can, but he neglects to tell you how. I did the research. You can change this with SEt STep.

Examples: ST ← Execute one High Level Language statement
ST OUT ← leave the current subroutine and return to the calling routine.

More Info: Enter Help COmmands STep.

Chapter 5 - Changing Things



Over-write the contents of one memory location with the contents of another memory location, or with a stated value. Shortcut command equivalent: "Z"

target The location in memory that you wish to over-write. This can be within the Private Area, the Common Area, a Dataspace, Foreign Address spaces, real storage, the registers, and the PSW. For zapping read-only storage, see Help COmmands Set Zap.

,addr-expr The address-expression whose value is to be written to the target. You must use either a comma, or a blank to delimit the second address-expression (but not both!).

=string A stated value that you wish overwrite the target with. There can be NO space between "target" and the =string value.

=&string The string is to be AND'd, byte by byte, into the "target address-expression."

=|string The string is to be OR'd, byte by byte, into the "target address-expression." You can use "|" or "¦".

=#string The string is to be Exclusive OR'd, byte by byte, into the "target address-expression."

Remarks: Zap is one of the most useful and powerful of z/XDC's commands. Accordingly, Zap must be used with caution, and care. You can zap literally anywhere if your security system will allow you to do so. **THIS MEANS THAT YOU CAN CRASH A SYSTEM WITH AN INCORRECT ZAP!**

To use Zap as a shortcut command, put a "Z" in column one of a z/XDC display, tab to the item you wish to change, and merely overtype the old value with the new value! You may overtype any field on the display in which a TAB operation stops the cursor.

If you wish to zap the PSW to change the execution location, then Zap will work on ONLY resume address portion of the PSW (which cannot be "overtyped" as just described). In that case ZAP may ONLY be issued from the command line. That is, "ZAP PSW <address-expression>". For other changes to the PSW, see the SET PSW command.

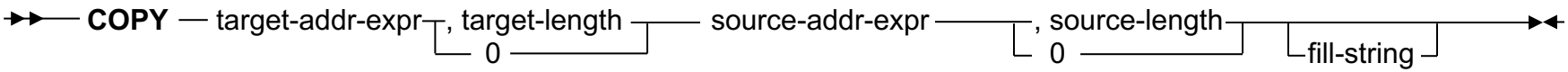
This command may be used in Foreign Address Space Mode. This is powerful, and scary. ***Vaya con Dios, pero tenga CUIDADO!***

For zapping read-only storage, see Help COmmands Set Zap.

z/XDC is now case-agnostic and accepts mixed-case zap strings. For idle research, see Help COMmands SEt ASIS and LISt ASIS.

Examples: Z R0 0FFFFFFF ← Overwrites R0 with the new value of "0FFFFFFF".
ZAP .LABEL='SOME TEXT' ←write the upper case letters "SOME TEXT" into the address expression ".LABEL".

More Info: Enter Help COmmands Zap and Help COmmands Zap Targets. See also Help COmmands Zap Boolean, Help Commands Verify, and SET PSW. If you want to move "chunks" of storage around, see Help COmmands COpy.



Propagate the contents of memory to one location from another (with optional filling).

target-addr-expr Where the user wishes to copy the source memory contents to. Can also be a retry level register, a general register, an Access Register or a Floating point register.

target-length Length of the target location. Can be a constant, a hexadecimal number, a decimal number (when suffixed by "N"), the contents of a register, an arithmetic expression, the contents of a location in storage or the resolved value of an address expression.

source-addr-expr Where the user wishes to copy from. If source-addr-expr is omitted, then source-length must also be omitted.

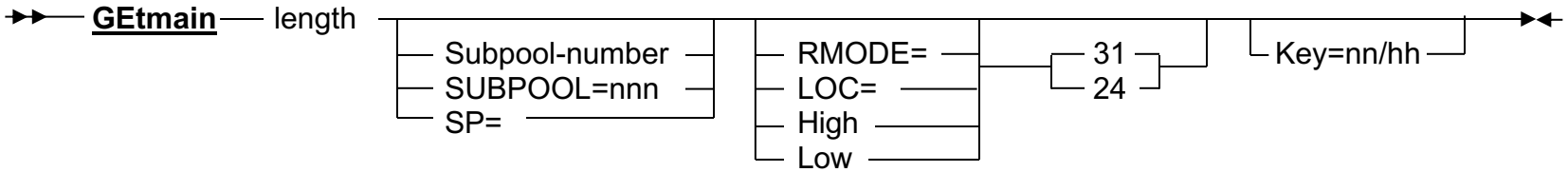
source-length Length of the source location. Can be a constant, a hexadecimal number, a decimal number (when suffixed by "N"), the contents of a register, an arithmetic expression, the contents of a location in storage or the resolved value of an address expression. If source-addr-expr and source-length is omitted, then target-length will be used as the default source-length.

fill-string An optional padding string of up to 256 characters. Character data is prefaced by a single quote. If omitted, then X'00' will be used as the default fill-string.

Remarks: If the target length is shorter than the source length, then the source is truncated. If the source-length is shorter than the target-length, then padding from a fill string is copied to the target location immediately following the data copied from the source. A source length of zero causes the target location to be filled entirely with padding from the fill string.
This command may not be used in Foreign Address Space Mode.

Examples: COPY R12! 300N mymod.mycsect.buffer 300N ← Copies 300 decimal bytes of data in a buffer to the memory pointed to (in the existing address mode, of 24 or 31-bit memory) by general register 12.
COPY R4?~ASID(pasn) 200N R3?~ALET(AR5) 200N ← Copies 200 (decimal) bytes from the location pointed to by general register 3 in the address/data space indicated by access register 5. The data is copied to the location pointed to by general register 4 in the retry level Primary Address Space.

More Info: Enter Help COmmands COPy.



Obtain a portion of memory from a specified subpool.

length The desired amount of memory, given as a hexadecimal number. The STORAGE OBTAIN service will automatically round this up to an 8-byte multiple.

Subpool-number/ SUBPOOL=nnn/SP= The desired subpool identifier, given as a decimal number.

High The desired memory is to be located "above" the 16MB line, in 31-bit storage. An alias is RMODE=31 or LOC=31.

Low The desired memory is to be located "below" the 16MB line, in 24-bit storage. An alias is RMODE=24 or LOC=24.

omitted If neither High or Low are specified, z/XDC will obtain the memory from either above or below the line depending on the addressing mode of the program being debugged. The user may issue LIST PSW FORMAT to inspect the address-mode bit, if desired.

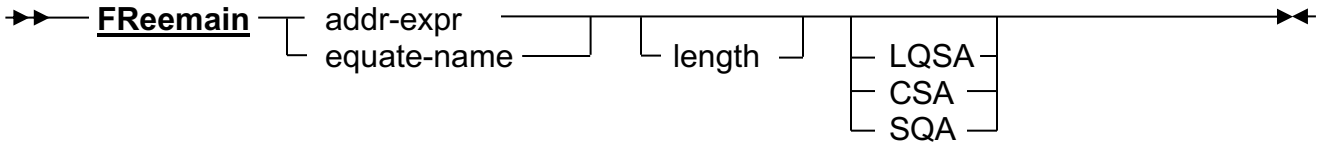
Key=nn/hh Requests that the storage be assigned to a particular key. The parameter may be specified in decimal (Key=11) or hexadecimal (Key=0B, or key=B). HOWEVER, certain rules apply to using this parameter. For one thing, it is effective only for multi-key subpools. Also, when z/XDC is APF-authorized, the System will only accept key values permitted by your program's key mask. Typically, you would be limited to Key=8 or =9. Read the Help Subsystem for important information on this parameter.

Remarks When GEtmain is issued, z/XDC creates two equates in memory: "AREA" and AREA #nn (where nn is the number of times you've issued the GEtmain command in this debugging session). On any subsequent GEtmain command, AREA will be updated to the most recently allocated storage and AREA#2 (or larger) will be created. This way, you can issue GEtmain in a script and refer to the most recently "got-mained" storage as "AREA".

This command may not be used in Foreign Address Space Mode. Well, you CAN use the command, but the memory will be allocated to YOUR address space, not the Foreign one.

Examples: GE 100,78 ← z/XDC will obtain X'100' bytes of storage from subpool 78. It will also create two Equates:
AREA ← This Equate is created or updated every time the GEtmain command is executed. It always points to the most recently obtained storage.
AREA#n ← This Equate uniquely identifies each area of storage created by a GEtmain command.

More Info: Enter Help COmmands GEtmain. See also Help COmmands FReemain.



Release an area of storage that was previously obtained by a GETMAIN macro or a z/XDC GETmain command.

addr-expr The address of the area to be freed. This can be an Equate that resolves at the start of or inside the storage to be freed.

Since any GETMAIN command issued within z/XDC automatically receives an equate name of “AREA” (for the most recently executed GETMAIN) and AREA#nn (where nn is the sequential number assigned by any GETMAIN command) it’s often handy to issue “FREEMAIN AREA” (for the most recent extent created with GETMAIN), or “FREEMAIN AREA#nn” (for any other storage extend created with any other GETMAIN).

FREEMAIN will also delete any other equates defined within this storage, Finally, FREEMAIN will fail unless addr-expr is doubleword aligned.

length This is a hexadecimal value giving the length of storage to be freed. The FREEMAIN SVC rounds this value up to an 8-byte multiple. This operand is required unless the address operand is a pure equate name in which case the length operand can be omitted and the length of the equate is used in its place. Bad news: If you FREE an area less than the length of the extent you created with GETMAIN, then *you will have created your own memory leak*, as “orphaned” storage will remain. Good news: At the end of the debugging session z/XDC will clean this up for you.

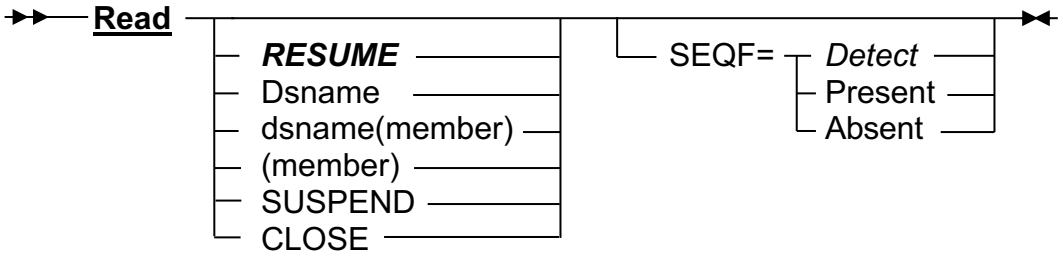
Equate-name If the user invokes the Freemain command with the z/XDC-defined equate name (via the GETMAIN command), then the addr-expr and length parameters need not be given (z/XDC always assigns an equate name of AREA#n to the memory accessed via an z/XDC GETmain command).

LQSA, CSA, SQA These parms may be specified when z/XDC is running APF-authorized. If the storage to be freed resides within one of these areas, then you MUST specify this parameter to avoid freeing storage from these areas accidentally.

Examples:

FR AREA#3	← The storage represented by the equate AREA#3 is freed, along with any equate names that occur within the memory’s extent.
FREEMAIN FRED 50	← z/XDC will free X’50’ bytes of storage beginning at the storage location beginning at the address of the equate “FRED”.
FR F00000,100,SQA	← If z/XDC is running authorized, and if location X'00F00000' falls within the System Queue Area (SQA), and if that storage currently is allocated, then this command frees it. (This command fails when any of the above listed conditions are not met.)

More Info: Enter Help COmmands FReemain. See also Help Commands GEtmain.



Cause z/XDC to obtain and execute commands from a previously created script.

RESUME Default. If a dataset has already been opened for reading, then was suspended (due to an error), execution is resumed.

dsname The name of a sequential dataset containing subsequent z/XDC commands to be read.

dsname(member) The name of a specific member of a partitioned dataset. It must be fully qualified and unquoted.

(member) If a previous Read command referenced a member of a partitioned dataset, then specifying Read (member) causes z/XDC to access the same PDS for the indicated member.

SUSPEND If a dataset is currently being processed, then execution is suspended. This parameter can only be issued from within the Read dataset itself.

CLOSE If a Read dataset has been suspended, then this parameter will close the dataset. If the user wishes to re-initiate a Read dataset that had previously been SUSPENDed, then the Read CLOSE command must be entered before the original READ command may be entered again.

SEQF= If you have decided to number the separate statements in your script, then by convention sequence numbers take up 8 decimal digits. But with the decline of the importance of sequence numbers, this may be vestigial. In any case, you can let z/XDC handle your sequence numbers for you, and NOT screw up your script by attempting to treat sequence numbers as commands in the script. SO, bottom line: it's best NOT to use sequence numbers in scripts. If you already have them in your scripts, you'll have to deal with them.

SEQF has two synonyms, each involving more key-strokes: SEQUENCEFIELD, and SEQFIELD.

SEQF= may have three parameters:

- ***Detect*** This is the default. z/XDC will do its best to determine the presence or absence of sequence fields.
- **Present** The sequence field is present in all records, and will be ignored by the READ command.
- **Absent** The sequence field is absent, and z/XDC will attempt to process everything it finds in the script dataset.

Remarks: Whenever you find yourself typing a series of commands over and over, DON'T. Take the time to create a script of those commands, and issue the READ command to execute them en masse! Scripts save tons of time! In many, if not most of our customers' shops there are one or two users who build and maintain scripts. Go make friends with that person.

Some rules about scripts:

- Scripts are "lists" of declarative commands. There's no conditional logic, but Dave insists that you can get a surprising amount done with simple declaratives (See Help SCripts Ourscripts for examples).
- If you want to investigate conditional logic (outside of scripts), check out HELP REXX.
- Scripts CAN be recursive. There is no reason that a script cannot itself issue the READ command.
- A script will not survive the "GO" command. If you want other things to happen after a "GO" command, make a second script.

More Info: Enter Help COmmands Read. See also Help SCripts Ourscripts and Help Scripts Ryoscripts (for "write your own").

The SET PSW Command

If you issue “ZAP PSW <address-expression> you can change where the Instruction Location Counter portion of the psw points to, in storage. In effect you’re changing the execution location of the program. This is directly equivalent to the “J” (Jump) Shortcut command.

However, if you want to change OTHER portions of the PSW, you must issue SET PSW.

In the Railroad Track reference, you will find that there are EIGHT variations of the SET PSW command. Each affects another portion of the PSW (such as the AMODE bit, the DAT ON/OFF bit, etc.)

If I were to include each of these eight variants of the SET PSW command, this booklet would be way too long for its purpose as a “quick reference”. So, I leave it to you to start a z/XDC session and issue Help COmmands SEt PSw.

Chapter 6 - Working In Other Address Spaces

z/XDC allows you to work in other addresses in two distinct ways:

Foreign Address Space Mode

Accessed via the SET ASID command, this allows you to go anywhere/look at anything based upon your security rules. However, this environment is constrained. You can look, but you can't touch. That said, if security lets you, you CAN ZAP and you CAN Hook. This is an advanced topic that I go over in my classes, and exceeds the mandate of this booklet. Ask me, I'll tell you how to do this.

Cross Domain Facility

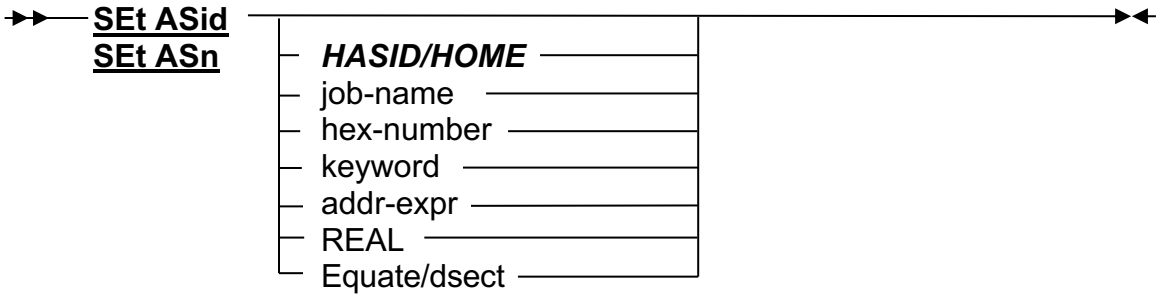
If you alter your EXEC statement from:

"EXEC PGM=<my-program and its parms>"
to
"EXEC PGM=XDCCALL, PARM=<my-program and its parms>"

Then z/XDC will execute your program in a batch address space and you may connect to it using Option 3 from the z/XDC Startup Panel.

You may invoke other DD statements in your batch job's JCL, but changing the EXEC statement is the only change that you MUST make.

Here are some of the commands you may use in Foreign Address Space Mode and Cross Domain Facility:



Use Foreign Address Space Mode to change from one address space to another.

HASID/HOME **(or omitted)** If one is currently in another address space, z/XDC will return the user to the "home" address space. So, "SET ASID" gets you "home".

job-name The name of a particular job, such as "JES2", **"**VTAM", "CICS" and "DB2".

hex-number The 1 to 4-digit hexadecimal number that identifies an existing address space (an index into the Address Space Vector Table).

keyword **HASID** - The user's "home address space". Mentioned above as the default. **PASID** - The retry level of the Primary address space. **SASID** - The secondary address space. **EPASID** - The error-level instance of the Primary address space. **ESASID** - The error-level instance of the Secondary address space.

Addr-expr A address expression that points to an ASID number.

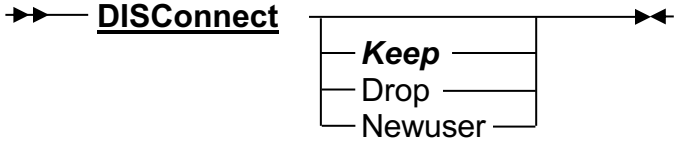
REAL The user wishes to display (and perhaps zap) real storage. This choice is NOT for amateurs, and (of course) is subject to security rules AND your personal discretion. Tread carefully here.

Equate/dsect An equate or dsect that has been assigned to represent storage in an address space.

Remarks: **The user must have APF-authorization in order to use this command (as well as appropriate permission from the security system).** Foreign Address Space Mode (AKA "FASM") allows the user to investigate programs running in other address spaces on an ad-hoc basis. This is an extremely powerful and useful facility.

If you wish (and if allowed by your security system), you may access another address space using FASM, and HOOK a running program, then get out of FASM and connect to it using Cross Domain Facility (Option 3 from the z/XDC Startup Panel). Now HOLD ON: This is powerful/scary stuff, and you really must know what you're doing when you use FASM to hook another address space. Proceed with caution. If you crash anything important, it will be on YOU, not z/XDC.

More Info: Enter Help COmmands SEt ASId. Also Help CDf Fasm, and Help FUNctions ASId. In addition, see Help COmmands SYntax ASids. While you're at it, see Help Security.



Sever the connection between a batch debugging session and a TSO or VTAM user without terminating the session.

Keep Default. (same as omitting any parm). The user's terminal is disconnected from the session but is returned to the Cross Domain Facility (XDC-CDF) job selection panel. The user may then select another job to debug or issue the "END" command to return to the z/XDC Startup Panel.

Drop The user's terminal is disconnected from both the debugging session and from XDC-CDF entirely.

Newuser The user's terminal is disconnected from the debugging session. XDC-CDF then displays a logon panel from which a user may log back into XDC-CDF using a different TSO userid.

Remarks: If the user has connected to XDC-CDF from a TSO session, using the Newuser parameter does not affect the "old" TSO session.

Note: If you want to let the batch program run, AND, you want to disconnect from it, enter "**DISC;GO**". This will sever the connection, allowing you to go your separate ways. HOWEVER, A DISConnected batch job may persist even if "DISC;GO" has been issued. The best practice is to cancel the batch job if you really want things to "end and go away entirely".

Examples: DISC ← The user's terminal will be disconnected from the session, and will return to the XDC-CDF job selection panel.
DISC D ← The user's terminal will be disconnected from the session, and will leave the XDC-CDF environment completely.

More Info: Enter Help COmmands DISConnect. See also Help Debugging Batchjobs.

Chapter 7 - The Help Subsystem

Everything in z/XDC is documented in Dave Cole's excellent Help Subsystem. There are MANY pages in help, all contained in ISPF panels that are uniquely named, and accessible by minimum abbreviations. Along the way, he discusses z/OS architecture in Help, because he has to. z/XDC runs so low in the operating system that he HAS to explain these concepts. This means you can actually increase your knowledge of z/OS architecture as you work in Help. Dave says, "You're welcome!"

HOWEVER, if you don't know Dave's indexing structure you're going to get frustrated very quickly. That's why the LIST HELP command exists. It allows you to quickly drill down into the Help Subsystem.

Our William Cole recently put LIST HELP online at our website. Check this out: https://colesoft.com/help. Pretty cool, William.

If I were to document each and every portion of Help that would (again) defeat the purpose of this book. So let me make some helpful suggestions within our Help Subsystem:

List Help Allows you to see the indexes within the Help Subsystem. Drill in with an "L" shortcut command. Use "H" to drill into any sub-topic of interest.

Help DEbugging Tells you how to deploy our product in a variety of situations.

Help COmmands Access EVERY ONE of z/XDC's commands. We display commands relevant to c/XDC and dump/XDC as well.

Help COmmands <command-name> Drill down to get the best information on each command, what it does, what the parameters do and useful examples. For the LIST commands there is often a separate page detailing what the output of the command means.

Help SUpport How to reach us for support, how to send us dumps, etc.

Help SCripts You can create sequential files containing z/XDC commands and run them "in batch" by using our REAd command. If you want to streamline your work process, scripts are a great thing to look into. You'll find the explanation in Chapter 7 (Changing things).

Help MEssages Investigate any z/XDC Help Message. HOWEVER, please don't drill down into Help MEssages directly. That is WAY too cumbersome. Instead, whenever z/XDC displays a message, navigate to the left of it and enter an "H". That will take you straight to the message, bypassing a lot of typing on your part.

Chapter 8 - The Wrap-up

And that's basically it; all that I think that you need, and nothing that I think you don't. HOWEVER, z/XDC and it's other features (c/XDC and dump/XDC) go much, much deeper into the z/OS architecture because our tools were designed to work "down there".

If you get curious, and begin to explore z/XDC's capabilities the rewards will the yours. You'll get more work done in less time, and at a high level of excellence. You may realize at the end of an effort that ONLY z/XDC was able to get you "this far". But it does take your active curiosity and willingness to try z/XDC's many powerful commands.

When z/XDC is running APF-authorized and IF your security system allows you, you may range freely across the address space or the system itself. I urge you to BE CAREFUL when you work in such environments because z/XDC gives you the power to do many things, including destroying a running OS!! For this reason we urge our customers to run z/XDC ONLY in development or test environments - never a production environment (unless the user is very, very knowledgeable).

If I just scared you, that was entirely on purpose. z/XDC is like a motorcycle or a chain-saw.

Cole Software is a tiny company dedicated to one thing: Helping knowledgeable programmers to be more productive in demanding and complex environments. We TOTALLY support what we offer, and we encourage you to contact us whenever you have a question.

First level support: Bob Shimizu - bob@colesoft.com. Also often available at (800) 932-5150 when I'm not in the field.

Second Level Support - Dave Cole, Frank Chu and Calin Cole all monitor support@colesoft.com.

- Our corporate telephone number is (540) 456-8210.
- Our website is at https://www.colesoft.com
- Interesting pages there:
- Documentation: https://www.colesoft.com/support/zxdc-release-z22
- Training videos: https://www.colesoft.com/training/training-videos
- Contacting us: https://www.colesoft.com/contact

THANK YOU for being our customer.